

UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Realizing Practical LLM-assisted AI Assistant in the Semiconductor Domain

Permalink

<https://escholarship.org/uc/item/14x4v0zb>

Author

Zeng, Yueling

Publication Date

2024

Peer reviewed|Thesis/dissertation

University of California
Santa Barbara

Realizing Practical LLM-assisted AI Assistant in the Semiconductor Domain

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Yueling (Jenny) Zeng

Committee in charge:

Professor Li-C. Wang, Chair
Professor Kerem Camsari
Professor Luke Theogarajan
Professor Zheng Zhang

June 2024

The Dissertation of Yueling (Jenny) Zeng is approved.

Professor Kerem Camsari

Professor Luke Theogarajan

Professor Zheng Zhang

Professor Li-C. Wang, Committee Chair

June 2024

Realizing Practical LLM-assisted AI Assistant
in the Semiconductor Domain

Copyright © 2024

by

Yueling (Jenny) Zeng

To all the truth, goodness and beauty of this world.

Acknowledgements

I can still vividly recall the first day of arriving in Santa Barbara in the summer of 2014. As I traveled down the coast from LAX, the beautiful landscape of Southern California gradually came into view. Gentle breezes carried the scent of saltwater from the nearby Pacific Ocean, blending with the warmth of the Mediterranean climate. It was in this enchantment that the ten-year story of my college life began to unfold.

The four years of undergraduate studies were like a vibrant fantasy, filled with youthful recklessness, yet lingering with delightful surprises and cherished memories. What followed was the unexpected turning point of deciding to continue with a PhD program. Even now, I still wonder what kind of magical forces led me down this rabbit hole.

If the undergraduate years were like painting freely on a colorful canvas, then the PhD studies were like constructing a grand castle from the ground up. This construction was by no means a linear process. It was on a certain day when I suddenly realized that the highest tower of the castle had long been thrusting into the clouds. The spectacular and magnificent scale of this endeavor has undoubtedly left a profound mark on my life. Only in retrospect can I truly grasp the extent of what I've learned throughout this journey—not just in terms of the research subject, but also in becoming aware of the chaos that exists in our world while still appreciating its inherent beauty.

No single word can fully express my gratitude to my advisor, Professor Li-C. Wang, for his support and guidance throughout the journey. His non-stopping curiosity about the world and his passion for thinking have inspired his students to delve into great depths of research, often immersing themselves in the realms of innovation and creativity. I will surely miss our discussions, where we would brainstorm topics ranging from AI to quantum mechanics to the wisdom in Buddhism. These vibrant conversations would often extend several hours, with no one paying attention to the passage of time.

He wholeheartedly takes the responsibility of supporting his students in all aspects of their lives, guiding them through every milestone toward graduation, assisting them in seeking promising career opportunities, and ensuring their happiness both in the present and for the future. It was under his support that my PhD journey was undoubtedly a pleasant experience, where I had the freedom to focus on my personal development, which naturally culminated in the attainment of the PhD degree.

I want to extend my gratitude to all my co-authors, reviewers, professors, and university staffs who have supported me throughout my academic journey. In particular, I want to express my deep appreciation to my lab mates, Min Jian Yang, Thomas Ibbetson, and Matthew Dupree, for their contributions to completion of this thesis. Min and Matthew played pivotal roles in developing the backend scripts and knowledge graph in IEA-Plot, as well as conducting extensive experiments. Thomas took the lead as the primary investigator in the entity extraction work detailed in Chapter 8.

This journey was also accompanied by my beloved families and friends. Thanks to my mom and dad for their enormous support, for sending me overseas to explore the world, for respecting all my decisions, and for always providing me a warm shelter whenever I wanted to be home as if I were a little girl again. Thanks to my roommates who brought great joy to my leisure moments and comforted me during the tough times. And thanks to my cohorts and friends—from the States to China and beyond—who I know will always be there for me, now and forever.

I believe this acknowledgment only scratches the surface of the invaluable people and experiences I am grateful for. As I reflect on these memories, vividly flashing before my eyes, I gaze out my window to see the sunshine piercing through the mist and fog, a typical early summer morning in Santa Barbara. In this very moment, I am filled with certainty that today will be another bright day, as will tomorrow and all the days to come.

Curriculum Vitæ

Yueling (Jenny) Zeng

Education

- 2020 - 2024 Ph.D. in Computer Engineering,
University of California, Santa Barbara.
- 2018 - 2020 M.S. in Computer Engineering,
University of California, Santa Barbara.
- 2014 - 2018 B.S. in Electrical Engineering,
University of California, Santa Barbara.

Publications

1. Wang, LC., **Zeng, Y.** (2023). Machine Learning Support for Wafer-Level Failure Pattern Analytics. In: Girard, P., Blanton, S., Wang, LC. (eds) Machine Learning Support for Fault Diagnosis of System-on-Chip . Springer, Cham.
2. M. Dupree, M. J. Yang, **Y. J. Zeng** and L. -C. Wang, “IEA-Plot: Conducting Wafer-Based Data Analytics Through Chat,” *2023 IEEE International Test Conference (ITC)*, Anaheim, CA, USA, 2023, pp. 122-131, doi: 10.1109/ITC51656.2023.00028.
3. **Y. J. Zeng**, M. J. Yang and L. -C. Wang, “Wafer Map Pattern Analytics Driven By Natural Language Queries,” *2022 IEEE International Test Conference in Asia (ITC-Asia)*, Taipei, Taiwan, 2022, pp. 31-36, doi: 10.1109/ITCA55616.2022.00016.
4. M. J. Yang, **Y. Zeng** and L. -C. Wang, “Language Driven Analytics for Failure Pattern Feedforward and Feedback,” *2022 IEEE International Test Conference (ITC)*, Anaheim, CA, USA, 2022, pp. 288-297, doi: 10.1109/ITC50671.2022.00037.
5. **Y. J. Zeng**, L. -C. Wang and C. J. Shan, “MINIature Interactive Offset Networks (MINIONs) for Wafer Map Classification,” *2021 IEEE International Test Conference (ITC)*, 2021, pp. 190-199, doi: 10.1109/ITC50571.2021.00027.
6. **Y. J. Zeng**, L. -C. Wang, C. J. Shan and N. Sumikawa, “Learning A Wafer Feature With One Training Sample,” *2020 IEEE International Test Conference (ITC)*, 2020, pp. 1-10, doi: 10.1109/ITC44778.2020.9325254.
7. L. Liang, L. Deng, **Y. Zeng**, X. Hu, Y. Ji, X. Ma, G. Li, Y. Xie, “Crossbar-Aware Neural Network Pruning,” in *IEEE Access*, vol. 6, pp. 58324-58337, 2018, doi: 10.1109/ACCESS.2018.2874823.

Preprints

1. **Yueling Zeng**, Li-C Wang and Thomas Ibbetson. 2024. “Per-Sample Examination of LLM’s Responses by Treating LLM as an Oracle,” under-review by NeurIPS 2024.

2. M. J. Yang, **Y. J. Zeng** and L. -C. Wang, “WM-Graph: Graph-Based Approach for Wafermap Analytics,” under-review by ITC 2024.
3. **Yueling Zeng**, Li-C Wang and Thomas Ibbetson. 2024. “Oracle-Checker Scheme for Evaluating a Generative Large Language Model”. arXiv preprint arXiv:2405.03170 (2024).
4. **Yueling Zeng** and Li-C Wang. 2023. “Domain Knowledge Graph Construction Via A Simple Checker”. arXiv preprint arXiv:2310.04949 (2023).

Best Paper Awards

1. “Language Driven Analytics for Failure Pattern Feedforward and Feedback”, *IEEE International Test Conference (ITC)*, 2022
2. “Learning A Wafer Feature With One Training Sample”, *IEEE International Test Conference (ITC)*, 2020

Other Awards

1. **IEEE International Test Conference 2023 Gerald W. Gordon Student Award:** The Gerald W. Gordon Award is a certificate presented annually to a current student in good standing at an accredited university or college for credible service to IEEE test technology related activities, programs and organizations, given by IEEE Computer Society at IEEE International Test Conference..
2. **ECE Department - Outstanding Teaching Assistant Awards:** The recipient was recognized for the outstanding service and dedication to student success in UCSB’s Computer Engineering department, received in three academic quarters for Winter 2019, Spring 2021, and Winter 2022.

Work Experience

- | | |
|-------------------|---|
| 2023.06 - 2023.09 | Machine Learning Engineer Intern,
LinkedIn Corporation, Sunnyvale, CA. |
| 2022.06 - 2022.09 | Machine Learning Engineer Intern,
LinkedIn Corporation, Sunnyvale, CA. |
| 2021.06 - 2021.09 | Data Scientist Intern,
NXP Semiconductor, Austin, TX. |
| 2020.06 - 2020.09 | Data Scientist Intern,
NXP Semiconductor, Austin, TX. |

Abstract

Realizing Practical LLM-assisted AI Assistant in the Semiconductor Domain

by

Yueling (Jenny) Zeng

The emergence of Large Language Models (LLMs) offers new opportunities for applying Machine Learning (ML) and Artificial Intelligence (AI) in semiconductor chip design and test (D&T). Realizing these opportunities requires a fundamentally different thinking from the past. For more than two decades, the semiconductor industry has been exploring applications of ML in D&T. Despite many promises, notable challenges remain in most of the application contexts.

The first part of the thesis (Chapter 2, 3 and 4) includes a review of works for applying ML in D&T, starting in 2003, and describes the journey leading to the current development of an AI Assistant called Intelligent Engineering Assistant (IEA). The journey evolved from one view to another, where each view perceived applying ML in D&T differently. In the first decade, the research took a *data-driven view* similar to that in common ML practices. This view was changed to a *knowledge-driven view* in 2014, due to the experience of solving a production yield problem for an automotive chip supply company. This experience is highlighted in the thesis, together with learning lessons from a variety of other works in the first decade.

The knowledge-driven view then lasted for four years. During the period, the research focused on finding ways to incorporate domain knowledge into the data learning process. It was in this period, the idea of Co-ML (complementary ML) first emerged. Co-ML formulates a ML problem as a decision problem where the outcome of the learning can

be either a model (an answer) or no model (no answer). Then, in 2018 the idea of IEA, as an autonomous system, was first construed. This changed the knowledge-driven view to an *autonomous system view*.

In 2022, the autonomous system view was once again revised. It was realized that in order to achieve a practical IEA, one had to take a fundamentally different view from the past and perceived problem and solution *as a pair*, rather than perceived problem as given for finding a solution. We call this thinking the *problem-solution dual view* (Chapter 5).

Under our problem-solution dual view, applying ML in D&T is no longer seen as “static” in the sense that for a given problem, there is an ML tool for it. It is seen as a data exploration process, a search process driven by user, where each search step comprises a pair of problem instruction and problem solver. Consequently, there are two requirements for an IEA: (1) to provide a language for specifying problem instructions and (2) to provide a software platform capable of solving each acceptable problem instruction. This novel IEA thinking was realized in our first end-to-end IEA in 2022 (IEA-2022) in the application context of wafermap analytics.

The development of IEA-2022 preceded the release of ChatGPT. At the time, IEA-2022 utilizes its predecessor GPT-3 model, only in a restricted way because of the limitations of the LLM. Then, the release of ChatGPT and its later models fundamentally changed design of IEA again (Chapter 6). In the latest IEA, called IEA-Plot, a *knowledge graph* (KG) is in place as the central piece to connect problem instruction to problem solver (Chapter 7). With a powerful LLM, the problem instruction can therefore be given in natural language. The instruction is then *grounded* by the KG internal to IEA in order to find a matching solution based upon a collection of solvers in the backend of IEA. IEA-Plot, again focusing on wafermap analytics, was demonstrated based on test data collected from several product lines. In the thesis, four chapters are devoted to

discuss the development of IEA-Plot which is the central piece of this thesis work.

IEA-Plot is the first step moving forward to build a practical LLM-assisted AI Assistant in the semiconductor domain. There is one essential issue with the development of IEA-Plot: the construction of the KG. This motivated us to explore the possibility of using an LLM to assist the development of KG. Furthermore, in the current IEA the domain knowledge is stored explicitly in the KG. The LLM is used off-the-shelf. This raises the question whether or not it is possible to ingest the domain knowledge into an LLM and remove the dependency on KG. The last part (Chapter 8) of the thesis will touch base on these two aspects. In particular, we present a novel idea called *oracle-checker scheme* (OC scheme) for utilizing an LLM by treating the LLM as an *oracle*. Findings for using LLM for KG development are summarized. Then, in the last Chapter 9 before the conclusion we paint a picture for how to ingest domain knowledge into an LLM by taking a generative AI approach.

While IEA-Plot is at the center of this thesis, it should not be seen as a standalone invention. IEA-Plot is a direct consequence from two decades of research on trying to apply ML in D&T. It exemplifies how to build an LLM-assist AI Assistant in practice. It marks the end of a two-decade journey and opens a new one toward generative AI.

Contents

Curriculum Vitae	vii
Abstract	ix
List of Figures	xv
List of Tables	xx
1 Introduction	1
1.1 An Example of Engineering Work Content	4
1.2 The Search Process	11
1.3 Reflection in Other Contexts	13
1.4 Ten Points from ML to Our AI Assistant	20
1.5 Conclusion	31
2 Journey to IEA - The 1st Decade	33
2.1 Data-Driven View (2004 – 2013)	35
2.2 Design-Silicon Timing Correlation	37
2.3 Speedpath Analysis	44
2.4 RTL Functional Verification	45
2.5 Customer Return Analysis	55
2.6 Other Areas to Apply Machine Learning	64
2.7 Lessons Learned from the First Ten Years	72
2.8 A Chronological Remark for the First Ten Years	78
3 Journey to IEA - The 2nd Decade	81
3.1 Knowledge-Driven View (2014 – 2017)	81
3.2 Modeling domain knowledge with an executable workflow	83
3.3 Plot-based analytics	84
3.4 Learning Domain Knowledge in Functional Verification	92
3.5 Three Challenges Motivating Language-Driven Analytics	96
3.6 Co-ML Capabilities	98

3.7	Monomial Learning	102
3.8	Uniqueness in View of Occam’s Learning	104
3.9	Local No-Free-Lunch (L-NFL)	106
3.10	Autonomous System View (2018 – 2021)	117
3.11	The Latest Three Views (2022, 2023, 2024)	120
3.12	DSML in View of Computational Complexity	121
4	Wafermap Analytics	127
4.1	Yield Excursion	128
4.2	A Yield Excursion Example	130
4.3	ML View to Wafermap Analytics	136
4.4	Types of Analytic Questions	145
4.5	From ML Classifier to DSML Oracle	147
4.6	Concept Recognition	149
4.7	The MINIONs Approach	155
4.8	Experiment Results	170
5	Problem-Solution Dual View	173
5.1	Dual View of DSML	174
5.2	Language-Driven Analytics	179
5.3	IEA 2022	183
5.4	Connecting LLM via Semantic Parsing	184
5.5	Implementation of Backend API	196
6	IEA 2023	214
6.1	Introduction	214
6.2	Use of Knowledge Graph	220
6.3	Wafermap Analytics in IEA-Plot	221
6.4	Analytics Driven by a Dialog	225
6.5	Frontend Parser for Task Grounding	226
6.6	A Remark about IEA-Plot	232
7	Knowledge Graph	234
7.1	Development of the KG in IEA-Plot	235
7.2	The Importance of Having the First KG	247
7.3	A Formalism Regarding the KG in IEA	248
7.4	Toward Automating KG Construction	252
8	Oracle-Checker Scheme	268
8.1	Per-Sample Examination of LLM’s Responses	269
8.2	Realizing GH-check In Entity Extraction	273
8.3	Realizing GI-check In Paraphrase Decision	279
8.4	Experiments	285

8.5	Related Work	295
8.6	Limitations	296
8.7	Summary	296
9	Journey from IEA - The Next Decade	297
9.1	The Generative AI View	298
9.2	The Oracle-Checker View	303
10	Conclusion	305
10.1	Takeaways	306
10.2	Ten Questions And Their Answer	308
10.3	Philosophical Remarks	310
	Bibliography	311

List of Figures

1.1	Application space developed from two perspectives	1
1.2	Finding a correlation between E-test and wafer probe failure	6
1.3	Yield distribution as a density plot based on 2000+ wafers	7
1.4	Yield improvement in silicon validation of recommended process adjustments	10
1.5	The illustration of a "Choose-and-Bound" search process	11
1.6	Reflecting Choose-and-Bound search in a functional verification context . .	13
1.7	Reflecting Choose-and-Bound search in a speedpath analysis context	16
1.8	Reflecting Choose-and-Bound search in an outlier analysis context	17
1.9	An end-to-end training view of machine learning	20
1.10	ML view vs. Choose-and-Bound view to approach the yield problem	21
1.11	Benchmark thinking vs. Specification thinking	21
1.12	Objective machine view vs. subjective human views	23
1.13	The task of ML and Co-ML perform on a given dataset	25
1.14	Contrasting to Figure 1.9, an AI Assistant is to help find a simple answer .	26
1.15	Drawing a line to separate the knowledge to be captured in an assistant . .	28
1.16	An AI Assistant works in two worlds: Free-Lunch and No-Free-Lunch	29
1.17	The data generator is evolving over time	29
1.18	Building an automation flow vs. building an assistant	30
1.19	IEA-Plot: from vague problem description to concrete solution	32
2.1	Evolution of views along the journey to IEA	35
2.2	10 categories of works carried out under the data-driven view	35
2.3	Data learning based diagnosis as summarized in [1]	38
2.4	Transduction diagnosis vs. traditional diagnosis, as explained in [1]	40
2.5	A rule learning example, reported in [2]	41
2.6	A simple dataset example to illustrate over-fitting	43
2.7	Manual learning in the context of functional verification	45
2.8	Proposed automatic learning in the context of functional verification	46
2.9	A notable result on improving simulation efficiency, later reported in [3] . .	50
2.10	The fundamental issue of kernel-based learning in functional verification . .	50
2.11	Two learning points in a functional verification environment	52

2.12	A notable result on improving verification coverage, reported in [4]	54
2.13	A notable result on improvement from zero coverage, report in [5]	54
2.14	Justifying an outlier model with multiple CQIs	56
2.15	Justifying an outlier model with domain knowledge	57
2.16	CQI model effectiveness shown in time view, reported in [6]	58
2.17	Being an outlier does not imply being abnormal [7]	59
2.18	The reason why we need a 2nd CQI to justify a CQI outlier model [7]	60
2.19	The reason why in test, one prefers an outlier model based on correlated tests [7]	61
2.20	For outlier screening a simple co-variance model might suffice [7]	63
2.21	Seeing a D&T problem as a ML problem	72
2.22	Seeing a D&T problem with a more realistic big picture	73
2.23	A chronological view of the first ten years (2004-2013)	79
3.1	Two types of domain knowledge to drive the search	83
3.2	Modeling domain knowledge as a workflow based on a plot-based view	84
3.3	Plot-based view to enable definition of primitive analytic steps	84
3.4	Plot-based view to enable definition of primitive steps in a workflow [8]	85
3.5	Summarizing the main idea in [8]	86
3.6	An example to illustrate process discovery [8]	87
3.7	Illustration of the issue yield addressed in [8]	89
3.8	The process learned from the work in [9], reported in [8]	91
3.9	Findings to explain the yield issue in Figure 3.7, reported in [8]	91
3.10	Constrained process discovery for functional test generation in [10]	95
3.11	A-KCCA risk evaluation on a target process parameter PP [9]	99
3.12	Many tests showing no outlier based on the minimum consistent threshold [11]	102
3.13	Solving the monomial learning problem in practice, as suggested in [12]	105
3.14	A no-free-lunch situation in the context of outlier screening	106
3.15	Local No-Free-Lunch (L-NFL) when learning from production data	107
3.16	Measuring applicability of an outlier model	109
3.17	Over-fitting illustrated in terms of model complexity	111
3.18	Illustration of Occam’s learning in DSML	112
3.19	“Learning” in traditional ML vs. “learning” in DSML	113
3.20	The original IEA proposed in 2018 aimed for an autonomous system	117
3.21	An optimization view in DSML	122
4.1	(a) Show the failure pattern constituting a yield excursion; (b) Show the number of impacted wafers over weeks of the production [13]	130
4.2	Result after using method in [13] and evidence from Failure Analysis report	134
4.3	Eight pattern classes and one “None” class in the WM-811K dataset	136
4.4	Two common ML approaches to solve a multi-class image classification problem	137

4.5	Examples of mistakes in Table 4.2	141
4.6	Two-step classification employed by the two trained VGG models [14] . . .	143
4.7	Examples of questionable classification (the shown labels were reported by the model and were considered questionable in the manual review) [14] . .	143
4.8	ML view vs. DSML view to approach wafermap analytics	145
4.9	Within-class pattern variations seen in WM-811K dataset	147
4.10	An example of pattern class varying within a single lot in WM-811K dataset	148
4.11	A DSML oracle provides decision support at the concept level	151
4.12	High-level idea of the MMINIONs approach	159
4.13	An example connected component extracted from a MINIONs recognition graph	160
4.14	Different ways to attain one-shot learning	161
4.15	The idea of manifestation learning presented in [15]	162
4.16	Visualization of latent space learned with MNIST samples in [15]	164
4.17	Codebook mapping in MINION training with augmented training data. . .	166
4.18	The neural network architecture of a MINION model.	167
4.19	The two parts in loss function for training a MINION model.	168
4.20	An example of finding a problematic lot	170
4.21	An example of searching for a similar lot based on a given wafermap	172
5.1	Problem and solution dual view in DSML	174
5.2	Search flow driven by user inputs to achieve a high-level analytic goal . . .	179
5.3	NLP is a necessary component for interpreting the analytic question	182
5.4	Overview of IEA 2022	183
5.5	A semantic parser interfacing with natural language queries	184
5.6	The workflow from queries to a summary plot	185
5.7	Constrained parsing from user query to meaning representations with LM	192
5.8	Merged summary plot of Table 5.2 and Table 5.5	193
5.9	An example of Minions' recognition graph on wafer maps	196
5.10	Attaining describable sets through NLI	197
5.11	An example of cluster and <i>cluster core</i>	199
5.12	The parsing tree for one wafer map in Figure 5.11. The wafer map is described as: " Null length thin thickness arc type at 12 o'clock direction null spread along edge extend to ... ", and at high level (using the high-level descriptor " something "), can be captured as: " something at 12 o'clock direction null spread along edge extend to ... "	201
5.13	The definition of REGION on the wafer map	205
5.14	Flow to investigate a correlation between wafer probe and final test where each try starts with a simple query to find a group of wafers.	211
5.15	Finding of a pattern trend lasting over three periods	212
5.16	E-test correlation plot based on wafer maps describable by "cluster fails at direction from 11 o'clock to 12 o'clock along edge"	212

6.1	Dialog with ChatGPT (03/26/2023) based on questions for how to correlate wafer map patterns to E-test parameters	215
6.2	Overview of IEA 2023	215
6.3	Task grounding problem: How to confine model responses within the scope of an embodiment with admissible actions $\{a_1, a_2, \dots, a_n\}$?	216
6.4	Using a knowledge graph to connect LLM and admissible actions	219
6.5	A user query corresponds to a subgraph in the KG	221
6.6	Identifying a pattern group satisfying a pattern Concept	222
6.7	Results based on the Center pattern concept	223
6.8	IEA-Plot outputs based on the two consecutive queries	224
6.9	A dialog example and IEA-Plot’s outputs shown in Figure 6.10	225
6.10	IEA-Plot output screenshots for queries listed in Figure 6.9, screenshots for Q4 and Q5 displayed in Figure 6.8 previously	226
6.11	Generating acceptable queries using a GPT model [16]	227
6.12	An example to obtain a list of acceptable queries	227
6.13	Intent capture by comparing pairwise SBERT embeddings	228
6.14	Fine-tuning SBERT improves our intent capture	229
6.15	Phrase matching by comparing pairwise BERT embeddings	230
6.16	Examples of parsing a query into a KG configuration	231
6.17	LLM enables data exploration through dialog	231
7.1	Hierarchy in our knowledge graph design	236
7.2	Axioms in the domain graph, representing the ontology we use	237
7.3	A conceptual example to illustrate our graph construction	239
7.4	Semantic meanings of the six relations for subgraph management	241
7.5	Subgraph for “ <i>Show me the Arc pattern at 11 o’clock direction</i> ”	243
7.6	Intent Capture and Phrase Matching for parsing a query	244
7.7	A constraint graph is separately maintained in our KG	245
7.8	Extending tool functionality by adding nodes in KG	246
7.9	An DSML oracle decides on the value of a Boolean concept x	248
7.10	The DSML hierarchy where k can be treated as a constant	249
7.11	An example of KGC using a prompt to GPT3.5. The input is a paragraph from the RISC-V Spec and the output is in RDF TTL format.	252
7.12	Improved RDF by repeating the example in Figure 7.11 and supplying the <i>background facts</i> (BFs).	254
7.13	The KGC study is based on the oracle-checker view	258
7.14	Results of consistency check without BFs provided; ■ : Show the most consistent group in 10 repeated runs; ■ : # of runs that failed	261
7.15	Results of consistency check with BFs provided; ■ : Show the most consistent group in 10 repeated runs; ■ : # of runs that failed	262
7.16	Two prompts used in the entailment check	263

7.17	Results of entailment check for the two chapters of paragraphs; Each chart shows overlapping of two results from the runs without and with BFs provided; ■: showing % of RDF Facts passing the check where those Facts are obtained with no BFs provided; ■: showing % of RDF Facts passing with BFs provided; ■: With BFs provided, some Facts fail the check (mostly because of including the auxiliary entities not given in the paragraph) and are bypassed after manual review.	264
7.18	No correlation between the size of largest group from consistency check and the entailment check score	265
7.19	Chapter 1 (With BFs): there are 75 concepts shared by at least two paragraph and the total number of edges is 454; In the bipartite graph, the bottom dots each represents a paragraph from chapter 1 and the upper squares each represents a subject concept. More transparent the color indicates more edges are connected between the concepts and paragraphs.	267
7.20	Chapter 2 (With BFs): there are 82 concepts shared by at least two paragraph and the total number of edges is 388; In the bipartite graph, the bottom dots each represents a paragraph from chapter 2 and the upper squares each represents a subject concept.	267
8.1	Realizing the two theoretical checks in practice by treating GPT-3.5 as an oracle.	271
8.2	Illustration of the reference function h , the entity extraction g , and the function-under-check f , with an example starting with E_s containing four extracted entities from original sentence s	275
8.3	Two ways for realizing GI-check (yes and no claims separately) in paraphrase decision	279
8.4	Probabilistic check by querying the oracle: $\pi_i(h) = g_1$ or $\pi_i(h) = g_2$?	281
8.5	Finding candidates (U, V) . μ_1 to μ_4 form a decomposition of s_a , and $(\mu_i \rightarrow \nu_i, i = 1 \dots 4)$ are the matching pairs, so the checker can ask the LLM the four questions: if $\mu_i \equiv \nu_i, i = 1 \dots 4$	282
9.1	IEA-Plot is a query generator	299
9.2	MINIONs with a Natural Language Interpreter (NLI) as a wafermap generator	300
9.3	With a wafermap generator, retrieval of wafermaps according to a user description can be made simpler — similarity search achieved (and simplified) by a generative approach	300
9.4	Does a retrained LLM provide generalization from A to B?	301
10.1	Evolution of views along the journey to IEA	306

List of Tables

2.1	The number of possible outlier models can grow substantially [7]	63
3.1	Number of H-paths included in the learned model for each i -prefix rule [8].	90
3.2	A simple Boolean learning L-NFL example	108
4.1	Labeled wafer maps from the WM-811K dataset, used in [14]	140
4.2	Confusion Matrix (On All 967 Wafer Maps): ⇒: Given Label, ⇓: Predicted Label	141
4.3	Classification on the 741 unlabeled wafer maps	143
5.1	A snapshot of the wafermap database table	186
5.2	Example queries, their parses and the generated plot	188
5.3	A snippet of the grammar and lexicon	190
5.4	A snippet of operators and their expressions	191
5.5	Second result	194
5.6	Third result	195
5.7	The lexicon for the formal language \mathcal{L}_0	202
5.8	The Grammar for \mathcal{L}_0	204
5.9	Salient wafer maps from a cluster core and their canonical utterances . . .	208
5.10	Examples of wafer grouping	209
5.11	Examples of wafer pattern interpretation	210
8.1	(1) GH-check on RISC-V sentences fails more than DOCRED sentences (51.6% vs. 20.46%). (2) GPT3.5 extracted about 70% ($= \frac{8830+2349}{15931}$) of the human-labeled entities and also extracted 7457 not labeled as entities in DOCRED.	289
8.2	(1) In-sample acceptance rate (A_{rate}) is correlated to the in-sample consistency measure (Con_{rs}), and not as correlated to $ E_s $. (2) Cross-sample correlations: Con_o is the % of consistency on the original sentence. Con_{rs} is the % of smallest consistency on the 15 modified sentences. A_{rate} is the % of passing out of the 11^3 combinations.	290

8.3	GPT3.5 claims compared with MSRP labels. * indicates GPT disagreements with the labels. GPT3.5 disagrees on 25.16% of the human labels in MSRP.	292
8.4	% of provable “Yes” cases is up to 57%.	292
8.5	On 500 GPT3.5’s own generated paraphrases, GPT3.5 answers “Yes” on 495 and their provable % is shown below.	292
8.6	Show % of those “No” cases, where our checker found a proof for their semantic equivalence.	293
8.7	On 1832 GPT3.5 claimed “No” cases, show % rejected by the checker in two scenarios: <i>with</i> (“w/”) or <i>without</i> (“w/o”) using an indifferentiable paraphrase p (using WP or SP)	294

Chapter 1

Introduction

十年磨一劍，霜刃未曾試。今日把示君，誰有不平事。

Ten years sharpening a sword, its frosty blade untested. Today, I present it to you, to whom does impediment persist.

— 《劍客》 A poem from Tang Dynasty



Figure 1.1: Application space developed from two perspectives

The application space of an emerging technology can be viewed from two distinct perspectives: the technology provider perspective and the technology adopter perspective (Figure 1.1). For a technology provider, their interests include exploration of new applications that can be enabled by the technology. For a technology adopter, their interests often lie in leveraging the technology, for example to make improvements in an existing application context. Exploring a new application and improving an existing application involve different considerations and constraints. The focus of this thesis is

on the latter, in particular, on adopting the latest technologies in Machine Learning in application contexts existing in the semiconductor industry.

Machine Learning (ML), including the broader domain of Artificial Intelligence (AI), is one of the fast-growing technological areas since the breakthrough performance of deep learning in image-related tasks in 2012 [17, 18]. The launch of ChatGPT in late 2023 [19, 16], powered by a type of large language model (LLM) based on deep learning, with its ubiquitous use in natural language dialogues, has set off a new wave of technological advancements. Generative AI (genAI) [20, 21], a term denoting the AI's capability of generating text, images videos, or other data in general, has become one of the industry's most trending words. This is evidenced by substantial investments from industry giants such as Google, Microsoft, and Meta etc., along with a funding topping \$21.8B across 426 deals for investment in generative AI startups in the year of 2023 [22]. This commercial booming in genAI has been driving a dramatic exploration in the breadth of its applications. A report from Deloitte [23] listed 60 example use cases of genAI, assisting applications in various areas including legal, healthcare, finance, and human resource etc. The particular use of LLMs as fundamental controllers for autonomous AI agent has also become an active R&D area, with the number of papers published growing exponentially each year [24, 25].

On the other hand, applying ML and AI in semiconductor design and test flows has been an important area of R&D within the semiconductor industry for decades. There are many potential applications of ML/AI technologies in design and test, which can be categorized into modeling, simulation, optimization, debugging, diagnosis, and so on. Recently, there has been a growing interest in leveraging the power of LLMs in the semiconductor industry. For example, researchers from Nvidia has explored the applications of LLMs for industrial chip design [26]. The work [26] adopted various domain adaptive fine-tuning techniques for enabling three selected LLM applications for chip design: an

engineering assistant chatbot, EDA script generation, and bug summarization and analysis. A recent article [27] featured insights from multiple senior executives at leading EDA vendors regarding the utilization of LLMs in electronic design automation (EDA). LabVIEW from National Instruments, a graphical programming software largely used in automated test system development, has witnessed a transformative shift with LLM-powered tools like Github Copilot, an AI code developing tool capable of converting verbal descriptions or even rough sketches into functional code [28].

In view of LLMs, the two perspectives in Figure 1.1 can be more precisely defined as the following. A technology provider has ownership of the LLM and has the resources to train the LLM model. As a result, they can make an LLM model to fit an application context by developing an appropriate dataset and by training or finetuning the LLM model. In contrast, a technology adopter does not own an LLM model nor has the ability or resources to train or retrain an LLM model. They take an LLM model as it and try to utilize the model in an application. For example, this usage can be through a web service provided by another company. The adopter has no access to the model itself nor has the ability to alter the model.

In this thesis, we consider building an LLM-assisted AI assistant from the perspective of a technology adopter. In other words, the LLM model is given as it. The application space we consider includes engineering work contexts existing in a semiconductor company: We are interested in building an AI assistant to assist engineers working in the semiconductor company. We give a special name of our AI assistant and call it Intelligent Engineering Assistant, or IEA. For building a practical IEA, the trust aspect is a crucial concern. This is because engineers often have much more stringent requirements for an AI assistant than casual users. If an engineer cannot trust the AI assistant and is frequently required to debug its work or check its result, then the utilization of the assistant can diminish, thus reducing its practical value.

While our focus is on leveraging the latest LLM development, it is worth noting that despite decades of efforts trying to apply ML and AI technologies, the semiconductor industry has not observed a similar pace of expansion in the application space of ML and AI as those seen in some other industries. In this introduction chapter, we will point out the fundamental barriers that impede the adoption of ML/AI technologies in the semiconductor industry. These barriers will be seen as the *gaps* between ML and practicing ML in domain-specific applications. Semiconductor industry demands solutions beyond standard ML practices. In this thesis, we will describe IEA as a framework for building a domain-specific AI Assistant. Key innovative components will be elaborated to explain the design of IEA for closing the aforementioned gaps. On top of that, we will explain why building an IEA is a natural way to realize a ML/AI solution to address those domain-specific needs, and more importantly the design of IEA demonstrates why LLMs play a key role for building a domain-specific AI Assistant.

1.1 An Example of Engineering Work Content

To build an AI assistant to help an engineer, we begin by understanding the work content of the engineer. In this section, we discuss an example to illustrate a type of work content in a semiconductor company.

1.1.1 Optimization of yield

For a semiconductor company, the *yield* is an important figure to optimize, which directly affects the profitability of a production line. Yield is defined as the ratio of the total number of good chips over the total number of manufactured chips. Deciding if a chip is good (passing) or bad (failing) is done by testing. In a typical test flow, *wafer probe*, which tests silicon dies on wafers, is the first stage of testing, followed by *final test*

which tests standalone packaged chips, and *system-level test* which tests packaged chips in a system. A semiconductor design company can carry out the test process internally or provide the test content to contract a specialized test service company to run the process. In either case, test data is collected and made available for engineers to analyze.

It is important for a semiconductor company to analyze the test data throughout the test flow in order to understand and improve their yield. Yield optimization typically starts by understanding the failures exhibited on the failing chips¹ In testing, failing chips are sorted according to a list of pre-defined *failure bins*. If a failure bin includes a large number of chips, it is desirable to find ways to reduce the number and thus improve the yield. Failure bins can be grouped according to the tests performed, for example, those used to test some leakage characteristics of an analog block can belong to one test group. There are three areas of changes to improve the yield: changing the test itself, changing the design, or changing the manufacturing process.

For a fabless company, the manufacturing process is operated by a foundry company. To change the process, the foundry company needs to be convinced that the cause for the yield issue is due to the manufacturing process. In other words, the foundry needs to see a clear evidence before taking an action to adjust their process.

One type of such evidences can be presented as a correlation between an *E-test* parameter and a type of failure in wafer probe, measured across wafers produced over time, as shown in Figure 1.2. An E-test is a specialized measurement provided by the foundry to measure some characteristic of the manufacturing process. Collectively, E-tests intend to measure the health of an entire wafer. The measurements are usually carried out on multiple *sites* on a wafer and on each site, the measurements can include a large number

¹We use the term “failing chips” rather than “bad chips” because a chip failing the test does not imply the chip is bad for sure. For example, the failure can be due to an issue in the test itself. To be more precise, we will use the term “failing chips” throughout the thesis. When we use the term “bad chip”, we refer to a *confirmed* bad chip, e.g. confirmed by failure analysis (FA) on the silicon chip.

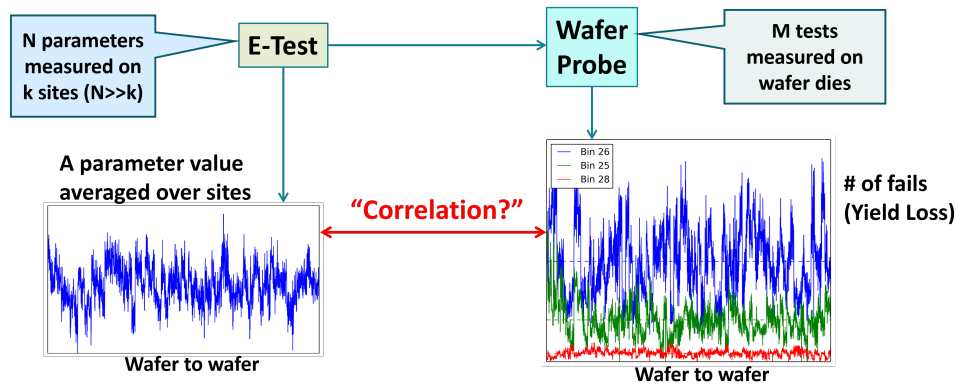


Figure 1.2: Finding a correlation between E-test and wafer probe failure

of E-tests. A strong correlation from a failure type to an E-test parameter can indicate the impact of some wafer-to-wafer process variations to the particular type of yield loss.

1.1.2 A success story

The story is about a real yield optimization task as reported in [9]. The task is about finding strong correlations between E-tests and wafer probe failures.

In 2013, a team at Freescale Semiconductor was trying to solve a yield problem that significantly affected a product line already in high-volume production. The product was an automotive SoC (System-on-Chip), more specifically a tire pressure monitor sensor (TPMS). A TPMS constantly measures the pressure of a tire and transmits the measured value wirelessly to a central control unit mounted in the car. A TPMS is supposed to operate with extremely low power consumption, in order to have a long lifetime with its mounted battery. Hence, this extreme low-power constraint typically is a critical optimization objective for this type of design.

Before the yield problem was handed over to our lab, it had lasted for a long period. During this period, the product team had gone through one design revision, multiple test revisions, and even asking the contracted foundry to conduct several process adjustments.

Despite all these efforts, the yield problem persisted.

The yield problem is illustrated in Figure 1.3. The figure shows a density plot and the density is based on wafer yield numbers calculated over 2000+ wafers. Due to confidentiality, the x-axis has no label. It suffices to say that the yield was not satisfactory and significantly fluctuated from wafer to wafer. The desired outcome had two aspects, as shown in the figure: (1) we desired the entire yield distribution to shift right, and (2) we desired the width of the distribution to be minimized.

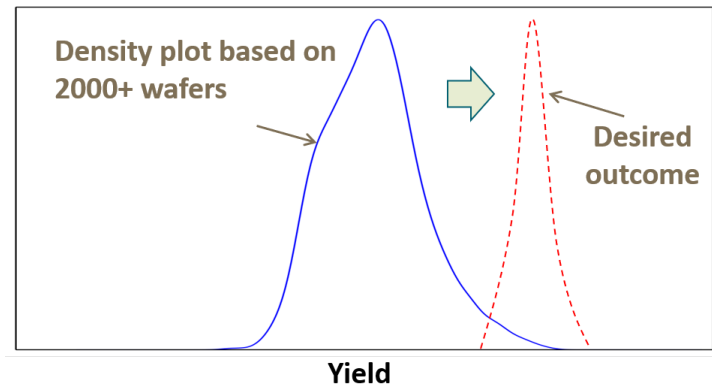


Figure 1.3: Yield distribution as a density plot based on 2000+ wafers

To our lab, the underlying problem was the same correlation problem as depicted in Figure 1.2 before. Such an analytic problem includes three aspects: the E-test data, the wafer probe data, and the correlation methods. It should be noted that other than the two types of data, our lab had no other data or further knowledge about the problem.

The E-test for this product had 130 process parameters measured on five sites on a wafer. The passing and failing chips were sorted into various test bins. As a common start, we could aggregate the value of an E-test parameter by taking average on the five sites. And we could use some standard statistical correlation methods such as Pearson correlation. Then, the problem could be specified as “finding a strong Pearson correlation between an average E-test value and a test bin”. Note that this correlation could be for

a passing bin or failing bin.

If we were lucky, the simple problem formulation could lead to a high correlation result and we would not need to pursue further. However, the reality was that the simple problem formulation was far from being sufficient. As reported in [9], the best correlation found with this simple formulation was merely 0.463. This correlation was not considered as a strong evidence, specifically, not meaningful enough for the foundry to be willing to make an adjustment to their manufacturing process.

It is important to see that the simple formulation is just one of many ways to formulate the analytic problem. For example, we could consider a more fine-grained view of the E-test data by considering measurements on each site individually, or collectively on a subset of sites based on their physical proximity on the wafer.

On the wafer probe side, we could also consider failures based on a specific test rather than a test bin (note: a test bin can be based on multiple tests). Further, we could calculate the failure number by focusing on a particular wafer region. Wafers were grouped in *lot* throughout production, so we could also introduce a lot-based view in addition to the wafer-based view. Given that the wafers were produced and tested over time, it was possible that a correlation itself could fluctuate over time. In other words, while the correlation might be low when considering an entire period, it could be high when examining specific windows within that period. These windows could be defined based on the production time or the test time.

The various aspects listed above provided choices in the formulation of an analytic problem. The combinations of those choices could render an enormous space of potential problem formulations. Therefore, the underlying challenge to solve the yield problem was to navigate in this space, form a specific and concrete *search space*, and to reach a satisfactory answer in the search space (if the search space contains the answer).

It is important to point out that satisfaction of an answer was subjective. Different

people involved might have a different requirement for this satisfaction. Hence, a “high correlation” result did not mean that the result would be automatically accepted by multiple parties. Our job was to discover results “interesting enough to be presented to the product team”. If the product team could be convinced by the meaningfulness of a result, they would then present the result to the foundry. Keep in mind that both the product team and the foundry team were organizations with a hierarchical management structure. Hence, one could imagine a *communication chain* from us as one end all the way to the decision makers in the foundry side as the other end. In other words, as the data analyst working in one end of this communication chain, our real job was to present results that were *likely to get through the chain and convince the decision maker to produce an action on the other end*. Under this realistic view, it is not hard to see that deciding if a correlation was high enough or not, involved judgment calls.

To reach a satisfactory answer quickly in a given search space, one needs two strategies: one to efficiently identify where the answer might locate and the other to efficiently decide where is unlikely to have the answer. In other words, one needs a *filter-in* strategy and a *filter-out* strategy and both have to be efficient.

Note that the filter-in and filter-out strategies are analogous to the *branch* and *bound* in a traditional search algorithm, respectively. More importantly, the efficiency of both strategies largely depends on the communication chain. Given such complications, it is therefore not uncommon that a less-experienced person can get trapped in the space and unable to find an answer while a more-experienced person can successfully find the answer that results in a meaningful action.

This was what happened even within our lab. Initially, the students involved in the project spent a month on analyzing the data and did not reach a meaningful answer (did not pass through the product team, i.e. the first level of the communication chain). Then, my advisor got involved and within a week, more results were found. Their

meaningfulness was confirmed by the product team. Based on those results, it took several additional weeks to expand them into further findings before the foundry was convinced to take action. Then, implementing the process adjustments and collecting the silicon data spanned several months. The final confirmation that the yield problem was resolved arrived many months later.

The first set of meaningful results were found after my advisor decided to try out a new failure type he called X_4 . This X_4 failure type was never separately considered in all previous attempts. This was because the X_4 failure, together with a few other failures, was already aggregated into a failure type in the original test data. As a result, no one thought about separating X_4 as a failure type by itself.

By focusing on the newly-defined X_4 failure type, our lab was able to reach a result of -0.766 correlation to an E-test parameter called PP1. This was the turning point for the entire effort [9].

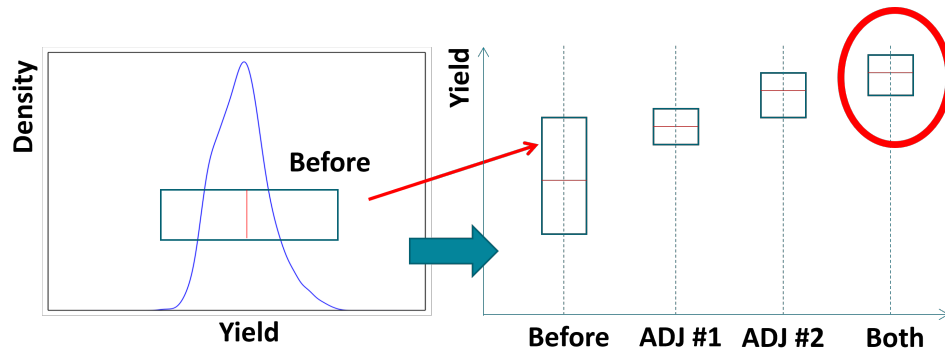


Figure 1.4: Yield improvement in silicon validation of recommended process adjustments

At the end, our lab suggested five process parameters for process adjustments. The foundry responded with two adjustments. This was because some parameters had to be adjusted together. Let us call these two adjustments ADJ #1 and ADJ #2.

The foundry implemented three silicon experiments: by applying ADJ #1 alone, by applying ADJ #2 alone, and by applying them both together. In each experiment,

multiple lots of wafers were manufactured and the wafer yield statistics were collected. The overall result is summarized in Figure 1.4. As shown, each adjustment improved the yield from the original yield distribution and the yield was the best when both adjustments were applied. Following this confirmation, both adjustments were adopted in mass production and the production yield thereafter indicated that the yield problem had indeed been resolved.

1.2 The Search Process

The story described in section 1.1.2 reveals several interesting points regarding an analytic process in practice. At the high level, we see that the analytic process is conducting a search. We can call this search a *Choose-and-Bound* search process.

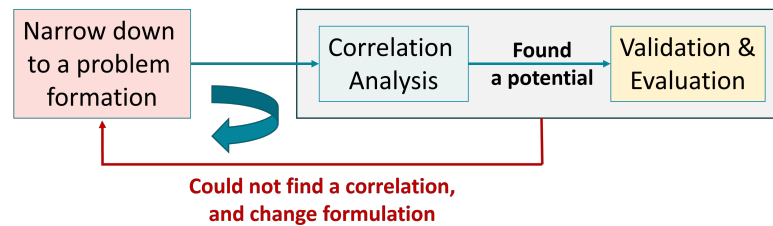


Figure 1.5: The illustration of a "Choose-and-Bound" search process

Figure 1.5 illustrates the Choose-and-Bound search flow. First, one specifies a problem formulation, for example, narrows down to a particular search scope with a particular failure type. Then, within the search scope one searches for a high correlation result. If a potential result is found, then the result is sent for further validation and evaluation, i.e. passing through the communication chain. For almost all the time, either one could not reach a result worthy of initiating a discussion in the communication chain or the result was turned down somewhere in the communication chain. At some point, one has to decide to give up the problem formulation and re-formulate a new one.

The search is iterative in nature. Each iteration involves an adjustment to the problem formulation, resulting in the preparation of a new dataset. Different problem formulations can be the results by taking different *perspectives*. One perspective can lead to multiple problem formulations each defining a unique dataset. Then, a *search space* as seen by a person conducting the analytics, includes all possible formulations based on the perspectives the person can think of. Consequently, different persons can come out with different search spaces based on the same data, due to different experience and domain knowledge they possess.

In this iterative search, deciding whether or not one particular perspective would eventually lead to a satisfactory answer, involves various judgment calls and can be tricky. Because of this, one can get trapped in certain perspectives and consequently be trapped in a given search space.

A key reason why previous efforts failed to discover the adjustment and we did [9], was due to the fact that we examined the data from a perspective that was not previously considered in the search, i.e. the X_4 perspective. In other words, the search spaces explored by other attempts did not include the X_4 perspective. Ideally, if there was a way to exhaustively enumerate all possible problem formulations based on given data to form a “universal” search space, and suppose the answer was included with one of the formulations in the search space, then given enough time, anyone would be able to reach the answer, e.g. reaching the answer with the failure type X_4 as we did. In practice, exhaustively enumerating all possible formulations is usually infeasible, and deciding whether or not a satisfactory answer is contained in a given search space can be hard.

Even by assuming that the formulation with the answer is reachable in the search, whether or not the answer can be reached within a given time frame is another issue. This means that we require an approach to systematically and efficiently explore all problem formulations in a given search space.

As mentioned earlier, the efficiency can be associated with two strategies: filter-in and filter-out. The filter-in corresponds to the *Choose* and the filter-out corresponds to the *Bound*. Because most problem formulations in a search space contains no answer, the efficiency of the Bound is critical. In the context of finding a strong correlation, this can mean that we need a method to tell there is “no high correlation” exists in a given dataset. In other words, we desire a robust method for making this ”no” decision, i.e. to bound the effort spent in a correlation analysis attempt. This bounding is to indicate that the current dataset, formulated under a particular perspective, is inadequate for reaching a satisfactory answer we are looking for.

1.3 Reflection in Other Contexts

The Choose-and-Bound search is at the core of many analytic tasks in the contexts of semiconductor design and test [2]. Below we provide three additional example contexts to reflect such a search process.

1.3.1 Functional verification

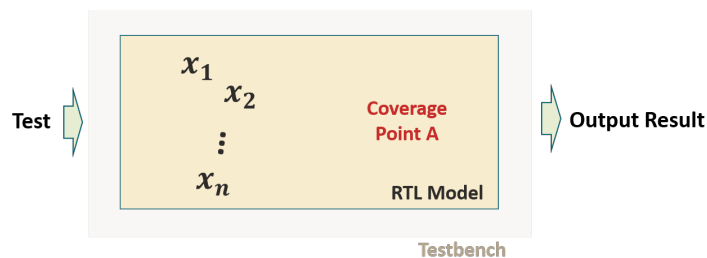


Figure 1.6: Reflecting Choose-and-Bound search in a functional verification context

In a typical design flow, the primary goal of functional verification is to reach *RTL golden* which means that the RTL simulation model is signed off and ready for its physical

implementation. This implementation can be done through synthesis and custom design effort to obtain the final layout model for chip production.

To justify RTL sign-off, a common methodology is to check on a set of coverage metrics in RTL simulation. A coverage metric can be defined based on a set of *coverage points*. For example, Figure 1.6 shows a scenario where the RTL model is simulated under a testbench, i.e. a simulation environment. A *test* is used to drive the input to the RTL model and the output result is observed. For an SoC design, a test typically is represented as a software program, e.g. compiled into a sequence of machine instructions. In Figure 1.6, there is a coverage point named A. Suppose this coverage point had not been hit by previous tests. Hence, the underlying task is to reach a test that can cover this coverage point A.

In the figure, x_1, \dots, x_n represents a set of microarchitecture states that are *controllable* at the software test level. In other words, an engineer can alter some options in the test to affect the values of those states. These microarchitecture states are usually described in the microarchitecture specifications. In view of these microarchitecture states, the underlying problem can be thought of as *finding the combinations of state values* that enable hitting the coverage point A [3][4].

The search process is iterative. An engineer runs some tests, observes and analyzes the output results, and decides how to adjust the tests to have a better chance of hitting the coverage point. The Choose aspect in this search can be a decision to focus on a subset of microarchitecture states. The Bound aspect in this search can be the decision to give up on this subset and move on to another one. It is important to note that even under the narrower focus with a subset of microarchitecture states, there can still be many options to fine-tune the tests. Hence, it is not an easy step to decide on giving up a chosen subset of states and looking for a new one.

In this context, a success means that the Choose-and-Bound search process eventually

reaches a subset of microarchitecture states and asserts their relevancy to the coverage of point A. In parallel, a test for hitting A is obtained and confirmed through RTL simulation.

It is interesting to note that the Choose-and-Bound search process effectively forces a verification engineer to learn more about the microarchitecture specifications. It is not uncommon that this learning reveals some inaccuracies in the specifications and results in their change. From this perspective, we can say that functional verification helps converging the design and its specifications [4]. This view is in contrast to the traditional view that functional verification is to verify a design against a specification.

1.3.2 Speedpath analysis

For a high-performance chip product, the design effort continues in the post-silicon stages [2]. It is a common practice in the industry to improve the performance of a design by examining a set of so-called *speed-limiting paths*, or simply *speedpaths* [48][45]. A speedpath is a *timing path* that limits the operation speed of a chip. Hence, by removing the path through re-design, the operation frequency can be pushed higher.

Suppose through silicon samples, a set of speedpaths are identified. To help the re-design effort, it is desirable to understand the underlying causes for those speedpaths. A speedpath can be due to its unique reason or can share a cause with other speedpaths. This aspect complicates the analysis on a group of speedpaths [29], i.e. an analysis needs to reveal two things: (1) which group of speedpaths share the same cause, and (2) what the cause is. A cause is usually represented as a combination of *design features* [2].

Figure 1.7 illustrates the analytic task reported in [48]. The speedpath analysis task is associated with a multi-core high-performance processor product of AMD at the time.

The left picture in Figure 1.7 shows a distribution of timing paths over four frequency

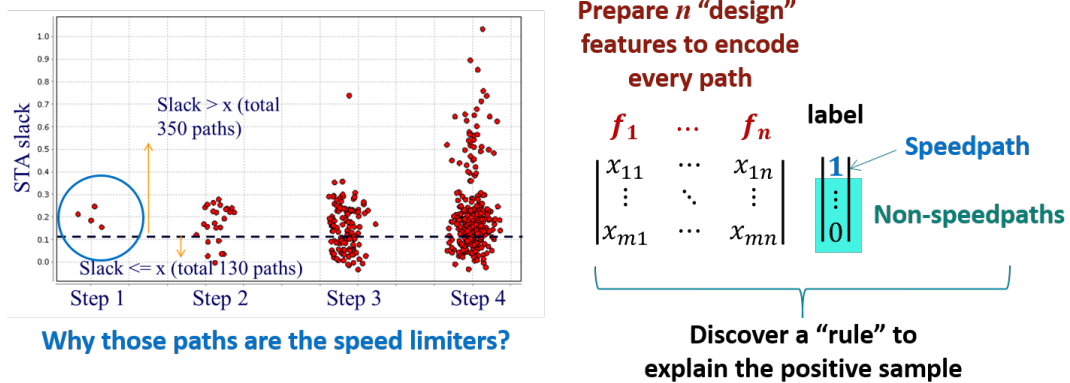


Figure 1.7: Reflecting Choose-and-Bound search in a speedpath analysis context

steps: Step 1 to Step 4. Each dot represents a timing path. The frequency of Step 1 is the lowest. There are four dots in this category. These paths are shown in the Step 1 category because they were the failing paths identified when the chips were operated in the lowest frequency. In other words, these four paths were the first set of speed limiters. To improve performance, it was therefore desired to understand the causes for those four speedpaths and seek a way to re-design and remove those paths.

Analyzing the cause of a speedpath can be based on the method depicted by the right picture in Figure 1.7. This method can be called feature-based analytics [2]. To analyze a speedpath P_1 , one collects a set of non-speedpaths P_2, \dots, P_m as the contrasting set. Then, one chooses a set of design features f_1, \dots, f_n to encode each path into a feature vector. Then, the analytic question becomes: what combination of feature values separate P_1 from P_2, \dots, P_m ? This can be seen as a *rule learning* type of analytics [30].

In practice, there can be a large number of potential design features to choose from. Considering all of them at once is practically infeasible. This infeasibility has two aspects. First, it is difficult to enumerate all possible features in advance. For example, there can be many ways to enumerate features based on the layout. To think in advance all possible features not only is hard but also is ineffective because most of the features would not

be relevant to the cause. The second aspect is about the rule learning which is applied to a specialized dataset containing one positive sample (speedpath) with many negative samples (non-speedpaths). Due to the scarcity of the positive samples, using too many features can easily lead to over-fitting a rule.

Consequently, in practice one chooses a set of features to conduct an analysis. If the analysis does not lead to a satisfactory answer, one moves on to another set of features. Hence, each iteration of the Choose-and-Bound search in this context involves choosing a set of design features, and making a decision that the current dataset contains no satisfactory answer.

1.3.3 Outlier analysis

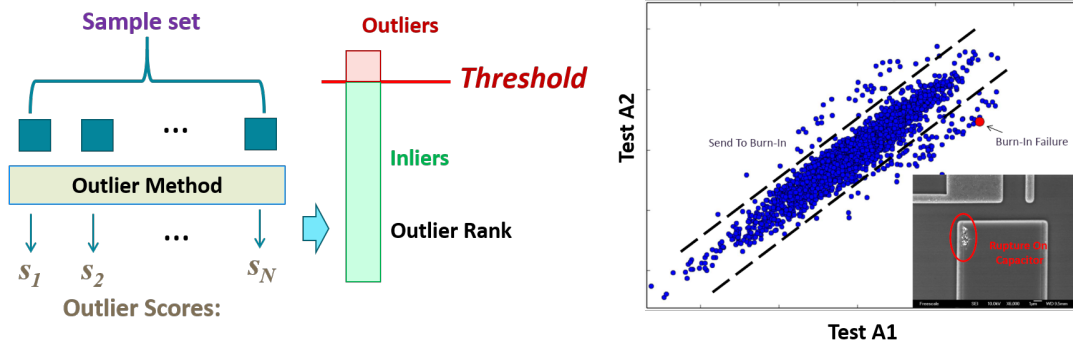


Figure 1.8: Reflecting Choose-and-Bound search in an outlier analysis context

In semiconductor testing, outlier analysis is a common approach for screening out potential bad chips [31]. The idea of outlier screening is simple: If the behavior of a chip, as measured by some test(s), is “significantly different” from others, then the chip is screened out. To apply outlier screening, one develops an outlier model. The left picture of Figure 1.8 illustrates the basic three elements for developing an outlier model: (1) the sample set, (2) the outlier method, and (3) the threshold.

An outlier is with respect to a sample set. For example, a chip can be outlying with

respect to other chips from the same wafer but not as outlying with respect to other chips from the same lot (a lot contains 25 wafers). Hence, the meaning of an outlier very much depends on the sample set used to determine the outlying property.

In semiconductor testing, there are many outlier methods to choose from [31]. An outlier method basically computes an *outlier score* for each chip. Then, the chips can be ranked according to those scores. Different methods result in different rankings of chips.

Once an *ourlier rank* is obtained, one still needs to decide on a threshold to separate outliers from inliers. For example, a simple Gaussian modeling can say that a chip is 5.9σ away from the mean. This number 5.9 is the outlier score. However, whether 5.9 should be called an outlier or not is a subjective decision. If one insists on a 6-sigma rule, then it is not an outlier.

Because in practice it is challenging to make outlier screening robust, one usually only considers an outlier model that screens out very obvious outliers. Outlier screening is also considered under a *yield budget*. For example, a product team can decide that they are willing to allocate 0.5% of the yield loss for outlier screening. Then, the test team needs to develop outlier models that in total screen out no more than 0.5% of the chips.

Because outlier screening is applied in a rather conservative way initially, over time when more test data become available, by learning from the data one may desire to introduce additional outlier models to either improve the test quality [32] or to improve the test efficiency [6].

The right picture in Figure 1.8 shows an example in the second context, trying to improve the test efficiency. In this context, one desires to develop outlier models that can identify chips that require going through the burn-in process. More importantly, one desires to identify chips that do not require the expensive burn-in stress test. Hence, minimizing its use can reduce the test cost substantially.

The picture shows a two-dimensional outlier model based on two tests A1 and A2.

The outlier boundary is shown as two dash lines defining an inside region for those inliers. All dots (chips) outside the region are outliers. The red dot shows a chip known to fail in the burn-in. The bottom-right shows an in-picture illustrating the root cause of the fail. This in-picture was generated by the failure analysis team.

The model would not have been meaningful if tests A1 and A2 were not relevant to testing the particular capacitor. Hence, reaching an outlier model to screen out the given failing chip does not imply the model is meaningful. One has to validate that the tests used by the model are meaningful in view of the underlying cause for the failure.

Given a set of known fails, the search for outlier models to screen them can also be seen as a Choose-and-Bound iterative process. An outlier model is based on a *test space* defined by one or more tests. Hence, the search space comprises all possible test spaces. In each iteration, one makes a choice and tries to use the test data to find a meaningful outlier model in the selected test space(s). The bounding part means that one gives up on the selected test space(s) and move on to another selection.

1.3.4 Summary

We see that the Choose-and-Bound search paradigm can capture the essence of many engineering practices in a semiconductor company. In general, the Choose involves choosing a problem formulation, a hypothesis for the answer, and/or a dataset. The Bound means giving up the formulation, the hypothesis, and/or the dataset. The iterative process is to search for a satisfactory answer. The *acceptability* of an answer might be determined by a group of people, and sometime through a communication chain across different organizations. Our goal is to develop an AI assistant that provides added-values to engineers working in a Choose-and-Bound search context.

Developing such an AI assistant requires fundamentally different thinking than to-

day’s common practices for applying machine learning in design & test and for developing an AI assistant in general. In the next section, we will highlight some key points to differentiate our thinking from others. To be more specific, we will elaborate our thinking in view of the thinking in common practices of machine learning.

1.4 Ten Points from ML to Our AI Assistant

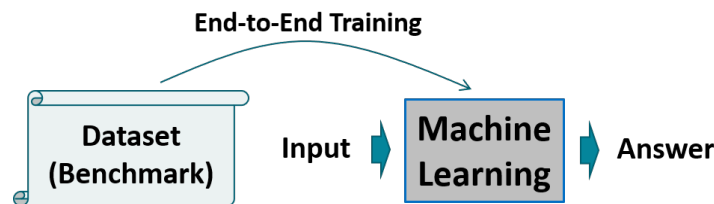


Figure 1.9: An end-to-end training view of machine learning

It is common that when thinking of “machine learning” (ML), one thinks about building a machine learning model. Figure 1.9 depicts this view. A dataset is prepared and used to build and validate a model. This dataset can be seen as the *benchmark* for defining the underlying problem to be solved. In application, an input is given to the model and the model provides an answer for the problem.

1.4.1 Well-defined problem vs. Ill-defined problem

By taking the ML view, an implicit assumption is that the dataset defines the underlying problem to be solved. If we took this view to approach the yield problem discussed in section 1.1.2, we would come out with a very different picture.

Figure 1.10 illustrates how taking the ML view would be different from taking the Choose-and-Bound view as discussed in section 1.2 before. The ML view starts with a benchmark to define the problem.

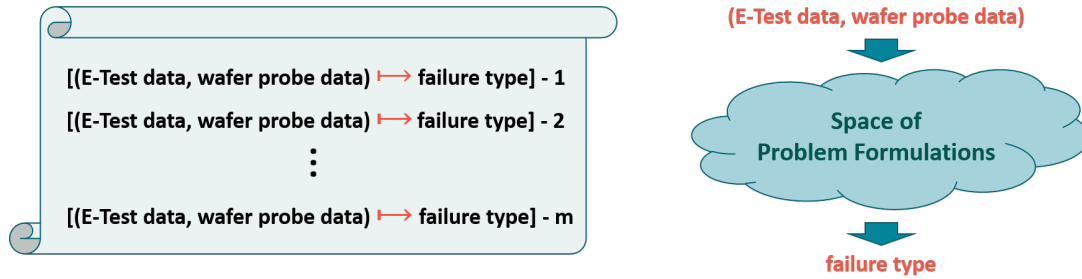


Figure 1.10: ML view vs. Choose-and-Bound view to approach the yield problem

The overall problem can be stated as the following: Given a pair of E-test data and wafer probe data, find a particular failure type such that it is highly correlated to an E-test parameter. Hence, the input to the problem is given as two pieces of data and the output is a failure type. With the ML thinking, we prepare a benchmark as illustrated in the left of Figure 1.10. There are m samples in this benchmark and each sample is in the form of “[input \mapsto output]”.

In contrast, the right plot of Figure 1.10 depicts our Choose-and-Bound view. Instead of preparing m pairs of data as the benchmark, this view focuses on learning with only one pair of data. Note that this learning focuses on finding one specific mapping “ \mapsto ” between the specific pair of data. In our Choose-and-Bound view, this mapping is seen as one instance in a search space comprising problem formulations that can be defined based on the pair of data. Hence, the learning is about expanding and exploring this search space. The assumption is that, in this search space there exists at least one problem formulation that is represented by a dataset extracted under a limited scope from the original pair of data, and analyzing this limited dataset leads to the failure type we are looking for.



Figure 1.11: Benchmark thinking vs. Specification thinking

Figure 1.11 summarizes the discussion above. Given some data to analyze, the ML view calls for collecting more data to define the problem in general. In contrast, the Choose-and-Bound view focuses on the given data by specifying the problem space that can be defined by the data. The problem space becomes our search space. This specification requires “domain knowledge”.

In other words, if we were to take a ML view, we would try to define the *general* problem first by developing a benchmark. Our Choose-and-Bound view does not seek for defining the general problem. Instead, it starts with a problem space *specific* to the given data. To obtain this space, our view is to enumerate a set of problem formulations that can be specified from the given data. This space defines the *scope of the problem*, and searching for the answer translates into searching for the right problem formulation.

An obvious reason why taking the ML view is not feasible is because in practice it is difficult, if not impossible, to obtain a reliable benchmark as depicted in Figure 1.10. For example, in the yield optimization context the input-output mapping “ \rightarrow ” can be extremely complex. To learn this complex mapping, one would desire a very large and comprehensive benchmark. This means that we need to collect data on many known and resolved yield issues from the past, i.e. m needs to be sufficiently large. This is difficult to do as the number of yield issues (worthy of the yield optimization) is not supposed to be large for a product line. Otherwise, there is something seriously wrong with the design, the test, or the manufacturing to begin with. In view of the challenges in obtaining a reliable benchmark, we can say that the analytic problems in the semiconductor field are mostly ill-defined.

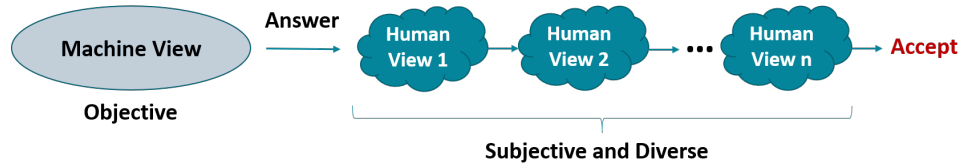


Figure 1.12: Objective machine view vs. subjective human views

1.4.2 Objective evaluation vs. Subjective evaluation

There is another reason why pursuing the left direction in Figure 1.11 is difficult. As discussed earlier, in many contexts the acceptability of an answer depends on a group of people. This is further illustrated with Figure 1.12. For a machine to compute an optimal answer, it needs to start with a definition for the optimization objective and evaluates potential answers according to the objective. However, such an optimal answer from a machine might or might not be an acceptable answer from a domain expert’s perspective. As mentioned before, the human evaluation might involve a chain of parties each having its own subjective view regarding the acceptability of an answer. Consequently, an answer is acceptable when it can pass all these diverse subjective evaluations.

When the acceptability of an answer is not based on a unified objective view, attempting to completely define a problem for evaluation becomes difficult. If we were to prepare a dataset with the goal to completely define the problem in an application context, we would need to consider all those subjective and diverse views, leaving the benchmark preparation effort somewhat open-ended.

1.4.3 Optimization vs. Exploration

A key difference between the two opposite directions in Figure 1.11 is that going left means the focus is on optimization while going right means the focus is on exploration.

Given the general problem as represented by a benchmark, the underlying goal be-

comes finding the best model to solve the problem. The merit of a model can be evaluated with the dataset and hence, is defined as well. Consequently, learning is achieved through optimization.

When we go right in Figure 1.11, learning is done by continuous search, i.e. exploration. Hence, we desire efficient methods to help explore the search space.

Building an AI Assistant to assist in optimization and building an AI Assistant to assist in exploration are different. This thesis focuses on the later. However, it should be noted that focusing on assisting in an exploration context does not mean that our AI Assistant would not involve a tool or a ML model for solving an optimization problem. On the contrary, the AI Assistant may still perform steps that are formulated as solving an optimization problem.

1.4.4 ML vs. Co-ML

In this thesis, we use the term *machine learning (ML)* to broadly refer to a tool or an algorithm that takes data as input and produces the “best” answer or model. In other words, it is data-driven and it performs some optimization.

In our Choose-and-Bound view, as shown in the right of Figure 1.10 earlier, we can think of the search space as comprising a set of datasets derived from the original data. Each dataset corresponds to a specific problem formulation. Hence, the search is about searching for the dataset with the specific problem formulation such that solving the problem leads to an acceptable answer.

Figure 1.13 illustrates this perspective. Note that in the search process, only one dataset is under focus at a time. Typically, one applies an ML tool or model to find the best answer on the dataset. This “best” can be thought of as according to some pre-defined optimization objective. To find the best answer across all datasets, one needs

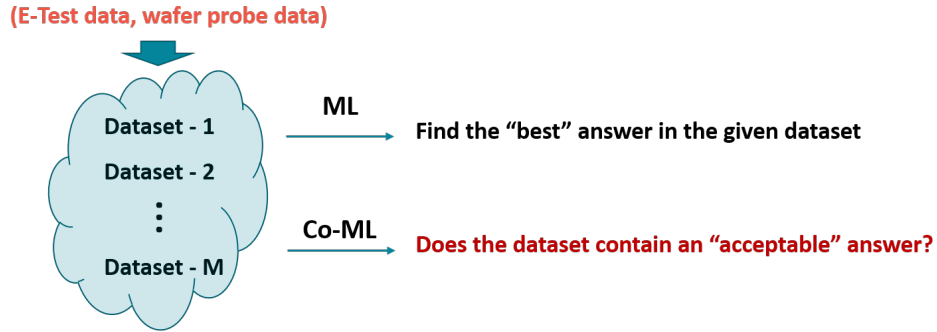


Figure 1.13: The task of ML and Co-ML perform on a given dataset

to apply the ML tool or model to each dataset. However, as mentioned before, the search space can be enormous and hence, going over every dataset is not practical. Moreover, applying ML on each dataset might involving trying on various ML methods and various options associated with each method. On top of that, as depicted in Figure 1.12 before the best answer from a machine view does not imply the answer is acceptable.

To facilitate the search, we therefore desire a complementary tool or model that can help identify those datasets likely to contain no acceptable answer. We call the second type of the tool or model as *Co-ML* (complementary ML). Hence, in an ideal situation in Figure 1.13, we desire to apply Co-ML to prune out as many choices as possible.

It is important to note that ML and Co-ML focus on different objectives. ML focuses on optimization and Co-ML focuses on *unacceptability*. It is possible that a Co-ML tool or model is built upon a ML tool or model. Hence, demanding Co-ML tools and models does not mean that the importance of ML is diminishing. The emphasis is that for building an AI Assistant in practice, we desire the Assistant to have ML as well as Co-ML capabilities — both are important.

The idea of Co-ML can be broadly applied to situations outside the scope of our Choose-and-Bound search view. For example, in yield optimization it is possible that the original data provided contains no meaningful answer at all, regardless how we conduct

the search. In other words, failing to find an answer due to insufficient data can be a common outcome from an analytic task. Reaching such a conclusion quickly can prevent wasted efforts and prompt the pursuit of an alternative sooner, such as collecting a different set of data.

1.4.5 Complex model vs. Simple answer

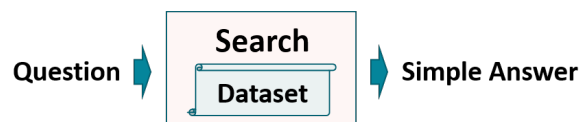


Figure 1.14: Contrasting to Figure 1.9, an AI Assistant is to help find a simple answer

In view of Figure 1.9 before, Figure 1.14 provides a contrasting view. The goal of an AI Assistant is to help find a simple answer than build a complex model. This is because simpler answer is more likely to be understood, justified, and consequently accepted by people. While simplicity does not imply acceptance, it is important to keep in mind that in practice, when someone does not understand or cannot justify an answer, it is unlikely for the person to accept and act upon it.

1.4.6 Looking for a trend vs. Looking for a signal

In a company, analytics is often referred as *looking for a signal* in the data. This is in contrast to building a complex model for capturing a trend or pattern in the data. Looking for a signal can sometime be like solving a needle-in-a-haystack problem.

This so-called “signal” can mean different things in different contexts. It is important to note that the interest level of a signal can depend on its novelty to a person. For example, for a signal representing a common phenomenon frequently appearing in the given type of data, it is not very interesting to a person because the person learns very

little new information from the analytics. Therefore, while a simple signal is more likely to be understood and accepted, a simple and commonly-occurring signal is less interesting. In other words, a simple and uncommon signal is more desirable.

1.4.7 Prediction vs. Prescription

In practice, a signal also needs to be actionable in order to be meaningful and accepted. In other words, we can say that the type of analytics is *prescriptive*. This is in view of the three types of analytics commonly mentioned: *predictive*, *descriptive*, and *prescriptive*. Predictive analytics is about learning from the current data to predict the future. Descriptive is about understanding the current data and prescriptive is about analytics leading to meaningful actions.

For the most part of our work, we consider assisting in a prescriptive type of analytics job content. As discussed before, moving from an analytic result to a meaningful action can involve decisions from diverse groups of people. Consequently, to achieve prescriptive analytics we need to, in one way or another, consider the decision making process.

1.4.8 Data-driven vs. Knowledge-driven

Because the success of an analytic task cannot completely ignore human decisions and judgment calls, a pure data-driven approach would not work effectively (or simply would not work). An effective AI Assistant needs to capture the *domain knowledge* involved in the specific application context. There can be various sources of this domain knowledge, from how an individual engineer performs the analytic task to how a decision maker reaches an action. For an assistant to effectively assist an engineer in the context of an analytic job, the assistant needs to incorporate some of these various types of knowledge, especially how a successful outcome was attained in the past. Possessing the

requisite knowledge to attain success in a job is important. The success story reviewed in section 1.1.2 demonstrates this aspect.

In addition to the individual engineer’s perspective, the effectiveness of an AI Assistant can also be considered from a company perspective. Consider three companies: company A supplies high-performance processor chips to be used in a data server, company Q supplies high-performance chips for mobile devices, and company N supplies micro-controller chips in the automotive market. Each of them can have their own unique emphasis on yield optimization. For example, company A emphasizes on performance yield. Company Q emphasizes on power consumption and company N emphasizes on quality and reliability. Consequently, the view towards the effectiveness of an AI Assistant for one company might not be the same for another company. It is intuitive to see that an effective AI Assistant for a company might need to consider the company-specific knowledge, e.g. their specific operations and business model.



Figure 1.15: Drawing a line to separate the knowledge to be captured in an assistant

If developing an AI Assistant universally effective across diverse companies is difficult and suppose we do not want our AI Assistant to “over-fit” a company, then it becomes important to consider drawing a line to separate out the company-specific knowledge that is not to be captured in the assistant. Figure 1.15 depicts this aspect of consideration. If this line is drawn overly close to the left end, transferring the assistant built for one company to another might require substantial modifications. If this line is drawn overly close to the right end, the assistant might not be seen as effective by an engineer. Hence, there is a tradeoff between the effectiveness of an assistant and its transferability.

While the above discussion stays at the intuition level, it should be noted that the term

“knowledge” remains vague in the above discussion. Consequently, “domain knowledge”, “company knowledge”, “individual knowledge” are all vague terms. When we say that the AI Assistant needs to *capture the domain knowledge*, it remains unclear what exactly the knowledge is, not to mention how to “capture” it. These are some fundamental questions that we will answer in the rest of the thesis in order to build a practical AI Assistant.

1.4.9 Free Lunch vs. No Free Lunch

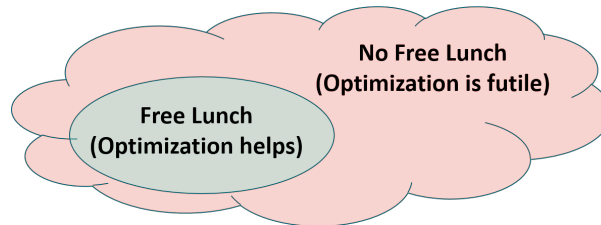


Figure 1.16: An AI Assistant works in two worlds: Free-Lunch and No-Free-Lunch

In machine learning, there can be two camps of beliefs: one believes a world of *free-lunch* and the other believes a world of *no-free-lunch* [33][34]. In a free-lunch world, optimization can help. In a no-free-lunch world, optimization is futile. The AI Assistant we envisioned needs to consider both worlds.

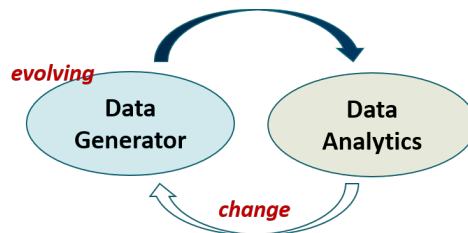


Figure 1.17: The data generator is evolving over time

Loosely speaking, we say that a no-free-lunch situation occurs when the current data is not representative to the future data. In other words, what we learn from the current

data might not be applicable to the future data. This aspect can be inherent in the analytics of design and test data.

Figure 1.17 brings the *data generator* into the picture for illustration. This data generator depends on the specific design, manufacturing process, and/or test where each can evolve over time. In addition, an action resulting from an analytics task can lead to changes in the design, manufacturing process, and/or test. Consequently, the data changes over time and the current data might not be representative to the future data.

To give an example, consider the three common stages to bring up a production line: *yield learning*, *ramp-up*, and *mass production*. In the yield learning stage, the goal is to improve the yield to a satisfactory level. In the ramp-up, the production volume is gradually increased. When the mass production starts, the yield is expected to reach a sufficiently high level and also stabilize. In this context, it is intuitive to see that yield issues learned during the yield learning stage may not necessarily be representative of the yield issues encountered during the mass production. A yield issue during the mass production is supposed to be a new issue not seen before. Therefore, if one desires to train a model with the data collected in the yield learning stage and use the model to predict yield issues in the mass production stage, it is possible that the learning task falls into a no-free-lunch world.

1.4.10 Automation vs. Assistance

Automation	Assistance
Make decisions	Make suggestions (assisting)
Driven by optimization	Driven by understanding
Assume a free lunch world	Include a no-free-lunch world

Figure 1.18: Building an automation flow vs. building an assistant

In design and test, it is common that one aims for building a flow to achieve *design automation* and/or *test automation*. Software companies supplying products to serve the semiconductor industry often call their tool an *automation tool*. An automation tool aims to automate the end-to-end process from an input to an output. It is important to emphasize that an assistance fulfill that role.

Figure 1.18 highlights a few points to separate our AI Assistance view from the traditional view of building an automation flow. An automation flow makes various best decisions for its user, driven by the underlying optimization objectives and assuming a free-lunch world. In contrast, an assistant makes a suggestion to its user and leaves the ultimate decision to the user. What driving the assistance is for achieving a better understanding of the data. And as mentioned in the previous section, an assistance includes consideration of a no-free-lunch world.

Contrasting our AI Assistant view with traditional automation thinking is important because designing a tool to assist in a job and designing a tool to take over the job are fundamentally different thinkings. The former thinking aims to aid the engineer while the ultimate goal of the latter is to replace the engineer, e.g. providing a “push-button” solution to perform the job. Hence, it should be emphasized that our AI Assistance view stands in stark contrast to building a push-button solution.

1.5 Conclusion

Given all the complications discussed above for building an effective AI Assistant in practice, it would not be surprised that the development of the first prototype took years. This first AI Assistant is called IEA-Plot and was publicly announced in 2023 [35]. In the rest of the thesis, we will discuss the journey leading to the completion of IEA-Plot, explain the key ideas behind several milestones along the way, elaborate

the technological innovations for realizing those ideas, and lay down a framework for further extensions and enhancements of IEA-Plot in the foreseeable future. Chapter 2 and Chapter 3 provide a retrospective view of the early works in applying ML and AI in design and test (conducted by our lab), prior to the first prototype of IEA in 2018. Readers may opt to skip ahead to Chapter 4, which marks the start of the primary focus of this thesis: the realization of the LLM-assisted AI assistant.

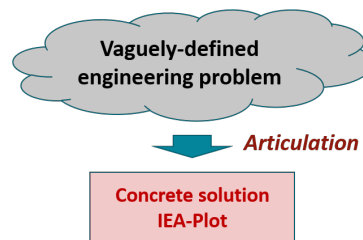


Figure 1.19: IEA-Plot: from vague problem description to concrete solution

It is understandable that the scope of the problem discussed in this introduction chapter remains vague, especially since some terms used in the discussion might seem somewhat loosely defined. Indeed, it is quite difficult to precisely describe an engineering job content in practice. In fact, the very essence of developing the IEA-Plot is to overcome this difficulty. The development process starts with a vaguely-defined engineering problem and concludes with a concrete solution. Articulating the problem scope is central to this development. Figure 1.19 illustrates this point.

Therefore, the contribution of IEA-Plot should not be seen as just providing a solution to solve a given problem or to solve it more efficiently. The development of IEA-Plot includes an articulation of the problem itself and implementation of innovative ML/Co-ML methods for realizing an efficient and practical solution. This dual problem-solution view is fundamental to the development of IEA-Plot and it is under this view that leveraging power of LLMs in IEA becomes crucial. This aspect will be elaborated in more detail throughout the thesis.

Chapter 2

Journey to IEA - The 1st Decade

橫看成嶺側成峰, 遠近高低各不同。

不識廬山真面目, 只緣身在此山中。

A range viewed in front and many peaks viewed from the side,

Assuming different shapes viewed from far and wide;

Of Mountain Lu we do not see clearly its true face,

For we are lost inside the very place.

— 《題西林壁》蘇軾, A poem from Song Dynasty

The journey to IEA can be traced back to the starting point two decades ago where the works in [36][37] tried to apply ML to solve a problem in design and test. The specific ML method used in those works was Support Vector Machine (SVM) [38] which was considered as state-of-the-art at the time. The specific problem was critical timing path selection [39]. These works were among the earliest, if not the earliest, attempts to apply modern ML in design and test.

The critical path selection problem can be stated as the following: Given a design, find all timing paths under statistical process variations, such that each path has the chance to

impact the timing performance of the design [39]. These paths are called critical timing path. The *static* version of the problem does not consider the inputs to the design. The *dynamic* version does and hence, is much more complicated. This is because a static critical path might never be activated by an input and hence, can become a non-critical path under the dynamic view. In other words, static analysis provides a bound to the timing analysis but is not accurate as dynamic analysis.

The main idea in [36][37] was to formulate the path selection problem as a *feature selection* problem. In the setup, a large set of paths were used as the set of features, i.e. treating each path as a feature. A ML model (more specifically, an SVM model) was built from statistical timing simulation data to predict the circuit delay. Then, from the model a subset of important features were selected, under the constraint that modeling with the selected subset achieved almost the same accuracy as modeling with the original feature set. In other words, the selected features are the critical timing paths identified using the model.

“Applying ML in Design and Test (D&T)” was exemplified by these early works as taking the following four steps:

1. Formulating the D&T problem as a ML problem
2. Preparing the dataset for learning a model
3. Choosing a learning method in view of the ML problem
4. Utilizing the learning model to solve the original D&T problem

We can call this the *data-driven view* to apply ML. This can be seen as the standard ML view, i.e. to apply ML we first prepare the dataset to train a model. As discussed in section 1.4, this dataset implicitly formulates the underlying problem to be solved. In

other words, in the data-driven view, one begins by formulating the problem as a ML problem and then formulating the ML problem through a dataset (e.g. a benchmark).

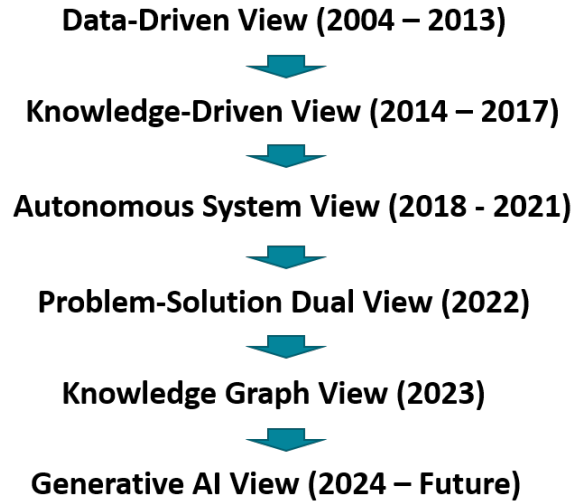


Figure 2.1: Evolution of views along the journey to IEA

Starting from the data-driven view in 2004, the journey to IEA then underwent an evolution of views to reach the IEA-Plot in 2023. Figure 2.1 depicts this evolution. In the rest of this chapter, we will elaborate the data-driven view taken in the first decade and discuss the lessons learned which motivated the works in the second decade. The second decade will be reviewed in Chapter 3.

2.1 Data-Driven View (2004 – 2013)

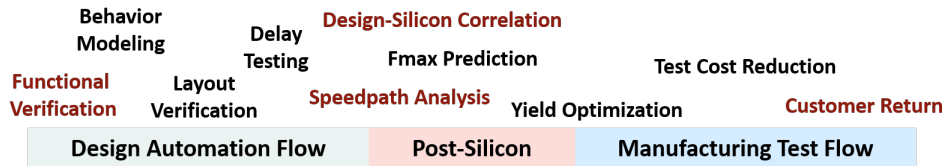


Figure 2.2: 10 categories of works carried out under the data-driven view

The era of data-driven view lasted about a decade. In this period, various categories of works were carried out to tackle problems originated from design and test. Figure 2.2 depicts ten categories across three stages: the design automation flow, the post-silicon stage, and the manufacturing test flow. We consider *post-silicon* as the stage after the first silicon and before the start of mass production. Note that the categorization shown in the figure is not precise. A particular category of work might be done in a different stage.

It is interesting to note that in Figure 2.2, most of the works conducted were situated at the boundary of a stage: The leftmost category that entering the design flow (functional verification), the categories crossing the design-silicon boundaries (design-silicon correlation and speedpath analysis), and the rightmost category leaving the test flow (customer return analysis). These occurrences were not accidental. These areas were critical and demanded a lot of engineering efforts and hence, companies were highly motivated to explore ML to improve their operations.

In practice, functional verification was often tedious and time-consuming, limited by the simulation speed and requiring substantial manual efforts to fine-tune the functional tests and the specifications during the process. From the time the 1st silicon was obtained, much engineering efforts were spent on debugging issues due to mismatches between design model and manufacturing process. These issues could be manifested as low yield, unexpected low performance, and/or unexpected functional bugs. These issues needed to be fully resolved before proceeding with mass production.

For a company, defective parts per million (DPPM) or per billion (DPPB), is an important metric promised to its customers. Consequently, throughout the production cycle, continuously driving down DPPM/DPPB was a top priority. This was usually done through analyzing parts returned by the customer(s). Analyzing these returned parts might lead to inclusion of new tests to screen out future parts with similar characteristics.

Traditional customer return analysis involved going through a dedicated failure analysis (FA) team where a part was opened up and physically inspected to identify the root cause of failure. FA was considered as an expensive process and hence, companies were motivated to find a feasible alternative.

Next, we will provide a retrospective review of the ten categories of works from the ten-year period, with a particular focus on elaborating more on the four categories highlighted in Figure 2.2.

2.2 Design-Silicon Timing Correlation

In an ideal situation, models used in a design should accurately reflect the outcomes from the manufacturing process. In reality, it is not the case. For example, after the models are released to the design team, the manufacturing process can continue going through some adjustments. It is not always the case that changes to the process resulting in model updates to the design team.

For efficiency, models used in a design are usually approximate models. For example, a timing model captures the worst-case, the nominal-case, and the best-case of the delay and ignores the statistics in between them. In practice, models used in the design can only be accurate to a certain extent [1].

As manufacturing technologies continued to scale and designs became increasingly complex in the period, predictability of silicon behavior based on models and simulations degraded. This degradation of predictability had been known for a while within companies that designed high-performance products. For these companies, it was therefore a common practice that optimization of a design continued into the post-silicon stage, utilizing information collected from silicon samples.

Design-silicon correlation, pioneered by the work in [40], was about learning from

silicon samples to resolve design-silicon mismatches. It was usually applied in the context of yield learning and optimization. At the time, operation frequency remained to be the top concern for a high-performance design. Hence, the works in this period concerned mostly *design-silicon timing correlation*, e.g. see [41][40][42][43][44][1][29].

2.2.1 Data learning based diagnosis

The methodology to achieve design-silicon timing correlation was summarized as *data learning based diagnosis* [1] in contrast to traditional fault-model based diagnosis.

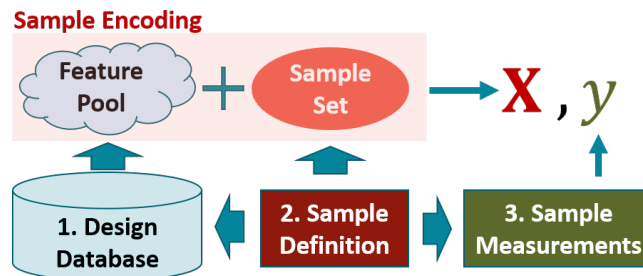


Figure 2.3: Data learning based diagnosis as summarized in [1]

To implement the methodology, three components are required as highlighted in Figure 2.3: (1) design database, (2) sample definition, and (3) sample measurement results.

Each application is based on a particular *sample definition*. For example, a sample can be a timing path. Based on the definition, a set of samples are decided. Some property on each sample is measured on silicon chips, resulting in a measured value for each sample. Based on this value, the “y” label can be obtained in the dataset. For example, the y value can be the difference between the measured value and the value predicted by a pre-silicon simulation model.

To obtain the “x” vectors, each sample is encoded based on a pool of features selected from the design database which supposedly contains all models on the design and data generated from the design process (e.g. from simulation and verification, etc.). For

example, the work in [40] defined features in terms of cells and interconnects. The work in [43] gave a more comprehensive list of features in five categories: cell-based, interconnect-based, location-based, dynamic effects (e.g. cross-talk, IR drop, temperature), and tool-related effects. For dynamic effect features, they are based on models and simulations. Hence, a feature value is only a predicted value, not the actual value.

In practice, the bottleneck for executing the methodology is on preparation of the learning dataset (\mathbf{X}, y) . Preparing y requires preparing the test content and running the test flow. This means a requirement to access and utilize the test infrastructure. Preparing \mathbf{X} requires a feature extraction and encoding process which accesses and utilizes the design infrastructure. Comparing to data preparation, analyzing the data is much more straightforward because it does not need to deal with the complex design and test infrastructures in a company. The analysis can be done locally by an individual.

Once the dataset is available, the analysis can be based on different approaches. It can be formulated as a (binary) classification problem [40] or as a regression problem [43]. In both cases, one can pursue a *feature importance ranking* [40][42][45][43] to identify the top design features relevant to the “y” labels. When it is formulated as a classification problem, one can also choose to apply *classification rule learning* to extract a specific rule to separate the classes [29][44].

2.2.2 Transduction diagnosis

The summary paper [1] calls the type of diagnosis performed by the methodology as *transduction diagnosis*, inspired by the concept *transduction* proposed in statistical learning theory [46]. Figure 2.4 illustrates the idea.

Traditional diagnosis goes through an *induction* process, followed by a *deduction* process. To analyze an unexpected silicon behavior, the goal of the induction is getting

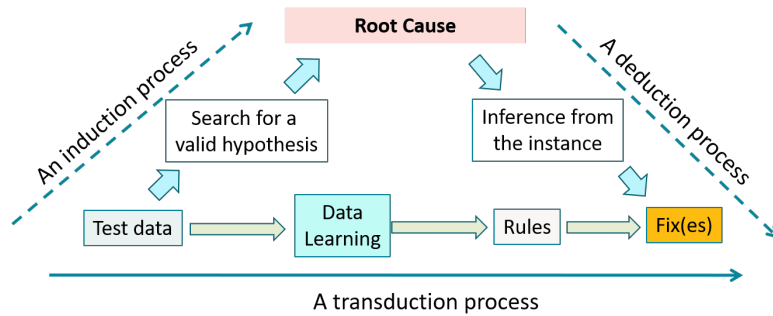


Figure 2.4: Transduction diagnosis vs. traditional diagnosis, as explained in [1]

to the *root cause*. Then, the goal of deduction is to inference from the particular instance of the root cause to obtain a fix or fixes. In general, a fix can mean to modify the design, the test, the manufacturing process, or a combination of them. Getting to the root cause involves forming a set of hypotheses and search for a valid hypothesis based on the data.

For example, a common practice in traditional defect diagnosis is by assuming a *fault model* consisting of *defect hypotheses* (or “faults”) [47]. Given the test data showing an unexpected result (a failure), *fault simulation* on those faults is used to match a fault to the failure signature. A fault is a hypothesis and hence, it does not necessarily mean the root case. In practice, the root cause is identified through physical inspection during FA (failure analysis). For example, the right picture in Figure 1.8 shown in section 1.3.3 earlier includes a demonstration of the root cause for the burn-in failing chip.

The transduction diagnosis intends to bypass the complex root-causing process. The idea is to apply machine learning on the data directly to obtain some rules (or relevant features), and from them to hypothesize a fix.

2.2.3 A binary classification case

Figure 2.5 depicts an example of transduction diagnosis applied in practice. In this case, it was formulated as a binary classification problem.

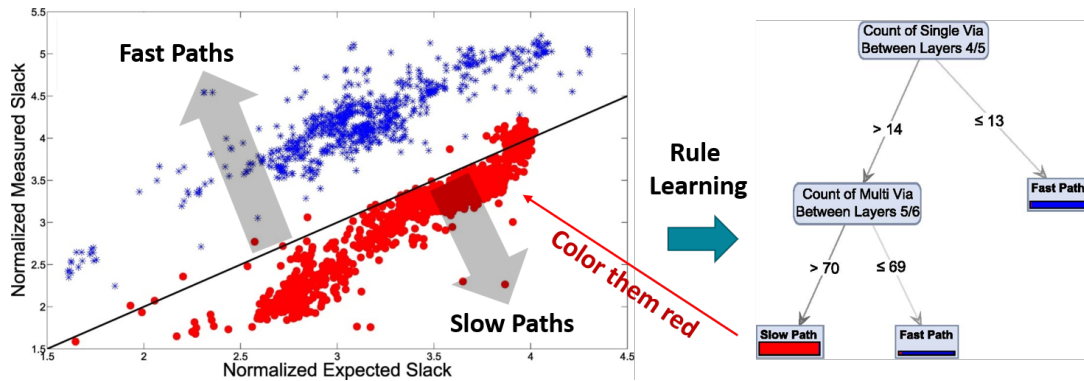


Figure 2.5: A rule learning example, reported in [2]

The data comprise two sets of timing paths: slow paths and fast paths. A slow path means that the observed delay of the path is slower than the delay predicted by the static timing analyzer (STA) (or equivalently, the actual timing *slack* is smaller than predicted slack)). A fast path means the opposite. The picture shows a diagonal black line to indicate where the observed slack matches the predicted slack.

In an ideal situation, the design team thought that these paths should fall closer to the diagonal black line. However, out of their expectation these paths form two clusters, one above the line (fast paths) and the other below the line (slow paths).

The transduction diagnosis methodology was applied to resolve this issue. One of the results found is shown on the right in Figure 2.5. It is a decision-tree type of rule. The rule classifies the paths into two classes, marked by one “slow path” bucket (in red color) and two “fast path” buckets (in blue color). In the picture on the left, the paths are colored accordingly based on the rule as red dots and blue dots. As seen, most of the red dots fall below the diagonal black line, indicating a good fit of the rule to the data.

The two features used in the rule concern about the via occurrences on a path between metal layer 4 and 5 and between metal layer 5 and 6. This was an evidence pointing to some issue related to metal layer 5. Later we were informed by the process team that there was a known issue on metal layer 5 which went through a fixing. The data we saw

was before the fix and afterwards, the issue disappeared.

2.2.4 Scarcity of the positive samples

The binary classification formulation is common for applying the transduction diagnosis methodology. Samples are divided into two classes to begin with, such as pass and fail, fast and slow, expected and unexpected, etc. They can be called *negative samples* and *positive samples*, respectively. The positive samples (fail, slow, unexpected) are the focus, i.e. one desires to understand what causes the positive samples.

The case shown in Figure 2.5 confirms that transduction diagnosis can be effective in practice. However, the case represents a relatively easier scenario because the dataset is more balanced on the positive and negative samples. In other situations, the number of positive samples can be very small, sometime with only one positive sample available for the analysis. Dealing with such extremely-unbalanced dataset requires a specialized rule learning approach [30].

2.2.5 Feature extraction to enable a data-driven analysis

Implementing a feature extraction process to support the transduction diagnosis obviously involves some domain knowledge about the design, the test, and even the manufacturing process. It is also apparent that the effectiveness of the methodology depends on the features used in the analysis. Nevertheless, the transduction diagnosis itself is data-driven. This view follows the same data-driven view in machine learning where a dataset is prepared to enable the learning. Under this view, feature extraction is done as a pre-processing step for dataset preparation. More importantly, a large number of irrelevant features can be included in the dataset and then, it is up to the learning algorithm to filter out those unimportant features efficiently.

2.2.6 A retrospective remark

	f_1	f_2	f_3	f_4	f_5	\dots	f_n
Positive sample	1	1	1	1	0	\dots	0
Negative sample	1	0	0	0	■	\dots	■
Negative sample	0	1	0	0	■	\dots	■
Negative sample	0	0	1	0	■	\dots	■
Negative sample	0	0	0	1	■	\dots	■

Figure 2.6: A simple dataset example to illustrate over-fitting

Ideally, suppose one can exhaustively list all possible features and one has a dataset with sufficient information to filter out all irrelevant features, plus there existing a learning algorithm to do so, then the transduction diagnosis methodology should work. In practice, this ideal situation rarely happens. In most scenarios, a person works with a set of features known to the person and the analysis is done with a limited set of samples. Consequently, over-fitting a rule to the dataset can be quite common.

Figure 2.6 uses a simple example to illustrate the issues. There is one positive sample and four negative samples in this dataset. Each feature f_i is a binary feature, indicating whether the feature is present or not on the sample. Because our interest is on the positive sample and only features f_1, \dots, f_4 appear on the sample, we can ignore all other features f_5, \dots, f_n .

To differentiate the positive sample from the four negative samples, we see that the reason cannot be a single feature f_1, f_2, f_3 and f_4 alone. For example, f_1 cannot differentiate the positive sample from the first negative sample. By removing those single-feature reasons, there are six possible two-feature reasons: $\{f_1 \wedge f_2, f_1 \wedge f_3, f_1 \wedge f_4, f_2 \wedge f_3, f_2 \wedge f_4, f_3 \wedge f_4\}$. While each of them can be used to differentiate the positive samples from the four negative samples, choosing any one of the four reasons is over-fitting.

One can look at the over-fitting from two perspectives. The obvious perspective is that

we need more data, i.e. more negative samples. Suppose in the practical environment, this is hard to do. Then, we are left with the second perspective. That is, we need a different set of features than $\{ f_1, f_2, f_3, f_4 \}$. There can be two ways to get new features, either by creating a new one or by deriving more detailed features from a given one, e.g. splitting f_1 into f_{11} and f_{12} . Both approaches aim to describe the samples with more detailed features, potentially corresponding to more domain knowledge involved in preparing the dataset. Consequently, these features are more effective in discerning the positive versus the negative.

The Choose-and-Bound search (e.g. the example discussed in section 1.3.2 before) is required because over-fitting happens frequently in practice, and it is challenging to determine in advance if a dataset is sufficient to differentiate the positive sample from the negative ones. Consequently, selecting the feature set to run transduction diagnosis becomes the most crucial step to influence the outcome of an analysis. This step is tedious and may demand substantial domain knowledge, thus requiring most of the help from an assistant.

2.3 Speedpath Analysis

There are two types of speedpath analysis: one to explain speedpaths and the other to predict speedpaths based on a given small set of speedpath examples.

The works on explaining speedpaths [48][30] followed the same transduction methodology discussed in the previous section, except that the number of speedpaths (i.e. positive samples) is much smaller than the number of non-speedpaths. To the extreme, one can face a dataset with only one speedpath to analyze. In this case, the positive sample is used to form a hypothesis space and the negative samples are used to prune out hypotheses [30]. When there are multiple speedpaths, one can also face a *subgroup*

discovery problem [29][49] to decide which paths share the same reason and which paths are due to its own unique reason. The example reviewed with Figure 1.7 in section 1.3.2 belongs to this type of speedpath analysis [48].

The works on predicting speedpaths [45][50] followed a methodology similar to that depicted in Figure 2.3, except for no “y” labels in the dataset to be learned, i.e. the learning is unsupervised. Instead of getting to a rule or finding the set of relevant features, the goal is to build a model to identify paths similar to the speedpaths, i.e. a model to enable similarity search. In [45], the model was built as a one-class SVM model [38]. The work in [50] proposed a more sophisticated method. It is important to note that like the transduction diagnosis methodology, the effectiveness of a similarity search model also largely depends on the features in use. Hence, selecting the feature set was still the most important step in the works reported in [45][50].

When there is only one or few speedpaths to learn from, the over-fitting scenario as illustrated in section 2.2.6 before, is likely to happen. Hence, while the transduction diagnosis and building one-class model were both promising, in reality selecting the most relevant features to start the learning remained to be the key to success.

2.4 RTL Functional Verification

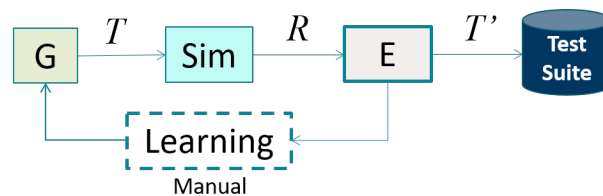


Figure 2.7: Manual learning in the context of functional verification

Figure 2.7 illustrates the context of (RTL) functional verification. There is a generator

G that is capable of producing a set of functional tests T . This generator can be a person, i.e. tests being manually developed, or an automatic tool such as a Constrained Random Test Generator. For an SoC or processor design, each test is a sequence of instructions/transactions, i.e. a software program. The simulator Sim simulates the set of tests and produces simulation result R which is evaluated through a component E . This evaluation can be based on a *coverage metric*, for example how many *coverage points* are covered through the simulation. The evaluation of tests is usually used to identify important tests T' and include them into a test suite. For example, an important test could be the one that excites a bug or provides a unique coverage. The test suite can be seen as the ultimate asset accumulated from the verification efforts.

Verification process is highly iterative, driven by a coverage objective. The job of a verification engineer is to develop tests that achieve a required coverage level, e.g. covering every coverage point in a given set at least once. Hence, when a coverage point is not yet covered, the engineer inspects the simulation result, traces the design model itself, and reads the design-related documents to come out with a test that can hit the coverage point. This learning is manual and based on three sources: simulation traces, design model (RTL model), and design-related documents. In many cases, obtaining the desired test can be based on modifying an existing test or test template.

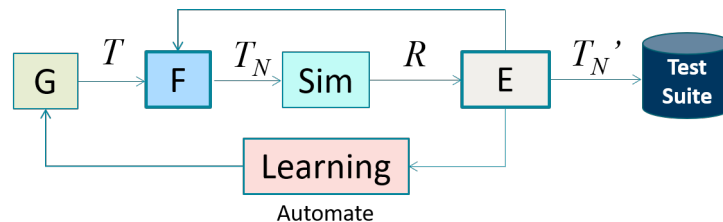


Figure 2.8: Proposed automatic learning in the context of functional verification

Figure 2.8 then illustrates two possible components to add an automatic learning capability, marked as the F box and “Learning” box. The F stands for a filter component

between the test generator G and the simulation Sim . The goal of F is to filter out “unimportant” tests before the simulation, avoiding their simulation to save simulation resources.

In addition to improving simulation efficiency, the F component can also be seen from the viewpoint of improving verification effectiveness. For example, suppose the number of tests to be simulated is limited at N . Without the filtering, N tests would be generated and simulated. With the filtering, $K * N$ tests could be generated and N selected important tests would be simulated. The idea is that the N *selected* tests from the $K * N$ tests should provide a higher coverage than the *first* N tests from the $K * N$ tests. Consequently, a more effective test suite can be obtained.

The second component intends to automate the learning process for guiding the test generation G . The goal is to provide information on how to improve coverage on a coverage point or a family of coverage points. The learning approach in this application context is very similar to the transduction diagnosis methodology discussed earlier. Essentially, it is still a feature-based diagnosis approach [4].

Just like the application contexts reviewed above, to apply machine learning in the two functional verification contexts, the most critical and time-consuming step is also to set up the learning environment, e.g. preparing the dataset. This setup needs to consider a number of questions [3]:

- What is a sample?
- What information is in the data?
- How is a sample represented (or encoded)?
- How is the learning model represented?
- How will the learning model be used?

In setting up a learning environment, perhaps the most important aspect is to provide a definition of the sample. For example, a sample can be an entire test or selected instructions from a test. A sample can also be a state vector based on a selected set of microarchitecture states appearing in a test.

Generally speaking, the data includes the simulation trace and the corresponding coverage report. In RTL simulation, a simulation trace is a cycle-based simulation dump file based on a set of selected signal names from the design. If one desires to learn a relationship between a microarchitecture state and a target signal, both the state signal name and the target signal name need to be included in the simulation dump instruction file.

Sample representation is another important aspect of consideration. If the goal is to learn a rule, a sample needs to be encoded as a vector form based on the selected features. If the goal is to build a one-class novelty detection model, a sample might not necessarily be represented as a vector [3].

Different learning algorithms produce learning models in different forms. The representation of a learning model can be based on rule, tree, equation, a collection of the samples (e.g. an SVM model), etc. This leads to the next consideration for how will a learning model be used. For example, if the model is first to be interpreted by a person, then a complex model like an SVM model would not work.

2.4.1 Test filtering to improve verification efficiency

The work in [51] pioneered the idea of building a test filtering component, by solving it as a novelty detection problem with kernel-based unsupervised learning. The idea is simple and can be summarized as the following. Suppose a set of tests T has been simulated. In novelty detection the learning machine builds a model to capture the

“coverage space” represented by the tests in T . The model is then used to filter future tests. If a future test falls inside the modeled coverage space, it is filtered out. If a future test falls outside, it is predicted as a novel test and selected for simulation.

It was recognized in [51] that the effectiveness of a test filtering component based on a kernel-based learning model, largely depends on the kernel in use. A kernel in this context computes a similarity score between two tests. As a result, a test is deemed novel or not depends on this computation.

The work [52] extended the idea to learn novel tests where tests were given as assembly programs. Each program was represented as a *program flow graph* and a kernel between two flow graphs was defined based on an *edited distance* between them, i.e. the number of graph operations to transform one graph to another. While the graph-based kernel worked well on small designs with short-length programs, it was unknown how well it might perform on a complex commercial processor where the tests could be relatively much longer programs in length.

Then, the work in [53] applied the idea on a commercial dual-thread low-power processor core. The verification context was unit-level verification on the fixed-point unit and the load-store unit. Load-store unit was among the most complex units in a typical processor design. It is interesting to note that the work in [53] abandoned the graph-based kernel idea proposed in [52] earlier. This was because converting the tests for the commercial processor into program flow graphs, by itself was a difficult problem.

The central idea in [53] was to measure the similarity between two tests based on approximate coverage estimations on the tests. The work proposed to build a coverage database comprising a large number of instances (t, c) where t was a short test up to three instructions and c was the coverage table of the test. Then, approximate coverage estimation on a given test is calculated based on information provided in this database. The work demonstrated that the novelty program detection model based on such a kernel

could be quite effective.

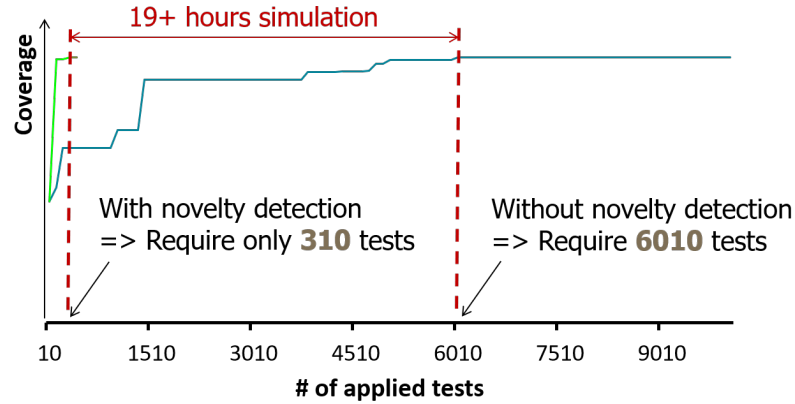


Figure 2.9: A notable result on improving simulation efficiency, later reported in [3]

Figure 2.9 shows a result reported later in [3] based on the same experiment setup in [53]. Without the novelty program detection model, the maximal coverage was achieved with 6010 tests. With it, the maximal coverage was achieved with only 310 tests. The actual saving of simulation resources was measured in run time, and was estimated about 19 hours on a high-end simulation server.

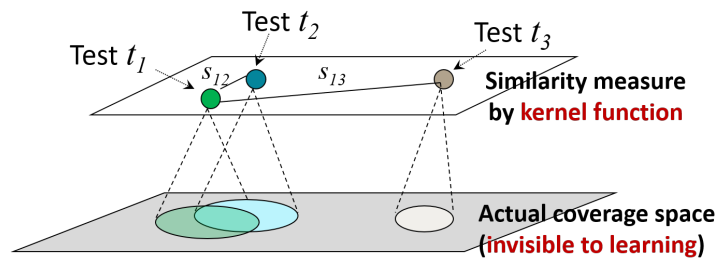


Figure 2.10: The fundamental issue of kernel-based learning in functional verification

From the early work [51] to the work [52] and then to the work [53], we could say that the underlying issue was to come out with a kernel function that made sense for the application context. The coverage database idea proposed in [53] was more robust and easier to implement in practice. It was also discovered that the similarity measure did

not need to be close to perfect for the filter component to work effectively. [52][53].

Figure 2.10 illustrates the underlying issue faced in the approach. A kernel function measures the similarity between tests in some space. However, this measure only makes sense if it reflects the actual similarity situations in the actual coverage space that is invisible to the learning. For example, if t_1 is deemed more similar to t_2 than to t_3 by the kernel function, then intersection of t_1 's covered set to the t_2 's covered set should be larger than that to the t_3 covered set. In a sense, the actual coverage situations are unknown to the learning, the kernel function is only “assuming” what would happen in the actual coverage space about different tests. If this assumption is far from the reality in the coverage space, then the novelty detection model would not work well.

Because the actual coverage space depends on the set of coverage points under evaluation, it is not hard to see that a kernel function ignoring the set of coverage points would not work well. This was why the coverage database idea proposed in [53] was more effective than the graph-based kernel idea, as the coverage database idea took into account the set of coverage points in a given application context.

2.4.2 Learning to guide test generation

The scenario reviewed in section 1.3.1 before belongs to this category where learning is to guide the test generation for obtaining a test that can help improve the coverage. In most scenarios, the test is obtained by refining an existing test or test template.

Figure 2.11 illustrates the learning setup with two levels of learning: input level and template level. The early work in [54] applied learning at the input level and the later works in [4][5] applied learning at the test template level.

The *implicant learning* is a simplified view of the actual learning problem. In practice, hitting a verification target usually requires controlling multiple signals, resulting in

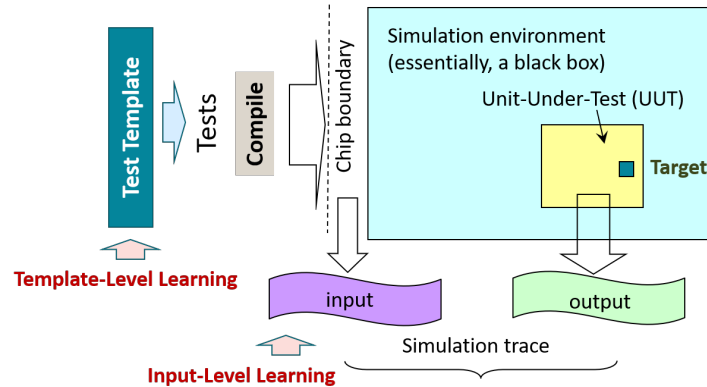


Figure 2.11: Two learning points in a functional verification environment

multiple implicant learning problems combined.

Suppose the input \mathbf{X} comprises m vectors $\vec{x}_1, \dots, \vec{x}_m$ where each \vec{x}_i essentially can be seen as a binary vector of length n . For simplicity, the output \vec{y} concerns only one particular signal and with m input vectors, \vec{y} can be seen as an m -value vector. Essentially, the learning is based on the dataset (\mathbf{X}, \vec{y}) where the m samples is drawn from an unknown Boolean function. The learning can be seen as learning the *implicants* of this function based on the samples [54].

In general, the implicant learning problem can be thought of as learning the Boolean function itself. However, as studied in [55] learning a Boolean function works only when the function has limited complexity. In fact, learning a Boolean function in general is a difficult problem [56]. As a result, the work in [54] applied the concepts from Association Rule Mining [57] to implement a method for learning implicants.

The works in [4][5] were applied in a Constrained Random Test Generation (CRTG) verification environment. In CRTG, a test is randomly instantiated from a *test template* that is constrained. For example, a test template is an assembly problem where some operands can be randomly selected from a set. In verification, each test template is instantiated into some number of tests. These tests are called *constrained random* tests.

They are in contrast to *direct tests* composed entirely manually by a verification engineer.

In [4][5], the learning is based on dividing a set of constrained random tests into two group: important tests (positive samples) and unimportant tests (negative samples). Important tests are those interesting for learning a property. Like other scenarios discussed before, typically one has few important tests and many more unimportant tests. Hence, the learning has the same challenge discussed before, i.e. scarcity of the positive samples.

The work in [4] used features separated into two levels: microarchitecture state features and instruction-based features. When the analysis was carried out with state features, the learning tried to uncover important state feature combinations. Then, with the instruction-based features, the learning tried to uncover what instruction properties were likely to achieve those combinations.

Learning at the test template level avoided the complicated implicant learning problem. The features were used to learn a rule to guide modification of the test template for hitting the target. Once the two classes of tests were represented in terms of the features, classification rule learning could be applied to uncover properties of the positive samples as rules [4].

Because the emphasis was on the discovery of properties of the positive samples, the number of the positive samples was much more important than the number of the negative samples. As discussed in [5], as the number of negative samples grows beyond a point (e.g. 10K), adding more negative samples would not help the learning.

Figure 2.12 shows an example to demonstrate the effects of the learning [4]. The experiments were conducted on a commercial processor verification environment. Originally, the tests based on the test template written by the verification team were not able to deliver good coverage on a family of five assertions - only assertion IV was hit once by 2000 tests generated. By learning the properties on the test hitting the assertion IV, the test template was modified and used to generate 100 more tests. This resulted in

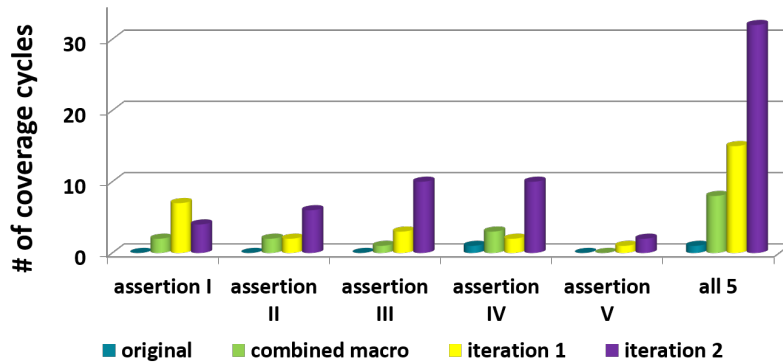


Figure 2.12: A notable result on improving verification coverage, reported in [4]

(labeled as "combined macro") coverage of assertions I to IV. With more positive samples available, the learning was carried out again. Test template was modified accordingly and 100 more tests were produced. With the 100 tests (labeled as "iteration 1") all five assertions could be covered. The learning was applied again and the result (labeled as "iteration 2") showed further improved coverage.

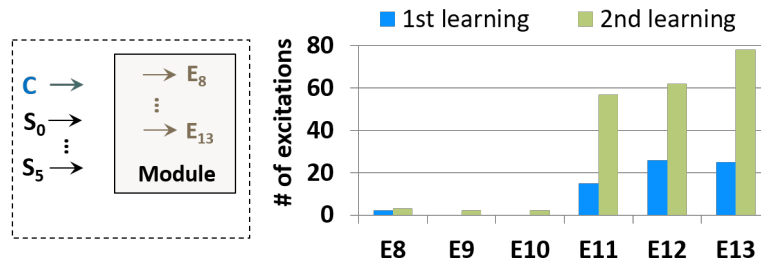


Figure 2.13: A notable result on improvement from zero coverage, report in [5]

Figure 2.13 then shows an example to explain how the learning could be applied when the initial simulation had zero coverage on the events under consideration [5]. In this example, the events were to cover E_8 to E_{13} . Simulation of more than 30K tests generated from a given test template could not cover any of the six events. Because there was no test to cover any of the events, there was no positive sample to learn from.

For this example, domain knowledge was applied to recognize that signal values on C and on S_0 to S_5 were highly related to the activation of the six E 's events. Hence, learning was used to learn about the properties related to the signal C and the signals S_0 to S_5 . Figure 2.13 shows that after one iteration of learning, the modified test template could cover 4 out of the 6 events (with 1K new tests). By learning on the new positive tests, the modified test template could cover all 6 events with 100 tests [5].

2.5 Customer Return Analysis

Customer returns are parts that pass all the tests before shipment but are determined as failing parts by the customer. Customer returns are sometime called *Customer Quality Incidents* (CQIs) because returning by the customer does not necessarily mean that the parts are defective from the part supplier's perspective. As reviewed before in section 1.3.3, a typical way to analyze customer returns is failure analysis through physical inspection. The desired outcome of CQI analysis is to obtain a test to prevent similar incidents from happening in the future.

Physical inspection through failure analysis is an expensive process. Hence, it is desirable to pursue an alternative to develop a test capable of screening out similar failing chips. One popular thinking is to construct an outlier model for the screening, which is based on learning from the CQI in view of the test data. For several years, this thinking led to the outlier screening studies for CQIs with the automotive product lines in Freescale Semiconductor, Inc. [58][6][32]. Preventing CQIs is a critical consideration for automotive product lines as their quality requirements are usually much higher than other consumer product lines. At the time, companies were aiming for sub-one DPPM quality, i.e. no more than 1 CQI per million chips shipped.

As reviewed before in section 1.3.3, to find an outlier model for a CQI, one needs to

define a *test space* for the outlier model to operate in. A test space is defined based on one or more tests [6]. Searching for a proper test space is part of the outlier modeling effort. Even with a fixed test space, construction of an outlier model can still be quite subjective, depending on various choices such as the base sample set, the outlier method, and the outlier threshold. Consequently, the real challenge of outlier model development is not about getting to an outlier model, but about justifying a given model [6][32].

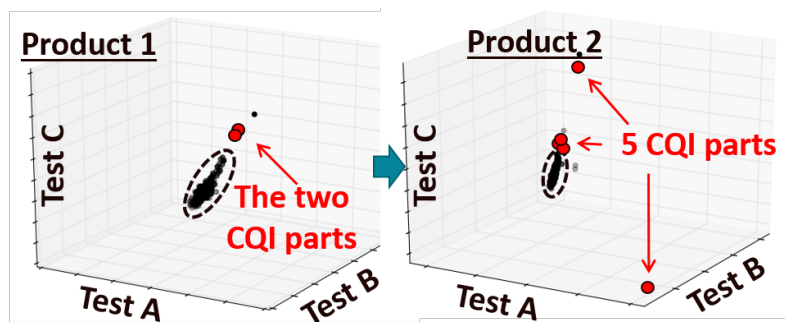


Figure 2.14: Justifying an outlier model with multiple CQIs

An outlier model might be justified in different ways. For example, Figure 2.14 shows that a 3-dimensional outlier space was learned based on one CQI, which also projected another CQI as an outlier in view of the same model. The second CQI provided a justification of the model. When the model was applied to a sister product, it could capture 5 more CQIs as outliers, hence providing further justification. These products were SoCs sold to the automotive market [32].

Figure 2.15 shows another example where the outlier model was justified validated through design knowledge and failure analysis report [7]. In this example, 10 CQIs were analyzed. It was known that they all failed due to some defective DC pins on the sensor interface. 7 out of 10 CQIs could be projected as (marginal) outliers in a test space comprising two tests related to the DC line.

In practice, justification of an outlier model and deployment of the model are two

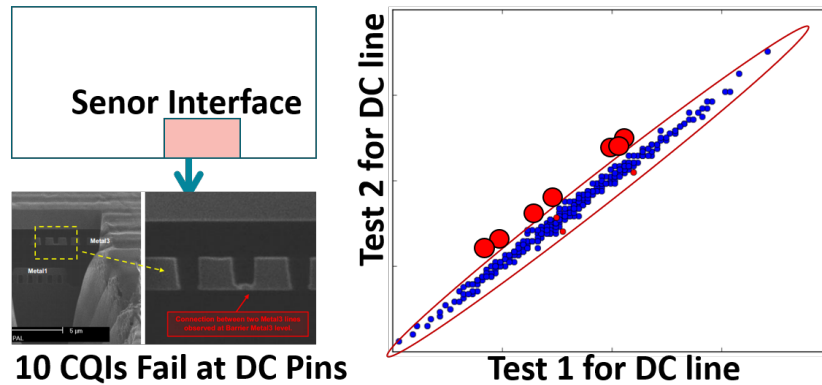


Figure 2.15: Justifying an outlier model with domain knowledge

different things. Model deployment needs to take another consideration into account: the resulting yield loss by the model. For example, the model in Figure 2.15 had about 600 PPM loss estimated based on the wafers where those CQIs were located. 600 PPM could be an acceptable figure. However, when the model was applied on 500+ wafers, the observed loss was about 12.1K PPM which could not be accepted by the product team. As a result, the team had to look for an alternative way to screen those CQIs, e.g. by developing a specialized test focusing on the type of failure.

The work in [32] studied 62 CQIs over a time window. One question was whether it was possible to build an outlier model using an earlier CQI to screen out a later CQI. If possible, then the question concerned what was the response time for learning the model. This response time spanned from the date the earlier CQI was reported back to the company to the date the later CQI was tested.

Figure 2.16 shows that 7 CQIs could have been prevented, i.e. it would have been possible to find the outlier models based on earlier CQIs within the response times and screen out those 7 CQIs at their test times.

In the figure, CQIs are named as R# according to their chronological order in the time window. The green triangle pointed downward marks the shipped date of a part

that would eventually fail in the field. There was a period of time before the part was returned by the customer. This period of time is represented by the light blue bar. When the part was returned, the date is marked by the yellow triangle pointed upward. At this time, it was possible to learn from the CQI to build an outlier model. The red arrow pointed downward marks the last test date of the future CQI that could have been screened out by the outlier model, if it was ready. Hence, the response time is bounded by the period between the yellow arrow and the red arrow.

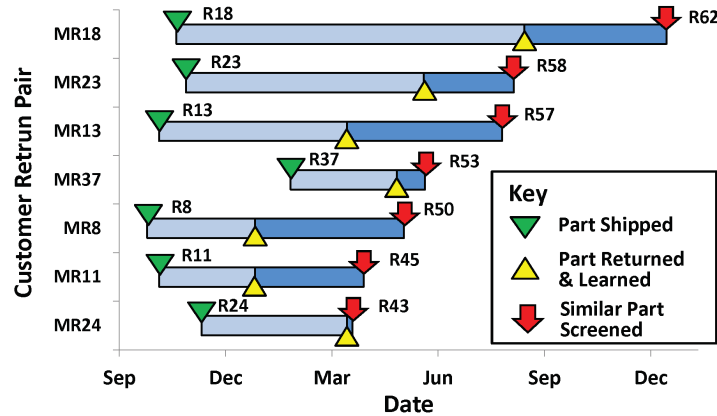


Figure 2.16: CQI model effectiveness shown in time view, reported in [6]

The response time was 5 days between R24 was returned and the last test date of R43. This was the minimal turn-around time required for the learning. The response time was more than 3 weeks between R37 and R53. Hence, if learning and deploying an outlier model required more than a week, R43 would be missed but the other six could still be prevented.

It is interesting to note that in practice, a simpler outlier model is more likely to be accepted for deployment. For example, it would be harder to accept and deploy a 3-test (3-dimensional) model than a 2-test model (2-dimensional). This was why along the line of the studies, after all the works in [58][6][32], the later work in [7] advocated that the model search should be constrained with simple models.

2.5.1 Considerations in outlier model development

Outlier analysis is similar to novelty detection, and could be thought of as a form of unsupervised learning. In practice, an outlier score is only a relative measure, i.e. relative to some sample population used to judge the outlier property. Hence, to construct an outlier screening model, one has to first decide on the population of the parts used to calculate the outlier scores. Earlier, we called this set the *base set*.

Being an outlier does not mean being abnormal

In testing, it is common to construct an outlier model based on wafer probe tests. Hence, it is natural to use all dies on a single wafer as the base set. In practice, one desires to screen out “gross” outliers. Figure 2.17-(a) illustrates this aspect.

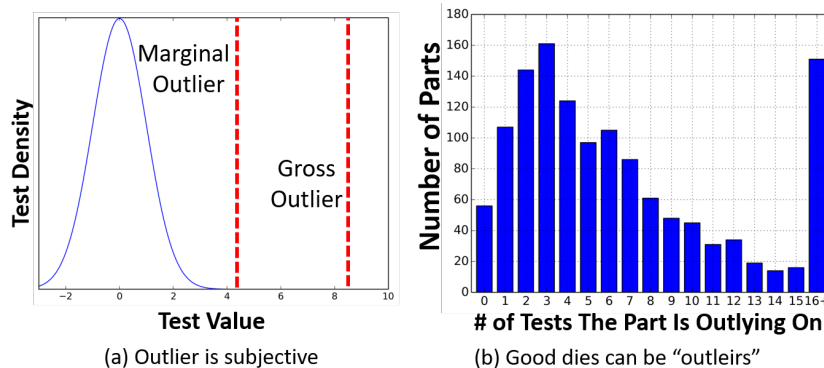


Figure 2.17: Being an outlier does not imply being abnormal [7]

The figure shows a distribution of all samples from a base set according to their test values. This is a hypothetical picture for illustration purpose only. Typically, one desires to draw the outlier threshold away from the distribution. This decision can be subjective. In practice, this decision is impacted by the concern of yield loss due to over screening. Because the threshold is set far away from the distribution, it is possible that a part whose value falls to the left of the leftmost red dash line gets shipped to the customer

and becomes a CQI, i.e. the part is a “marginal” outlier under this test. Hence, when the part is returned, one desires to adjust the threshold to screen out “marginal” outlier.

Finding an outlier model for a given CQI can easily run into an over-fitting situation. Figure 2.17-(b) illustrates this point. The plot was based on 1000 randomly selected good dies. The test data was based on an airbag sensor product line and the number of wafer probe tests was greater than 950. Those tests were parametric tests, i.e. measurements that give values rather than logical tests that output pass/fail.

The x-axis shows the number of tests on which a die shows outlying behavior. Here, being an “outlier” on a given test was defined as being among the top five most outlying dies on the wafer according to their test values. In this case, the die could be seen as a marginal outlier based on the test. As the plot shows, only less than 60 dies were not classified as an outlier with any test. All other dies were outliers based on one or more tests. More than 150 dies were outliers with respect to ≥ 16 tests.

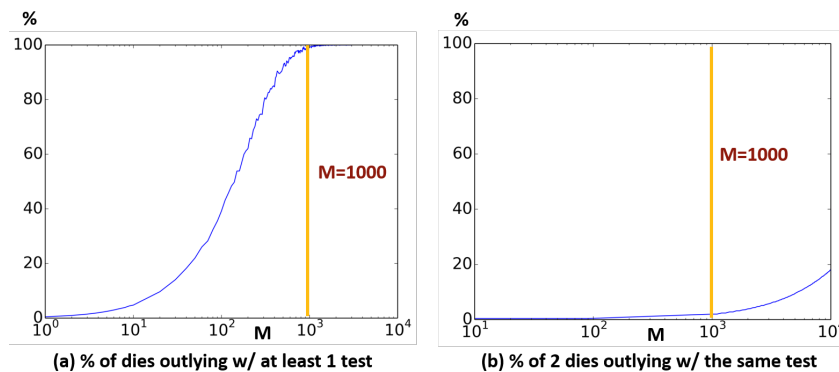


Figure 2.18: The reason why we need a 2nd CQI to justify a CQI outlier model [7]

Plot in Figure 2.17-(b) could be explained with a simple intuition. Assume that there are M tests and 1000 selected dies. Further assume that the measured values of each test follow a given Gaussian distribution. In a Monte Carlo simulation, let the measured values of each die be randomly drawn from the Gaussian distribution. Figure 2.18-(a) shows a plot where the y-axis is the percentage of dies found to be an outlier based on at

least one test and the x-axis is the assumed M . Again, an outlier is defined to be among the top five most outlying dies with respect to a particular test distribution. Observe that, as the number of tests M grows passing 1000, almost all dies become outliers, outlying based on at least one test. The intuitive is that with enough number of tests, any die can become a marginal outlier according to one of the tests.

Figure 2.18-(b) shows a different plot by ignoring tests in Figure 2.18-(a), which are associated with only one outlier. In other words, only tests with two or more outliers are considered. In this case, we see the percentage of outliers drops significantly. This is why in CQI outlier modeling, an outlier model for a given CQI needs to be validated with some other CQIs. Otherwise, the model is likely to be over-fitting.

Outlier modeling based on correlated tests

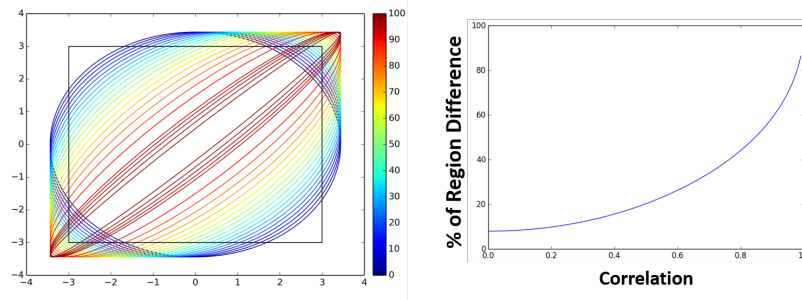


Figure 2.19: The reason why in test, one prefers an outlier model based on correlated tests [7]

In practice, parametric tests are often grouped into families of tests and within each family, values of the tests can be correlated. In outlier modeling, it is more often to use correlated tests (e.g. from a family) to define an outlier space than to use uncorrelated tests. Figure 2.19 provides an intuition for this practice.

The left plot in Figure 2.19 shows a 2-dimensional test space based on two hypothetical tests, with their test values across x-axis and y-axis. The rectangle box in the test space

shows the “good die” region based on some assumed upper and lower limits of each test *individually*. In other words, any die whose test values falling outside the rectangle box is screened out.

Suppose test values on the two tests follow the same Gaussian model assumption. For a 2-dimensional outlier model, we can simply consider a co-variance model, e.g. within the 3σ bound [7]. Then, the inlier region of this co-variance model depends on how the two tests correlated. The different inlier regions based on different correlation assumptions are shown in the plot with different colors.

Let A be the inlier region based on the rectangle box (region inside the box) and let B be the inlier region based on a correlation assumption (region inside the colored oval), the region difference $D = A - B$ can be seen as the actual screening region based on the co-variance outlier model. As seen this D is larger as the two tests become more correlated. The right plot illustrate this point by plotting how $\frac{D}{A}$ (y-axis) changes according to the correlation of the two tests (x-axis). As seen, $\frac{D}{A}$ is larger when the correlation is higher.

High-dimensional and/or complex models not preferred

Consider the wafer probe data from the airbag sensor product again [7], Table 2.1 shows how the number of possible models can grow fast as the dimensionality of the test space grows. This was based on one CQI on one wafer with about 1300 dies, i.e. the base set. Recall that the number of tests is 950+. The definition of being an outlier is the same as above, among the top 5 most outlying dies. The models were built using one-class SVM [38]. As seen, with a 3-dimensional space there can be 795K possible models (out of about 142M 3-test combinations).

It is a myth that people often might think using a more “advanced” algorithm is “better”. Figure 2.20 provides an opposite view to this myth. The figure compares an SVM one-class model to a simple co-variance model. In [7], it was found that 93% of the

Table 2.1: The number of possible outlier models can grow substantially [7]

Test space dimensionality	1	2	3
Number of possible SVM one-class outlier models	11	3027	795128

tests had their test values follow a Gaussian distribution and 1.3% follow a exponential distribution. Others failed the two statistical hypothesis tests for the two distributions.

For a test whose test values follow a Gaussian distribution, the benefit of having a more complex model like one-class SVM diminishes. One of the reasons why a test value distribution (based on a wafer) does not follow a Gaussian distribution is due to some *within-wafer variations* which can be caused by the manufacturing process or the wafer probe testing. For example, in wafer probe four dies might be tested in parallel at once and there are some systematic variations across the four test site.

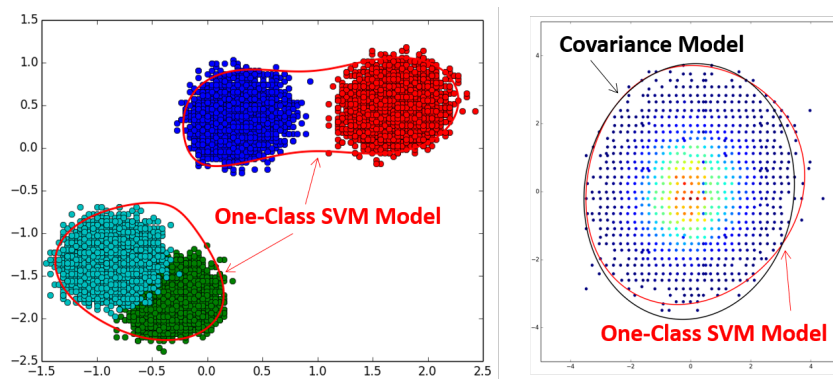


Figure 2.20: For outlier screening a simple co-variance model might suffice [7]

The left plot in Figure 2.20 shows an example of this site-to-site variations. The test values obviously do not follow a Gaussian distribution. Hence, a one-class SVM model, as shown, could take the clusters into account while a co-variance model could not.

On the other hand, if one knew site-to-site variations exist and this would happen, then one could easily align the test values from different sites according to their means. After the four clusters are aligned, a co-covariance model would still suffice, which is

shown in the right plot. In the right plot, a co-variance model and an SVM one-class model are compared. As seen, their inlier regions overlap significantly.

While pursuing a higher dimensional model or employing a more complicated model-building algorithm might be considered advancements in problem-solving for many application scenarios, this is usually not the case in customer return analysis (and outlier screening in general). Outlier screening prefers simpler models and this simplicity can manifest in several ways: constructing low dimensional test space, using correlated tests, and building a model with a simple outlier algorithm.

2.6 Other Areas to Apply Machine Learning

The above reviews the works in the four areas highlighted in Figure 2.2 before: design-silicon correlation, speedpath analysis, functional verification, and customer return analysis. In the following we will provide a brief review for each of the other six areas. Lessons learned from the works in those six areas echo the experience learned in the previous four. The overall conclusion is that, in general, applying ML to an application context in design and test was far more complicated than initially anticipated when starting on the work in this context.

2.6.1 Yield optimization

The success story reviewed in section 1.1.2 was the only work done in the area of yield optimization in the period. The work also marked the end of the period, which will be explained in more detail in section 3.1 later. It should be noted that yield optimization is an important area for a product line and the reason that there was not much works published in the period, was mostly due to lack of data for the research. Yield data was often regarded highly confidential and without access to this data, data-driven research

would not be possible.

2.6.2 Test cost reduction

Another important area of test data analytics is for test cost reduction. Test cost reduction can take place within a test stage [59] or across multiple stages [60]. During within-stage test cost reduction, one desires to remove a test or a test insertion. For example, in [59] the goal was to remove some of the parametric tests while maintaining the same quality level. This type of test cost reduction is typical, as testing usually starts with a higher volume of test content at the beginning of production, which is gradually optimized during the early period of the production. The intuition behind this is that when initial knowledge is limited, a more conservative approach is needed, whereas as knowledge accumulates, a more aggressive strategy can be adopted.

During cross-stages test cost reduction, one desires to use the test data from one test stage to predict the potential failing parts of a future stage. For example, one desires to use the wafer probe test data to predict potential parts that may fail in the burn-in test stage [60]. By predicting these parts, only those parts identified need to go through the burn-in process, thus saving burn-in costs on other parts. However, as explained in [2], predicting future burn-in fails can be a difficult task. A part may appear perfectly normal in wafer probe testing, yet exhibit obvious defective behavior after burn-in. This is because the burn-in process alters some characteristic on the part. Indeed, a weak part is among those whose characteristics are changed by burn-in. Before burn-in, these characteristics have not yet been altered, making it difficult for using wafer probe test data to predict such changes.

In a subsequent paper [61], it was explained that achieving test cost reduction while aiming for a minimal DPPM impact can be exceedingly challenging. For example, if the

DPPM impact is limited to 1, this means that by implementing a cost reduction method, the method is guaranteed to miss at most 1 defective part per million. Defective parts often occur at the tail end of a test distribution, and meeting such a requirement demands a model capable of capturing the behavior of this tail in high accuracy, which can be quite difficult [61].

2.6.3 Fmax prediction

The term “Fmax” stands for the maximum frequency a chip can be operated on. *Speed binning* is the process to categorize chips according to their Fmax. Speed binning is usually carried out in a system test environment and hence, commonly considered as an expensive step. As a result, one desires to minimize the use of this step by finding an alternative on deciding the Fmax. In the early dates, this alternative could be based on establishing a correlation between a low-cost on-chip measurement and the Fmax. For example, the measurement can be for measuring the speed of a ring oscillator, the delay of a critical timing path, the maximum frequency based on a set of transition pattern set, and/or performance of some circuitry designed specific for the purpose. We can call this the *univariate Fmax modeling* era, see e.g. the work in [62].

When ML became more noticeable in the semiconductor design industry, companies started to be interested in more “advanced” Fmax modeling methods and hence, started the studies focusing on *multivariate Fmax modeling*, i.e. predicting Fmax based on a set of low-cost measurements collectively, see e.g. [63][64][65].

An Fmax prediction model is essentially a regression model [63] to map low-cost measurements $\mathbf{x} = (x_1, \dots, x_n)$ to Fmax y . The work in [63] studied five different learning methods: nearest neighbor, least-square fit, ridge regression [66], SVM regression [38], and Gaussian Process (GP) regression [67]. The focus was on the *applicability* of a model,

i.e. asking the question whether or not a given model should be used to predict the Fmax of a given part. The work suggested to use a confidence band based on GP modeling to help assess model applicability. In addition, the work proposed an idea called *conformity check* to identify training samples as “noisy samples” that ought to be removed from the training dataset.

There were three issues that prevent adopting a ML-based Fmax prediction model. First, it was hard to bound and characterize the prediction error of a model. Fmax for a chip represents a promise from the chip supplier to its customer. For example, if a chip is labeled as 2.4GHz, it is promised that the chip will function correctly with that frequency. Hence, a prediction error that results in labeling a chip as faster than it should be means a defective part and can become a customer return. If the DPPM requirement is 50 DPPM, this means that the one-side error has to be below 0.005%. This is hard to achieve.

Second, the number of samples used to learn a model can be small, e.g. a hundred [64][65]. This was because chips with Fmax labels were usually characterized from a dedicated system test lab. The characterization was comprehensive and complicated. One of these chips’ focuses was to achieve a reliable Fmax test suite to be used for determining the Fmax. The job was not to provide samples for building a reliable ML-based Fmax prediction model.

Third, due to process variations, the regression function to map \mathbf{x} to y could change from one production period to another. For example, a process recipe change might be the cause. As a result, it is hard to ensure that a model learned from chips produced in one period has the required prediction power for chips produced in the future, especially under the extreme accuracy requirement mentioned above.

Overcoming the three issues above in practice was very challenging. As a result, the work on applying ML in Fmax prediction in our lab terminated after 2010.

2.6.4 Layout hot-spot prediction

A *layout hot-spot* is a piece of layout which has a high chance to cause an issue in chip fabrication. An “issue” usually means that the actual geometric shape of the piece can deviate significantly from the shape in the layout model. Hot-spot detection is an important step during library cell development. A typical way to identify layout hot-spot is through lithography simulation, which is complicated and time-consuming. Hence, there has been a desire to build a ML hot-spot prediction model as a faster alternative [68][69].

As one of the pioneering works in this area, the work in [68] applied SVM binary classification to build a prediction model. The training data comprised good layout samples and bad layout samples where good and bad were determined by the lithography simulation. These samples were extracted from a given layout by applying Raster scanning, i.e. moving window with overlapping boundary [68].

Similar to the filtering component discussed in section 2.4.1, the main challenge for learning an SVM model was on choosing an effective kernel function. The work in [68] used the Histogram Intersection (HI) kernel and experimentally showed its effectiveness. Another important issue was on selecting of the window size in the Raster scan, where in [68], the size was determined experimentally with cross validation.

A fast hot-spot predictor, even if not highly accurate, can still be useful during the cell library development process. The design development process is iterative, where a designer makes adjustments to the design, runs simulations, and makes further adjustments. Once the design is finalized, it then undergoes the final verification through simulation again. By using a fast and approximate predictor, the iteration process can be expedited, reserving the slow lithography simulation for the final verification stage. Since the final verification is still guarded by the lithography simulation, any prediction

errors are guaranteed to be detected later. Hence, a prediction error does not translate into a quality issue but only affects the efficiency of the process.

However, assessing the added-value of using a predictor can be difficult. Even during the development process, the slow lithography simulation does not need to be applied to the entire design but only to the local regions currently being modified. In addition, a prediction error might cause a designer to expend effort on optimizing the design unnecessarily. Assessing this unnecessary engineering cost is not a straightforward task.

2.6.5 Delay testing

In delay testing, one of the main issues is to decide on the clock frequency used in testing. Note that this frequency is usually slower than the F_{max} . This is because delay testing applies test vectors structurally through scan structures, not functionally from the primary input of a chip. Often, the delay test frequency is determined via timing analysis and further learning from silicon samples. The work in [70] therefore tried to improve the characterization of delay test frequency through statistical timing analysis.

However, with a statistical timing view, the subsequent works in [71][72] provided an alternative to look at delay testing from a binary classification perspective. By applying a test pattern with a *faster-than-speed* test clock and collecting the result on a set of n flip-flops, a delay test characterization vector (c_1, \dots, c_n) could be obtained, where each c_i is either 0 (pass under the clock) or 1 (fail under the clock). Essentially, the idea was to turn testing into characterization. Then, by characterizing m sample chips, one could obtain a dataset of $m \times n$ characterization matrix. Suppose the pass and fail labels on these chips could be determined with another more rigorous and expensive test process. Then, one had a dataset for learning a binary classification model to predict the pass or fail of a chip based on its characterization vector, bypassing the difficulty for determining

the delay test frequency. The work in [70] used SVM for learning the model and the work in [72] used *random forests* [73].

The work in [74] explained the above idea in more detail and suggested to use multiple *faster-than-speed* test clocks for the characterization, essentially making the feature vector go beyond binary. Instead of taking a binary classification view, the work took an outlier analysis view to remove the necessity of having the pass/fail labels. In this way, an outlier deemed from a characterization matrix was considered a failing chip. The work studied three outlier methods: Principle Component Analysis (PCA) based analysis [75], random forests, and one-class SVM. The works in [76] and [77] later then extended the idea by optimizing the number of clocks, the test pattern set, and the chip samples in use for building models and for screening delay defects.

A barrier to adopt a ML model in delay testing is the cost. Traditional delay testing follows a deterministic process. A test pattern is applied with a test clock. If a chip fails the test pattern under the clock, the chip is considered as defective. Adopting a ML model means that the application of a test is treated only as a characterization of chip's behavior. To accurately determine the pass and fail of a chip, more characterization steps are required to enable learning the behavior. While the thinking is not scientifically flawed, adding even a single additional step can result in doubling the test application cost, which is practically prohibitive. As a result, it is difficult to adopt the above ML view in an actual delay test practice.

2.6.6 Circuit behavior modeling

Building a behavior model for a circuit is a common practice. A behavior model can be simulated faster than simulating the original circuit model. For example, a supplier of analog products provides behavior models (e.g. equation based models) for their products

to enable a potential customer to assess the functionality of a product in the simulation environment on the customer site. The works in [78][79] were done in this application context and the work in [80] applied Gaussian Process regression to learn behavior models based on SPICE simulation.

Given a circuit, its behavior can be observed by injecting various inputs and obtaining the outputs through simulation, e.g. SPICE simulation. Hence, creating a training dataset is straightforward, and it is intuitive to think about applying ML to build a behavior model. However, as pointed out in [81], traditional regression modeling may not be adequate for the application. This is mainly because the input of a circuit is often represented as a waveform, and similarly, the output is also a waveform. Further, even a slight change on the input waveform can result in a significant change in the output waveform. Such type of drastic behavior is usually of interest to engineers, while typical behavior that conforms to a trend is less interesting. Developing a model to cover all those corner behaviors can be challenging.

To overcome the challenge, the work in [81] then suggested building a collection of “micro” models where each micro model covered only a rather restricted input space. Then, on a given input the most appropriate micro model was used for its prediction. Accordingly, a method was needed to determine that no micro model was suitable and hence, the input could be deemed as unpredictable. The intuition for building micro models was that, within a restricted input space, the input-output mapping could be more accurately captured with a simple ML model, e.g. a linear SVM model. Moreover, using a collection of micro models enabled a way to determine that some input sub-spaces were not yet predictable because of lack of training samples [81].

The thinking for building a collection of simpler but more reliable micro models exemplifies the point mentioned in Section 1.4.6. In practice, it is desired to build a ML model for predicting simple and uncommon behavior rather than generalizing to common

trend.

2.7 Lessons Learned from the First Ten Years

The most important lesson learned during the first ten years was that the data-driven view was not sufficient for applying ML in design and test. In other words, applying ML in design and test was not as simple as solving a stand-alone ML problem. Consequently, the period began by taking this simple view, and ending with a view that was much more complicated. Figure 2.21 to Figure 2.22 illustrates the transition of the view.

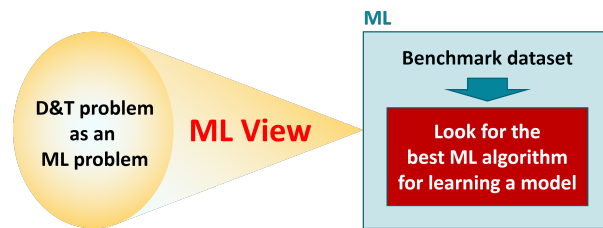


Figure 2.21: Seeing a D&T problem as a ML problem

As introduced at the beginning of this chapter, the data-driven view follows a standard ML practice, i.e. starting with a benchmark dataset that defines the problem. Applying ML to solve a D&T problem is to formulate the D&T problem in terms of a dataset for ML. Once the dataset is given and treated as a benchmark, the focus becomes finding (or developing) the best ML algorithm for learning a model with respect to the benchmark. The effectiveness of this practice largely depends on the assumption that the dataset is a benchmark representing the underlying D&T problem to be solved. In reality, this is rarely the case.

Figure 2.22 depicts a more realistic picture based on what we learned in the period. First of all, practicing ML in view of a D&T problem needs to consider three important aspects: (1) data sample sufficiency, (2) behavior complexity, (3) feature space adequacy.

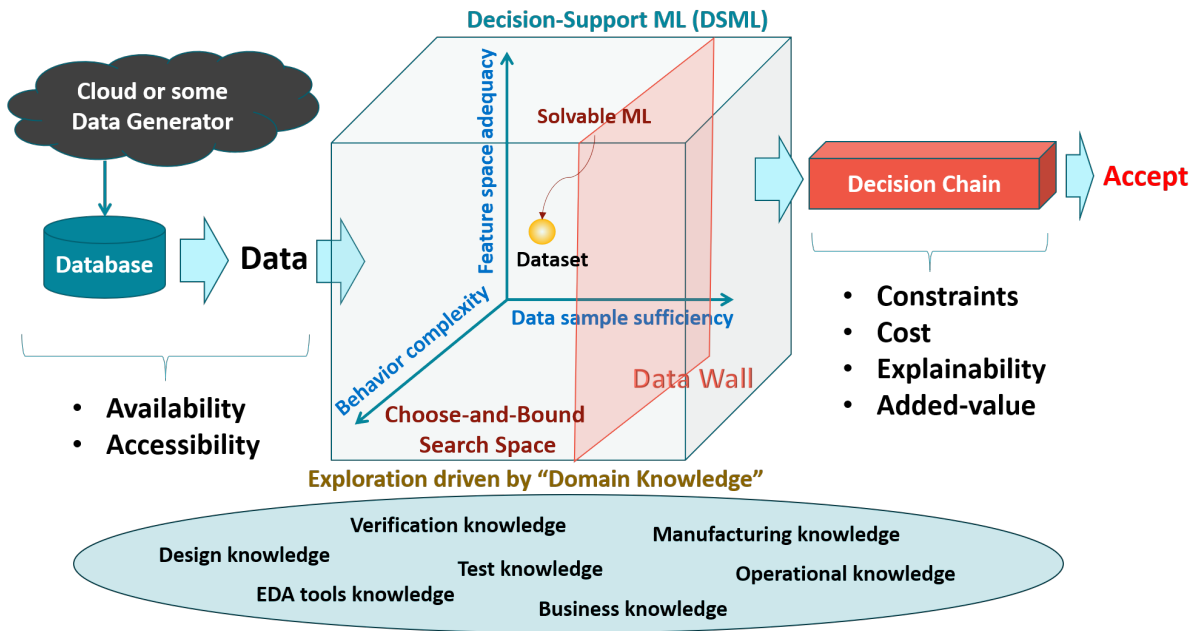


Figure 2.22: Seeing a D&T problem with a more realistic big picture

These three aspects are illustrated as three dimensions in Figure 2.22.

2.7.1 Data wall

It is important to note that for all D&T problems studied in this period, there was a *data wall* limiting the sufficiency of data samples. In a D&T environment, data can come from various sources. For example, production test data usually come from somewhere in the cloud. It is common that a design company contracts a test service company to run their test flow. The test data is collected by the test service company and managed by a separation data organization, e.g. a 3rd-party data service company.

When one desires to use the test data in a certain application context, e.g. yield optimization, the *context-relevant* data needs to be downloaded into a local database. When one desires to conduct a task, the *task-relevant* data needs to be extracted from the database. This task-relevant data can be seen as the input to the “learning”. Notably,

this learning may not necessarily be machine learning but could be *human learning*, i.e. the learning is manually conducted by human and the knowledge learned stays with the human. Given the data, a dataset is then created for machine learning.

In a pre-silicon design environment, there can be various types of data: simulation results, design models, testbenches, design specifications, test plans, bug reports, reports generated by an electronic design automation (EDA) tool, etc. For tasks involving learning from data, if this learning was human learning, it could involve any of these data types. However, if this learning was machine learning, to our experience in this period, the data mostly referred to simulation results.

Regardless of the application context, a data wall for ML exists because of two aspects of issues: availability and accessibility. Availability concerns if the data needed to solve the problem is available or not. Unavailability can be inherent, i.e. for a problem there is no data containing the information for solving the problem. For example, a future yield issue might not be predictable by current data because the yield is caused by a process recipe change. A burn-in fail might not be predictable by wafer probe data because the fail is caused by unique characteristics in burn-in.

Unavailability can also be due to some barriers in the generation of data. For example, in a pre-silicon context, limitations such as simulation efficiency and time constraints to complete a task can restrict the amount of available data. In a post-silicon context, limitations like data downloading efficiency and time constraints can also impede data availability. Moreover, both simulation data and production test data are contingent on the applied tests. Consequently, due to a lack of proper tests, certain interesting behaviors may never manifest in the data.

Assuming the required data is available somewhere, accessing the data is another issue. Unaccessibility can be due to lack of the security privilege to access the data or in some cases, lack of accessing the right personnel to get to the data. Therefore, even

assuming the data is available, it does not imply that the data is always accessible.

2.7.2 Implications of the data wall

Ideally, end-to-end ML would become feasible in D&T application contexts if the data wall could be pushed to the right end infinitely, i.e. effectively there was no data wall. However, due to the presence of the data wall, the three dimensions of consideration shown in Figure 2.22—data sample sufficiency, behavior complexity, and feature space adequacy—all become part of the learning process itself. In essence, learning becomes an exploration along these three dimensions, driven by domain knowledge. Note that domain knowledge can involve knowledge from diverse areas, including knowledge from the design and design models, from verification practices, from test practices, from manufacturing process, from company specific operations, and from its business models. Further, a single area includes diverse knowledge by itself. For example, test practices can encompass logical testing, delay testing, analog testing, and memory testing etc., each associated with its specialized domain knowledge. We characterize this complex exploration bounded by the data wall as the Choose-and-Bound search earlier in section 1.2. In Figure 2.22 here, the Choose-and-Bound search is re-illustrated with a more complete picture.

It is a common step in machine learning to formulate and prepare a dataset under the assumption that it represents a problem solvable by applying machine learning. However, it's important to recognize that this assumption can often be false. Learning that such an assumption is false is part of the learning process. For example, when dealing with a particular behavior of interest, one formulates a particular dataset. However, the corresponding ML problem might be unsolvable due to either an inadequate feature space or insufficient data samples. Recognizing and reaching this conclusion requires

considerable effort. Once the conclusion is reached, one might take a step back and reformulate a different dataset. Alternatively, one might adjust the behavior of interest to be learned.

2.7.3 Decision chain

Suppose one reaches a dataset and produces an answer (a model, a rule, etc.). This answer is only locally acceptable from the learning perspective depicted in Figure 2.22. For the answer to be accepted, e.g. to trigger model deployment or an action, the answer needs to pass through a *decision chain*. In some contexts, the decision chain can involve people in different organizations. Acceptance of an answer depends on four aspects of considerations: constraints, cost, explainability, and added-value.

For example, a model is only acceptable when it meets a certain constraint. As reviewed before, a customer return screening model needs to meet the yield budget constraint. A burn-in fail prediction model (or Fmax prediction model) needs to guarantee no miss of any part that may fail the burn-in (no Fmax miss-labeling). These are domain-specific constraints usually not considered in a standard ML practice.

Cost is another important consideration. For example, in yield optimization changing the process recipe is quite expensive and hence, can only be done when the evidence is clear that the yield issue is due to the manufacturing process. In design-silicon correlation, changing the design is also expensive because silicon re-spin is expensive, in addition to all the verification cost associated with a change of design. In delay testing, increasing the test pattern size and/or increasing the number of test clocks represent significant increase of test cost.

Because the acceptance of an answer depends on people, explainability of the answer is important. It is hard for people to accept an answer and take an action without a

certain extent of understanding the answer, especially when there is a cost associated with the answer.

Even an answer is somewhat accepted under all of the above considerations, the answer can still be rejected if its added-value is not evident. In many cases studied during this period, existing tools and methodologies were already available for addressing the problems at hand. Therefore, an answer is only deemed valuable if it cannot be obtained using existing tools and methodologies within a reasonable timeframe. Furthermore, adopting an ML-based flow often means modifications to existing tools and methodologies. Without a clear understanding of the added value, it can be challenging to justify taking such actions.

2.7.4 Decision-Support ML (DSML)

At the end of this period, we could conclude that applying ML in a domain-specific scope as depicted in Figure 2.22 was fundamentally different from ML. To emphasize this, we could call the learning in Figure 2.22 *Decision-Support Machine Learning* (DSML). In view of the ten points highlighted in section 1.4 before and based on what we learned in the period, we could say that DSML was to apply (machine) learning with data in an application context where

- The problem definition is vague to begin with.
- There is no unified objective evaluation for the problem answer.
- The learning is in the form of exploration.
- The learning is to find explainable solution to support decision making.
- The learning includes applying ML and Co-ML.

- The learning cannot be just data-driven and it has to involve domain knowledge.
- The learning can enter a world of no-free-lunch.
- The success of the learning is only when a meaningful action is taken.

2.7.5 Fundamental challenges in DSML

If we were going to use one equation to define DSML, we could say that:

$$\text{DSML} = \{\text{ML} + \text{Co-ML}\} | \text{Domain Knowledge} \quad (2.1)$$

where $|$ stands for “conditioning on”. Then, it is intuitive to see that to develop a successful DSML solution, in addition to harnessing existing ML capabilities, one needs to answer four essential questions:

- What is the required Co-ML capability?
- How to implement the Co-ML capability?
- What is the required domain knowledge?
- How to incorporate the domain knowledge in a DSML solution?

Consequently, in the next period our research was driven by this set of questions.

2.8 A Chronological Remark for the First Ten Years

Figure 2.23 depicts the first ten-year period in a chronological view of key ML developments leading up to the emergence of *deep learning*, marked by the famous AlexNet

[18] in 2012. It is interesting to see that in the period reviewed above, key ML techniques employed in the studies—such as SVM classification, SVM regression, SVM one-class, Gaussian Process regression, and CN2 subgroup discovery rule learning, random forests—all originated around the early 2000s. Hence, they could be seen as state-of-the-art techniques for ML from a technology consumer’s perspective at the time. Notably, none of the works in the period involved deep learning. This absence is understandable, considering that deep learning emerged in 2012, and its widespread applications became apparent only thereafter.

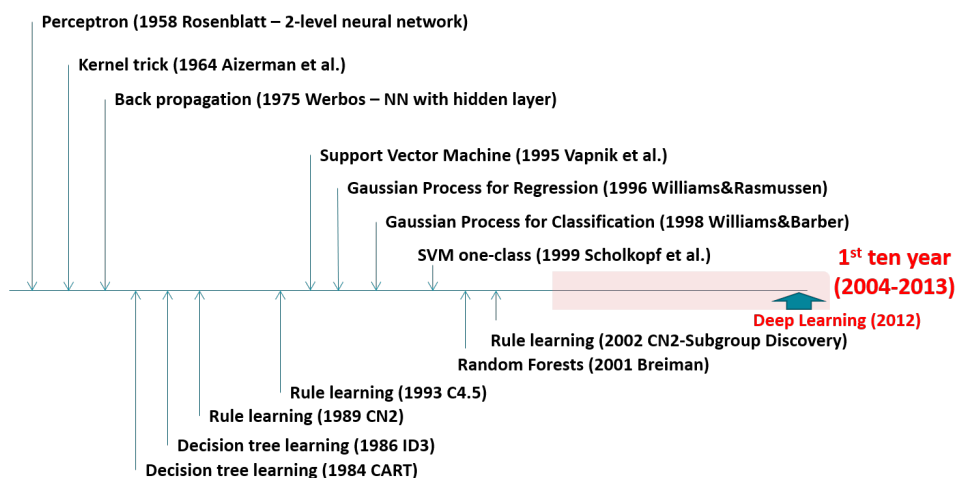


Figure 2.23: A chronological view of the first ten years (2004-2013)

2.8.1 Further clarification of terms, e.g. “Machine Learning”

Some of the works reviewed in the first ten years was not strictly classified as “machine learning”. For example, the yield optimization work [9] mostly involved finding statistical correlations. Outlier analysis mostly involved in finding a low-dimensional test space. In design-silicon timing correlation, sometime it was sufficient to find a simple decision tree. In some cases, using a sophisticated learning algorithm like SVM, Gaussian Process, or random forests, was not necessary. Therefore, some works reviewed above were referred as

data learning rather than machine learning. The term data learning was used to reference a more general sense of learning from data, which may include statistical learning and human learning.

As mentioned in section 1.4.4, we use the term *machine learning* to broadly refer to a tool or an algorithm that performs learning from data in terms of optimization. Then, depending on how “optimization” is defined, machine learning can mean differently. Under one definition, finding the optimal correlation or identifying the best outlier model might be considered machine learning. However, under a different interpretation, these tasks may not necessarily fall within the scope of machine learning.

To avoid such a debate, we employ the term DSML, which refers to applying ML for decision support in view of Figure 2.22. Later in Section 3.12 we will provide further clarification to differentiate DSML from ML. We will explain that the ML stated in equation (2.1) above is not with a traditional sense and in Section 3.12, the ML and Co-ML will be more specifically referred to as ML* and Co-ML*.

Throughout the first ten-year period, the term used was data learning. Towards the conclusion of this period, the transition to using the term DSML became somewhat apparent. This shift was prompted by the lessons learned, which enabled us to achieve a clearer understanding of why applying machine learning in design and test was fundamentally distinct from conventional ML practices.

Chapter 3

Journey to IEA - The 2nd Decade

半畝方塘一鑑開，天光雲影共徘徊。

問渠那得清如許，為有源頭活水來。

Half a mu of square pond, one mirror opens wide.

Heavenly light and clouds' shadows side by side.

Ask the stream, how it stays so clear?

For there's a source, where living waters appear.

— 《觀書有感 其一》 朱熹, A poem from Song Dynasty

3.1 Knowledge-Driven View (2014 – 2017)

Although the yield optimization paper [9] was published in 2014, most of the work was done in 2013. It took more than six months to obtain the silicon results, deploy the new recipe into the manufacturing process, and finally confirm the removal of the yield issue in mass production. The work marked the ending of the the first ten-year period, i.e. the ending of the data-driven view. From the work, it had become clear that learning from data demanded domain knowledge. As a result, in the next four years

“applying ML” was seen from a *knowledge-driven view*, i.e. a DSML view as described in sections 2.7.4 and 2.7.5 above.

After the tremendous success for resolving the difficult yield issue, the company immediately desired to duplicate the success on yield issues seen on other product lines. Other companies were also interested in the technology transfer. Often, people asked for a “tool” that could duplicate the success reported in [9].

In fact, this was not the first time of requests for a tool in view of technology transfer. As reviewed in section 2.1, over the first ten-year period, numerous successes were reported across multiple practices: design-silicon timing correlation, speedpath analysis, and functional verification in terms of both simulation efficiency and coverage, among others. Many sought a replication of these achievements through technology transfer. However, we could not provide a “tool” to such transfer because these successes were not solely attributed to the tools in use. In each case, the essential component for success was “domain knowledge”, predominantly residing in the human mind and not encapsulated within the tool itself.

3.1.1 Two types of domain knowledge to drive the search

In view of the yield optimization effort reported in [9], Figure 3.1 depicts two types of domain knowledge. While the concept of domain knowledge itself may seem elusive, its practical application suggests two types of domain knowledge: one to drive the ML and the other to drive the Co-ML. (Note: In the figure, “ML” is used to include correlation analysis, despite some potential interpretations differing from traditional ML definitions).

To drive ML, the knowledge is used to create a dataset and perform correlation analysis utilizing that dataset. To drive Co-ML, the knowledge is used to decide when the dataset is inadequate for finding a satisfactory answer, prompting the creation of

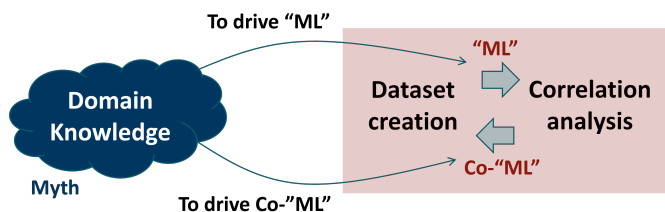


Figure 3.1: Two types of domain knowledge to drive the search

a new dataset. Overall, the domain knowledge is used to drive the Choose-and-Bound search.

If we were going to provide a piece of software to duplicate the success, we needed to incorporate a model of the domain knowledge into the software.

3.2 Modeling domain knowledge with an executable workflow

In a company, it is common to refer to a workflow for task execution as comprising both tools and methodology. Here, the methodology can be thought of as a flowchart, while each tool is associated with a specific step in that flowchart. An automation flow refers to a workflow that can be executed automatically. Therefore, it is intuitive to consider providing an automated workflow to execute our Choose-and-Bound search process conducted in the yield optimization work [9]. This thinking requires us to convert our domain knowledge into an executable workflow. The work in [8] follows this thinking. The high-level idea behind the work is depicted in Figure 3.2.

As illustrated with Figure 1.13 in section 1.4.4 before, the Choose-and-Bound search space could be thought of as a collection of datasets. In section 1.4.4, we emphasize that exhaustively enumeration of all possible datasets is impractical. Nonetheless, at the time the work in [8] made an attempt to enumerate this search space as much as possible.

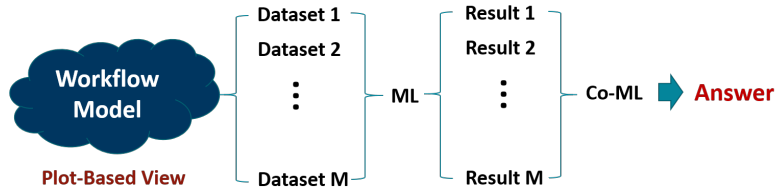


Figure 3.2: Modeling domain knowledge as a workflow based on a plot-based view

Hence, the idea in [8] was to capture the domain knowledge into a workflow model such that execution of this workflow generated a collection of datasets to be analyzed. Each analysis went through a “ML” analytic tool and produced the “best” result. Then, a “Co-ML” step was applied to select which results to report. The core of the analytic tool in the yield optimization context was *canonical correlation analysis* (CCA) [83] and the Co-ML step was based on the *risk evaluation* method implemented in the work [9]. Further detail for the Co-ML step is provided in Section 3.6.1 below.

3.3 Plot-based analytics

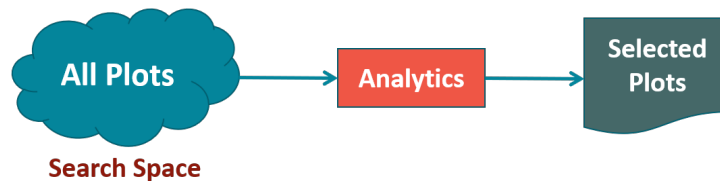


Figure 3.3: Plot-based view to enable definition of primitive analytic steps

An important idea proposed in work [8] can be called *plot-based analytics*. Although the concept was initially proposed in [8] and subsequently in [84][85], the specific term “plot-based analytics” was not explicitly articulated until the keynote presentation in 2020 [86]. Figure 3.3 illustrates the idea. Plot-based analytics conceptualizes the input for analytics as encompassing all conceivable plots, with the job of the analytics being to select certain plots for output. From this view, the search space comprises all possible

plots.

Given that the search space is defined in terms of plots, we can then consider a workflow that generates all possible plots derived from the data. Note that this enumeration of plots implicitly includes enumeration of all possible datasets derived from the data and hence, can be thought of as obtaining a representation of the Choose-and-Bound search space for a given task.

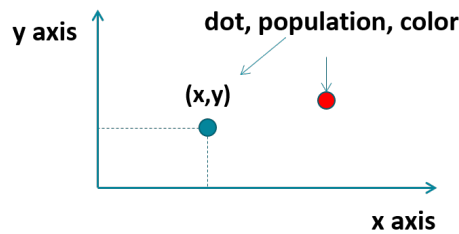


Figure 3.4: Plot-based view to enable definition of primitive steps in a workflow [8]

For example, in the yield optimization work [9], the prevalent plots utilized were two-dimensional scatter plots. Figure 3.4 then illustrates seven facets that might be associated with a scatter plot [8]. To formulate a specific scatter plot, a definition is needed for each facet. A step used to furnish this definition can be regarded as a *primitive step* executed by a workflow. As a result, the seven facets can be viewed as seven groups of primitive steps. The definitions of each facet are listed below:

1. Defining the meaning of a dot shown in the plot. For example, a dot can represent a single part or represent a single wafer.
2. Defining the population of the dots shown in the plot. For example, they are from a single wafer or from a single lot.
3. Defining the meaning of the x axis. For example, the x axis is based on an E-test.

4. Defining how the x value is calculated. For example, the value is calculated as the average across all sites.
5. Defining the meaning of the y axis. For example, the y axis is based on a test.
6. Defining how the y value is calculated. For example, the value is the test value.
7. Optionally, defining how to color a dot. For example, classifying the dots into two pass and fail and color them differently.

Each group includes multiple primitive steps to provide an *executable* definition for the facet. Suppose group i has n_i primitive steps. In total, the number of possible plots that can be generated is bounded by $n_1 \times \dots \times n_i$. Note that this bound represents an estimation because some primitive steps can have a dependency among them, e.g. one step might only be invoked after another has been previously executed.

3.3.1 Process discovery

Suppose a set of primitive steps are developed and each step is represented as a piece of Python code so that it is also executable. Then, constructing a plot corresponds to execution of a sequence of primitive steps. Such a sequence was called *hypothesis path* or *H-path* in [8]. A set of H-paths was called a *process log*.



Figure 3.5: Summarizing the main idea in [8]

The main idea in [8], summarized in Figure 3.5, was to construct a process log as the model for past analytic experience and apply *process discovery* [87][88] on the process log

to learn a workflow graph. For example, in work [8], a process log was built based on the procedures detailed in [9], and the learned workflow was then applied to address another production yield issue.

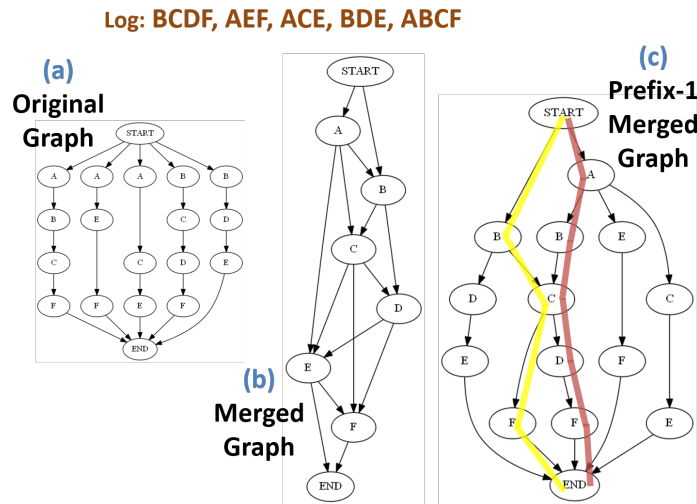


Figure 3.6: An example to illustrate process discovery [8]

Figure 3.6 shows a simple example to illustrate the basic idea of process discovery. In this example, the log file contains five H-paths based on primitive steps labeled A to F. Graph (a) basically depicts each of these H-paths individually within a single graph. Graph (b) consolidates by merging nodes with identical labels from graph (a). It is noteworthy that graph (b) contains paths (from **START** to **END**) that extend beyond the original five H-paths. In a sense, graph (b) generalizes from the process log to include new H-paths.

Generalization was needed because in [8], the goal of obtaining an executable workflow was to apply it, along with the domain knowledge modeled within, to solve a different yield issue. The resolution of a new yield issue was unlikely to be the same as that in [9]. In other words, following each of the H-paths in the process log built upon the work [9] was not sufficient for resolving the new yield issue. Hence, there was a need for some generalization to include new H-paths.

Generalization of a process log aims to encompass a broader search space. However, an excessive number of H-paths might not be desirable due to the increased costs and potential irrelevance. Hence, an essential consideration in learning a workflow model is the trade-off between *over-fitting* and *under-fitting*. Over-fitting means little or no generalization (e.g. graph (a) in Figure 3.6). Under-fitting means excessive generalizing (e.g. graph (b) in Figure 3.6). Since the primary objective is *Generalization*, overfitting is undesirable. However, under-fitting can become problematic if it is not controlled [8].

3.3.2 Prefix constrained merging

The solution to addressing under-fitting lies in imposing constraints on whether node merging is permissible. Assuming no constraint are imposed on primitive steps to begin with, any steps with multiple occurrences in the process log can be treated as a single node in the process graph. To tackle the under-fitting issue, one way is to constrain which steps are allowed to merge by considering their preceding steps. A prefix rule [88] can be applied to decide whether two steps with identical names should be represented with the same node in the process model. Suppose one H-path contains a segment αX and another H-path contains a segment βX , where α and β each is a sequence of one or more steps and X is a single step. Given a length requirement $i \geq 0$, let α_i be the last i steps in α and β_i be the last i steps in β . An *i-prefix rule* means that the two X steps would be represented by one node in the process model if $\alpha_i = \beta_i$.

For example, consider two H-paths containing the subsequences “BA” and “CA”, respectively. Without any constraint, both occurrences of step A would be treated identically. However, with the *1-prefix rule*, the two would be treated as different nodes. For another pair of H-paths including the subsequences “BDA” and “CDA”, respectively. In this case, the two occurrences of step A are represented as a single node because for both

occurrences of A, the preceding step is D. Therefore, it satisfies the 1-prefix rule.

The length of the matching *prefix* is a variable that inversely controls the degree of generalization. To demonstrate this idea, the graph (c) in Figure 3.6 shows a construction based on the 1-prefix rule. Compared to graph (a) which produced 22 new H-paths, the 1-prefix graph (c) only includes two new H-paths, BCF and ABCDF, highlighted in two different colors.

3.3.3 An application example

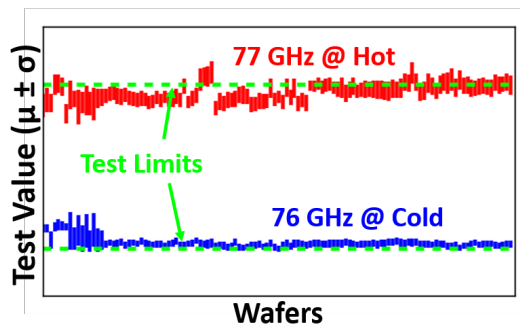


Figure 3.7: Illustration of the issue yield addressed in [8]

The yield issue addressed in [8] is illustrated in Figure 3.7 and is described as follows. The product was an automotive part that operated in the 76-77 GHz band allocated for vehicular radars on an unlicensed basis (see e.g. FCC page [89]). To meet the specification, packaged chips were tested under cold temperatures, operating at 76 GHz, and under hot temperatures, operating at 77 GHz. In both conditions, the voltage required to drive the oscillator was measured. Upper and lower limits were set for the measured voltage during hot and cold testing, respectively. The yield issue arose from an unexpected drop in yield, which was observed in some assembly lots during the cold and hot temperature tests.

Final test data was originally organized by assembly lots. Using each chip's ECID

(Electronic Chip ID), the data was then reorganized into their production lots. In total, there were 175 wafers arranged by their production lots. Figure 3.7 shows wafer-to-wafer variations of the test values at cold and hot temperatures. In the figure, two vertical bars are depicted for each wafer, denoting the cold (blue) and hot (red) results, respectively. Each bar shows the range of measured values obtained from dies on the corresponding wafer. This range spans from $\mu - \sigma$ to $\mu + \sigma$ where μ stands for the mean and σ stands for the standard deviation. Additionally, the upper and lower limits are shown as two horizontal green dashed lines. As seen, hot values exhibit a more frequent drift beyond the limit compared to cold values.

Based on the past work in [9], a process log with 39 H-paths was constructed. Process discovery was applied to learn a workflow model based on an i -prefix rule. Table 3.1 shows the number of H-paths included in the resulting model for each of the i -prefix rule, for $i = 0, \dots, 6$.

i -prefix rule, where $i =$	0	1	2	3	4	5	6
Number of H-paths	98990	1271	160	63	53	42	39

Table 3.1: Number of H-paths included in the learned model for each i -prefix rule [8].

As seen in the table, the number of H-paths can significantly decrease as i increases. In practice, one may desire to start with a large i -prefix rule to produce analytic H-paths closely resembling those already present in the log. Subsequently, as resources allow, one could explore H-paths generalized from smaller i -prefix rules.

It is important to note that in process discovery, the H-paths produced by a larger i -prefix rule are always contained within the H-paths produced by a smaller i -prefix rule, i.e. if P_i is the set of H-paths for i -prefix, then $P_{i+1} \subseteq P_i$. Figure 3.8 shows the process model learned using the 2-prefix rule [8]. Under this model, recall that an H-path is a sequence of multiple primitive steps, each generating a plot in order.

Figure 3.8 highlights two H-paths within the model. The first, highlighted in green,

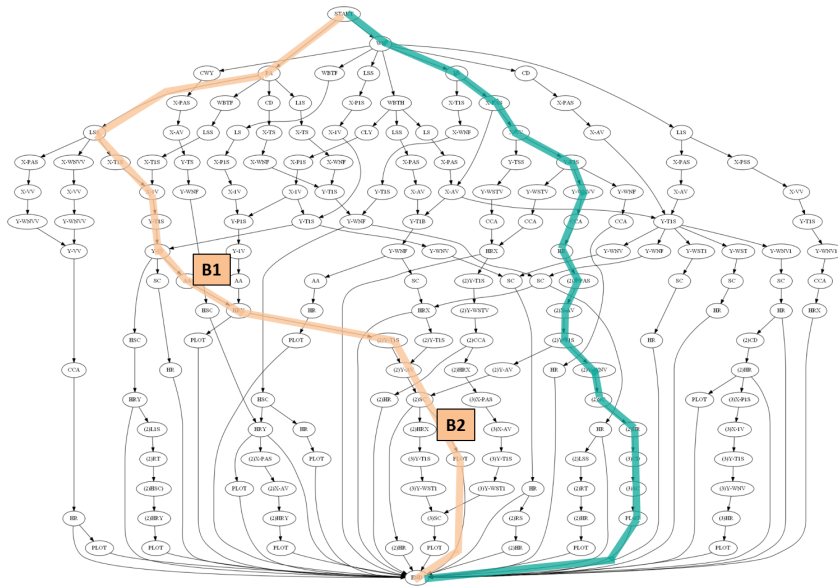


Figure 3.8: The process learned from the work in [9], reported in [8]

is the H-path generating several meaningful plots initially reported in [9]. The second highlighted H-path is a generalized path identified by the process discovery. This H-path was *never* used in the analytic work reported in [9]. However, it is important to note that this H-path is only included when i is at most 2 according to the i -prefix rule. With the 2-prefix model, there are 121 ($= 160 - 39$) new H-paths to explore. The highlighted H-path results in two findings, marked as B1 and B2 in the model. These findings are shown in Figure 3.9, and their meanings are explained below.

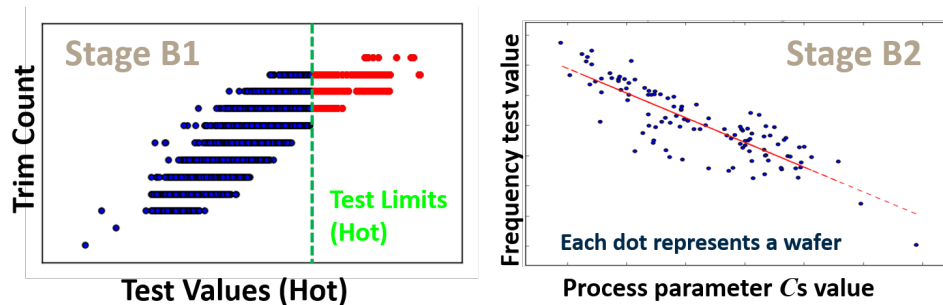


Figure 3.9: Findings to explain the yield issue in Figure 3.7, reported in [8]

Before conducting the cold/hot final test, the frequency of the on-chip oscillator was

measured at room temperature. Then, the oscillator underwent a tuning process known as a *trim* process, which is standard for components such as an oscillator. The trim count was treated as a different test value. The left plot of Figure 3.9 shows the result from stage B1, where test-to-test association was analyzed. In this plot, each dot represents a part. It shows an association between the trim count and the hot test with a high confidence, indicating that parts failing the hot test had a high trim count.

In stage B2, the right plot of Figure 3.9 was obtained, showing a strong correlation between the room temperature frequency test and a process parameter C. We knew that trim count was related to the room temperature frequency test, with higher frequency implying a higher trim count. Consequently, based on both figures, we could conclude a possible resolution to the yield: pushing parameter C to the right corner. This adjustment could effectively reduce the trim count and mitigate the occurrences of parts failing the hot test.

3.4 Learning Domain Knowledge in Functional Verification

In addition to learning a yield optimization workflow model [8], the process discovery approach also found application in functional verification. Specifically, the work in [10] applied an improved approach called *constrained process discovery* in the context of functional test generation. The goal was to produce functional tests that were challenging to obtain using other existing approaches. The approach was useful in the area of security verification (e.g. verifying the security features of an SoC) where it was usually difficult to construct *penetration tests* capable of exposing unspecified design spaces within a security design. In-depth domain knowledge of both the design and the security features

was often required to develop the penetration tests that could successfully expose system vulnerabilities.

This work was therefore motivated by the observation that developing successful penetration tests was challenging in practice, and this challenge could be viewed in two aspects. Firstly, there was often a deficiency in domain knowledge necessary for writing a penetration test. This knowledge gap might manifest in various scenarios, such as a security specialist lacking familiarity with the design or a verification engineer lacking insight into security features. Secondly, there existed a scarcity of personnel possessing the requisite domain knowledge. If only a handful of people were proficient in writing penetration tests, scaling up the practice to obtain a large collection of such tests could be difficult.

The approach to overcome the challenge was to learn from the test examples written by an expert. After the learning, the model was able to produce tests resembling those crafted by the expert. The clear advantage of this approach lay in its ability to effectively “clone” the domain knowledge of an expert, making this knowledge readily accessible for test generation at any time. Consequently, a greater number of tests could be obtained within a specified timeframe, thereby boosting the productivity of security verification efforts.

Without loss of generality, the work in [10] considered each expert’s test as a C program. To learn from a set of C programs, one needed a way to represent the programs, as this formed the basis of the learning process. Similar to the approach in [8], where an analytic path was modeled as a sequence of primitive steps, [10] represented a C program using a set of *primitive transactions*. These primitive transactions can be conceptualized as parameterized scripts capable of generating specific pieces of C code when invoked. In practice, these primitives served as a TPI (Test Programming Interface) for a person to write direct tests.

3.4.1 Relation to grammatical inference

With the primitives defined, a direct test was represented symbolically. Similar to that in [8] for representing an H-path, a test could also be represented as a sequence of *steps*, e.g. [A,B,C,...]. Then, each test could be viewed as a “sentence” example derived from an unknown formal language, as described in the field of formal language theory [90]. In other words, primitives were seen as *words* of the unknown language. Then, methods from *grammatical inference* [91] could be applied to discover an automata model, such as a finite automato, to describe this language based on a given set of examples.

The *learnability* problem in grammatical inference asks whether a model could be learned with a finite number of samples. In this context, two types of samples are defined: in-model samples, which comply with the model (i.e. the language) being learned, and out-model samples, which do not comply with the model. The main result of [92] points out that if only in-model samples are available, the only learnable class is the set of finite-length languages, i.e. there is a bound on the maximum length of a sentence. If both in-model and out-model samples are both available, then all classes up to the Context-Sensitive grammar in the Chomsky Hierarchy can be learned [90].

Since the work [92], the learnability of a finite automaton has been among those that received the most attention [93]. Later, *process discovery* emerged as a separate field, primarily targeting business applications [94]. In this context, a process model is derived from an event log that records instances of business transactions. In process discovery, a common representation for the process model is Petri Net [94], wherein the graph model allows loops and concurrency. Notably, learning such a process model is as hard as learning a finite automaton.

Process discovery and the approach in [10] had fundamentally different objectives. Process discovery was for discovering business intelligence from event logs. Hence, it

was important for the learning model to be interpretable, i.e. simplicity of the model to enable visualization was a key consideration. The work in [10] was for test generation, where it was not necessary to condense all learned information into a single interpretable process model. This difference enabled the development of the novel *constrained process discovery* approach.

The work in [10] considered only in-model samples, i.e. tests that were meaningful in view of the verification. As a result, only finite-length language models were learnable. The work also excluded loops and concurrency in the model and otherwise, the learning would be too difficult to pursue.

3.4.2 Constrained process discovery

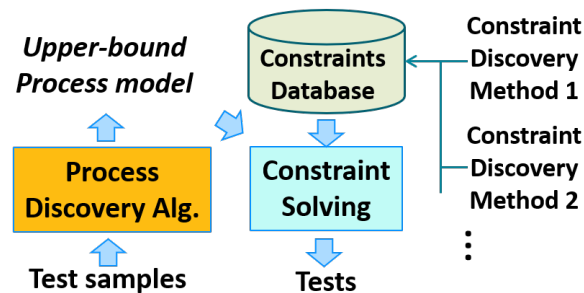


Figure 3.10: Constrained process discovery for functional test generation in [10]

Figure 3.10 illustrates the constrained process discovery approach. The approach splits the learning into two parts: (1) learning an *upper-bound model*, and (2) learning a set of constraints.

The goal of an upper-bound model is to capture a bound on the space of all possible tests such that a desired test is ensured to fall within this space. However, due to its nature as an upper bound, the model can include many undesirable tests. To address this issue, a separate *constraints database* is maintained to impose constraints between and

among primitives. In the methodology outlined, constraints are learned independently of the process learning. Then, for test generation, the upper-bound model and the constraints from the database are combined to formulate a *constraint solving* problem. Each solution to this problem corresponds to a viable test. In this work, a Boolean satisfiability (SAT) solver [95] was used for the constraint solving.

The approach enabled a desirable feature in the context of the verification: The constraints database served as an innate repository for accumulating and disseminating verification knowledge. This knowledge could be added manually, based on previous learning sessions, or based on a separate constraint discovery method. With the incorporation of additional constraints, the test space was constricted, thereby enabling the generation of more targeted and focused tests.

3.5 Three Challenges Motivating Language-Driven Analytics

The process discovery approaches [8][10] for learning domain knowledge into a process model encountered three fundamental challenges:

Completeness The first challenge concerns the completeness of the set of primitive steps. The effectiveness of the approach depends on this completeness. Developing a comprehensive list of primitive steps demands extensive domain experience and deciding its completeness can be an open-ended issue.

Abstraction The second challenge concerns the right abstraction level to define primitive steps. In [8], the primitive steps were defined at a rather detailed level for constructing a plot. In practice, an investigation might involve a sequence of plot types, one after another, to reach a finding. Using low-level primitive steps to

model such an analytic path can be tedious. Applying learning with a log based on these low-level steps might be inefficient. In [10], the primitive steps were defined at a transaction level. This limited the test generation from reaching a test that requires modifications of a transaction.

Recognition The third challenge concerns recognizing the meaningfulness of the instances generated from a process model. In [8], this challenge involved selecting meaningful plots in view of plot-based analytics depicted in Figure 3.3. While the approach in [8] generated a comprehensive search space for exploring the data, it did not address the issues of automatically selecting plots for output. To automate a plot-based analytic workflow, a separate component is needed, which should be capable of determining the meaningfulness of a plot and selecting interesting plots for output. In [10], the meaningfulness of a generated test could be determined through simulation and in this sense, the issue might be less severe than that in the plot-based analytics context. Nevertheless, the meaningfulness of generated tests from a model depends on the constraints stored in the constraint database. As a result, this aspect of challenge is still reflected in the recognition of the usefulness of the stored constraints.

The original proposal of the Intelligent Engineering Assistant (IEA) in [84] tried to address some of these challenges, particularly focusing on the third challenge in view of the plot-based analytics. However, a solution addressing all three challenges was not realized until the work presented in [96], where the idea of *language-driven analytics* was proposed. Language-driven analytics will be discussed in detail in Chapter 5.

3.6 Co-ML Capabilities

As discussed with Figure 3.1 in section 3.1.1, two types of domain knowledge can be considered: one to drive ML and the other to drive Co-ML. The process discovery approaches reviewed earlier in [8] and [10] were primarily geared towards modeling the first type of domain knowledge. Specifically, they operated in the context of test data analytics and functional verification, respectively.

The “recognition” challenge, as highlighted in section 3.5 above, can be considered part of the challenges associated with modeling the second type of the domain knowledge. Specifically, recognizing whether an output is meaningful includes some Co-ML capability of determining the acceptability of an answer. As mentioned earlier, the challenge was not completely addressed until the proposal of language-driven analytics in [96]. Nevertheless, in this period (2014-2017), two methods were developed for Co-ML, each tailored to a specific context. In the context of correlation analysis, a method was provided to determine “no correlation” [9]. In the context of outlier analysis, a method was provided to determine “no outlier” [11]. Below, we will review both methods and their thinking..

3.6.1 Showing “no correlation”

When adjusting a process parameter to improve yield, it is important to show that the adjustment, while solving one yield issue, is unlikely to cause another issue [9]. One type of the evidences is to show that the target E-test parameter to be adjusted has “no correlation” to the failure types that are not currently under concern. The work in [9] developed a *risk evaluation* method to provide this type of evidence.

The essence of risk evaluation is to show that an E-test parameter and a failure type is likely to be statistically independent. A well-known method for assessing statistical independence is the Canonical Correlation Analysis (CCA) [83]. CCA finds the maximum

correlation between two random vectors across all possible linear transformations of them. The more complex version, the kernel CCA (KCCA), applies the so-called “kernel trick” [97] to extend CCA to consider non-linear transforms.

However, KCCA is not very useful in practice. Given a kernel function that is complex enough, KCCA can always find a mapping to over-fit the data such that the resulting correlation is 1.0 [98]. To resolve this issue, the work [9] adopted a different implementation based on the idea in [99]. The idea was to approximate KCCA (A-KCCA) by running kernel Principal Component Analysis (KPCA), followed by regular CCA on a dataset transformed with the first C principal components. With $C = 0$, A-KCCA becomes regular CCA. This setup allowed sweeping the number of principal components, thereby incrementally increasing the model complexity for finding the correlation.

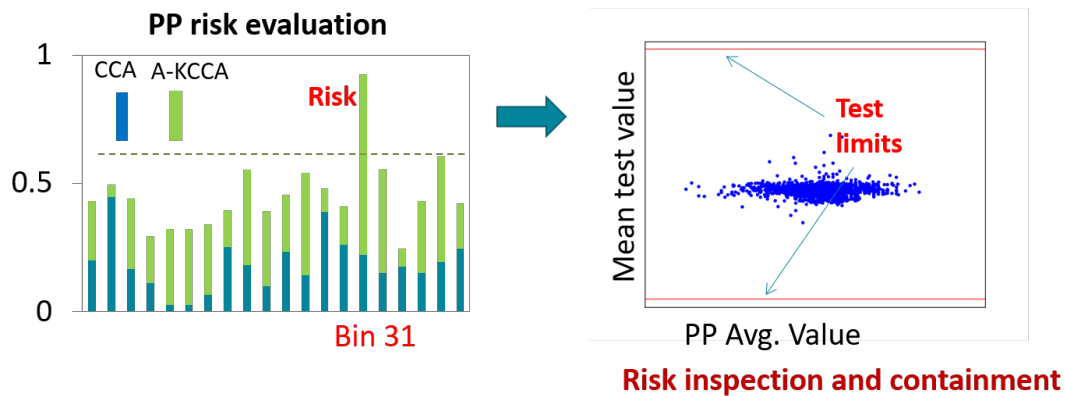


Figure 3.11: A-KCCA risk evaluation on a target process parameter PP [9]

Figure 3.11 shows an example of A-KCCA risk evaluation. In this example, we evaluated the risk of adjusting a process parameter PP by assessing the dependence between the value of PP and the number of failures from different test bins (each bin categorizes a type of failure). Initially, the complexity control C is swept from 0 until the correlation with at least one of the test bins becomes high, say above a threshold of 0.6. In this case, the A-KCCA correlation on Bin 31 jumps above 0.6 when $C = 17$. Then,

based on a higher complexity of $C = 50$, the A-KCCA correlations to different bins are shown in the left plot of Figure 3.11. As seen, even with this high complexity, only Bin 31 stands out. Thus, Bin 31 is considered as the *risky bin*. Others are considered not risky because even with a high complexity assumption, their correlations are still below the threshold.

The next step is therefore to visually inspect the risk on Bin 31. Bin 31 had only 1 test and the risk inspection on this test is shown in the right plot of Figure 3.11. The plot shows a scatter plot where each dot is a wafer. Let M be the average of test values across dies on a wafer. The y-axis indicates the wafer's M value and the x-axis shows the average PP value across sites on a wafer. The plot shows that the adjustment of PP has a minimal impact on the test value distribution, as it remains distant from both the upper and lower test limits. Hence, even though adjusting PP may somehow affect the test result, the risk would be very low.

In essence, the idea behind the A-KCCA method is as follows: Showing “no correlation” means to show that the A-KCCA correlation falls below a predefined threshold (e.g., 0.6), while adhering to a specified complexity bound. The essence lies in devising a method for measuring complexity, such as the number of kernel principle components used in this case. Although it might be impossible to assert that a dataset contains no answer, we could say that a dataset contains no target answer (e.g. exceeding a threshold) that is not overly complicated (e.g. below a complexity level).

3.6.2 Showing “no outlier”

As discussed in Section 1.3.3 earlier and further elaborated in Section 2.5, outlier analysis can be quite subjective and over-fitting a model can be a common occurrence. Consider a scenario where a known failing part p (e.g. a customer return) and a list of

wafer probe tests t_1, \dots, t_n are provided. One asks the question: on which 2-dimensional test space, based on $t_i t_j$, does the failing part p appear as a wafer-based outlier? Addressing this question involves considering $\frac{n(n-1)}{2}$ test spaces.

A simple answer is to choose an outlier method and find the test space where p is the most outlying die on the wafer. However, as discussed before, being the most outlying die does not mean the outlier model is justifiable. Instead, based on the outlying property, the test spaces can be ranked accordingly. Then, one may need to systematically justify them from the top of the list, one by one, until a satisfactory test space is found.

A Co-ML method in this context is the capability to assert that a given test space contains “no outlier”. This assertion should be independent of the subjective threshold used to calculate the outlying property in the justification process. As a result, for a high-ranking test space—signifying a greater degree of outlying behavior—if it is deemed to have “no outlier” by the separate Co-ML method, then there is no need to pursue further justification. Ideally, the Co-ML method can eliminate many test spaces in the search space, thereby reducing both the efforts in search and outlier model justification.

This was one of the motivations for work [11] that proposed a method for asserting that a given test space contained no outlier. The essence of the method lies in a concept called *consistency check*. The check is based on the assumption that outlier decision on one wafer should be *consistent* with outlier decisions on other wafers. For a test value, instead of calculating one outlier score per die based on the wafer it locates in, a vector of outlier scores are calculated based on a set of other wafers. At the high level, we say a given threshold to classify one die as outlier and the other die as inlier is consistent, if the outlier scores for the two dies across a set of wafers do not violate these classification decisions. Given n dies with n outlier score vectors calculated (each vector has m dimensions where m is the number of wafers), the method searches for a *minimum* threshold such that the threshold is consistent across the n vectors. On a test space, if

no die can be deemed as an outlier according to the minimum consistent threshold, then the test space is asserted to have no outlier [11].

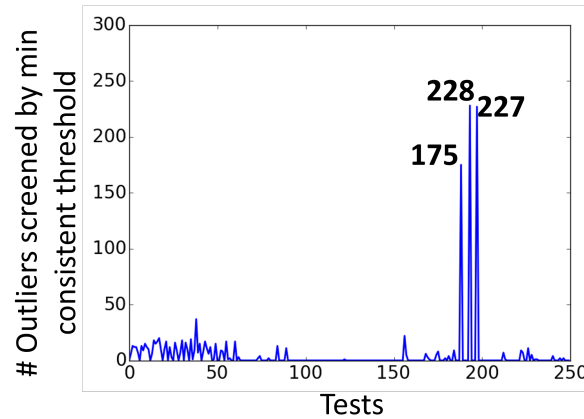


Figure 3.12: Many tests showing no outlier based on the minimum consistent threshold [11]

For example, Figure 3.12 shows the number of outliers identified by the *minimum consistent threshold* for each test based on the Dynamic Part Average Testing (DPAT) method [100]. Notice that many tests in Figure 3.12 shows no outlier, i.e. no die screened by using the minimum consistent threshold. Most of them have a small number of outliers screened out by the minimum consistent threshold. In customer return analysis, if the returned part is not identified as one of the consistent outliers, then there is no need to consider the corresponding test spaces. Hence, the method can effectively eliminate many tests from consideration in a customer return analysis case.

3.7 Monomial Learning

Discussion in Section 2.2.6 with Figure 2.6 illustrates a special case of *monomial learning* where there is only one positive sample. Monomial learning can be seen as learning a simple form of Boolean function. Learning a Boolean function in general is discussed in *computational learning theory* (CLT) [56].

Given n Boolean features f_1, \dots, f_n , a monomial is a conjunction of some features. The *Boolean hypothesis space* contains 2^n hypotheses, including the empty hypothesis with no feature. To the extreme, given n Boolean features and without any constraint, the Boolean hypothesis space contains all 2^{2^n} Boolean functions. In CLT, a hypothesis space is usually specified with a representation. For example, a k -term DNF (disjunctive normal form) restricts the functions to those representable with the sum of k monomials (product terms). Restricting the hypothesis space does not necessarily avoid the computational hardness, though. For example, learning DNF formulas is as hard as solving a random K-SAT problem [101]. Even for $k = 3$, the DNF learning problem remains hard [56], in the sense that it is hard to find a polynomial-time algorithm unless $\mathbf{RP} = \mathbf{NP}$ (note: $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$ [102]).

Due to the computational difficulties discussed in CLT, learning with a dataset based on Boolean features in practice (e.g. Figure 2.6) is often restricted to monomial learning. This is because from the perspective of CLT, monomials are efficiently learnable [56]. However, this does not imply that monomial learning is easy from a practical point of view.

To explain this point in a simple way, suppose the training samples are generated based on uniformly random sampling. Suppose the true answer is the monomial with j features for a large j . It is possible that no positive sample is generated in the training samples, i.e. the output labels across all samples are zero — they are all negative samples. Learning from such a dataset would result in a model where $f() = 0$, simply the constant 0. From the CLT perspective, $f()$ is a good answer because the error probability for any randomly drawn sample remains low. However, in practice a constant 0 most often means the learning has failed.

While a model that gives a small error probability is considered a good model from the viewpoint of CLT, merely considering the error probability is not sufficient in practice.

Often, one desires to learn the monomial that represents the underlying cause for the observed behavior. And as discussed before, this learning can be based on only one or few positive samples, making the learning problem more challenging. In fact, if there is only one positive sample and many negative samples, finding the shortest monomial is NP-hard [103].

3.8 Uniqueness in View of Occam's Learning

The work in [12] proposed a tool called VeSC-CoL (Version Space Cardinality based Concept Learning) [104] to address the practical challenges with monomial learning. The core idea can be stated as the following: Find the simplest monomial that fits the data (Occam's learning) while the monomial is also the unique answer at that complexity level (the uniqueness requirement). To justify the Occam's Razor principle in machine learning, the author in [105] already suggested that the simplest model was better because the model was more likely to be unique. In other words, it would be computationally harder to find another answer with the same complexity which could also fit the data. Hence, instead of implicitly hinted by adopting an Occam's learning approach, the idea proposed in [12] is to make the model uniqueness an explicit requirement for the learning. The VeSC-CoL tool is built upon Boolean Satisfiability (SAT) solving, achieving this by encoding a given monomial learning problem into a SAT problem [12][104].

3.8.1 A Co-ML view of monomial learning

Based on the VeSC-CoL tool, the work in [12] suggested a solution to solve the special monomial learning problem in practice, particularly when only one or few positive samples are available. The solution is illustrated in Figure 3.13.

Based on the one or few positive samples, a hypothesis space is formed. For monomial

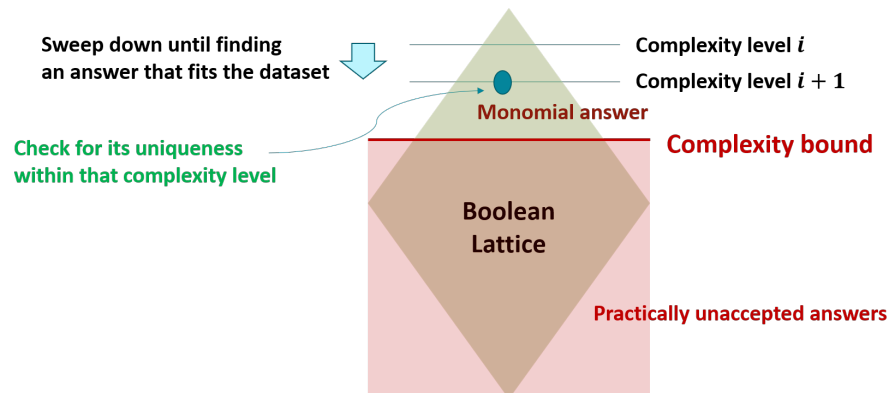


Figure 3.13: Solving the monomial learning problem in practice, as suggested in [12]

learning, this hypothesis space essentially is a Boolean lattice contains all 2^n monomial based on n Boolean features (see e.g. example in Figure 2.6). This hypothesis space can be divided into n levels of complexity. The i th level contains monomials with length i , i.e. with i features. First, a complexity bound is imposed to eliminate all hypotheses whose complexity is higher than the bound and treat them as practically unacceptable answers. For example, for any monomial whose complexity is higher than level 3, i.e. $i \geq 3$, is treated as unacceptable. This is usually a reasonable assumption as a complex answer is more difficult for people to accept (see discussion in Section 1.4.5).

It is interesting to note that, in view of the discussion with Figure 2.22 in Section 2.7 before, while the decision chain complicates the search for an acceptable answer in one way, it can also simplify the search space in another way. The complexity bound, arising from practical considerations in the context of monomial learning, helps eliminate a significant portion of the search space in the Boolean lattice, thus exemplifying this simplification.

For hypotheses whose complexity falls within the bound, we search for an answer with the lowest complexity. This process can be viewed as a sweep from complexity 1 downward, level by level, until we reach a hypothesis that fits the training dataset. In

Figure 3.13, this hypothesis is found at level $i+1$. Note that in the implementation of the VeSC-CoL tool, this search was achieved by SAT solving. Finding the lowest-complexity monomial answer that fits the data follows the Occam’s learning principle.

The VeSC-CoL tool has two ways to reach the conclusion that a given dataset is insufficient for solving the monomial problem, essentially achieving a Co-ML capability. First, if a monomial answer is found within the complexity level, the tool proceeds to check if the answer is unique. If the uniqueness is confirmed, the answer is accepted. If not, the answer is rejected, and the monomial learning problem is considered *unsolvable* with the dataset. Second, if no fitting hypothesis within the complexity bound is found in the first place, the problem is also considered unsolvable.

3.9 Local No-Free-Lunch (L-NFL)

No-free-lunch situations, as pointed out in Section 1.4.9 before, can happen when analyzing or modeling data from design and test processes. During the first decade of works reviewed in Chapter 2, no-free-lunch was not yet a main subject of concern. However, in the following decade it became apparent that accounting for no-free-lunch was imperative when implementing a practical solution.

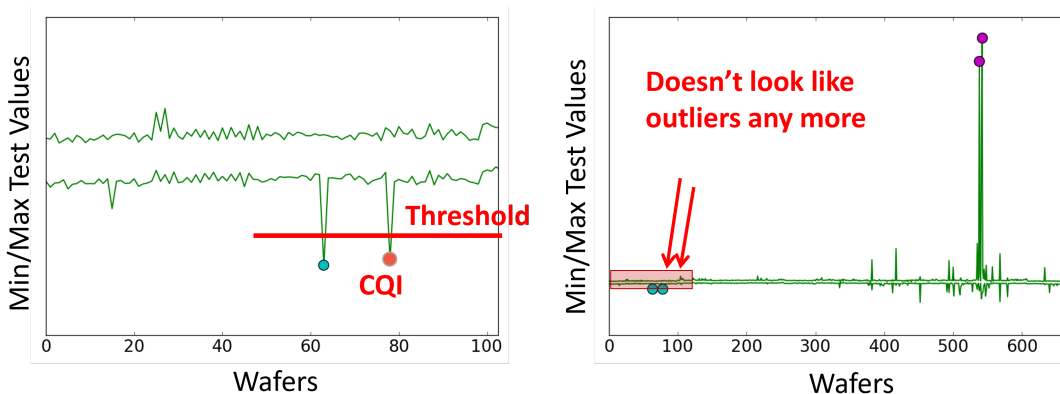


Figure 3.14: A no-free-lunch situation in the context of outlier screening

In the context of outlier screening, the concern for no-free-lunch is apparent, as illustrated in the example in Figure 3.14. The left plot shows the minimum and maximum test values for a particular test, across the first 100 wafers ordered chronologically. Note that the minimum and maximum test values on each wafer were calculated based on all the dies passing the test. There was a CQI (customer return) and based on the plot, the CQI could be seen as an outlier in the sense that its test value did not follow the min/max trend. A threshold could be set to screen out future dies behaving similarly to this CQI on the test.

However, the right plot shows what happens on the subsequent 500 wafers after the first 100 wafers. As depicted in this plot, outliers seen within the first 100 wafers no longer appear to be outliers when considering the perspective of the subsequent 500 wafers. Again, note that the minimum and maximum test values in the right plot were also calculated based on all the dies passing the test.

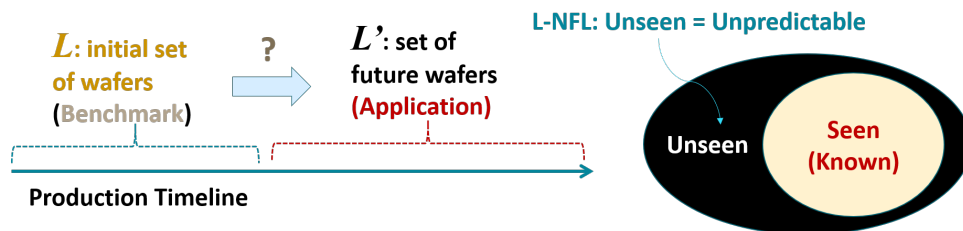


Figure 3.15: Local No-Free-Lunch (L-NFL) when learning from production data

Figure 3.15 illustrates the situation in general which can be called *local no-free-lunch* (L-NFL) [31]. In a process, in this case a chip production process, a model learned from the initial set of data (L) might or might not be *applicable* to the future data (L'). A L-NFL situation means that the behavior of the unseen data is not predictable based on learning from the seen data, i.e. “L-NFL: Unseen = Unpredictable”.

Even though L-NFL was not a focal point until the second decade of the journey to IEA, it had already been an issue encountered in the context of Boolean learning in an

early work [55]. This work tried to learn an abstract design model based on the input-output simulation samples from a given design. Table 3.2 shows a simple example to illustrate the point.

Table 3.2: A simple Boolean learning L-NFL example

x_1	x_2	x_3	y
1	1	1	1
1	0	1	1
0	1	1	1
0	0	1	1
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	?

The learning problem is based on a Boolean function with three inputs, x_1, x_2, x_3 and one output y . There are 8 possible input values where the data provides the output values on 7 of them already. The task is to learn a model to predict the unknown output value denoted by a question mark in the last row, corresponding to the input “110”.

If we assume that a simpler model is better (i.e. Occam’s learning), then we would choose the model $f_a(x) = x_3$ which will output “ $y=0$ ” as the answer for the input “110”. However, the true answer can be $f_b(x) = x_3 + x_1x_2$ which will output “ $y=1$ ” as the answer for the input “110”. Both models have zero error rate based on the samples in Table 3.2. Thus, without an acceptable assumption to choose between the two models, we simply would not know which one should be used. From the L-NFL perspective, both answers are possible and hence, the data is simply insufficient for answering the question.

As an early attempt to mitigate the L-NFL issue, the work in [55] used Ordered Binary Decision Diagram (OBDD) [106] as the representation for a Boolean function and used the OBDD size as a *regularization* objective in the learning. In other words, the OBDD size was kept as small as possible when the OBDD was used to fit the training samples.

The OBDD-based regularization helped learn a good model, yet this model was never 100% accurate [55]. At that time, the L-NFL issue had never been explicitly addressed.

3.9.1 Model applicability in view of L-NFL

The work [31] in 2017 explicitly addressed the L-NFL issue in the context of outlier analysis for defective part screening. The main idea, illustrated in Figure ??, was to develop a method for measuring the *applicability* of an outlier model. Given a model M and a piece of data D , applicability is to ask whether or not M is suitable to be applied on D .

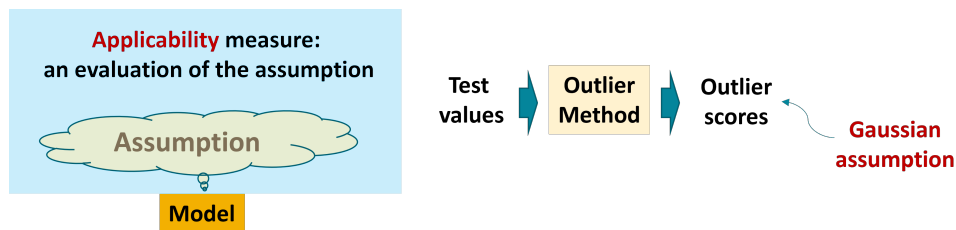


Figure 3.16: Measuring applicability of an outlier model

A key observation made in [31] was that a learning model was always associated with an implicit assumption. To implement an applicability measure, one needed to find a way to evaluate the data against this assumption. In the context of outlier screening, the work in [31] found that methods commonly practiced in the industry implicitly followed a Gaussian assumption. Figure 3.16 summarizes the key observations.

At the high level, the finding is that an outlier method *transforms* each test values into an outlier score following the assumption that the outlier scores should form a Gaussian distribution. It is then based on this assumption, a threshold is set to screen out outliers. The Gaussian assumption is the key for a practitioner to more intuitively capture the meaning of a threshold, e.g. a 6σ threshold. Based on this observation, the work in [31] then developed an *applicability measure* statistically, based on which a methodology was

implemented to accomplish two tasks: (1) For a given set of outlier models for a given wafer, decide if there is any model applicable to the wafer, and (2) If there is, then decide which model is the best.

It is interesting to note that the idea of model applicability was also considered in another work [81] in the context of circuit behavior modeling (reviewed in Section 2.6.6 before). The mirco-modeling approach suggested in [81] aimed to capture a sub-input-space wherein predicting the output is possible, thereby avoiding the challenge of making predictions outside the space.

3.9.2 A remark on the concept of over-fitting in DSML

Suppose in ML, the true answer to be learned is a function $f(x)$ and the learning yields a model $h(x)$ to approximate this function. In ML, the concern is usually how close $h(x)$ is to $f(x)$. This closeness is often measured in a probabilistic sense. For example, for a sample randomly drawn from the input space (or produced by a data generator G following a certain probabilistic distribution), the expected error $Err(h, f) = EXP(f(x) \neq h(x))$ can be an inverse measure for the closeness.

Consequently, the optimization in ML is to find a model $h(x)$ as close as to the function $f(x)$. Since $f(x)$ is unknown, the optimization objective cannot be formulated based on $f(x)$ directly. Instead, it must be based on a set of samples in the dataset. It is important to note that in ML, achieving strict equality between h and f , i.e. $h = f$, is not necessary for success. Rather, in a sense, the extent of success is gauged by the probabilistic performance of h in approximating the behavior of f .

Because learning is carried out on the samples in a given dataset D , the main concern in ML is *over-fitting*. Over-fitting can be illustrated as Figure 3.17 in view of *model complexity* [46]. In ML practice, over-fitting is often explained with two error rates in a

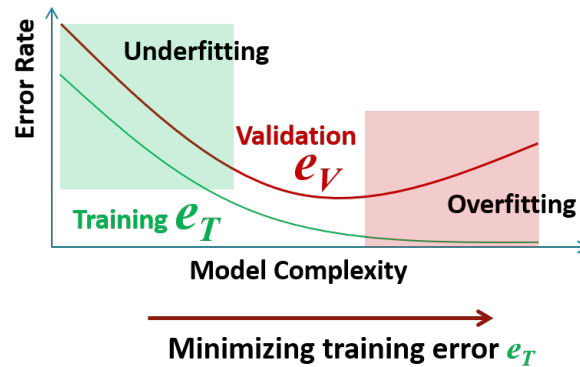


Figure 3.17: Over-fitting illustrated in terms of model complexity

cross-validation setting. Let D and D' denote a training dataset and a validation dataset, respectively. For a given hypothesis h , let $EmErr(h, dataset)$ be an error function to report an *empirical error rate* when applying h onto a dataset. Let the training error rate be $e_T = EmErr(h, D)$ and validation error rate be $e_V = EmErr(h, D')$. In ML, a learning algorithm operates solely on D . Hence, while the algorithm can try to improve on e_T , it remains unaware of the potential outcome for e_V .

To fit the dataset D better, a more complex model is picked. In this case, the learning process can be viewed as moving to the right along the x-axis in Figure 3.17. In optimization, this corresponds to minimize the *fitting error*. As shown in Figure 3.17, this minimization alone is problematic: As the model complexity continues to increase, while e_T continues to approach zero, the difference between e_T and e_V will continue to increase. In other words, e_T is no longer a good “tracker” for e_V . The situation where e_T deviates from e_V , is called *over-fitting* from the perspective of Figure 3.17.

In view of the DSML discussed with Figure 2.22 before, the scenario shown in Figure 3.17 is not likely to instantiate in practice. This is mainly due to the existence of data wall discussed in Section 2.7.1. Because of the data wall, cross-validation is not practically feasible in a DSML context. Consequently, the traditional ML perspective on over-fitting depicted in Figure 3.17 is no longer applicable for signifying over-fitting in

DSML practices. A more applicable perspective in DSML is illustrated in Figure 3.18.

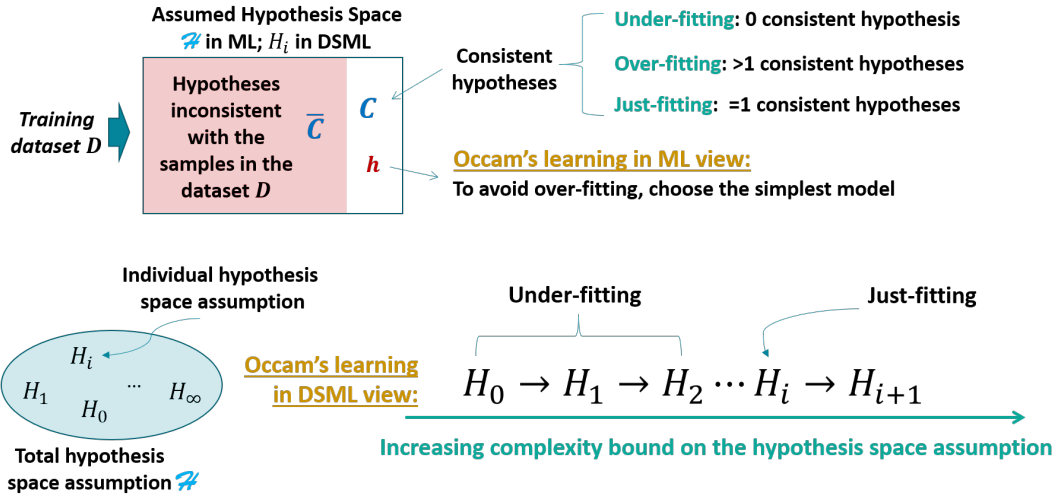


Figure 3.18: Illustration of Occam’s learning in DSML

In this view, ML starts with a *total hypothesis space assumption* \mathcal{H} . Samples in a given dataset D are used to *filter out* hypotheses in \mathcal{H} which are inconsistent with the dataset. What remains is a set C of *consistent hypotheses*, i.e. they are competing hypotheses that explain known observations equally well. To avoid over-fitting, with traditional Occam’s learning, it tries to pick the “simplest” model from C .

In Figure 3.18, a given hypothesis space assumption is divided into two sets: the set C of consistent hypotheses and the set \bar{C} of inconsistent hypotheses. If $|C| = 0$, this is called under-fitting, i.e. no hypothesis can fit the data. If $|C| > 1$, this is called over-fitting, i.e. many hypotheses can fit the data. In ML, the starting \mathcal{H} is usually complex enough to ensure the existence of at least one hypothesis for fitting the data. As a result, it is almost certain to have the situation where $|C| > 1$. To resolve an over-fitting situation, Occam’s learning is therefore applied to pick the simplest one as the best hypothesis.

3.9.3 The meaning of Occam’s learning in DSML

In DSML, both under-fitting and over-fitting are not desirable. The desirable situation is $|C| = 1$ and is called *just fitting* [12]. This is similar to the uniqueness requirement discussed above in Section 3.8.

In view of Figure 3.18, Occam’s learning has a different meaning in DSML than that in ML. As mentioned above, Occam’s learning in ML can be seen as finding the simplest hypothesis in C . In DSML, both under-fitting and over-fitting are considered not successful and the learning is to seek a just-fit model. As a result, Occam’s learning in DSML means to search for the simplest *hypothesis space assumption* to achieve just-fitting, i.e. there exists one just-fit hypothesis in an assumed hypothesis space.

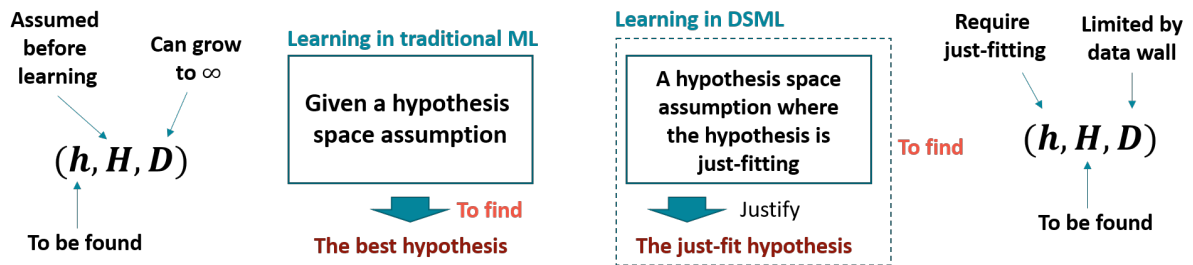


Figure 3.19: “Learning” in traditional ML vs. “learning” in DSML

Figure 3.19 shows the difference between DSML and ML from the perspective of learning. We can say that “learning” in traditional ML is to find the best hypothesis in a given hypothesis space assumption. “Learning” in DSML focuses on finding and justifying a hypothesis simultaneously. The justification is done by ensuring the just-fitting property. Because the just-fitting property of a hypothesis can only be evaluated with respect to a given hypothesis space assumption, the learning cannot be just about searching for a hypothesis. As a result, the learning has to include a search for the hypothesis space assumption, constrained by the just-fit requirement. The hypothesis space assumption is

then used to justify the found hypothesis by showing just-fitting. With Figure 3.19, we see that the so-called “learning” in ML takes place across hypotheses, i.e. at hypothesis level. In contrast, the “learning” in DSML takes place at the level of hypothesis space assumption.

Specifically, consider a triple (h, H, D) where D is a given dataset, H is an assumed hypothesis space, and $h \in H$ is a hypothesis fitting D . In ML, H is assumed and fixed before the learning (e.g. H is the total hypothesis space assumption \mathcal{H} mentioned above). D can grow to be extremely large. The goal is to find the best h fitting D according to Occam’s learning. In DSML, D is limited by a data wall. Just-fitting for a hypothesis is a requirement. The goal is to find the hypothesis space assumption H .

Refer to Figure 3.18 again for how “learning” in DSML might take place. Justification required in DSML is unlikely to be realized by starting with a complex hypothesis space assumption like \mathcal{H} . Due to the existence of data wall, it is unlikely to filter out all inconsistent hypotheses in \mathcal{H} and reach just one consistent hypothesis. Hence, DSML has to be accomplished at the level of individual hypothesis space assumptions.

Let us say that the total hypothesis space assumption \mathcal{H} consists of individual hypothesis space assumptions $H_0, H_1, \dots, H_\infty$ that can be used to justify a hypothesis. Learning in DSML needs to be justifiable with respect to some H_i . As stated above, ML starts with \mathcal{H} and uses the data to find the simplest hypothesis h in the entire \mathcal{H} . In DSML, to achieve justification, this h must be a just-fit hypothesis with respect to some individual hypothesis space assumption H_i . If h was found by ML based on \mathcal{H} , the requirement for justification would not be ensured. Even though h can be evaluated with each hypothesis space assumption H_i in \mathcal{H} individually, the h might not be a just-fit hypothesis with respect to any H_i , i.e. h is still not justifiable.

To satisfy the justification requirement, in DSML we therefore work on one hypothesis space assumption H_i at a time. Without loss of generality, let us assume that

H_0, H_1, H_2, \dots are ordered with increasing complexity. Here, the complexity of a hypothesis space assumption might be seen as the highest complexity allowed for a hypothesis in the space. For example, if we see each H_i as a set, then one way for the ordering might be that $H_0 \subset H_1 \subset \dots \subset H_i \subset H_{i+1} \subset \dots$. With these hypothesis space assumptions ordered in a certain way, Occam's learning in DSML can be explained as the following.

Occam's learning in DSML is applied to choose the simplest hypothesis space assumption H_i , rather than to choose the simplest hypothesis. We start with H_0 and evaluate one hypothesis space assumption at a time. Ideally, Occam's learning in DSML proceeds by sequentially evaluating these hypothesis assumptions, beginning with the simplest one and progressively increasing in complexity. The objective is to find the simplest hypothesis space assumption that yields a just-fit model. For example, for all hypothesis space assumptions up to H_{i-1} , we find that they are all under-fitting, i.e. $|C| = 0$. Then, the search continues until we find H_i that contains a just-fit hypothesis. The reason to find the simplest H_i where a just-fit hypothesis exists is that the found hypothesis is most acceptable with this simplest assumption.

In essence, Occam's learning in DSML is an iterative search process, i.e. a Choose-and-Bound search. One chooses a hypothesis space assumption and evaluates if it contains a just-fit hypothesis. If it is not just-fitting, one moves on to choose another hypothesis space assumption. It should be noted that in practice, within the same complexity level there might be many hypothesis space assumptions to search on. Across different complexity levels, the Occam's Razor principle is used to guide the search to ensure that we search on a simpler hypothesis space assumption before moving onto a complex one.

Occam's learning in DSML has a practical meaning as well. A just-fitting model with a hypothesis space assumption as simple as possible has a better chance to be accepted in practice. This goes back to our earlier discussion in Section 1.4.5 that a simpler answer is

more likely to be understood, justified, and consequently accepted by people. Notice that in DSML the simplicity of an answer is not measured on the answer itself, but measured on the simplicity of the hypothesis space assumption satisfying the just-fitting property.

In the context of monomial learning, the learning methodology proposed in [12] is to achieve Occam's learning under the DSML view. When practicing DSML in a broader learning context, as depicted in the overarching view of Figure 2.22, Occam's learning is more often guided by domain knowledge and achieved by a person. In other words, the person often chooses a simpler assumption to try before moving onto a more complex one. Thus, the Occam's learning in DSML can be mostly operated by a person. Consequently, to automate Occam's learning in DSML we have to provide automation at the AI Assistant level. Automation at the ML tool level simply would not work.

Another important reason behind the just-fit requirement in DSML can be explained as follows. Even though Occam's learning guides a ML algorithm to find the simplest model, this remains a difficult optimization problem. In ML practice, the limited *effective capacity* of a learning algorithm does not actually find the best model, but merely one that significantly reduces the training error [107]. Consequently, while ML promises to find answers that are *probably* correct about *most* member of the set they concern, DSML often requires a *deterministic* answer regarding a *particular* behavior of interest. For example, when finding a rule to explain a design-silicon mismatch, a rule that is 90% correct (evaluated based on limited data) and covers obvious patterns may not be deemed acceptable for diagnosing the hidden issue, especially as the underlying assumption of the hypothesis within the black-box ML model is incomprehensible to humans. We can say that such a DSML problem is essentially a decision problem, i.e. decides whether there is a rule that can explain the issue. It is because of this decision nature that achieving just-fitting becomes an important requirement in DSML. This thinking will be further explained in Section 3.12.

3.10 Autonomous System View (2018 – 2021)

To achieve automation at the AI Assistant level, the idea of Intelligent Engineering Assistant (IEA) was first proposed in 2018 [84]. The original IEA aimed at building an autonomous system, as depicted in Figure 3.20.

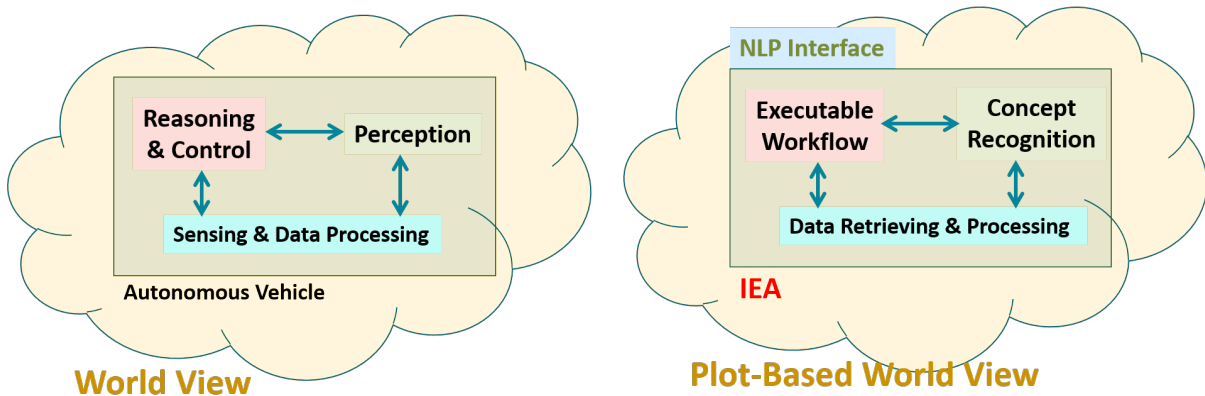


Figure 3.20: The original IEA proposed in 2018 aimed for an autonomous system

The autonomous system view of IEA was inspired by the autonomous system view of an intelligent vehicle [108]. The intelligence of an autonomous vehicle is essentially achieved with three components. A “Sensing & Data Processing” component that collects data from the environment. This component can operate a list of sensors such as short-range radar, long-range radar, LiDAR, ultrasonic, vision, stereo vision, etc. Once the data is collected, the system needs to process it to extract meaningful information. This is accomplished by the “Perception” component. The component interprets the data, for example to perform object recognition and lane detection, etc. The result of an interpretation can be establishing a *concept*, e.g. the vehicle is crossing the right lane boundary. The third component is the “Reasoning and Control” component that utilizes the information provided by the perception component to make decisions and control the vehicle. Functions of this component include calculating the free space and moving the

vehicle, planning its path, and controlling the speed/brake/wheel.

Essentially, the “Sensing & Data Processing” component collects data from the environment and processes them. The “Perception” component recognizes what the data mean. For example, with a deep learning neural network [109], a picture taken by a mounted camera is interpreted for its meaning. The “Reasoning and Control” component decides what to do next. Another important piece for the autonomous vehicle is the “World View” that defines a world for the vehicle to operate in [108].

The analogy to IEA is illustrated In Figure 3.20. IEA also comprises three components: a “Data Retrieving & Processing” component, a “Concept Recognition” component, and an “Executable Workflow” component, respectively. An important difference is that the “World View” is seen as a “Plot-Based World View” to realize the plot-based analytics as discussed in Section 3.3 before. Thus, while an autonomous vehicle is for driving in a world with roads and terrains, IEA was original thought of as an autonomous system that navigated in a world of analytic plots.

3.10.1 The inclusion of an NLP interface and its necessity

The original IEA 2018 included an Natural Language Processing (NLP) interface with voice recognition capability. Because IEA 2018 operated as an autonomous system, it was supposed to perform analytics automatically by itself. On given data, interesting analytic findings were discovered and stored in the system. A user only needed to query for the findings. The user asked the IEA what it had found but could not tell the IEA how to perform the analytics. Therefore, the NLP interface was only for facilitating *information retrieval*.

IEA 2018 was designed as a PowerPoint (PPT) slide generator [84]. A user is supposed to interact with the IEA and pick interesting plots to be included in a summary PPT

presentation. The IEA 2018 included the functionality for PPT generation [110].

If the NLP interface was used only for information retrieval, this raised the question whether or not the NLP component was necessary. For example, a user might prefer to use a GUI with mouse clicks to retrieve interesting plots. The necessity of the NLP interface was not justified until the later time when the approach of language-driven analytics was realized in IEA 2022 [96][111].

3.10.2 The executable workflow component in IEA 2018

The “Executable Workflow” component in IEA 2018 followed the same idea of modeling the domain knowledge as a workflow process [8][10] (see discussion in Sections 3.2 to 3.4 above). The original intent was to model “all domain knowledge” as a comprehensive analytic workflow that was executable.

Steps in the workflow involved *concept recognition* [84]. For example, a step might be an if-else statement such as “if a correlation exists, do something”. The word “correlation” here means a *concept*. This concept is represented in terms of a plot. To decide if there is a correlation or not, we need a *plot recognizer* to recognize those plots satisfying the particular correlation concept. This motivated the plot-based concept recognition work done in [85]. Concept recognition will be discussed in detail in Chapter 4 in the context of *wafermap analytics*.

Setting aside concept recognition for the moment, as discussed before, the pressing issue with a workflow model lies in its completeness. It is difficult, if not impossible, to ensure that a workflow model is complete, i.e. sufficient for a given analytic job content. In fact, in the next few years after the publication of IEA 2018, our lab tried to collaborate with a fabless company closely to implement a practical IEA, based on their test data analytic application context. What we learned was that it would be

very difficult to develop a complete analytic workflow sufficient enough to address all practical requirements. Consequently, we gradually abandoned the notion of pursuing an autonomous system view.

3.11 The Latest Three Views (2022, 2023, 2024)

Figure 2.1 at the beginning of Chapter 2 provides an overview of the journey to IEA. Chapter 2 and Chapter 3 so far cover the first three views in the journey: the data-driven view (2004-2013), the knowledge-driven view (2014-2017), and the autonomous system view (2018-2021). The latest three views in 2022, 2023, and 2024, will be discussed in detail in three separate chapters, respectively:

Problem-Solution Dual View (2022) This view will be discussed in Chapter 5. Realizing the importance of this view in order to move forward in the journey of IEA, perhaps is the most intriguing and critical discovery in the entire IEA journey. Chapter 5 will provide a detailed explanation of this view.

Knowledge Graph View (2023) This view will be discussed in Chapter 7. Adopting this view can be seen as a direct consequence following the Problem-Solution Dual view. The Knowledge Graph view also enabled our successful implementation of the first complete IEA called IEA-Plot, an end-to-end AI Assistant solution in the context of wafermap analytics. The implementation of IEA-Plot will be discussed in Chapter 6.

Generative AI View (2024) This view will be discussed in Chapter 9. In the development of IEA-Plot, the knowledge graph was manually constructed. One of the commonly-asked questions was: why we had to manually develop the knowledge graph rather than training an LLM to model those knowledge? In this chapter,

we will explain why taking the Generative AI view has to happen after taking the Knowledge Graph view. In other words, manual construction of a knowledge graph, in our journey, was seen as a must-taken step that enables the realization of generative AI capabilities eventually. In Chapter 9, we will elaborate on this observation.

Following the initial IEA in 2018 and preceding the adoption of the Problem-Solution Dual view in 2022, our works were primarily directed towards realizing the “Concept Recognition” component. In this period, we narrowed our focus on the application context of wafermap analytics. With a narrower application focus, we intended to develop a more comprehensive “Concept Recognition” component that could be deployed into an industrial production environment. In the next Chapter, we will go over the innovations during this period.

It is interesting to observe that it took us 10 years to progress beyond the first stage (the data-driven view), 4 years each to progress beyond the second and the third stages (knowledge-driven and autonomous system views), and only one year each to proceed with the last three stages. In a way, the views in the first three stages were “incomplete” views for us to learn the experience. It is also important to note that the last three views can be seen as one integrated view, despite being developed over three consecutive years. The culmination of our learning was encapsulated within these last three views, which have already been integrated into our current LLM-assisted AI Assistant, the IEA.

3.12 DSML in View of Computational Complexity

As mentioned in Section 1.4.4 and further elaborated in Section 2.8.1, we use the term *machine learning* to broadly refer to learning from data through optimization. In other words, a learning problem is seen as an optimization problem. The above discussion

regarding Occam’s learning in DSML hints that a DSML problem can also be seen as solving an optimization problem. However, the way we see DSML optimization is different from the way we see ML optimization.

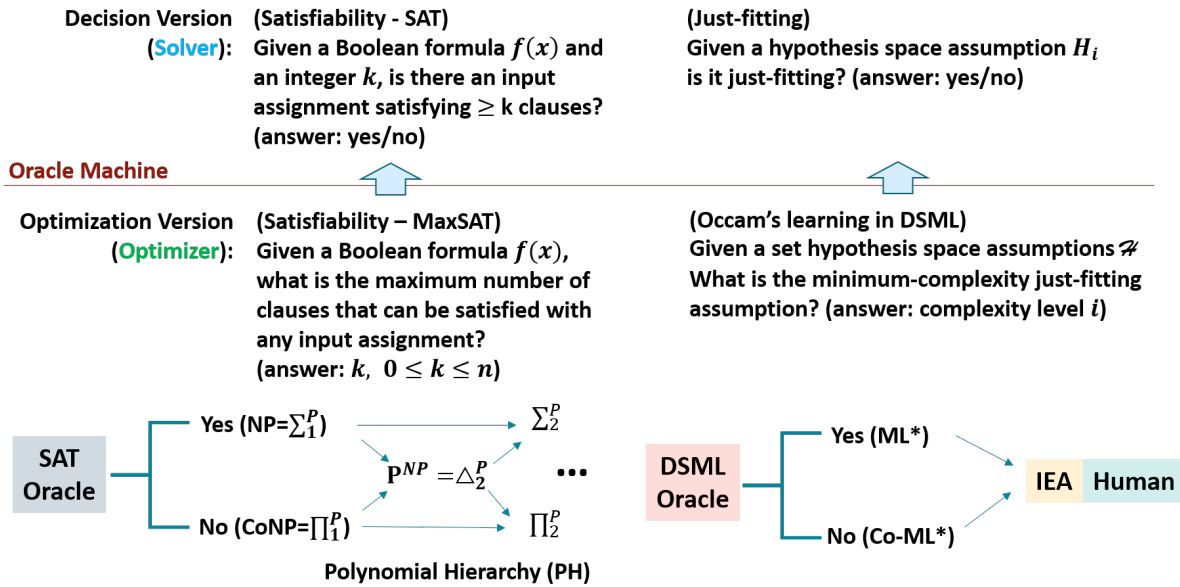


Figure 3.21: An optimization view in DSML

3.12.1 Decision problem vs. optimization problem

Figure 3.21 draws an analogy between computational complexity classes to the problems solved in DSML. On the left, we consider two versions of the Boolean Satisfiability (SAT) problem. In the *decision version* (based on a given threshold k), the SAT problem asks if $\geq k$ clauses can be satisfied for the given formula $f(x)$. The answer is a binary yes or no. In the *optimization version*, the problem asks for the k such that at most k clauses can be satisfied in $f(x)$. This is commonly-known as the *MaxSAT* problem [112].

It is well known that SAT is an NP-Complete problem. It is also known that the threshold version of SAT shown in Figure 3.21 is NP-Complete. However, it is unclear

whether the optimization version of SAT is in NP. In fact, the work in [112] defined the class FP^{SAT} and showed that MaxSAT is $FP^{SAT}[O(\log(n))]$ -complete. The class FP^{SAT} is widely known as the P^{NP} class (also denoted as Δ_2^P) [113] that includes (formal) language decidable in polynomial time (the “P”) with access to an NP oracle, e.g. a *SAT solver*. Indeed, if there exists a SAT solver to solve the decision version of the SAT problem in Figure 3.21, then the optimization version can be solved by making $O(\log(n))$ calls to the SAT solver, i.e. by performing a binary search to find k .

The intuition behind the definition of the P^{NP} class is commonly known as *oracle machine* [114], which is illustrated in the bottom of Figure 3.21. Suppose we have an oracle that has the ability to solve the SAT decision problem. Given $f(x)$ the oracle solves the problem on f and provides a *certificate* c (or a *proof*) to the verifier to verify the correctness of the answer. In the decision version, c can simply be an input assignment that satisfies $\geq k$ clauses. The verifier can check on this fact by applying the input assignment to the formula f and verify the number of satisfied clauses is $\geq k$. The verifier can do this in polynomial time and hence, the threshold version is in NP. In other words, the oracle only needs to show one input assignment as the certificate to prove the correctness of an answer. This is a classical definition of the NP class, i.e. \exists one input.

The optimization version is more difficult because to prove that k is the maximum number, the oracle needs to provide a certificate to show that there is no input assignment that can satisfy $> k$ clauses, i.e. \forall inputs, it is impossible to satisfy $> k$ clauses. The inclusion of the \forall quantifier introduces the CoNP aspect into the problem. In other words, the optimization version is at least as hard as both NP and CoNP. The inclusion of both quantifiers, \exists and \forall , make a problem go beyond NP and CoNP. Note that in Figure 3.21, the SAT oracle should be able to answer yes or no, i.e. be able to answer both an \exists (NP) and a \forall (Co-NP) question.

In general, the *polynomial hierarchy* (PH) $(\Sigma_1^P, \Pi_1^P, \Sigma_2^P, \Pi_2^P, \dots, \Sigma_i^P, \Pi_i^P, \dots)$ captures

the complexity of a sequence of *alternations* between the two quantifiers \exists and \forall . For example, for $i = 1$ the sequence contains one quantifier: \exists . For $i = 2$, the sequence contains two alternating quantifier: \exists, \forall . For $i = 3$, the sequence contains three alternating quantifier: $\exists, \forall, \exists$, and so on [114].

With the oracle machine notation [113], we have $\Sigma_{i+1}^P := \text{NP}^{\Sigma_i^P}$, i.e. Σ_{i+1}^P has access to an Σ_i^P oracle and with that access, the complexity from the verifier's perspective is NP. Similarly, we have $\Pi_{i+1}^P := \text{CoNP}^{\Pi_i^P}$. At the first level of PH, we have $\Sigma_1^P = \text{NP}$ and $\Pi_1^P = \text{CoNP}$, as shown in Figure 3.21. Note that through the PH, we also have an “optimization hierarchy” where $\Delta_{i+1}^P := \text{P}^{\Sigma_i^P}$ where P stands for the class of polynomial time complexity (in contrast to NP), i.e. with a Σ_i^P oracle, the complexity of the verifier is in P.

3.12.2 Decision and optimization in DSML

As mentioned above, Occam's learning in DSML can be seen as solving an optimization problem and DSML includes solving both an ML problem and a Co-ML problem. We will see that the problem solved by DSML is analogous to the PH class in general. In practice if the number of Choose-and-Bound iterations is limited to a constant, we can say that we are dealing with problems that is in the $O(1)^{DSML}$ class where *DSML* represents a *DSML oracle* solving a DSML problem (ML and Co-ML) and $O(1)$ represents a constant. If the number of iterations is in $O(\text{poly}(n))$ for some input size n , then the problem class is $O(\text{poly}(n))^{DSML}$.

However, a learning problem solved by the DSML oracle is not the same as those in traditional ML. Traditional ML solves an optimization problem by following the Occam's learning. The learning problem solved by the DSML oracle can be seen as a decision problem: Given a dataset and a decision question, the oracle *DSML* answers yes or no.

In rule learning, the uniqueness requirement is included and hence, the problem can

be converted into a decision version of the SAT problem [12] (see also discussion in Section 3.9.3). In correlation analysis, the problem can be viewed as follows: given a threshold, determine if, on a given hypothesis space assumption (i.e. a given dataset), there exists a correlation \geq the threshold. Similarly, in outlier analysis, a distribution-based threshold, e.g. 6σ can be used as the threshold. Hence, the ML problems solved by the oracle *DSML* are not an optimization problem in view of the analogy shown in Figure 3.21. We can say that in those ML problems solved by *DSML*, we are looking for the *existence* of a “just-fitting” answer. Hence, the “ML” in *DSML* corresponds to showing the existence of such an answer and “Co-ML” in *DSML* corresponds to showing the non-existence of such an answer. Because this “ML” is different from traditional ML, we can call it ML^* to avoid any confusion. To be more precise than before, we can rephrase the earlier equation stated in Section 2.7.5 as:

$$\mathbf{DSML} = \{\mathbf{ML}^* + \mathbf{Co-ML}^*\} \mid \{\mathbf{IEA} + \mathbf{Human\ Knowledge}\} \quad (3.1)$$

In essence, Decision Support ML includes a *DSML* oracle which is called by IEA and human. This is in contrast to equation 2.1 by (1) replacing the ML and Co-ML in the earlier *DSML* equation with ML^* and $Co-ML^*$ here, respectively, (2) replacing the “Domain Knowledge” with the “IEA + Human Knowledge”, and (3) interpreting the “|” as “calling the oracle” rather than “conditioning on” as that before. In the analogy shown in Figure 3.21, the “IEA + Human Knowledge” is analogous to solving a problem in PH, e.g. an optimization problem. Figure 3.21 shows the problem asked in one step of the optimization throughout a Choose-and-Bound search. This optimization problem is as follows: given a set of hypothesis space assumptions \mathcal{H} , what is the minimal-complexity assumption that contains a just-fit answer (or model)? Note that in *DSML* context mentioned in this thesis, the implementation of this search for the minimal-complexity

assumption may not be automatic and can involve human-in-the-loop.

3.12.3 ML vs DSML revisited

In view of the analogy in Figure 3.21, we can say that ML is to solve an optimization problem directly. DSML is to solve an optimization problem by making calls to an oracle for solving the decision version of the problem. The former resembles building an optimizer directly for solving the MaxSAT problem, while the latter involves relying on an oracle machine and solving the MaxSAT problem by devising a search algorithm that calls the oracle machine, e.g. using a SAT solver as its underlying engine. And the main reason that DSML takes the latter route to solve an optimization problem is due to the existence of a data wall.

In view of a DSML oracle machine, a key aspect for IEA, from the verifier's perspective, is to check the certificates for the answers provided by the oracle machine. In IEA 2018, this checking is supposed to be done by the concept recognition component (see Figure 3.20). To achieve DSML optimization, a search is also needed. In IEA 2018, this search is carried out by the executable workflow component. Later, it was realized that carrying out this search automatically at the data level was practically infeasible and hence, the human-in-the-loop aspect is included to achieve a Choose-and-Bound search. Note that the oracle-verifier view is crucial for the design of IEA in general and this view will be elaborated more in Chapter 8.

Chapter 4

Wafermap Analytics

愿乘冷風去，直出浮雲間。舉手可近月，前行若無山。

Wish to ride the cool breeze, through clouds I shall be free, like the moon is in my grasp, if no more mountain in front.

— 《登太白峰》李白，A poem from Tang Dynasty

Earlier in Section 1.1.1, we mentioned the problem of yield optimization. Yield is an essential metric closely tied to the profitability of a product line. Hence, yield optimization is a crucial consideration for a production line, during the early ramp-up stage and through out the entire mass production stage. Early in a production, yield optimization can include changes in the design, manufacturing process, and/or test. Later in the production, yield optimization concerns more on changing the test content, i.e. becoming part of the test optimization.

From the perspective of test data analytics, there can be two types of yield optimization: one to find correlations of a failure type to E-test parameters and the other to find a systematic *wafermap* pattern. The yield optimization story reviewed in Section 1.1.2 belongs to the first type. In this chapter, we will discuss the second type.

4.1 Yield Excursion

Semiconductor manufacturing is a complex process with many sources of variations, ranging from equipment to operation of the production line. The complex interactions among all these factors can significantly impact the yield. Therefore, even after initial efforts to stabilize the yield, through out the production the yield needs to be constantly monitored to ensure that it consistently stays above a desired level. When the yield deviates “significantly” from an expected norm, it is called a *yield excursion* [115]. A yield excursion may happen on individual wafers or lots, and during a period.

Yield is observed through testing. Wafer probe is the first testing stage to observe the yield. There is a time elapsed between a wafer being fabricated and the wafer being tested in wafer probe, and also a time elapsed between a yield excursion being observed and the excursion being escalated in an organization for investigation and action (if that happens). As a result, it is possible that when the severity of an excursion is confirmed, the underlying cause of the excursion have already impacted a large number of wafers. To shorten the time elapse, it is desirable to monitor the yield closely and as early as possible, to detect a yield excursion and confirm its severity.

While detecting a yield excursion is important, it is not always feasible to thoroughly investigate and correct every yield excursion. In practice, resources can be prioritized for excursions that have more significant impact, specifically those systematically affecting a large number of wafers. Practically, early detection of a yield excursion can mean detecting those excursions that are *worth* the investigation effort.

Determining if a yield excursion is worth investigating, similar to many problems we mentioned in this thesis, involves judgment calls. For example, yield excursions can either occur suddenly or develop gradually over time, and the amount of yield loss due to these excursions can vary significantly. Generally speaking, excursions characterized by

substantial and persistent yield loss are likely to be readily noticeable and prioritized for investigation resource allocation. Those excursions that are less significant or less persistent might go unnoticed initially, yet some of them could evolve and become significant over time. Hence, determining what constitutes a meaningful (“significant” and “persistent”) yield excursion can be a subjective judgment call that requires passing through the communication chain, rather than being solely characterized by a standalone metric.

One common approach to detect a yield excursion is by checking if there is a formation of a *failure pattern* on the wafer. A failure pattern is a region of failing dies that form a particular pattern on a wafer, such as a cluster, a line, or an arc etc. Wafers exhibiting a similar failure pattern can be considered as having the potential same cause for yield excursion. Hence, analyzing failure patterns shown on the wafers can indicate signatures for helping engineers to categorize a yield excursion.

For example, a concentrated failing region shown in the center of a large number of consecutive wafers can be caused by an misaligned step in the manufacturing process. A common way to analyze such patterns is to visualize the failure patterns on *wafermaps*, a graphical representation of some statistics based on the wafer shape, usually used to display various properties or characteristics of the chips fabricated or tested on the wafer. For example, wafermaps can be used to display the distribution of passing and failing dies in colors. This type of analytics is referred to as *wafermap analytics*.

In this chapter, we will begin by using a real yield excursion example to elucidate several important practical considerations in the problem space. While some of these have been discussed in previous chapters, we reiterate these points within the context of wafermap analytics. In Section 4.3, we will delve into the problem of wafermap pattern recognition (WMPR) and review the machine learning (ML) techniques used to address this problem. Then, we will identify and summarize the gaps between the capabilities offered by common existing ML technologies and the practical needs in wafermap an-

alytics in Section 4.4. To close the gaps, Section 4.6 will introduce the perspective of learning a *concept recognizer* and propose a novel *one-shot learning* approach for training a recognizer, which we refer to as the *Minions* model. Finally, Section 4.7 will describe the novel graph-based approach for wafermap analytics, enabled by the Minions model, and demonstrate its practical capabilities.

While the above discussion uses the term “failure pattern” and “similar failure pattern” to explain the ideas, it is important to note that the two essential questions in wafermap analytics are:

- What is the definition of a failure pattern?
- What does it mean by saying that two wafermaps are similar?

The usefulness of a tool to assist wafermap analytics largely depends on the answers to these two essential questions.

4.2 A Yield Excursion Example

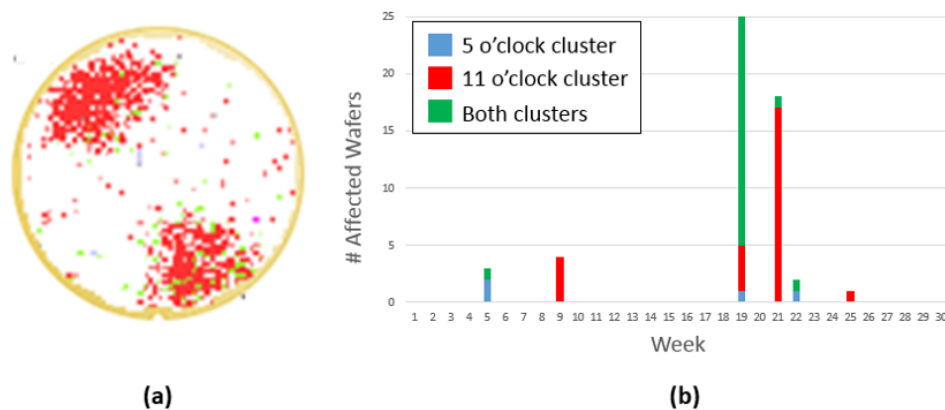


Figure 4.1: (a) Show the failure pattern constituting a yield excursion; (b) Show the number of impacted wafers over weeks of the production [13]

The work [13] reported a systematic yield event on a single product at NXP Semiconductors in 2016, as shown in Figure 4.1. Wafers affected by this issue contained a failure pattern as a cluster along the wafer edge at the 5 and/or 11 o'clock directions, as seen in Figure 4.1 (a). Note that the plot was obtained by stacking multiple wafermaps. On a single wafermap, the two failure clusters might not be as pronounced as what was shown on this plot.

In Figure 4.1 (b), the numbers of wafers that contained either the 5 o'clock cluster or the 11 o'clock cluster or both, are shown. The x-axis shows the week number when those wafers were manufactured. It can be observed that before week 19, this excursion occurred infrequently with multiple weeks spanning between occurrences. As a result, the first two occurrences were noticed but decided as one-off events due to the small number of affected wafers and the scattered locations of the clusters. The excursion was not escalated until week 19, when an entire wafer lot was affected.

There was a 14-week delay from the first occurrence to the escalation of the excursion for further investigation. During this time, the production line continued to produce wafers and the yield on those wafers might have been impacted. Further, those wafers could have an unknown quality risk due to the underlying yield issue. Identifying the issue earlier could have led to a corrective action in place earlier, thereby reducing the yield loss and the risk to product quality.

4.2.1 Industrial methods for detecting yield excursion

Detecting abnormal yield occurrences is common in semiconductor manufacturing. Statistical Bin Limits (SBL) [116] and Below Minimum Yield (BMV) are two widely adopted methods to identify wafers with abnormal yield. While BMV is a coarse anomaly detection method, SBL considers yield from individual test bins. Hence, SBL provides

additional resolution to detect yield abnormalities that may be missed by BMY.

There are also methods used to identify anomalies at a die level. Wafer spatial cluster detection algorithms such as Good Die in a Bad Neighborhood (GDBN) [117] and Unit Level Predictive Yield (ULPY) [118][119], are methods commonly employed in post processing of wafer level test data. GDBN evaluates the health of a die based on the failing statistics of its neighbors. On the other hand, ULPY also incorporates the neighbors' failing statistics but uses additionally distance information between dies to weigh the influence of neighbors, i.e. a failing die in close proximity has more influence than those located at a greater distance.

Image processing techniques have also been proposed for wafer spatial pattern detection. In [120], the Hough Transform was used to detect scratch patterns. The authors in [121] proposed transforming wafer maps into so-called spatial correlograms to aid in detection and classification of failure patterns such as clusters, circles and others. A method was proposed for identifying similar patterns by comparing the correlograms of future wafers against a reference set of patterns.

Overall, we can see that historically there are three common perspectives to check for an abnormal yield occurrence:

1. Checking some statistics of the yield
2. Checking the density of failures
3. Checking failure patterns

The first two are more commonly adopted in practice because their results are easier to be visualized and comprehended. As we mentioned, understanding the significance of a yield abnormality is important for making a decision to escalate the issue. Compared to the first two types of methods, understanding and deciding on the significance of a

failure pattern can be more complicated and remains an active area of research. Thus, the discussion in the rest of this chapter will focus on the problem of analyzing failure patterns, i.e. wafermap analytics.

4.2.2 2016 work on failure pattern detection

For the yield excursion shown in Figure 4.1, the authors in [13] proposed a methodology capable of grouping wafers that exhibit a cluster failure pattern. The methodology involved two main steps: (1) A density estimation method such as *kernel density estimation* (KDE) [122] was used to convert a wafermap into a density map. From the density map, a threshold was used to identify a failure cluster region on the wafer. (2) Some *features* were used to describe the location and direction of the cluster and possibly some other attributes, e.g. its size, total number of failures, and number of failures from each test bin, etc. Then, a *clustering* tool [123] was applied to classify wafers into groups. Note that *clustering* is a very common problem formulation in ML [123]. In this sense, the overall methodology might be seen as a ML-supported methodology.

As reported in [13], the methodology was evaluated based on 30 weeks of production data from 15 high-volume products manufactured with an analog circuit technology. These products were designed for a variety of applications. In total, around 40,000 wafers of various die sizes were analyzed.

One noticeable result from the methodology evaluation is summarized in Figure 4.2 (a), which shows a plot similar to that shown in Figure 4.1 before. The difference is that Figure 4.1 is based on one product and Figure 4.2 is based on all 15 products.

The result shows that identifying failure patterns across multiple products increased the visibility of a systematic issue that impacted an entire technology. When the analysis focused only on one product as that shown in Figure 4.1, during the first 11 weeks only

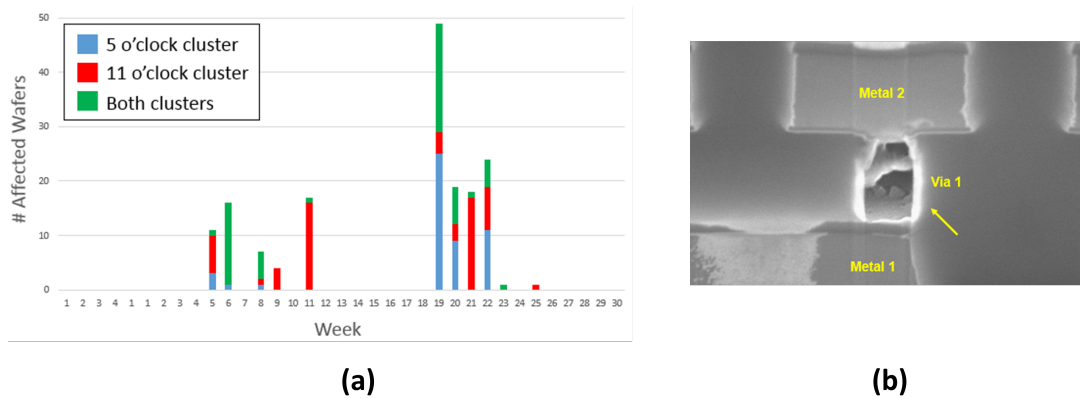


Figure 4.2: Result after using method in [13] and evidence from Failure Analysis report

9 wafers had the cluster patterns. After including the other 14 products in the analysis, this number jumped to 55 as shown in Figure 4.2.

In Figure 4.2-(a), during the first 11 weeks, there were 5 wafers with clusters at the 5 o'clock direction (highlighted in blue) and 28 wafers with clusters at the 11 o'clock direction (highlighted in red). Additionally, there were also 22 wafers with clusters in both directions (highlighted in green). In contrast to 9 total wafers in Figure 4.1, this total number of 55 wafers would have been sufficient to trigger an investigation with the in-house workflow [13].

Implementing the methodology as an automation tool can facilitate analysis of failure patterns across multiple products, thereby increasing visibility of a yield excursion. In this case, failure analysis (FA) could have been performed in week 11 or even earlier, instead of waiting until week 19, if the tool was in place. The FA report that revealed what had happened in the above example is shown in Figure 4.2-(b) (see [13]).

During the period, process changes were slowly rolled out to improve the de-vail process prior to via metal deposition. The de-vail process removes extra material that overhangs a via hole. This small obstruction is called a *vail*. Vail can partially cover a via hole, hindering the metal deposition. Although changes to the de-vail process

were thoroughly evaluated and approved, these changes resulted in significant yield loss whenever the process shifted to a particular corner.

This yield loss occurred very infrequently at first, affecting roughly one lot a week across the entire technology. Additionally, the severity of the yield loss varied. After the yield excursion was escalated, FA on failing parts found that the vails were not completely removed by the new de-vail process, resulting in partial-via defects. A cross section of one defective via is shown in the Figure 4.2-(b).

To summarize this example of yield excursion, it was initially overlooked due to its infrequent occurrences and varying severity of yield loss. The location of the cluster patterns were not consistent during their early occurrences and the issue only appeared infrequently. On one hand, the perspective of conducting analysis across multiple products had to be considered. On the other hand, an automatic pattern detection tool, such as the one provided by [13], had to be in place to monitor products throughout the entire technology and assist engineers in easily observing the excursion. As demonstrated in [13], a more extensive analysis across 15 products showed that a cross-product pattern analysis would have substantially enhanced the visibility of the yield excursion during the early weeks.

It is interesting to note that in the above yield excursion example, if the 11 o'clock cluster and 5 o'clock cluster were treated as two separate failure patterns as that in common practice of wafer pattern classification, then the significance of each individual pattern would be less than the significance of the both combined. Treating both clusters as one *signature* increased the significance of the yield excursion. This decision was hinted by the observation that there were some wafermaps containing both clusters. This example indicates that the definition of a failure pattern might be dynamic during an analytic process. Two patterns considered earlier might later be merged into a single signature (a single pattern) to facilitate the analytics.

4.3 ML View to Wafermap Analytics

The density estimation and clustering methodology utilized in [13] above can be viewed as applying ML techniques to detect failure patterns on wafermaps. A general problem, known as the Wafermap Pattern Recognition (WMPR) has been studied for decades in the field of semiconductor manufacturing [124]. The authors in [124] published a comprehensive wafermap dataset called WM-811K in 2015 for studying the WMPR problem. This dataset includes 811,457 wafer maps among which 172,950 are labeled and the rest are not. The work [124] introduced two ML-related approaches, one for classifying wafermaps and the other for searching similar wafermaps. The approaches were based on engineering a set of discriminative features to build a model for pattern recognition and a model for similarity ranking.

Multiple types of features are used, including those based on Radon transform and those based on analyzing geometric properties such as failing die counts, region labeling, line detection, etc. For model building, SVM classification [38] is used. The work reported 94.63% classification accuracy on the dataset, comparing to the deep learning approach at the time which achieved 89.64% accuracy.

4.3.1 The multi-class classification problem

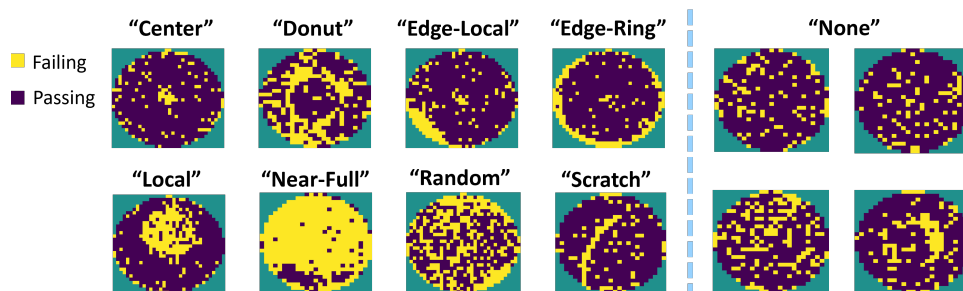


Figure 4.3: Eight pattern classes and one “None” class in the WM-811K dataset

Using the labeled portion of the WM-811K dataset, it is intuitive to treat the WMPR as a *multi-class classification* problem, a supervised learning problem in view of the general ML. The dataset pre-defines nine classes. Figure 4.3 illustrates eight pattern classes where the purple color marks the wafer and the yellow pixels indicate the locations of failing dies on the wafer. In addition to the eight pattern classes, a “None” class is used to denote wafermaps containing “no pattern”. The size of a wafermap depends on the manufacturing process. The most common size in the dataset has a width of up to 27 dies and a height of up to 25 dies.

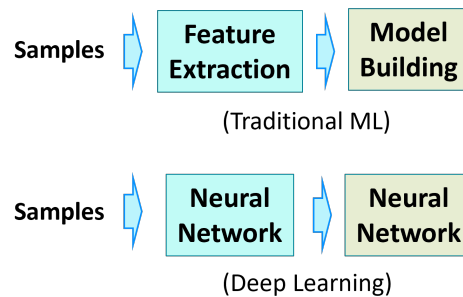


Figure 4.4: Two common ML approaches to solve a multi-class image classification problem

A number of works was published using the WM-811K dataset. These works can be roughly categorized into two types in view of general ML: (1) Those with a traditional ML approach, where a set of features were developed and used to convert each sample (a wafermap image) into a *feature vector*, followed by a model building method that operates on the set of feature vectors to learn a model; (2) Those with a *deep learning* approach, where the feature extraction step is automated by, for example, layers of a Convolution Neural Network (CNN), followed by one or more Fully-Connected (FC) layers for classification decision (corresponding to the model building step in the traditional ML). Figure 4.4 summarizes these two common ML approaches.

Early works from 2015 to 2017 followed the traditional feature engineering ML approach. The work [125] adopted the feature-based approach with a slightly different

feature set and aimed to improve the multi-pattern detection accuracy, i.e. a wafer map containing patterns from two or more pre-defined pattern classes. The work [126] advocated using more discriminant features based on Linear Discriminant Analysis (LDA) such that the model building step did not require a sophisticated method such as SVM. Instead, the work used Fisher discriminant analysis to replace SVM. Using Radon transform features, the work [127] proposed a special Decision Tree based ensemble learning method. Decision tree models are generally more interpretable than SVM. Note that the original work [124] also belongs to this category.

From 2018, applications of deep learning on the WM-811K dataset have begun to emerge. The author in [128] proposed a 2-stage classification: first to classify between having a pattern and having no pattern (i.e. the “None” class in WM-811K) and if there is a pattern, to classify which class it belongs to. Note that WM-811K is an extremely imbalanced dataset where some classes have many more samples than others [124]. This contributes to the inferior deep learning result reported in [124]. The work in [129] proposed a special data augmentation method based on Generative Adversarial Network (GAN) [130]. The network architecture also took the imbalance for learning a class into account, i.e. some classes are harder to learn than others. Instead of using GAN, the authors in [131] used an Auto Encoder (AE) [132] for data augmentation. Moreover, the author found that augmenting the samples with rotation could help. A CNN-based network was then used for training the classifier. In contrast, the authors in [133] used pre-determined methods to augment the dataset while using a deeper CNN architecture for training the classifier. The authors in [134] approached WM-811K dataset from a different angle. They tried to address the concern that a wafermap to be predicted might contain a new pattern class or a multi-pattern class not defined with the dataset. On those cases, the classifier might not be applicable. The work proposed using Selective Learning to determine an applicability of the model, where the deep learning model

included the choice to abstain from making a prediction, i.e. the neural network has an integrated option to reject some samples and only make prediction on others.

Recently, a work [135] proposed a semi-supervised learning approach to learn a neural network model with less amount of labeled data. The model includes an encoder to learn representations for wafermaps using unsupervised contrastive learning, and a supervised head for classification. Similar to previous works, a set of transformations was identified for data augmentation and used to adjust the imbalanced data. This recent result in [135] showed an average accuracy of 77.90% obtained when using different portion of labeled data ($\leq 50\%$) in training.

4.3.2 Study of a multi-class neural network classifier

In view of the big picture depicted with Figure 2.22 in Section 2.7, although we did not believe wafermap analytics should be treated as solving a multi-class classification problem as all the other works reviewed above, in [14] we did conduct a study on the WM-811K dataset based on training a neural network classifier. Our motivation was not to obtain a more accurate model as others did. Rather, we were interested in understanding the underlying dataset-related barriers causing the model accuracy loss. More generally, we were interested in showing that training a classifier was not the way to address the practical needs in wafermap analytics. This was already obviously based on the lessons learned during our first decade of the journey to IEA (see Section 2.7). Nevertheless, within the scope of wafermap analytics and in view of other WMPR, we would like to gather more evidences to strengthen our DSML view.

The classifier employed in our work [14] was based on the popular VGG-16 architecture [136]. The study selected two wafer sizes which have the largest number of wafermaps in the dataset. Table 4.1 summarizes the selected set of labeled samples from the eight

pattern classes for the experiment. In total, there were 967 wafer maps considered and they were called the “in-class” samples. In addition, there were 22,115 wafer maps belonging to the “None” class. These labeled wafermaps were used as the training set. From the unlabeled set, there were 19,086 wafermaps with the two wafer sizes. These unlabeled wafermaps were used as the test set to evaluate the classifier.

Table 4.1: Labeled wafer maps from the WM-811K dataset, used in [14]

Center	Donut	Edge-L	Edge-R	Loc	N-Full	Random	Scratch
81	10	402	9	345	16	28	76

Following a similar strategy as in [128] to tackle the imbalanced data problem, our work [14] also trained two models where one was a binary classifier for separating “in-class” wafermaps from ‘None’ wafermaps and the other was a multi-class classifier for separating the eight pattern classes. A simple data augmentation strategy involving image rotation was used to generate additional wafermaps for underrepresented classes, such as the “Donut” class and “Edge-Ring” class, as shown in Table 4.1. In our study, to make the dataset balanced across all eight pattern classes, rotated samples were added to make every class comparable to the largest number, i.e. the number of the “Edge-Local” class, 402 in Table 4.1.

Common cross-validation approach was used to train the classifiers. In our study, we obtained a multi-class model with validation accuracy of 92.5%. Then, this classifier was applied to the original 967 “in-class” samples used in training. The resulting confusion matrix is shown in Table 4.2.

In total, there were 64 mistakes reported in Table 4.2. We then manually inspected these mistakes to determine the sources of difficulties in predicting the class labels of these wafermaps. These sources can be categorized into three types:

- **Label Ambiguity:** Two very similar wafermaps have two different labels in the

Table 4.2: Confusion Matrix (On All 967 Wafer Maps): \Rightarrow : Given Label, \Downarrow : Predicted Label

	Center	Donut	Edge-L	Edge-R	Loc.	N-Full	Random	Scratch
Center	71	0	0	0	3	0	0	0
Donut	0	10	0	0	1	0	1	2
Edge-L	0	0	387	0	16	0	1	1
Edge-R	0	0	5	9	0	0	0	1
Loc.	10	0	7	0	319	0	0	7
N-Full	0	0	1	0	0	16	0	0
Random	0	0	1	0	1	0	26	0
Scratch	0	0	1	0	5	0	0	65
Total Mistakes	10	0	15	0	26	0	2	11

training dataset.

- **Under-specification:** The wafermap contains a unique pattern, which is quite different from those wafermaps included in the training dataset.
- **Model Deficiency:** Other mistakes that are not due to the above two reasons.

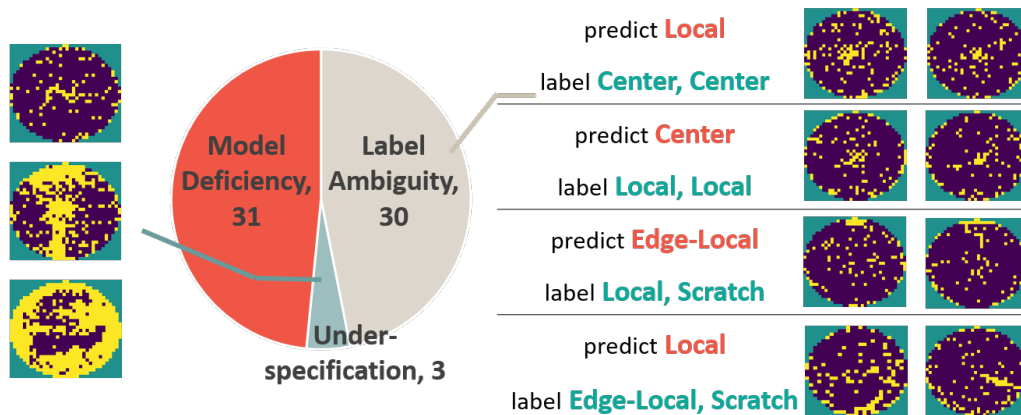


Figure 4.5: Examples of mistakes in Table 4.2

The number of mistakes in each category is shown in Figure 4.5. If we consider the 33 cases in the first two categories as problems with the dataset itself, over half of the mistakes was due to this dataset deficiency.

The right-hand-side of Figure 4.5 shows several mistakes from the label ambiguity category. Additional examples can be found in [14]. For each case, two wafermaps are shown. Note that for each wafermap in this category, we found a wafermap with a similar pattern labeled as the predicted class to justify the prediction. For example, if a wafer map was predicted by the model as “Local” but was labeled as “Center” (first case), we considered this as label ambiguity if we could find another similar-looking wafer map in the “Local” class to justify the prediction (second case). In other words, we cannot blame the model for the mistake because two very similar-looking wafermaps were assigned with two different class labels to begin with.

The left-hand-side of Figure 4.5 shows three mistakes in the under-specification category. Each of these patterns appeared only on one wafermap. If the wafermap were not in the training set, the model would never see the pattern. Hence, it is understandable that the model would have mistaken on them.

The same VGG architecture was used to train a binary classifier model to differentiate the “in-class” wafermaps from the “None” wafermaps [14]. The same image rotation strategy was used to make the training dataset more balanced. The training dataset comprised all 967 original “in-class” wafermaps and their rotated images. It also included 9K randomly-selected “None” wafer maps. The test dataset comprised the rest of the “None” wafermaps *without* any “in-class” wafermaps. The test dataset is used to evaluate how many wafermaps can be filtered out by the model. Ideally, we would like the model to filter out all wafermaps in the test dataset. Our work in [14] reported a binary classifier model with training accuracy of 95.25%. This model filtered out 99.33% of the “None” samples in the test dataset. However, it also filtered out 249 of the “in-class” wafermaps.

The binary classifier and the 8-class classifier were then applied to the 19086 unlabeled wafer maps in sequence, as illustrated in Figure 4.6. After the first step, only 741 wafer maps were left (they were supposed to have an “obvious” in-class pattern). Their



Figure 4.6: Two-step classification employed by the two trained VGG models [14]

classification result is shown in Table 4.3. After manually reviewing the result, those “questionable” were identified (see [14]). The number of questionable samples are also shown in the table and we consider them as potential mistakes.

Table 4.3: Classification on the 741 unlabeled wafer maps

Class	Center	Donut	Edge-L	Edge-R	Loc	N-Full	Random	Scratch
Classifier’s Result	114	37	211	44	233	9	34	59
Potential Mistakes	2	24	4	10	37	0	10	49

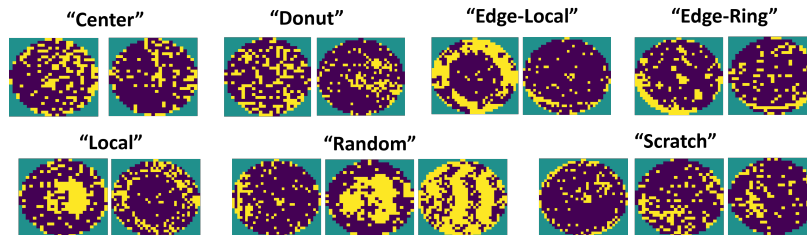


Figure 4.7: Examples of questionable classification (the shown labels were reported by the model and were considered questionable in the manual review) [14]

Figure 4.7 shows some examples of those potential mistakes. Additional examples can be found in [14]. An interesting aspect observed was that although the 8-class VGG classifier previously achieved a high validation accuracy of 92.25% on the labeled dataset, its performance on the set of 741 unlabeled wafer maps was considerably worse. In a sense, the earlier 92.25% accuracy result for the model might be somewhat misleading when considered from a practical application standpoint. Additionally, it was noted that the classifier’s performance on different classes varied significantly.

The two classifiers reported in [14] were by no means optimal. However, optimizing model’s classification accuracy was not the goal of the study. The study focused on

understanding the sources of the mistakes and suggested that for dealing with some mistakes, it might not be effective to look at the problem from the perspective of model optimization. This was because many mistakes were due to some issues in the dataset to begin with.

4.3.3 Bypassing the training dataset issues in ML

Based on our study in [14], if we were following a ML view to approach wafermap analytics, a logical conclusion would be to improve the quality of the training dataset. Indeed, this is a common ML thinking, beginning with defining the general problem using the training dataset, followed by two focuses of optimization: the quality of the dataset and the accuracy of the ML model. Hence, if the accuracy loss is due to the quality of the dataset, a logical action is to improve the dataset.

For all the other works reviewed in Section 4.3.1 above, their focus was on the model accuracy. Although some tried to address the issue of data sufficiency through data augmentation or using less training samples, they never questioned the validity of the WM-811K dataset to begin with. In contrast, we did in our work [14].

The validity can be questioned at two levels. At the dataset level, it had issues from label ambiguity and under-specification. At the problem definition level, the eight-class classification, in some cases, seemed quite arbitrary. In other words, the initial specification of the general problem might be questionable.

Our approach to resolve these two levels of questions is to *bypass them entirely* under our DSML view, as depicted in Figure 4.8. Instead of starting with a *general* problem definition, each step in the DSML view starts with a dataset associated with a *local* question on the dataset. This local “(dataset, question)” pair is sent to a DSML oracle (see discussion in Section 3.12.2) which either provides an answer to the question or

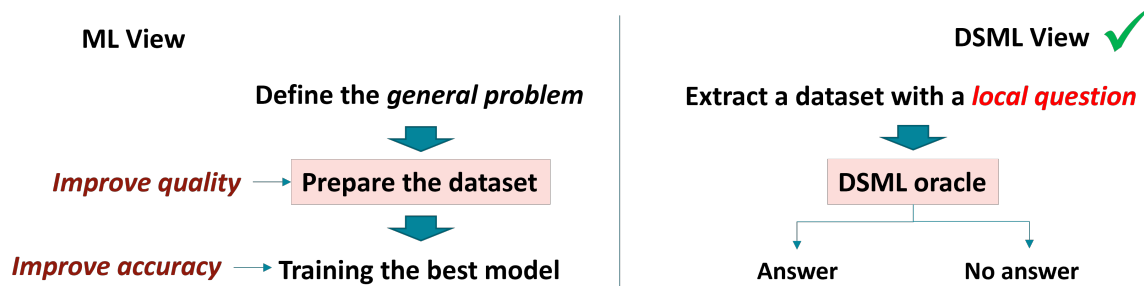


Figure 4.8: ML view vs. DSML view to approach wafermap analytics

decides that no answer can be found on the given dataset. To develop a wafermap analytics tool with the DSML view (i.e. to serve as the DSML oracle in the wafermap analytics context), the main concern is on the types of the questions to be supported by the tool. This is a key consideration for the development of the Minions approach and will be elaborated in detail in the following.

4.4 Types of Analytic Questions

For the application contexts reviewed in Chapter 2, there are many contexts where the analytic questions focus on one sample. For example, in speedpath analysis (Section 2.3), we are asking the question regarding the cause of a path. In functional coverage improvement (Section 2.4.2), we are asking the question regarding how to hit a coverage point. In customer return analysis (Section 2.5), we are asking the question how to project a known failing chip as an outlier.

In wafermap analytics, one can also ask a question based on one particular wafermap, e.g. find me all wafer lots containing the pattern shown on the wafermap. In addition, one can ask a variety of other types of questions. In wafermap analytics, we can consider analytic questions in two categories: *pattern based* and *lot based*.

Pattern based This category of questions asks the DSML oracle to analyze the data based on patterns, and what constitutes a “pattern” can be unspecified and left to the DSML oracle to decide. There can be three types of questions:

Existence type For example, the question can be “What patterns are in the dataset?”. Underlying this question is a more basic decision question: “Do you see a pattern in the dataset?”.

Search type The question is formed by a pair of “(dataset, wafermap)”, asking to find all wafers from the dataset, which contain the pattern shown on the given wafermap. Again, the decision version of the question is that “Do you see other wafers containing the pattern shown on this wafermap?”

Attention type This type of question is to ask the oracle to point out where in the data needs more attention. For example, the question can be “What patterns do I need to pay attention?” The decision version of the question can be that “Is there a pattern that I need to pay attention this week?”

Lot based Pattern-based questions above can also be asked from the lot perspective. Lot-based questions ask the DSML oracle to analyze the data based on wafer lots. Lot-based questions are different from pattern-based questions mainly because the definition of the word “pattern” can change.

Existence For example, the question can be “Which lots contain a significant pattern?”. Underlying this question is a more basic decision question: “Do you see a lot containing a significant pattern?”.

Search type The question is formed by a pair of “(dataset, lot)” to ask the oracle to find all lots “similar” to the given lot. Again, the decision version of the question is that “Do you see other lots similar to this lot?” The scenario might

be that a person has decided that one particular lot is reflecting a yield issue and desires to find all lots possibly exhibiting the same yield issue.

Attention type This type of question is to ask the oracle to point out which lots need more attention. For example, the question can be “Which lots do I need to pay attention?” The decision version of the question can be that “Is there a lot that I need to pay attention this week?”

4.5 From ML Classifier to DSML Oracle

In view of the types of analytic questions listed above, it is obvious that a multi-class wafermap classifier trained with a pre-defined set of pattern classes is insufficient for answering those questions. Rather, answering those questions necessitates a DSML oracle as shown in Figure 4.8

4.5.1 Definition of pattern classes

A dataset like WM-811K [124] is based on pre-defined pattern classes. This definition is subjective in view of an intended application context. With a fixed class definition, it is assumed that in an application, there is no need to differentiate wafer maps beyond those defined classes. This is not true in view of the analytic questions above.

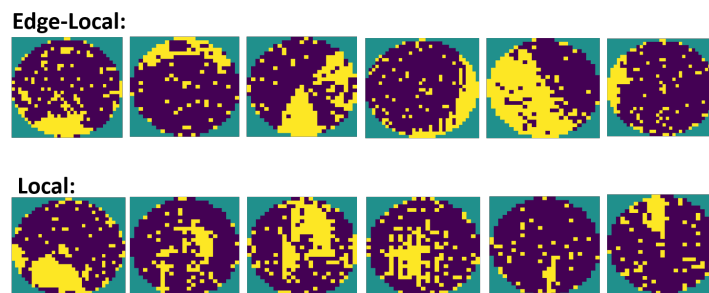


Figure 4.9: Within-class pattern variations seen in WM-811K dataset

The pattern classes defined in WM-811K can be a rather limited way to differentiate wafer maps. For example, Figure 4.9 shows six examples from each of the two classes, “Edge-Local” and “Local”. As seen, the six wafer maps in each set do not look similar. In one context, it might make sense to consider them as a single class. In another, it might make sense to separate them into different classes. In other words, the class definition should depend on the application context.

Deciding in advance a pre-defined set of pattern classes that makes sense universally in all contexts can be an unrealistic thinking. For example, Figure 4.10 shows six consecutive wafermaps from one single lot in the WM-811K unlabeled dataset. These six wafers clearly show a systematic trend. However, based on the WM-811K’s class definition, the 1st, 3rd, and 5th wafers might be called a “Local” pattern. The 2nd and the 4th might be a “Donut”. The rightmost one might be called a “Center” pattern or a “Local” pattern (subject to label ambiguity). From the pattern class definition, they are not the same. However, by the fact that they are consecutive wafers within the same lot, the systematic trend is clear. Hence, from the lot perspective all these wafermaps belong to a “single pattern” that they are likely caused by the same issue.

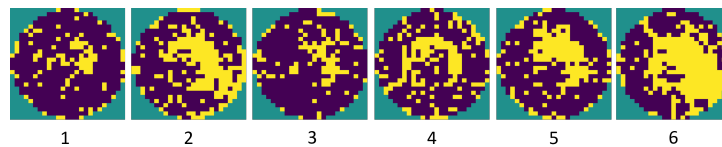


Figure 4.10: An example of pattern class varying within a single lot in WM-811K dataset

Example shown in Figure 4.10 illustrates why we earlier separate the analytic questions into two categories. The definitions for a pattern with and without taking the lot perspective, can be different. Thus, the DSML view in Figure 4.8 allows patterns to be defined based on the given question. To emphasize this point, we state three postulates, and in the rest of the chapter we will discuss how to achieve a DSML oracle in view of

these postulates.

1. Pattern class definition depends on the analytic question.
2. Pattern class definition depends on the dataset associated with the question.
3. A user can assert a pattern class definition with one wafermap.

4.5.2 (Dataset, Question) pair with pattern class constraints

Given a “(Dataset, Question)” pair, it is desired that a DSML oracle uses a pattern class definition that is most appropriate in view of the question. Therefore, it is important to note that on the same dataset, different questions may lead to different pattern class definitions. Similarly, on the same question, different datasets may lead to different pattern class definitions.

In practice, a pattern class definition derived by the DSML oracle on its own might not completely meet the need of a user. Hence, the DSML oracle should allow the user to enforce a pattern class. For example, the user can provide one or more wafermaps and enforce they being treated as a pattern class.

4.6 Concept Recognition

Section 3.10 briefly introduces the *concept recognition* component with Figure 3.20. In [84], the term “concept” is explained through several examples without a formal definition. In this section, we provide a more formal definition of the term “concept”.

4.6.1 What is a “concept”

In view of a DSML oracle discussed in Section 3.12, a *concept* C is a description used to phrase a decision problem given to the DSML oracle. The problem statement is an

existence question in the following form:

With the given dataset, does C exist?

The following shows some C examples in different contexts.

- A correlation ≥ 0.65 between an E-test and a failure type.
- A test space where the given failing chip is an outlier.
- A monomial with length ≤ 3 to explain a speedpath.
- A wafermap that has the same pattern shown on a given wafermap.

In each case, the existence of C in the given dataset is decided by the DSML oracle. If C exists, then the DSML oracle reports a set of instances to show its existence. Otherwise, the DSML oracle reports the non-existence of C in the dataset.

A concept C can be more complex than those shown above. For example, C can be called “a systematic trend” and “a problematic lot” in the wafermap analytics context. Regardless how a concept is called and defined, the important aspect is that a concept must be associated with a *decision procedure* in the DSML oracle to decide its existence in a given dataset. In other words, we define concept as the following.

A concept is something whose existence in a dataset
can be decided by one or more DSML oracles.

In other words, if IEA contains n DSML oracles, the system can support no more than $2^n - 1$ concepts. For each concept C_i , the corresponding DSML oracle D_i is to decide its existence or not. If a question asking for the simultaneous existence of C_i, C_j, C_k , then it is equivalent to asking the existence of a higher level concept $C_{i,j,l} = C_j \wedge C_j \wedge C_k$.

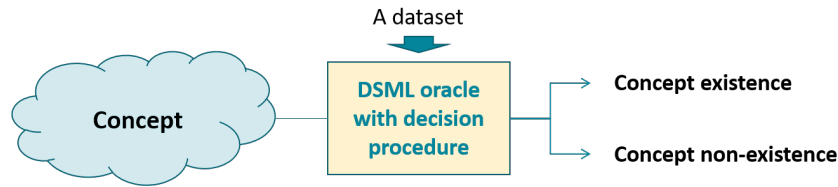


Figure 4.11: A DSML oracle provides decision support at the concept level

4.6.2 Decision support by a DSML oracle

Earlier in Section 2.7.4 we define DSML as decision-support ML in view of the big picture Figure 2.22. The term “decision-support” can be seen as supporting the decisions through the decision chain (see Section 2.7.3). Based on the discussion in this section, we see that there is a more basic level of decision support, provided by a DSML oracle. This is illustrated in Figure 4.11. In essence, to support decision making through the decision chain, we incorporate decision support starting at the basic concept level. This thinking is fundamental to the design of IEA, that analytic result for making a higher level decision should be recursively supported by decisions made at a lower level. Note that this thinking is in parallel to assembling concept hierarchically, wherein more complex concepts are constructed based on simpler ones..

4.6.3 Graph-based concepts in wafermap analytics

To enable answering the types of analytic questions discussed in Section 4.4 above, in the following we define a list of wafermap concepts. These concepts are defined based on two symmetric binary relations: *semi-equivalent* and *similar*, denoted as \cong and \simeq , respectively.

\simeq Given two wafermaps wp_1, wp_2 , a DSML oracle decides if they are similar to each other.

If they do, their relation are denoted as $wp_1 \simeq wp_2$.

\cong Given a group of wafermaps $E = \{wp_1, \dots, wp_k\}$, for $k \geq 3$, we say that they are

semi-equivalent if $\forall i, j, 1 \leq i, j \leq k$, we have $wp_i \simeq wp_j$. In this case, we say that $wp_i \cong wp_j \forall wp_i, wp_j$ in the group.

With the two binary relations, we can think of concepts defined in this section as *graph-based concepts*. With the similarity relation \simeq , a set of wafermaps can be converted into a wafermap graph where there is an edge between any two wafermaps wp_1, wp_2 in the set if $wp_1 \simeq wp_2$ holds. Then, the establishment of the semi-equivalent relation among a group of wafermaps means that the group forms a *clique* in the wafermap graph.

Note that the basis for deciding the two relations is to decide the similarity between two wafermaps. This similarity decision can depend on the implementation of the DSML oracle. Later in Section 4.7, we will discuss the Minions approach for implementing this similarity decision.

With the two wafermap relations defined, we can use them to define several pattern concepts used to support wafermap analytics.

Anchor Pattern Given a set of wafermaps, an anchor pattern is represented by a maximal group of semi-equivalent wafermaps $A = \{wp_1, \dots, wp_k\}$ s.t. $\forall wp \notin A, \exists wp_i \in A, wp$ and wp_i are not similar, i.e. A is a maximal clique in the corresponding wafermap graph.

Primitive Pattern A primitive pattern is represented by a group of anchor patterns A_1, \dots, A_l for $l \geq 1$ s.t. for any pair A_i, A_j , there exists a $wp \in A_i$ and $wp \in A_j$, i.e. the two cliques share the same wafermap wp .

Lot Pattern A lot pattern is a collection of primitive patterns appearing in the same lot. In this case, they are treated as a single pattern.

Pattern A pattern is dynamically decided based on a fixed and computable rule that is stated in terms of the pattern concepts and relations defined above.

With the two relations and the pattern concepts, the DSML oracle can further defines other relations and measurements to support the analytics.

i-Hop Relation i.e. $\simeq_i, \forall i \geq 0$. We say that there is an i-hop relation between two wafermaps wp_a and wp_b , denoted as $wp_a \simeq_i wp_b$, when there exists a sequence of wafermaps wp_1, \dots, wp_i s.t. $wp_a \simeq wp_1 \simeq wp_2 \simeq \dots \simeq wp_i \simeq wp_b$. We let \simeq_0 to be the same as \simeq .

Distance Given two wafermaps wp_a, wp_b , their distance, denoted as $dist(wp_a, wp_b)$ is $i + 1$ iff $w_a \simeq_i w_b$ holds. If wp_a, wp_b have no similarity relation between them for any i , then the distance is ∞ .

Group Similarity Strength Given a group of wafermaps wp_1, \dots, wp_n , we can measure a *similarity strength* for the entire group based on the distance concept defined above. For example, this strength can be measured as $\frac{1}{n} \sum_{\forall i, j} \frac{1}{dist(wp_i, wp_j)^2}$.

Group-to-Wafer Similarity Strength Given a wafermap wp , and a group of wafermaps wp_1, \dots, wp_n , we can also measure a group similarity strength with respect to wp . For example, this strength can be measured as $\sum_{\forall i} \frac{1}{dist(wp, wp_i)^2}$.

4.6.4 Answering wafermap questions

Wafermap concepts in the previous sections can be readily defined once the relation \simeq is implemented in the DSML oracle (equivalently, once a wafermap graph based on \simeq is constructed). In Section 4.7 below we will discuss an implementation of the similarity relation \simeq . Before getting to the implementation, in this section we provide a few examples to illustrate how those wafermap concepts can be used to answer questions discussed in Section 4.4 before.

Pattern based Below shows the three types of pattern-based questions discussed in Section 4.4 and examples for how to answer them.

Existence question “What patterns are in the dataset?”

The DSML oracle can output the primitive patterns if they exist. If no primitive pattern exists, the oracle simply says there is no pattern.

Search question “Are there other wafermaps in the dataset, containing the pattern shown on wp ?”

The question can be answered in two steps. For example, the first step is to determine if the pattern on wp can be deemed as a primitive pattern or close enough to a primitive pattern, say within a distance i for some i . If not, then the answer is “no pattern seen on wp ”. This is a required step to verify that wp indeed has a pattern from the oracle’s perspective. If the first step passes and it is determined that wp has some pattern on it, then the search can be carried out by finding all wafermaps whose distances are $\leq i$ for some pre-fixed i . In addition, the oracle can allow the user to specify a different i as an additional constraint to the question.

Attention question “Is there a pattern causing more than 0.5% of yield loss?”

This question can be answered by two steps. The first step is to define the pattern with a collection of wafermaps, followed by the second step to compute the average yield loss on those wafers. The collection of wafermaps can be selected, for example, by starting with a primitive pattern P and including all wafermap wp s.t. $dist(wp, wp_i) = 1$ for some $w_i \in P$.

Lot based Below shows the three types of lot-based questions discussed in Section 4.4 and how to answer them.

Existence question “Which lots contain a significant pattern?”

The significance of a pattern can be based on the average yield loss across a group of wafermaps containing the pattern. Then, with a given threshold, a pattern can be called significant or insignificant. If a pattern is deemed significant, those lots with the wafermaps containing the pattern can be reported. Again, the oracle might calculate a yield loss threshold by itself or take a user-defined threshold as the constraint to determine significance.

Search type Question: “Which lots contain wafermaps similar to this wp ?”

Answering this question is similar to answering the pattern-based search question above. The lot name of the found wafermaps are reported. In addition, the lots can be ranked based on their group-to-wafer similarity strength to wp .

Attention type Question: “Is there a lot with a pattern that causes more than 0.5% of yield loss?”

Answering this question is similar to answering the pattern-based attention question above. In addition, the lots can be ranked based on their group similarity strength.

4.7 The MINIONs Approach

The above discussion shows that in practical wafermap analytics, one can ask a variety of analytic questions. To answer those questions, we need to define a variety of different concepts. To achieve the required flexibility, at the most basic level we need a definition for the similarity relation \simeq . Once the similarity relation is there, we can use it to construct a *wafermap graph*. Then, based on a given wafermap graphs, various concepts can be defined to answer various analytic questions. To be clear, the following provides the problem statement fundamental to the wafermap analytics discussed so far.

What we need is a decision procedure that given two wafermaps decides if they are similar or not.

4.7.1 Previous works on learning a concept recognizer

The idea of concept recognition in test data analytics was first proposed in [84] from our lab and the work in [85] tried to implement concept recognition where concepts were represented as plots. In the context of wafermap analytics, the starting point is that a user provides a small set of wafermaps to represent a pattern class of their own. The user would like to have a model to recognize just the particular pattern class. In this scenario, the concept is represented by the set of wafermaps.

The works in [85] and its follow-up work [137] were both based on this perspective. In the first work [85], the approach was based on Generative Adversarial Networks (GANs) [130] for learning a concept recognizer. GANs are methods to learn a *generative model*. Given a dataset, a generative model synthesizes new samples similar to the training samples. A GAN's architecture consists of two neural networks. The *generator* network \mathbf{G} is trained to generate samples. The *discriminator* network \mathbf{D} is trained to differentiate the training samples from the generated samples.

In [137], the discriminator was a simple CNN with one convolution layer and one fully-connected (FC) layer. The generator network basically had a reverse-CNN architecture with more convolution layers and FC layers. The design in [137] used a simple discriminator and a more complex generator because generation of samples was harder than classification of samples. In other words, the generator was given with more *capacity* than the discriminator.

The implementation was inspired by the regularization ideas proposed in [138][139]. Note that in typical GANs training, the goal is to learn a good generator. The goal was

different in [137]. After learning, the discriminator was used as the concept recognizer. Hence, the training focused on the quality of the discriminator rather than the quality of the generator.

To train a discriminator as the concept recognizer, the experiments in [85] showed that a minimum requirement of five training samples. It should be noted that this minimum requirement depended on the pattern class to be learned. In [85], the five wafermaps used in the experiments were very similar. If we were to learning a recognizer for a pattern where within-pattern variation is large (e.g. those in Figure 4.9), then five training samples would not be enough.

For training GANs, attention is required to ensure two aspects: the output quality of both neural networks and the convergence of the training iteration. The work in [140] suggests several guidelines to improve the quality.

The work in [85] found that the performance of the CNNs were sensitive to whether or not (1) Batchnorm was used in both generator and discriminator CNNs, and (2) the Leaky ReLU activation function was used for all perceptrons. Implementation of a *Dropout* strategy was also found to be crucial [141]. For convergence, the feature matching technique [142] was crucial. Furthermore, in the discriminator network the Sigmoid function was used to convert the output of the last perceptron into a value between 0 and 1. In the generator network, the Hyperbolic Tangent function was used for adjusting the output value. The CNNs in [85] were implemented with Google TensorFlow [143]. The ADAM optimizer [144] was used for the training.

Lessons from previous works on learning a concept recognizer

The conclusion from [85][137] was that training GANs for building a concept recognizer could be quite tricky [140][142] because balancing the convergence between the generator and the discriminator could be challenging. An additional tricky aspect is de-

deciding when to stop the training. This aspect was studied in detail in [137]. If the goal was to obtain a good generator, the training could proceed until the discriminator failed to separate the training samples from the generated samples. This strategy would not work if the goal was to obtain a discriminator as the recognizer. In [137], deciding when to stop the training was based on so-called *separability*, a measure how well the in-class and out-of-class samples were separated by the discriminator.

Because of all the subtleties in training a GANs-based concept recognizer, it would be difficult to guarantee the performance of a concept recognizer. The robustness concern motivated the use of another method to check the performance of a concept recognizer [145]. A tensor computation based method was developed and refined through a sequence of works [146][147][148]. Initially, the method was for implementing an online checker for a concept recognizer. In [137], it was further extended as a way to extract training samples for training a GANs-based recognizer.

Because training a GANs model is complicated, the robustness of a GANs-based recognizer can be a concern. While the tensor-based containment check was proposed to mitigate the concern, it did not completely address all the challenges (see [137]). The flow presented in [137] was comprehensive. Nevertheless, its use was still limited to the application scope of multi-class classification. In other words, its output is still a set of “best-determined” pattern classes and the set of wafer maps identified in each class.

In view of the three postulates discussed in Section 4.5.1 above, the early works for concept recognition were not sufficient to address all the needs. All the challenges were addressed after the MINIONS approach was invented [15][14].

4.7.2 Learning a concept recognizer for one wafermap

The MINIONS approach enables a reliable implementation for realizing the similarity relation \simeq defined above. It started by finding a way to learning a concept recognizer for just one wafermap [15]. The MINIONS approach was presented in [14], which was based on the *manifestation learning* approach presented in [15]. MINIONS stand for MINiture Interactive Offset Networks. Figure 4.12 illustrates the high-level idea of the MINIONS approach.

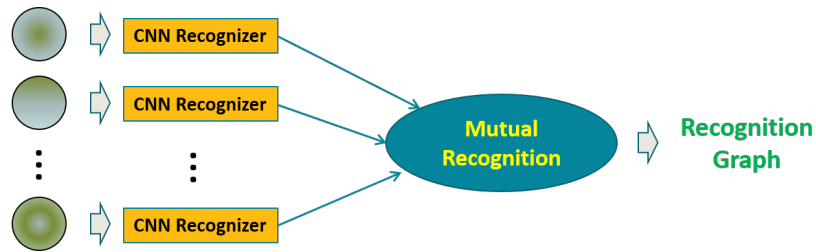


Figure 4.12: High-level idea of the MMINIONS approach

With the approach, a neural network (NN) model is independently learned for each wafermap. This NN model serves as a *concept recognizer* dedicated for the wafermap. In this sense, each wafermap by itself represents a concept at the most basic level. In [14], each recognizer is a neural network model called a MINION. The wafermap used to train a MINION is called its *anchor*. With one MINION for every wafermap, we can then perform *mutual recognition* on pairs of wafermaps, which will result in a *recognition graph*. In this graph, every node is a wafermap. Two nodes have an edge connecting them if their recognizer recognizes each other (i.e. mutual recognition). Therefore, in the MINIONS approach the similarity relation \simeq is decided by the mutual recognition.

Given a recognition graph, we can analyze wafer maps using well-known operations on graphs. For example, clusters of wafermaps can be attained by finding all Connected Components (CC) where each CC is treated as a separate cluster. Figure 4.13 shows a CC example based on wafermaps from the WM-811K dataset. It is interesting to

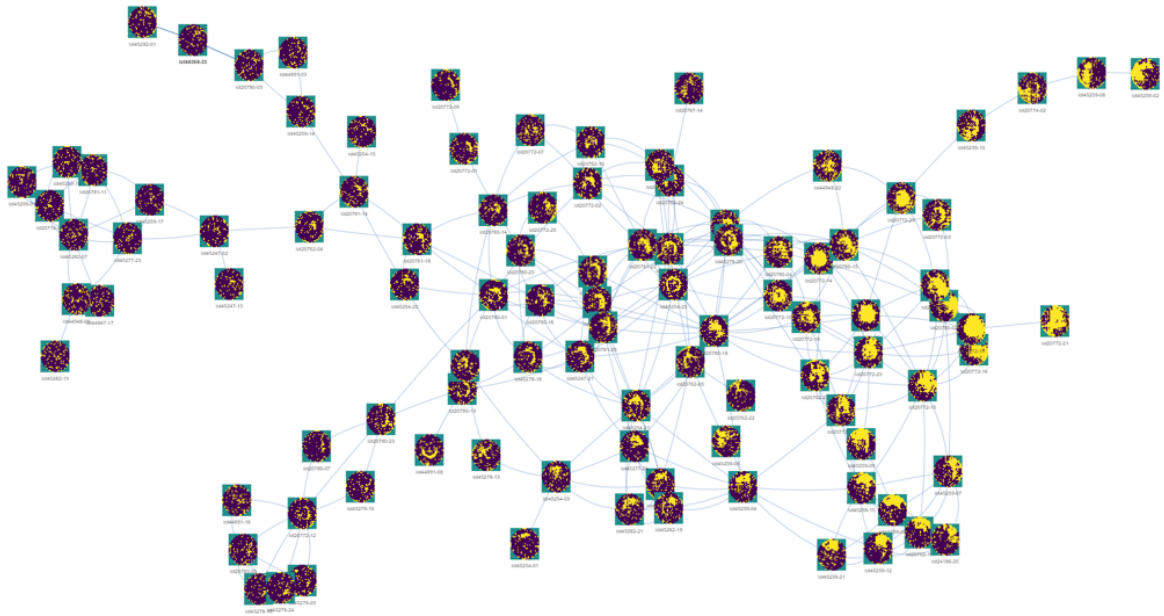


Figure 4.13: An example connected component extracted from a MINIONs recognition graph notice that while two wafermaps with a direct connection have a similar pattern, not all wafermaps in the CC have a similar pattern. This is because each MINION can recognize a wafermap with certain variations from its anchor. In addition, in the implementation an input wafermap is rotated with $\pm x^\circ$ to generate additional input wafermaps such that if any of them is recognized by a MINION, the original input wafermap is considered as recognized. The setting used to generate Figure 4.13 is $\pm 10^\circ$. Because each MINIONs can recognize small variations of the pattern shown on its anchor, as a result, in a CC when two wafermaps are connected through more edges, they tend to become more dissimilar. In other words, along a path of multiple edges in a CC it is possible to observe a *morphism* across the patterns shown on the wafermaps.

One major advantage of the MINIONs approach is that it turns a traditional analysis such as clustering from statistical to graph-based. After a mutual recognition graph is constructed, all analyses can be done with graph-based operations, which can improve

traceability and robustness of the analyses.

4.7.3 MINIONs' one-shot learning

A MINION is trained with one sample. Training with one sample is generally referred as *one-shot learning* [149][150]. In Machine Learning, three approaches have been proposed to tackle one-shot learning, as illustrated in Figure 4.14.

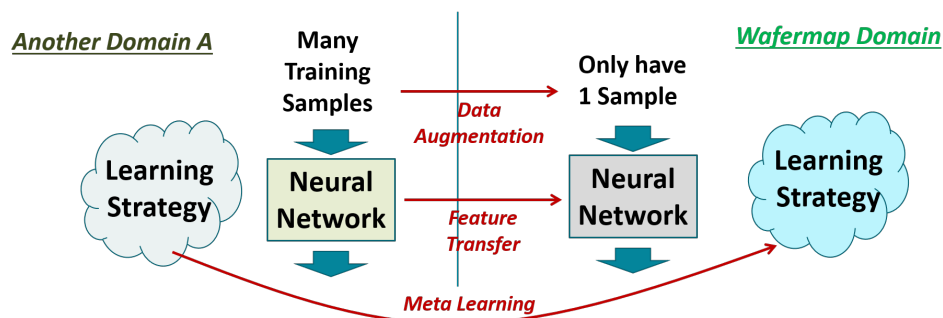


Figure 4.14: Different ways to attain one-shot learning

Two domains are shown, a source domain A on the left and the target wafermap domain on the right. Domain A has many training samples. Wafermap domain has only 1 sample. The idea of data augmentation is to learn from samples in domain A to generate more samples and augment the dataset in the wafermap domain [151][152][153]. In *feature augmentation* (e.g. [153][154][155][156]), generated samples are in the feature space rather than in the input space.

In *feature transfer* [157], neural network weights learned from the source domain are transferred to the wafermap domain. *Domain adaptation* [158][159] is a specialized approach where the transfer is between two domains for performing the same task and hence might not be applicable here. In *meta-learning* [160], a learning strategy learned in the source domain is transferred into the target domain. A recent study [161], though, shows that meta-learning is not as effective as people had claimed on a variety of tasks.

For learning a MINION model with one wafermap in the target domain, the work in [14] did not find the three approaches effective. Instead, a fourth approach called *manifestation learning* [15] was adopted. From the perspective of transferring between two domains, this approach transfers an “output vector space” (an *embedding space*) from domain A to the wafermap domain [15]. In work [14], a two-part training scheme was used based on a variant of the Triplet Loss Siamese Network [162].

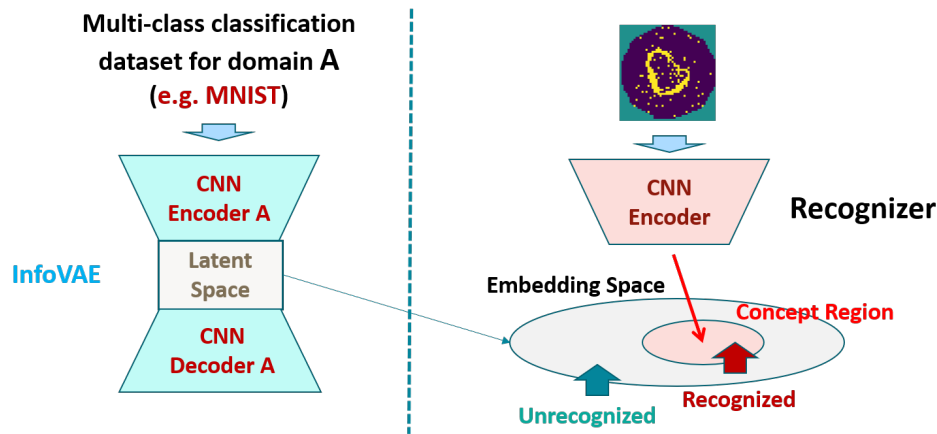


Figure 4.15: The idea of manifestation learning presented in [15]

Figure 4.15 explains the idea of manifestation learning presented in [15] and further improved in [14]. The training has two phases. In the preparation phase, popular handwriting digit classification dataset MNIST [163] is used. The MNIST dataset contains 5000 images for each digit, 0 to 9. In manifestation learning, this dataset is used first to train a Variational Auto-Encoder (VAE) model [164]. A VAE model comprises a CNN encoder and a CNN decoder. The CNN encoder maps each MNIST image onto a distribution of *embedding vectors* in the latent space. Then, the CNN decoder maps an embedding vector back to an image. In [15], the training is based on InfoVAE [165].

After the InfoVAE training to obtain the latent space with embedding vectors, on its latent space we build an SVM one-class model to capture a *concept region* for a selected class. The class used in [14] was the class of digit “1”. The SVM model is built in such

a way that it tries to capture all embedded vectors of digit “1” and no vector from any other digit. Hence, it is a conservative model.

Refer back to Figure 4.15. In the application phase, the training takes place in the wafermap domain with the purpose to train an encoder without the decoder. The latent space with embedded vectors from the MNIST training is transferred to serve as the target for the encoder. The same SVM model is reused for training a recognizer for every wafermap. The idea in this training is to map the wafermap to the center of the concept region defined by the SVM model. Detail will be explained in Section 4.7.4 below.

After the training, the CNN encoder is our recognizer (MINION) for one given wafermap. Then, for a wafermap given as input to the recognizer, it will be mapped to an embedding vector. If this vector falls inside the concept region as determined by the SVM model, it is recognized. Otherwise, it is unrecognized.

4.7.4 MINION’s training detail

While the initial works described some of the MINION’s training details [15, 14], our implementation has been enhanced over time with the development of IEA. Here, we consolidate the most updated training setup of the MINION models as follows.

As discussed in Section 4.7.3, the first phrase of training a MINION model is to prepare the latent space by training an InfoVAE model with MNIST dataset. A wafermap image was encoded with values 1 and -1, indicating the failing and passing dies, respectively. The digit images from MNIST dataset were normalized to use the same encoding scheme as the wafermap images.

An InfoVAE includes a encoder and a decoder. For our implementation, the encoder contained Convolutional Neural Network (CNN) layers and a dense layer that outputs a set of parameters for specifying the latent distribution. Specifically, two parameters

called the “mean” and “variance” were learned, each was specified as a 16-dimensional vector. Then, the reparameterization trick was applied to generate a latent sample for the decoder [164, 165]. The decoder contained several dense layers which takes the latent sample as input and transpose convolutional layers to reconstruct the original input image. The activation function used in the training was Leaky ReLU and an learning rate of $5e-4$ was adopted for Adam optimizer.

InfoVAE uses the Maximum Mean Discrepancy (MMD) as the loss function, instead of the traditional evidence lower bound (ELBO) used in VAEs. In this way, the latent distribution has to match the prior only in expectation, rather than for every input, thus enabling more meaningful latent representation [165]. This property is preferred in the manifestation learning because our goal was to obtain a latent space of embedded vectors which maintains class information. The original VAE is designed to learn a powerful decoder as the generator. Thus, the ELBO criterion can become overly restrictive, mapping every class of input distribution onto the same Gaussian distribution, i.e. random noise. This can result in the loss of information in the latent space.

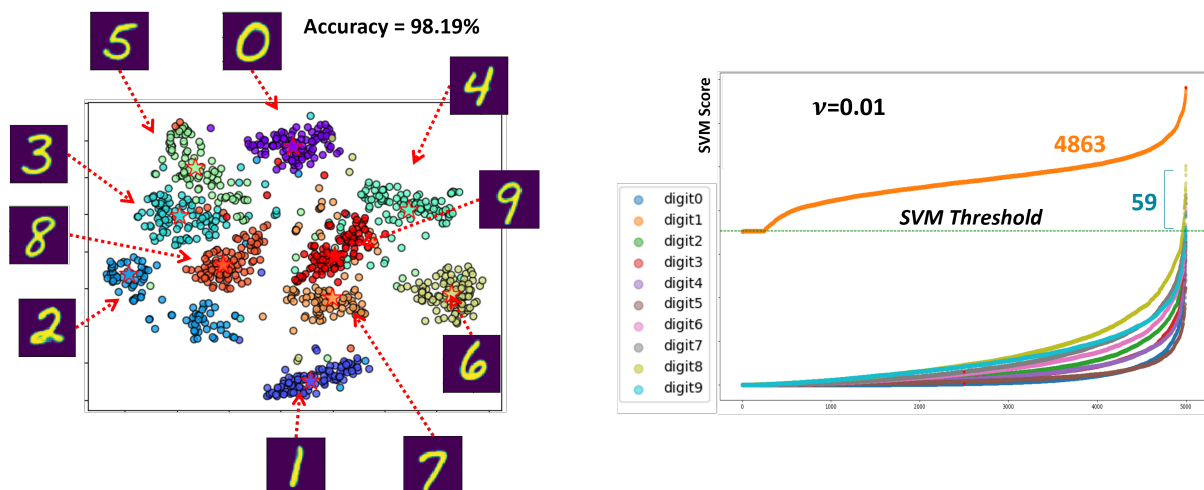


Figure 4.16: Visualization of latent space learned with MNIST samples in [15]

The learned encoder essentially transforms an input image into a 16-dimensional latent vector using the reparameterization trick. We refer to the collection of these latent vectors obtained from each digit image as the latent space. The left plot in Figure 4.16 shows a visualization of the learned latent space by projecting it into a 2D plot with TSNE [166]. 100 samples are shown for each digit. As seen, samples from the same digit class are grouped together.

Then, a SVM one-class model was used to define the concept region by fitting the distribution of digit class “1” in an unsupervised learning setting. An SVM parameter, ν , which sets an upper bound on the fraction of samples classified as outliers, was specified to a minimal value of 0.01 to fit samples in the “1” class as many as possible. The right plot in Figure 4.16 shows the sorted SVM scores obtained by applying the learned SVM model on each digit class. The obtained one-class SVM model has a decision threshold set at 0.705. Based on the threshold, the number of in-class samples obtained was 4863 and there were only 59 samples from other classes, i.e. false positives. The digit “1” was selected because the SVM model learned with it contained the most in-class samples and the least false positives. These in-class samples were used to represent the so-called concept region. As seen in the 2D projection plot, the cluster of digit “1” is also the farthest away from other clusters.

The learned latent space along with the SVM model are transferred to be used in the MINION’s training. Figure 4.17 shows the training data including the original one wafermap and some augmented samples used to train the MINION model. The training essentially learns an encoder to encode the *anchor* image into the latent vector with the highest SVM score in the concept region. Denote this latent vector as t_1 . Note that the anchor used is a special wafermap transformed from the original wafermap by extracting the top *large connected components* (LCCs) from the original wafermap based on a heuristic. A connected component (CC) is defined based on the 8-neighbor

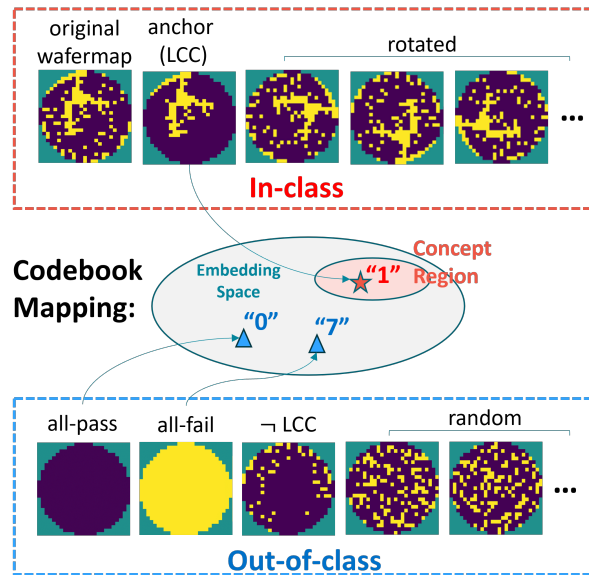


Figure 4.17: Codebook mapping in MINION training with augmented training data.

connectivity in image processing, i.e. each failing die (pixel with value 1) in a CC either shares an edge or a vertex with at least one other failing die in the same CC. The heuristic first extracts up to top 10 largest CCs where each contains at least 8 failing dies. The number 8 is the lower bound on the size of CC and is set to fit the WM-811K dataset so that most of the wafermaps with a “pattern” are covered. In practice, this number can be increased to 12 or 16 for a more reasonable threshold. Through experiments, we found that most of the wafermaps under our study contained a single LCC after applying the heuristic. Only few contained two or more LCCs.

Note that the LCC-based approach has its limitation. For example, for a grid-like pattern the approach would not work because a grid-like pattern contains failing dies that are not neighbors to each other (in any direction). Therefore, for special cases like this, we need to implement a custom script, on top of LCC, to take care of them.

In addition to the anchor, two types of generic training samples were created artificially. The first was referred to as the *in-class* training samples, which include the origi-

nal wafermap and additional wafermaps generated by transforming the original wafermap with data augmentation techniques such as rotation. Note that the anchor wafermap was also treated as in-class. The second was referred to as the *out-of-class* training samples, comprising of an “all-pass” wafermap, a “all-fail” wafermap, a special wafermap which contains failing dies from the original wafermap excluding the ones in the anchor (LCC), denoted as “¬ LCC”, and wafermaps randomly generated with the count of failing dies comparable to that of the original wafermap.

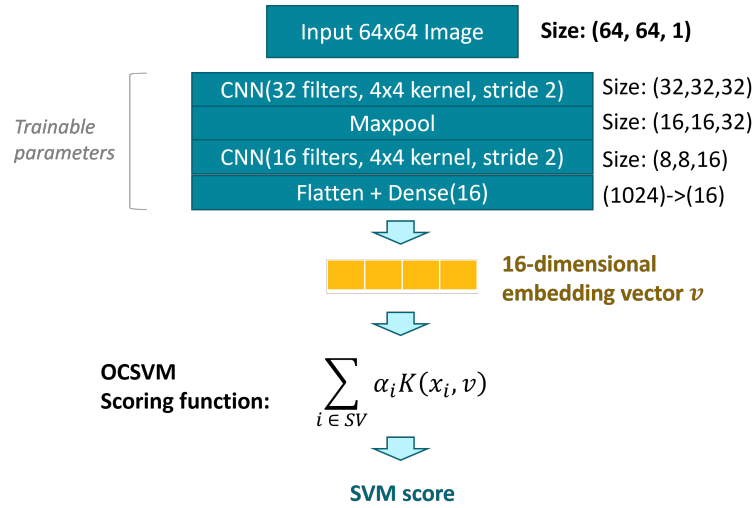


Figure 4.18: The neural network architecture of a MINION model.

All training data are resized to 64x64 images (with continuous values in the range $[-1, 1]$) and serve as input to a MINION model. Figure 4.18 shows the neural network layers used in the MINION model. The MINION model functions similarly to the encoder part of an InfoVAE model where the input image is encoded by two CNN layers along with a Max Pooling layer, and then transformed into a 16-dimensional *embedding vector* v by a dense layer. In addition, the SVM score of v with respect to the latent space can be calculated by applying the scoring function of the one-class SVM model. In the scoring function, K is the rbf kernel, x_i is a support vector, α_i is the coefficient of support vectors [38].

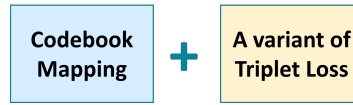


Figure 4.19: The two parts in loss function for training a MINION model.

The high-level training objective aims at aligning the output embedding vectors based on the concept region in the latent space. Specifically, we desire the embedding vectors labeled as in-class to align with the concept region as closely as possible, while the embedding vectors labeled as out-of-class should be positioned away from the concept region as farthest as possible. To achieve this objective, the loss function used in the MINION training contains two parts as shown in Figure 4.19.

The first part is the codebook mapping as illustrated in Figure 4.17. The detail is described in equation 4.1 where $f()$ represents applying the MINION model to get an embedding vector from an image, e.g. the anchor, all-pass, or all-fail wafermap. The codebook loss is described as three Euclidean distances between an output embedding vector and a predefined target latent vector in the latent space learned with MNIST data. As mentioned before, t_1 is the target latent vector with the highest SVM score in digit “1” class. We chose the latent vectors with the median SVM score in digit “0” class and with the median SVM score in digit “7” class as the target latent vectors for mapping the out-of-class samples, all-pass and all-fail, respectively. Denote these two target vectors as t_0 and t_7 , respectively. The reason is that the distributions of class “0” and “7” are the two farthest from class “1”, i.e. the concept region. Minimizing the codebook loss forces the model to generate the three embedding vectors that align closely with their corresponding target vectors.

$$\mathcal{L}_{codebook} = \|f(anchor) - t_1\|^2 + \|f(allPass) - t_0\|^2 + \|f(allFail) - t_7\|^2 \quad (4.1)$$

The second part of the loss function is a variant of the Triplet Loss [162]. Equation 4.2 describes the implemented triplet loss where $\phi()$ represents applying the SVM scoring function on an output embedding vector corresponding to an image from the in-class or out-of-class samples. T_{in} is the in-class threshold, which is a constant set to allow a slight margin from the target vector of the anchor, i.e. $T_{in} = \phi(t_1) - 0.05$. T_{out} is the out-of-class threshold, calculated as $T_{out} = \phi(t_0) + 0.2$. Minimizing the triplet loss penalizes the in-class samples that fall below T_{in} and the out-of-class samples that exceed T_{out} . The minimum value of the triplet loss is zero.

$$\mathcal{L}_{triplet} = \max(T_{in} - \phi(inClass), 0) + \max(\phi(outOfClass) - T_{out}, 0) \quad (4.2)$$

The final loss is the sum of $\mathcal{L}_{codebook}$ and $\mathcal{L}_{triplet}$. During training, the standard mini-batch stochastic optimization is employed to minimize the loss. Specifically, an Adam optimizer is used with a learning rate of 0.001 and batch size of 10. Model selection is based on selecting the model that obtains a loss value no more than 0.02 and has the lowest loss value among all models trained over a total of 300 epochs.

To apply a trained MINION model during inference, given a wafermap, its SVM score is calculated as shown in Figure 4.18 and compared with the SVM decision threshold. Wafermaps with a score greater than the SVM threshold are classified as in-class and therefore recognized. Otherwise, they are deemed unrecognized. Since MINION follows one-shot learning, the recognition of a wafermap is based on the single training sample used to train the MINION model.

In addition to the hyperparameter selections typically involved in standard neural network training, MINION's training includes consideration of several other important parameter choices. For example, the number of augmented in-class and out-of-class samples is a user-specified parameter. The image transformation of a wafermap can involve

more complicated settings than simple rotation. The random wafermap generation can be based on a fixed or a range of yield values. The heuristic used to extract the LCC wafermap involves several parameters such as the size of the CC and the number of CCs to retain in a wafermap etc. The parameters used in our MINION training were determined empirically through extensive experiments, based on performance observed on the public WM-811K dataset and private test datasets from three production lines.

4.8 Experiment Results

In this section, we use two experiment results to demonstrate the benefits of our MINIONs approach. The MINIONs approach has been implemented as part of our IEA-Plot AI Assistant [35]. More results will be shown with the IEA-Plot in Chapter 6. The two results shown below were based on the WM-811K dataset where 306 lots of wafermaps (both labeled and unlabeled) were used in the experiments by selecting one particular die count (518, represented in WM-811K as the expected number of dies on a given wafer) and one particular wafer shape (the 27×25 shape in the WM-811K dataset).

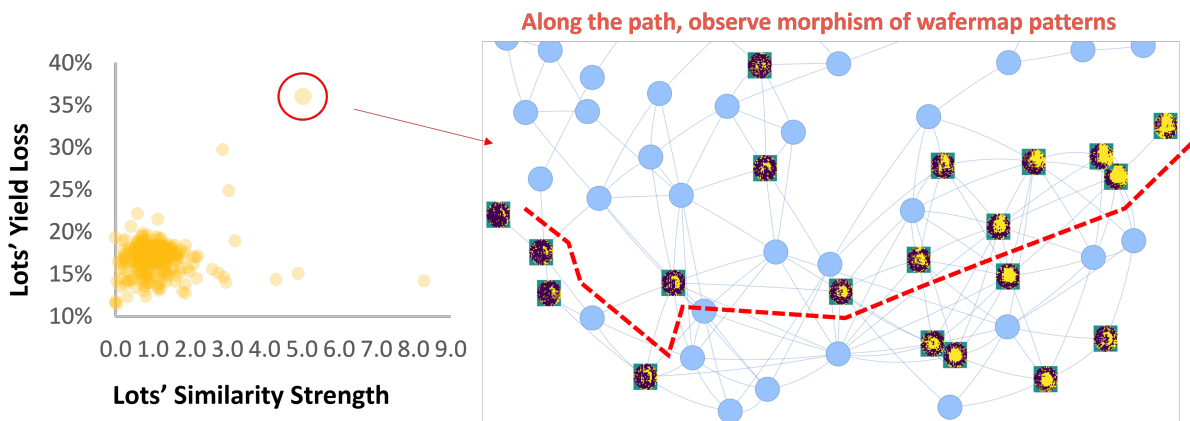


Figure 4.20: An example of finding a problematic lot

Earlier in Section 4.6.3, we define the concept of group similarity strength. The left of

Figure 4.20 shows a plot where each dot is a lot. Every lot is positioned with its similarity strength (the group is the set of wafers in the lot) and average yield loss. A large similarity strength means the lot contains a more systematic pattern. There is one lot circled in the plot, which has a large strength and high yield loss. The wafermaps from the lot are shown on the right of Figure 4.20, as part of the MINIONs recognition graph. We can see that the wafermaps exhibit a clear systematic trend. There is a “path” highlighted on the graph. Along the path we can observe morphism of wafermap patterns. The morphism path shows that those wafermap patterns are related, even though they do not all look the same. And because all these wafers appear in the same lot, it is likely that those wafermap patterns are caused by the same underlying issue in the manufacturing process. With the MINIONs graph, we can clearly observe a systematic trend on a group of wafers even though they contain “different patterns”. This is one of the capabilities which is challenging if implemented with a traditional multi-class classification approach.

Earlier in Section 4.6.4, one of the questions is: “Which lots contain wafermaps similar to this *w_p*?”. Figure 4.21 shows the answer for a particular wafermap “lot45254-23”. The figure shows one lot which has the highest wafer-to-group similarity strength as defined in Section 4.6.3 above. The wafermaps from this lot are shown as part of the MINIONs graph. This lot happens to be the lot shown in Figure 4.20 above.

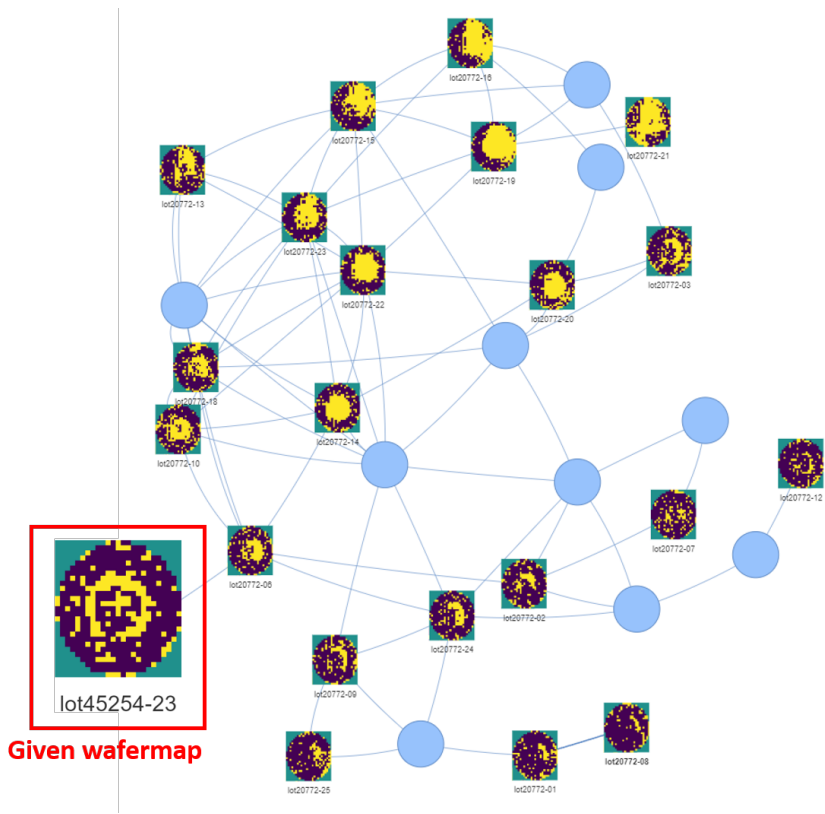


Figure 4.21: An example of searching for a similar lot based on a given wafermap

Chapter 5

Problem-Solution Dual View

因有故成無，因無故成有； 微塵分析事，不起色分別。

Because there is, there is not; Because there is not, there is; Analyzing everything to the end, There is nothing to discern.

— 《楞伽經》 Lankāvatāra Sūtra

In the previous chapter, we see that having a DSML oracle is fundamental to enable solving DSML problems such as wafermap analytics. A DSML oracle is built upon a decision procedure and is for solving a decision problem, i.e. with a binary answer. A DSML problem is considered always as solving a decision problem at its core. We can say that from the description perspective, each concept corresponds to a DSML oracle. Then, to solve a problem based on a problem description involving a number of concepts, essentially is to apply the corresponding DSML oracles. A DSML oracle can be built upon other DSML oracles at the lower level, corresponding to that a concept can be built upon other more primitive concepts.

In this section, we consolidate all these ideas in DSML, which have been gradually developed throughout the previous chapters, into a fundamental thinking called *Problem*

and *Solution Dual View* (referred to as “Dual View” in the following sections). From there, we will elaborate on the motivation behind language-driven analytics and present its initial implementation in 2022.

5.1 Dual View of DSML

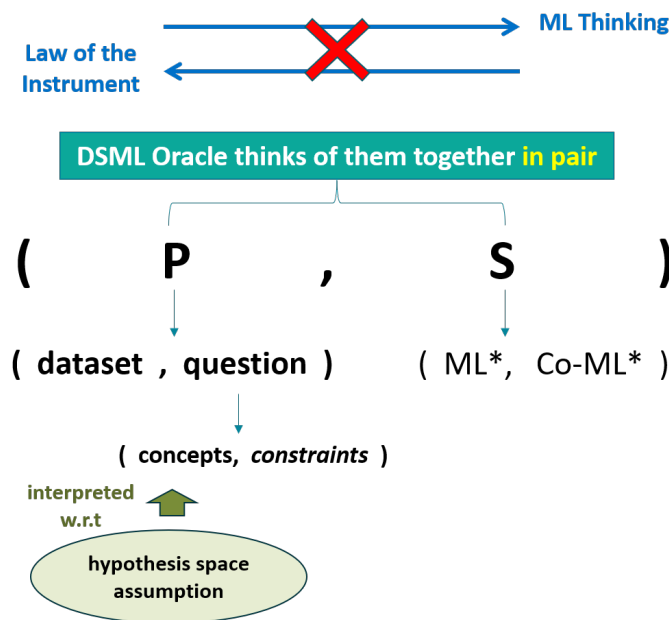


Figure 5.1: Problem and solution dual view in DSML

Figure 5.1 summarizes the Dual View seen when designing our DSML oracles. At the high level, the Dual View means that the Problem (P) and Solution (S) have to be considered together as a pair when solving a problem for domain-specific analytics.

In traditional ML, one starts with a problem formulation defined with a benchmark dataset. The problem is assumed well-defined. The focus is on solving the problem, i.e. obtaining an optimized learning model. In most of the ML applications today in the semiconductor industry, one goes in the reverse direction. Given a ML tool, a ML model, or a ML technology, one tries to apply the tool, the model or the technology

by formulating a given description of the analytic problem to *fit* the tool, the model or the technology. In other words, all those practices might be seen as following the *law-of-the-instrument*¹ An example of the *law-of-the-instrument* practice is described in the previous chapter, where all other researches deal with wafermap analytics by formulating it as a multi-class classification problem (because of the available benchmark WM-811K). As explained before, solving a multi-class classification problem does not address the practical needs appropriately, leaving a gap between the solution provided and the actual problem people care about solving in the first place.

It should be noted that law-of-the-instrument practices are visible everywhere in the semiconductor industry. Within a company, we can see the outcome of “applying ML” as a collection of so-called “ML tools”. Each tool generates a specific type of plot for visualization. There can be a large number of such tools made available to engineers. In reality, an engineer can see a large number of “interesting plots” but often in order to obtain an evidence to convince others to take an action, they need to do their own scripts and conduct the analytics manually. The underlying issue behind this phenomenon is that the tool providers make tools by assuming every problem in practice is like a ML problem but in reality, they are not — their are DSML problem which is fundamentally different from ML problem, as explained extensively in this thesis.

Therefore, our Dual View intends to emphasize the importance that defining the Problem is an integral part for solving an analytic problem. In Figure 5.1, we emphasize that between P and S, one should not think in either direction (P to S or S to P). Rather, *think P and S as a pair*. One of the reasons to justify this thinking is that an analytic problem is usually described by a domain expert, using words without a precise definition. We have seen in the previous chapter that a person might be interested in

¹It is a concept due Abraham Harold Maslow, an American psychologist who created the famous *Maslow's hierarchy of needs*, and it can be stated as the following: “If the only tool you have is a hammer, it is tempting to treat everything as if it were a nail.”

seeing a “systematic pattern”. Defining the meaning of this term is an integral part of the analytic problem.

While wafermap analytics described in the previous chapter exemplifies our Dual View thinking, if we go back to other application contexts reviewed earlier we can see that the Dual View also applies. For example, from the very beginning when we describe the yield optimization example in the introductory chapter in Section 1.2, we emphasize that key to success was due to taking a correct perspective for formulating the problem. The essence of the Choose-and-Bound search lies in searching for the problem formulation that’s most likely to render the satisfactory answer.

Then, DSML was formalized throughout Chapter 2 and Chapter 3. In equation 3.1, the “IEA + Human Knowledge” includes the domain knowledge used to provide problem formulation. Under the dual view, Equation 3.1 can be stated as the following:

```
DSML includes a DSML oracle, called by IEA, to formulate and solve
a decision problem with ML* and Co-ML* capabilities.
```

Although IEA provides precise definitions to interpret concepts, each interpreted with a DSML oracle, it is human who decides a specific problem formulation based on selected concepts and gives the problem to the IEA to solve. Therefore, it is not hard to see that IEA as a system, provide a *language* for its user to specify a problem to solve.

Specifically, the P in Figure 5.1 at the highest level, comprises a pair of inputs: “(dataset, question)”. In the Dual View, this entry-level P is formulated as an analytic question along with a dataset intended for the analysis. For example, in yield optimization (Section 1.1.1), the question can be: “Do you see a high correlation between a failure type and an E-test parameter?” and the question can be asked on a particular dataset. In that example, if the dataset contains the X_4 failure type, the answer would be “Yes”. If not, the answer should be “No”. The concept is “high correlation” and there should

be a DSML oracle to decide what that means. In the yield optimization example, most of the Choose-and-Bound search steps ask the same question on different datasets.

As mentioned in Section 4.5.2, in wafermap analytics, a DSML oracle takes a “(dataset, question)” pair as input and solves a decision problem, e.g. deciding if the given dataset contains some pattern class. In contrast to pre-defining the pattern classes with a general dataset as in traditional ML view, the pattern class definition is dynamically derived based on a local question and its associated dataset. One primary reason for this distinction is due to the data wall (see Figure 2.22). Section 2.7 illustrated two realistic constraints, namely the data wall and the decision chain, which underscore the importance of searching for a dataset across the three dimensions within the Choose-and-Bound search space in DSML. The fundamental thinking behind this perspective also traces back to the Dual View.

Figure 5.1 further shows that the “question” can be broken down into a pair of “(concepts, constraints)”. We consider *constraints* as those used to select samples from the dataset, i.e. they are *data point selectors*. For example, “yield $\geq 0.5\%$ ” is a constraint. “The last three months” is a constraint. “Failing tests in test bin X” is a constraint. On the other hand, “showing a donut pattern” is not a constraint even though the result is also a selected subset of wafermaps. This is because this phrase involves the concept “donut” that requires an interpretation through a DSML oracle. In other words, while both concepts and constraints both may result in selecting a subset of samples, a constraint does not involve a term that requires a DSML oracle to interpret its meaning. The meaning of words or terms used to describe a constraint is well defined, or at least universally agreeable in the given application context.

As mentioned in Section 4.6, the question used to query the DSML oracle can be stated as asking whether a concept exists in the given dataset, e.g. a correlation, an outlier, or a pattern etc. While an IEA system with DSML oracles can automatically

establish the concept definition, a user-imposed input constraint can enforce the concept to be applied on a selected scope of the provided dataset.

Note that while a DSML oracle always deals with a decision problem, not every decision problem in IEA should be seen as solving a DSML problem. We refer to DSML problems as those follow the formulation in Equation 3.1 where ML^* and $Co-ML^*$ are involved. For example, making decision based on a simple threshold is not considered as solving a DSML problem. However, if the threshold is ensured consistent based on the data, i.e. to decide if a consistent threshold exists or not, then it is solving a DSML problem.

Referring back to Figure 3.18 and 3.19, the learning in DSML can be formally stated as searching for the hypothesis space assumption that can justify a just-fitting hypothesis (answer). From the “question” perspective, this translates to searching for hypothesis space assumption with respect to which the concept definition can be interpreted. This aspect is further shown in Figure 5.1.

To determine the existence of a concept, a DSML oracle essentially functions as a concept recognizer in practice. In the context of wafermap analytics, various concepts can be defined based on the similarity relationship in the so-called MINIONs graph. Thus, the corresponding “(ML^* , $Co-ML^*$)” capabilities are implemented in terms of the various graph operations.

The dual view taken by our IEA design differs significantly from both ML perspective and common “applying ML” perspective. Interestingly, the Dual View is not just at the high level. The view is recursively applicable to all levels in our IEA design where at each level of problem solving, the Dual View manifests as a pair of entities. The Dual View is the fundamental thinking in IEA, evident throughout the entire IEA system, from the basic level of concept recognition to the outermost layer of language-driven analytics.

5.2 Language-Driven Analytics

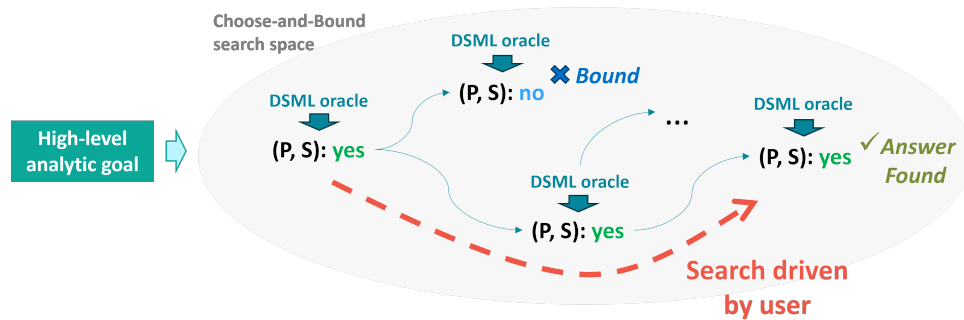


Figure 5.2: Search flow driven by user inputs to achieve a high-level analytic goal

In DSML, the Choose-and-Bound search space can be viewed as a space of (P, S) pairs, where a decision answer is determined for each pair by a DSML oracle. Figure 5.2 illustrates the search space. This is also the detailed view of the space of problem formulations as depicted in Figure 1.10 earlier. Given a high-level analytic goal, such as solving a yield issue, the search moves from one (P, S) to another sequentially until an acceptable answer is found. While the Co-ML capability helps bound unnecessary search branches, it is ultimately the user who drives the overall flow of the search process.

At the search flow level, the user determines the next step to investigate by posing an analytic question. For example, in yield optimization, one might start with analyzing the yield plot, followed by multiple correlation analysis between various test categories. The DSML oracle formulates the problem by interpreting the analytic question and provides the decision answer. To support the user-driven search, it is natural to take in user inputs by enabling them to specify their input in natural language.

As mentioned in Section 4.6.3, various concepts can be defined, each described as an English term. Multiple terms can be used to describe a higher-level concept. Since an analytic question involves concepts, it is natural to enable description of concepts and questions with natural language processing (NLP) techniques. Hence, we refer to the

theme of using NLP to facilitate user-driven search flow as the language-driven analytics.

5.2.1 Language-driven is a necessity

The adoption of NLP techniques in IEA originated in 2018 (IEA 2018), as mentioned in Section 3.10.1. At the time, the NLP interface was only used for facilitating information retrieval, thus it was not considered a necessity; however, in view the redesigned IEA in 2022 (IEA-2022) [96][111], we argue that language-driven analytics is now indispensable for the following reasons.

In Section 3.10.2, we discussed that IEA 2018 attempted to model “all domain knowledge” as a comprehensive analytic workflow. If realized, a user could directly query the high-level analytic goal, with the IEA system generating the results internally, and NLP interface used to retrieve the analytic results, such as those stored in a PowerPoint presentation. However, after three years of efforts trying to commercialize IEA within a partner company, by 2022, we concluded that it is impractical to assume IEA could have all the knowledge pre-stored in its system. The most important reason for reaching the above conclusion goes back to the Dual View.

In view of Figure 5.2, the scope of domain knowledge can be seen as enumerating the (P,S) pairs in the search space. IEA 2018 attempted to exhaustively incorporate all problem formulations and their solutions in a rule-based autonomous system and automatically conducted the search flow. Earlier in Section 1.2, we mention that this exhaustive enumeration is infeasible. This is because the search space can be enormous and complex, and engineers in the field usually have diverse ideas for conducting a search, i.e. how to explore the space or what part of the space is more important than others. We hope this point has become intuitive, as we dedicated a substantial portion of the first four chapters detailing various problem contexts in design and test.

The complexity of the search can be explained again from perspective of the Dual View. While a practitioner has a set of tools available to use, there is usually no well-defined problem formulation to begin with when dealing with a practical problem. For example, a practical problem can be stated as: “Do you see a yield issue this week?” followed by “If you see a yield issue, provide an explanation why it happens.” Consequently, there might be a follow-up problem: “Do we need to call TSMC?” As seen, none of these practical problems are well defined to begin with.

Due to the data wall, defining the problem with a benchmark dataset is usually not a feasible option. As a result, one can invent their own ideas to interpret the original problem statement and come out with diverse problem formulations to solve the original problem, leading to a variety of different search flows. To achieve an AI Assistant that can effectively assist a practitioner, the ability to articulate vaguely defined problem statements must be inherent in its intelligence. Essentially, the communication between an AI Assistant and its user should be *standardized* with a set of well-defined terms for the Assistant to know precisely what the user wants and provide the results that meet the user’s need. It is for this articulation that NLP becomes a necessary component in IEA-2022.

For the sake of argument, one can say that NLP is not necessary and can be replaced by defining a new formal language, say L_{IEA}^* , like a new type of scripting language to interact with IEA. This might be true. However, this means that a user needs to learn the new language L_{IEA}^* in order to use IEA. If the IEA can allow its user to use English, it is questionable why in our IEA design, we desire to take away that option and ask a user to learn a new formal language L_{IEA}^* . To a certain extent, the L_{IEA}^* counter argument to our claim that NLP is necessary, is almost like saying that English interface is not necessary in IEA because a Chinese (a different language) interface is possible. At any rate, when we say that NLP is a necessity, we are emphasizing the “language process” aspect more

than the “natural” aspect. This is why we call our approach Language-Driven Analytics, instead of “Natural Language Driven Analytics”.

5.2.2 IEA redesigned in 2022

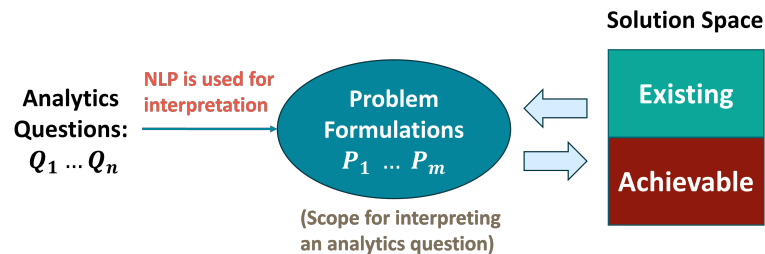


Figure 5.3: NLP is a necessary component for interpreting the analytic question

Starting from 2022, the IEA design leaves the “knowledge to drive the search” out of the system and lets user dictate the search flow. In this way, the NLP is used to interpret the analytic questions as specified by the user in each search step. Figure 5.3 illustrates the usage of NLP in the IEA-2022 design. The interpretation is based on domain knowledge and the intelligence for achieving this interpretation is retained in the IEA system. Although exhaustive enumeration of the scope for interpretation is infeasible, with NLP in place, modeling this domain knowledge can be accomplished through grammatical modeling. Just as in the English language, although enumeration of all possible sentences is infinite, we can define a English grammar to capture what might possibly form a proper sentence within the scope of the language.

In view of Figure 5.3, the scope for interpreting the questions is a set of problem formulations, such that they are sufficient for answering each of the analytic questions. Note that the mapping from analytic question to problem formulation is not necessarily one-to-one. When defining this problem space, we need to simultaneously consider the solution space, which contains existing solutions and those achievable with the state-of-

the-art technologies. From the solution to the problem, we need to formulate solvable problems constrained by the existing solutions. From the problem to the solution, we need to implement new solution achievable by current ML technologies. This is again seen with our Dual View, where it is this entanglement of dealing with both P and S simultaneously that poses the challenges in developing AI solutions in design and test.

5.3 IEA 2022

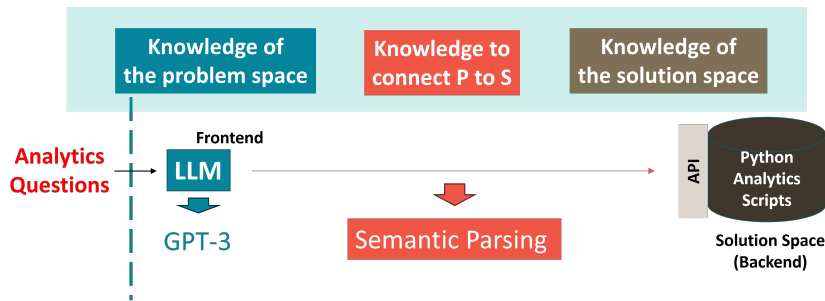


Figure 5.4: Overview of IEA 2022

The first implementation of the IEA based on the language-driven analytic approach was realized in 2022 (IEA-2022) [96][111]. Figure 5.4 shows the fundamental design thinking of IEA 2022 as incorporating three types of domain knowledge. With the dual view, IEA needs to have the knowledge of the problem space (P) for interpreting the analytics questions and the knowledge of solution space (S) such as the DSML oracles. Then, in-between, IEA needs the knowledge to connect the P to S. Figure 5.4 points out that the connection is realized by a technique called *semantic parsing*. The semantic problem existed because we used LLM as the frontend and the backend comprised an API of analytics software scripts. To connect the two, we formulated the connection problem as solving a semantic parsing problem. With the MINION’s approach as the backend solution for wafermap failure analytics, in the following sections, we describe how IEA

2022 was implemented to assist yield engineers in accomplishing the jobs described in Section 4.1.

5.4 Connecting LLM via Semantic Parsing

Semantic parsing is the process to assign real-world meanings to linguistic inputs [167] (e.g. words). Specifically, in *computational semantics*, formal structures called *meaning representations* are used to link the non-linguistic knowledge (e.g. data stored in database, API function calls, etc.) to linguistic elements such as English words. Semantic parsing is a wide field of study. In machine learning, most works to achieve semantic parsing centered on using supervised learning with large amounts of human-created semantic parses [167][168]. Such an approach is hard to be duplicated in a specialized domain like ours, because we lack the resources to create large amounts of training data.

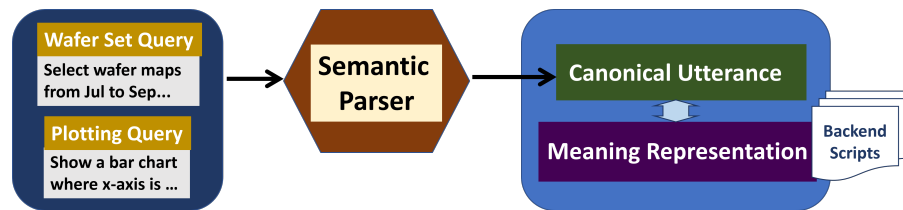


Figure 5.5: A semantic parser interfacing with natural language queries

Consequently, we do not take a supervised learning approach to implement our semantic parser. Instead, we adopt the approach called *semantic parsing via paraphrasing* [169]. The approach makes use of triples (*natural query* q , *canonical utterance* c , *meaning representation* m), where the parser maps $q \rightarrow c \rightarrow m$. It was observed in [169] that mapping $c \rightarrow m$ and vice-versa could be achieved by fixed rules, which make it more feasible in our application context.

Figure 5.5 depicts the idea in our context. IEA-2022 supports two types of queries²:

²Throughout the history of IEA development, we call the inputs to IEA with various names: “queries”,

wafer set query and plotting query. Then, the semantic parser translates queries into *canonical utterances* which are based on a restricted language whose *lexicon* is custom-defined by us. These utterances are interpreted with a *meaning representation* which defines the corresponding actions to be performed by the backend analytics component. In this flow, the “ $c \rightarrow m$ ” mapping are implemented with fixed rules. The “ $q \rightarrow c$ ” mapping are handled by the parser.

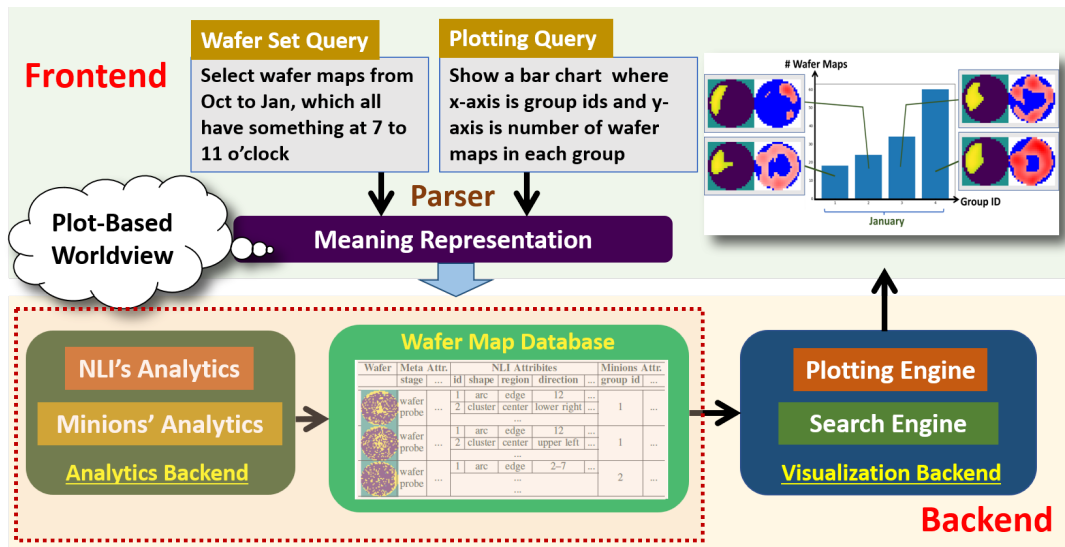





Figure 5.6: The workflow from queries to a summary plot

Realizing Figure 5.5 requires us to define a feasible meaning representation as the target for the parser. We can start by developing a *grammar* that defines the lexicon of the canonical utterances. After the lexicon and the meaning representation are defined, we can choose a way to implement the parser. In IEA-2022, the parser is implemented with *constrained semantic parsing* powered by a pretrained language model [170]. The subsequent Section 5.4.2 introduces the grammar and meaning representation and Section 5.4.3 introduces the visualization component. Overall, they are simply used to refer to the inputs to IEA. In one context, the word “query” may be more understandable and in another, the word “instruction” may be more appropriate. When we describe the DSML oracle before, we use the word “question” “problem”, or “problem statement” to refer to the inputs to an AI Assistant in a DSML application context.

Table 5.1: A snapshot of the wafermap database table

Wafer	Meta Attr.		NLI Attributes					Minions Attr.	
	stage	...	id	shape	region	direction	...	group id	...
	wafer probe	...	1	arc	edge	12	...	1	...
			2	cluster	center	lower right	...		
			...						
	wafer probe	...	1	arc	edge	12	...	1	...
			2	cluster	center	upper left	...		
			...						
	wafer probe	...	1	arc	edge	2-7	...	2	...
			...						
			...						

tion 5.4.3 discusses the detail of constrained semantic parsing.

Figure 5.6 shows the workflow from user queries to a summary plot [111]. When parsing a wafer set query, the corresponding meaning representation contains instructions in the backend for retrieving a set of wafers from the wafermap database. The wafermap database is developed in work [96] based on the MINION module and the Natural Language Interpreter (NLI) module and will be elaborated in Section 5.5.

Table 5.1 shows a snapshot of a wafermap database table. Essentially, wafermaps are each associated with one or more pattern descriptions (each comprising multiple NLI attributes, e.g. shape, region etc.), then grouped based on similarities and pre-stored in the database. For a wafermap pattern, the concatenation of its NLI attributes forms a canonical utterance describing the pattern concept. For example, a query might request to retrieve all wafer maps from wafer probe that have “arc along edge at 12 o’clock”. The first two wafer maps in Table 5.1 will be included in the return. The “group id” attribute is provided by the MINION’s approach [14]. The wafer maps annotated by the same group id generally exhibit a very similar pattern [96]. If the query requests “group 1”, the result will also include the first two wafer maps.

The software scripts executing the wafermap database search are implemented in the

search engine module. In addition, there are instructions for generating a certain type of plot and assigning plot attributes based on the retrieved data. The corresponding software scripts are implemented in the plotting engine module.

5.4.1 Example parsing and result

The work [111] considers an application context where in addition to knowing that a set of wafermaps all exhibiting some kind of pattern (as described in Chapter 4), a practitioner might also desire to know if some action can be taken based on the pattern. For example, the practitioner would like to query if there is some correlation to the final test result.

Consider what type of summary plot might be used to answer the request, Table 5.2 below shows an example. The example includes the wafer set query and the plotting query, their parsed **canonical utterances**, and the **meaning representations**, used for generating the result plot. Note that the canonical utterances are essentially paraphrases of the input queries with their wordings restricted by a pre-defined grammar (see Section 5.4.2). A meaning representation contains instructions, each corresponding to a step that calls a function in the backend API and returns some result.

It should be noted that the utterance “something left along edge” describes a certain type of wafer map pattern the query is looking for. The set of wafer maps with such a pattern will be retrieved from the wafermap database. Specifically, a pattern description such as “something left along edge”, once parsed into its canonical utterance, will match to a *group* of corresponding wafermaps by directly querying the wafermap database [96].

In the generated plot, the wafer maps used in this example were from a recent production line. The wafers were stamped with days in four months (from October to January). The four months had 211, 1501, 1337, and 3251 wafers, respectively.

Table 5.2: Example queries, their parses and the generated plot

Wafer Set Query
<p>Select wafer maps from wafer probe from bin \mathcal{Z} with component yield loss greater than or equal to 5% from Oct to Jan, where all wafer maps exhibit something <i>left</i> along the edge</p>
<p>((((wafer maps) from wafer probe) from bin \mathcal{Z}) with component yield loss greater than or equal to 5%) from Oct to Jan) which have something left along edge</p>
<p>A.1. Return wafer maps A.2. Return A.1 from wafer probe A.3. Return A.2 from bin \mathcal{Z} A.4. Return A.3 with component yield loss greater than or equal to 5% A.5. Return A.4 from Oct to Jan A.6. Return A.5 which have something left along edge</p>
Plotting Query
<p>Show a bar chart where x-axis is group ids, y-axis is the number of wafers in each group. For each bar, also show a sub plot of two wafer maps where the left is wafer probe heatmap, the right is final test heatmap</p>
<p>(((bar chart) where x-axis is (group id of wafer maps)) where y-axis is (the number of wafer maps for each (group id of wafer maps))) with ((wafermap subplot for each bar) where left figure is wafer probe heatmap) right figure is final test heatmap)</p>
<p>B.1. Return bar chart B.2. Return group id of A.1 B.3. Return the number of A.1 for each B.2 B.4. Return B.1 where x-axis is B.2 B.5. Return B.4 where y-axis is B.3 B.6. Return B.5 with wafer map subplot for each bar B.7. Return B.6 where left is wafer probe heatmap B.8. Return B.7 where right is final test heatmap</p>
Generated Plot (Same as in Figure 5.6)

In this example, the plotting query specifies the plot type as a bar chart. The bar chart shows that in January, IEA found four groups of wafers with *group id* as assigned in the wafermap database. The bar height shows the number of wafers in each group. For each group, two wafer images are shown. The left image is obtained by stacking wafer maps in the group where those wafer maps are based on wafer probe test. The right image is by stacking wafer maps where they are based on final test. Each stacked image shows a region including 60% of the density, i.e. the highlighted region including roughly 60% of the fails in the group.

The correlation between wafer probe and final test can be inspected in this summary plot. It can be observed that the failing pattern forms a “thick ring”. Fails from wafer probe concentrate on the left-side of the ring. Then, fails from final test tends to “complete” the ring. From the data, this observation occurred in November and affected more wafers in December.

5.4.2 Grammar Model

With the wafermap database, the semantic parser’s job is primarily centered on parsing the queries into canonical utterances that are related to data/concept retrieving and plotting operations. For this purpose, we define the following: (1) A grammar that captures data/concept retrieving and plotting operations in the backend. (2) From the grammar, we obtain the lexicon for defining the canonical utterance and meaning representation.

The grammar for describing the wafermap pattern concepts are developed in work [96] (see Section 5.5.3). For defining the grammar for data selection and plotting, we follow an approach similar to that used in [96]. We define a context-free grammar that describes the process for generating a plot. For example, the left-hand side of Table 5.3

Table 5.3: A snippet of the grammar and lexicon

Grammar Rules			Lexicon	
(i)	S	$\rightarrow Plot$	TYPEBAR	\rightarrow bar chart
(ii)	$Plot$	$\rightarrow Bar \mid Scatter$ $\mid Wafermap \mid \dots$	TITLE	\rightarrow title $\mid null$
(iii)	Bar	$\rightarrow Descr_{Bar}$ TYPEBAR	AXIS1	\rightarrow x-axis
(iv)	$Descr_{Bar}$	\rightarrow AXIS1 AXIS2 TITLE	AXIS2	\rightarrow y-axis
	

shows a snippet of the grammar, which consists of a set of production rules. Rule (i) starts the process to get a plot. Rule (ii) provides several plot options including *Bar*, *Scatter*, or *Wafermap* etc. In Rule (iii), each plot option defines the plot type and a descriptor for the specific type. For bar chart, the descriptor specifies three attributes: AXIS1, AXIS2, and TITLE.

Production rules which derive words as the terminal values are called the *lexicon*. A snippet of the lexicon is shown on the right-hand side of Table 5.3. The left part of a lexicon rule has a type or attribute that can be processed internally by the backend. For example, *TypeBar* corresponds to a script in the plotting engine to define a bar plot object. The lexicon rules provide the words that can be used in the utterances and in the meaning representation. For example, the plotting query in Figure 5.6 uses words “bar chart”, “x-axis”, and “y-axis”. Note that the lexicon related to wafer set queries is developed in the work [96]

The meaning representation uses words provided by the lexicon rules. In addition, it has words that express operations. For example, to generate the plot in Table 5.2, the query can use the word “select” to request a set of wafer maps from the database. Then, a separate query uses the word “show” to request a plot object. Table 5.4 shows a snippet of these so-called *operators*. For each operator, the natural language *template* used to express the operator is shown. The parameters denoted by “[]” are from the lexicon. The words outside are called *functional words* which indicate what functions to

Table 5.4: A snippet of operators and their expressions

Operator	Template	Meaning Representation
Select	Return [TypeData]	1. Return wafer maps
Filter	Return [ref] from [condition]	1. Return wafer maps 2. Return #1 from Jul to Sep
Aggregate	Return [aggregate] of [ref]	1. Return wafer maps 2. Return the number of #1
Show	Return [TypePlot]	1. Return bar chart
Group	Return [aggregate] [ref1] for each [ref2]	1. Return wafer maps 2. Return group ids of #1 3. Return the number of #1 for each #2
Specify	Return [ref1] where [plot attribute] is [ref2]	1. Return bar chart 2. Return wafer maps. 3. Return group ids of #2 4. Return #1 where x-axis is #3
	...	

be performed on the parameters. The corresponding meaning representation exemplifies these functions.

This meaning representation is called *Question Decomposition Meaning Representation* (QDMR) [171], chosen for our current implementation. QDMR contains natural utterances, which is easier to understand than complex logic forms used in [168]. Using QDMR facilitates data preparation for us to bootstrap a semantic parser.

In general, QDMR constitutes an ordered list of steps, each corresponds to an operator. In our case, each operator can be executed by some backend scripts. The steps all together accomplish the task specified in the user query. The lexicon defined by our grammar restricts the scope of meaning representations to those supported by the backend. A valid meaning representation should contain words only from the pre-defined lexicon, the functional words used in the operator template such as “for each”, and possibly a *reference token* that refers to the result of a previous step. In this way, we make sure that any step in the meaning representation can be mapped into a valid execution flow with the backend scripts. Moreover, some operators can be merged to a *high level*

QDMR [171] to reduce the total number of steps involved. The detailed definition of QDMR and more types of operators can be found in [171].

5.4.3 Constrained parsing with GPT-3

With the meaning representation defined, the semantic parser’s job is to parse a user query into its corresponding canonical utterance. We adopted the approach proposed in [170], illustrated in Figure 5.7. The method leverages a pretrained language model (LM) for parsing input queries into canonical utterances. With a LM, we can use *few-shot learning* to teach the LM about our specific utterances.

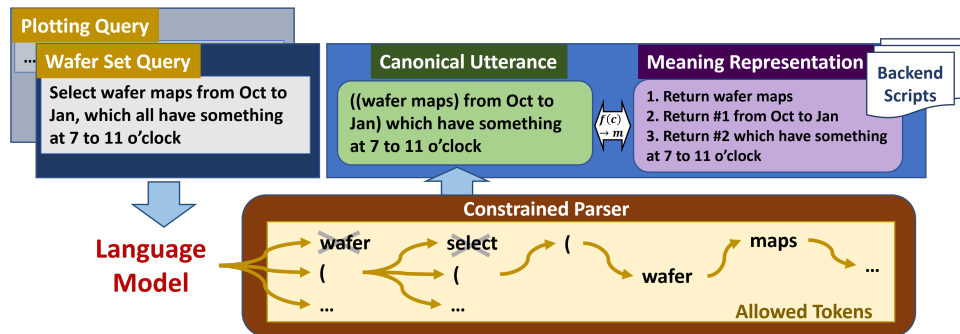


Figure 5.7: Constrained parsing from user query to meaning representations with LM

Following [170], we use the state-of-the-art language model GPT-3 [172] that has 175 billion parameters as the natural language interface. The GPT-3 handles the variation of wordings in the spoken language. Moreover, by utilizing GPT-3’s powerful *in-context learning*, we finetune the GPT-3 to learn the *translation* from the input natural query to its corresponding canonical utterance with only 20 demonstrating examples (i.e. few-shot learning). The translation is essentially a many-to-one mapping function that generates a single unique interpretation of various sayings.

As shown in Figure 5.7, the constrained parsing restricts the output space of GPT-3 by a set of *allowed tokens*, i.e. words that should be generated by GPT-3 based on the

input query. The allowed tokens include: (i) words or their inflections that appear in the query (words from the lexicon), (ii) a pre-defined set of functional words (words used in operator’s template), (iii) opening and closing parentheses.

Given a natural query as input, GPT-3 will search through all valid tokens that belong to one of these three categories and output the token that has the highest probability conditioned on the tokens already been generated. In addition, the parser will ensure any parentheses used in the output string are balanced. The generated string by GPT-3 becomes our canonical utterance for the query.

Then, it is straightforward to convert a canonical utterance into a meaning representation in QDMR format. For example, a converting function $f(c) \rightarrow m$ can be implemented by: for each utterance in the parentheses, replacing the return in inner parentheses with reference token, removing all the parentheses, and adding the word “return” to the front and new line to the end. Each meaning representation entails an executable flow in our backend to generate a plot.

5.4.4 More parsing and analytic results

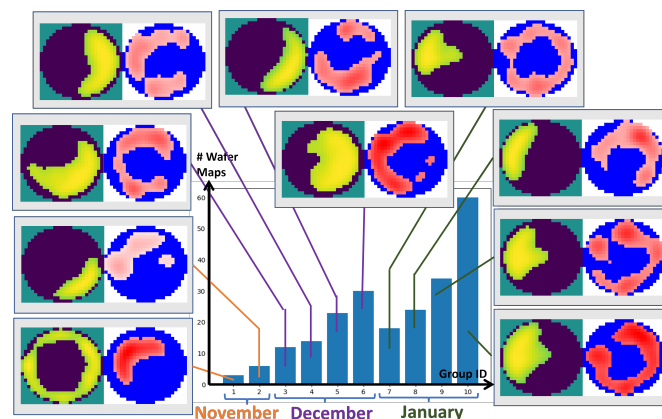


Figure 5.8: Merged summary plot of Table 5.2 and Table 5.5

In work [111], IEA 2022 with the constrained parsing approach was used to analyze

Table 5.5: Second result

Wafer Set Query
<p>Get wafer maps from bin \mathcal{Z} of wafer probe with yield greater than or equal to 5% from Oct to Jan, which all have something right along the edge</p> <p>(((((wafer maps) from wafer probe) from bin \mathcal{Z}) with component yield loss greater than or equal to 5%) from Oct to Jan) which have something right along edge</p>
Plotting Query
<p>Plot a bar chart such that x-axis is group ids, y-axis is the number of wafers in each group. Also show a sub plot of two wafer maps for each bar, where the left is wafer probe heatmap, the right is final test heatmap</p> <p>(((bar chart) where x-axis is (group id of wafer maps)) where y-axis is (the number of wafer maps for each (group id of wafer maps))) with (((wafermap subplot for each bar) where left figure is wafer probe heatmap) right figure is final test heatmap)</p>
Generated Plot

four months of wafer maps from a recent production line. This “analysis” was essentially a query-driven search process. Each time, we provided a query and inspected the resulting plot. In the process, we collected the plots that indicated us a potentially interesting finding.

Note that in addition to using all-fail, the work [111] conducted the search with wafermaps based on fails from individual *test bins* in the analytic backend, Hence, in our analytic backend, each wafer has multiple wafermaps defined, based on total fails and based on a set of test bins. In the search, all these wafer maps are analyzed together. For example, two wafermaps showing a “grid pattern” could be from two different test bins. Nevertheless, when a user requests a “grid pattern”, they both can be put into a

Table 5.6: Third result

Wafer Set Query

Fetch wafer maps from wafer probe with component yield loss greater than or equal to 5% from Oct to Jan, which all demonstrate a **grid** pattern

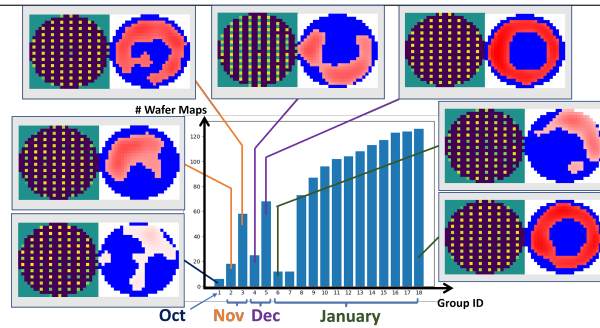
(((wafer maps) from wafer probe) with component yield loss greater than or equal to 5%) from Oct to Jan) which have grid

Plotting Query

Generate a bar chart where y-axis is the number of wafers in each group, x-axis is group ids. And near each bar, also draw a sub plot of two wafer maps the right is final test heatmap, the left is wafer probe heatmap.

(((bar chart) where x-axis is (group id of wafer maps)) where y-axis is (the number of wafer maps for each (group id of wafer maps))) with (((wafermap subplot for each bar) where left figure is wafer probe heatmap) right figure is final test heatmap)

Generated Plot



single set.

Continuing the result shown in Section 5.4.1, Table 5.5 shows a result that also indicates a “thick ring” fail pattern. Noticing those density maps from wafer probe data (left figure), the patterns are on the right side (in contrast to left side in figure in Table 5.2). However, when we look at both from wafer probe and final test data, we can see, together all pairs tend to form a “thick ring” pattern.

Then, we combine the two plots with another finding which has only 1 wafer group, to produce a final summary plot shown in Figure 5.8. The figure shows that the “thick ring” fail pattern affected many wafers at both wafer probe and final test and lasted from November to January.

Table 5.6 shows another interesting finding we discovered on the dataset, by searching

for a “grid pattern”. The plot shows that many wafers were affected (with this pattern) at wafer probe stage, lasting from October to January. However, there is no apparent correlation to the final test. This indicates that the issue might be with the tester, rather than with the manufacturing process.

5.5 Implementation of Backend API

In Figure 5.6, the analytics backend comprises two components: the MINION’s analytic module and the Natural Language interpreter (NLI) analytic module. The implementation of these two modules in IEA 2022 was described in work [96]. Behind the MINION’s analytic module is the MINION approach detailed in Chapter 4. As explained in the following, the NLI analytic module further attaches natural language descriptions to the groups of patterns to enable language interactions with the wafermap data.

5.5.1 Describable set based on MINION’s graph

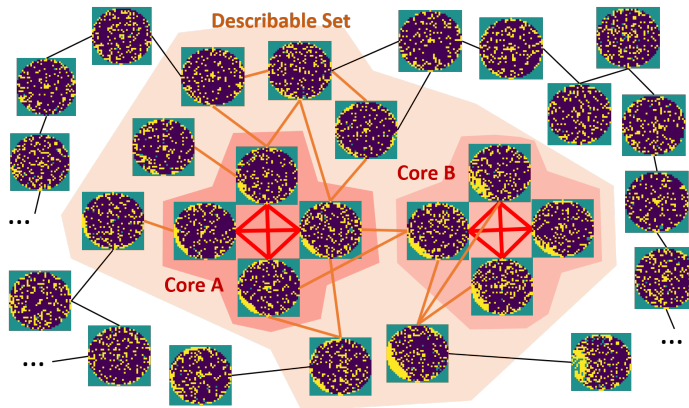


Figure 5.9: An example of Minions’ recognition graph on wafer maps

Groups of wafermap patterns can be defined on a MINION graph (see Section 4.7 for detail). The work [96] was based on a simple graph operation of finding maximal cliques.

Figure 5.9 highlighted two maximal cliques on the corresponding MINION graph. In work [96], a maximal clique is called a *cluster core*. The two cliques are shown as “Core A” and “Core B”. Notice that within each clique, the wafer maps look very similar.

In the backend of IEA 2022, a cluster core is treated as a *primitive pattern*. A pattern class can be specified and extended from a primitive pattern. Each extension is through a language interpretation based on one primitive pattern, and the interpretation result is captured in a *describable set*.

For example, given a description: “more fails spread from 6 o’clock to 9 o’clock along edge”, suppose the NLI finds Core A in Figure 5.9 satisfying this description, and then extends from the core to find all other wafer maps also satisfying this description. The *Describable Set* is highlighted in the figure, which includes Core B and 7 other wafer maps. These 15 wafer maps are those shown in Figure 5.9 before.

In our analytics backend, a describable set corresponds to a primitive pattern and satisfies three conditions: (1) It includes the maximal clique of the primitive pattern; (2) Within itself, all wafers are connected; (3) It is describable through our *natural language interpreter* (NLI).

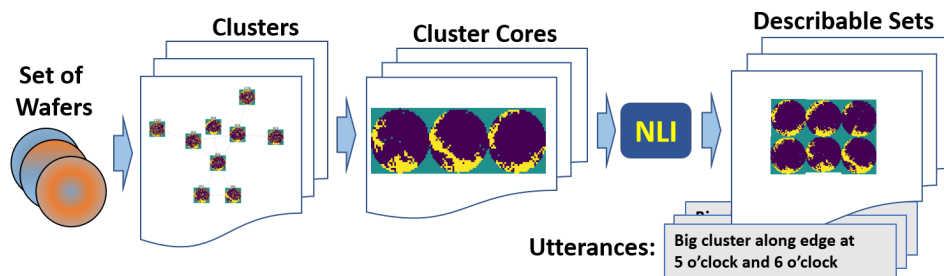


Figure 5.10: Attaining describable sets through NLI

Figure 5.10 illustrates the analytics performed in the backend. Given a set of wafer maps, first we obtain their MINION graph. From the graph, we extract connected components, and each becomes a *cluster*. Then, from each cluster we extract maximal

cliques as the *cluster cores*. Based on a core, the NLI interprets a wafer map by assigning values to a set of *wafer attributes*. These wafer attributes are to be selected by an *utterance* that describes a group of wafer maps. For simplicity, in this work we rely on utterance as input query to extract the wafer group for a plotting request (as shown in Section 5.4.1). Given an utterance describing a pattern, the utterance is broken down into a list of constraints on the attribute values. Then, a describable set is the set of the wafers whose attribute values are based on the same core and satisfy all the constraints.

The search space comprises all describable sets which are extended from the primitive patterns. In a try, one or more describable sets can be selected based on the query. Also, a selected group can be further refined with other pattern-independent constraints, such as a yield constraint or a selected period. For example, the user can request using wafer maps only from July, with yield loss greater than 30%.

Details of describing a cluster core

Figure 5.11 provides an example to explain more detail of the flow shown in Figure 5.10. Figure 5.11 shows a cluster, i.e. a connected component. One of the cluster cores (i.e. the maximal clique) has four wafer maps. A salient region analysis is applied to these four maps. This analysis can be based on the techniques reported in [124]. The goal is to extract an attention region on each map. Then, the four salient maps are stacked to create an *attention mask*. This mask can be obtained using a density estimation technique [13] and a threshold to include, say 80% of the density mass.

For a wafer map, values of its wafer attributes depend on the attention mask in use. Since a CC can contain multiple cluster cores, each with a different attention mask, a wafer map in the CC can receive multiple interpretations from the NLI. Note that wafers in one describable set are based on one interpretation. For an input query, all satisfying describable sets are included in the group of wafer maps.

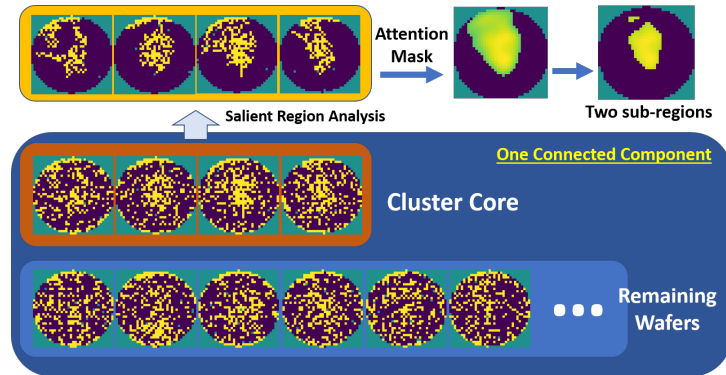


Figure 5.11: An example of cluster and *cluster core*

Consider the four wafer maps in the cluster core shown in Figure 5.11. Suppose the focus is on the top-left sub-region. For the four wafer maps, the NLI may interpret them as an “arc at x o’clock along edge” where x could be 11, 12, 10.5, and 12, respectively, depending on where the center of the arc pattern is located. These four wafer maps can all be captured in a less-constrained utterance like: “arc at 11 o’clock to 12 o’clock along edge”.

For the middle sub-region, the four descriptions may all be: “big cluster at the center”. Then, the capturing utterance would be the same. Furthermore, to describe the entire wafer, the NLI needs to decide a phrase that connects these two sub-regions. For example, the NLI can use a phrase like “extend to”, i.e. the “arc” “extend to” the “big cluster”.

5.5.2 The Natural Language Interpreter (NLI)

In the example above, we see that the NLI needs to support utterances using those *descriptive terms* such as “arc”, “along edge”, “11 o’clock”, “cluster”, “big”, “at the center” and “extend to”. Such vocabulary determines the scope of possible utterances that can be interpreted by the NLI. For building our NLI, we follow a grammatical

approach [167]: (1) We use a grammar to define the scope of all possible utterances; (2) For each descriptive term, we use a software script to check for its existence on a given wafer map (i.e. setting the value of the wafer attribute).

The grammar includes the capability to support describing patterns at two levels, at individual wafer level and at multi-wafer level. Earlier we see an example where different numerical values for a descriptive term can be merged into a value range to form a less-constrained utterance. When combining two utterances that have different pattern descriptions, we can use a high-level descriptive term. For example, suppose one utterance describes an “arc at 11 o’clock along edge” and another utterance describes a “cluster at 11 o’clock along edge”. In this case, we may use a high-level term “something” to capture both “arc” and “cluster”. Hence, both wafer maps can be described with the utterance “something at 11 o’clock along edge”.

To illustrate how the NLI follows a grammar to interpret a given wafer map, Figure 5.12 depicts the *parsing tree* for one wafer map (the 2nd map in the cluster core in Figure 5.11). For the core, there are two sub-regions. Therefore, in the parsing tree the wafer map is first partitioned into two components: $Comp_1$ and $Comp_2$. The detailed tree for the $Comp_1$ is shown in Figure 5.12 .

Below $Comp_1$, there are two nodes: $Comp$ and RELATION. The $Comp$ is for describing the component. The RELATION is for describing the relationship to $Comp_2$.

The $Comp$ node has two child nodes: $Check_{Arc}$ and $Descr_{Location}$. The $Check_{Arc}$ checks the type of the pattern to determine if it is an “arc”. If it is, then $Descr_{Arc}$ is used to describe what kind of “arc”. The $Descr_{Location}$ describes the location of the pattern. Three *attributes* are used: DIRECTION, SPREAD, REGION, and additional two are for associated PREPOSITION (PP).

The value of an attribute is determined by its corresponding software script. For example, the value of the LENGTH can be: `short | long | null`. The value of the

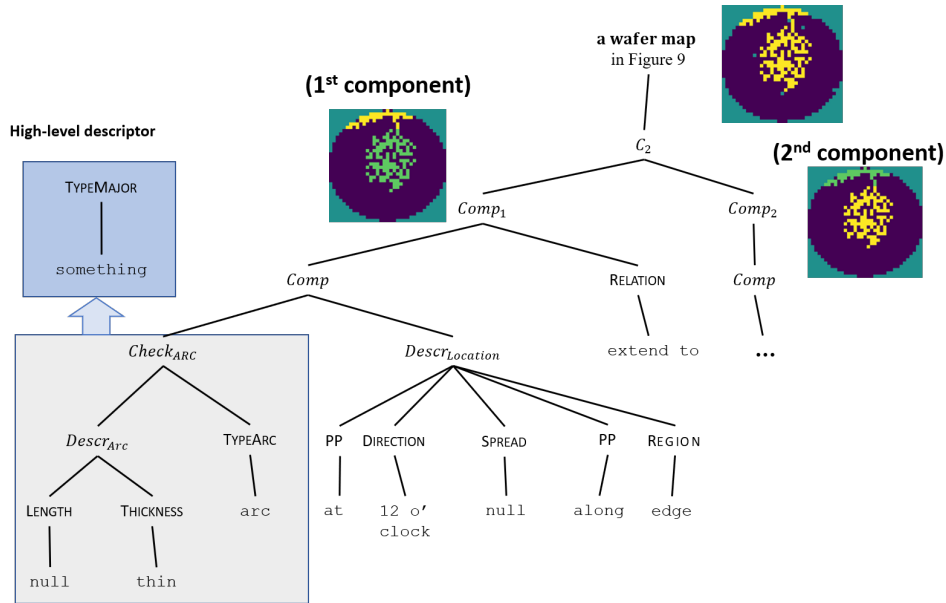


Figure 5.12: The parsing tree for one wafer map in Figure 5.11. The wafer map is described as: **“Null length thin thickness arc type at 12 o’clock direction null spread along edge extend to ...”**, and at high level (using the high-level descriptor “something”), can be captured as: **“something at 12 o’clock direction null spread along edge extend to ...”**

THICKNESS can be: thin | thick | null. The attributes and their possible values are part of the *lexicon* in the grammar, as that exemplified in Table 5.7.

Sub-tree below the $Check_{ARC}$ node is highlighted with a shaded box. This is to indicate that the box can be replaced by a high-level descriptor called “something”. With the high-level descriptor, detail of the pattern shape is ignored and the focus is on its other attributes.

In the parsing tree, the leaf nodes are values of the attributes. A *canonical utterance* can be obtained by concatenating the leaf node values and for some, their attribute names from left to right. The utterance is shown in the caption of the figure. A high-level utterance can also be constructed using the high-level descriptor “something”.

5.5.3 NLI’s grammar for describing patterns

Our NLI uses a context-free grammar (CFG) [167] to define a formal language \mathcal{L}_0 , which essentially models the interpretation process. A CFG is formally specified by 4-tuple (V, Σ, R, S) where V is a set of non-terminal symbols, Σ is a set of terminal symbols, $R = \{V \times (\Sigma \cup V)^*\}$ is a set of rules, and S is the starting symbol. A CFG includes a list of *production rules* that expand a non-terminal $v \in V$ into a string which can contain both non-terminals and terminals. The subset of rules that turns non-terminals into terminals, is called the *lexicon* of the grammar.

Table 5.7: The lexicon for the formal language \mathcal{L}_0

TYPEARC	→ arc
TYPERING	→ ring
TYPELINE	→ line
TYPEDONUT	→ donut
TYPECLUSTER	→ cluster
TYPEMAJOR	→ something
TYPEMINOR	→ minor component
RELATION	→ and and extend to
DIRECTION	→ x o’clock left right upward downward lower left upper left lower right upper right
LENGTH	→ short long <i>null</i>
THICKNESS	→ thin thick <i>null</i>
WAVINESS	→ straight wavy <i>null</i>
SIZE	→ small big huge massive <i>null</i>
DENSITY	→ solid somewhat solid more fails some fails
PREPOSITION	→ at near from to along around touch on
REGION	→ center edge in-between
SPREAD	→ wide <i>null</i>
CONNECTIVITY	→ broken <i>null</i>
COMPLETENESS	→ half full <i>null</i>
SIGNIFICANCE	→ not obvious
SUBSET	→ only exhibit on some wafers

Table 5.7 shows our current lexicon for \mathcal{L}_0 , which can be extended as needed. Each rule is of the form: $\text{ATTRIBUTE} \rightarrow \{\text{value1} \mid \text{value2} \mid \dots\}$ (i.e. wafer attributes and values). Note that a *null* value means that there is no description for the attribute and thus can

be omitted from the utterance. To use this grammar as an interpreter, a software script is implemented for each lexicon rule. The script determines which value should be selected for the wafer attribute.

Table 5.8 shows our current grammar rules for \mathcal{L}_0 . The grammar rules can be used to generate various strings by recursively expanding non-terminals starting from S until every non-terminal is rewritten into a terminal according to the lexicon. All the non-terminals that trigger a rule in the lexicon (Table 5.7) are presented in small capital font.

The grammar rules represent the working logic in the interpretation *workflow* implemented in our NLI. For example, the first rule ($G0$) expresses the fact that a wafer map can consist of up to two major components (C_1 for one and C_2 for two components) and some minor components (C^*). ($G5$) to ($G7$) are for a minor component without detailed description of its shape. ($P0$) captures the generic procedure for describing a major component. A major component can be described in terms of its shape and location descriptors along with other optional descriptions. Specifically, each type of shape is associated with a dedicated “check” function for its determination, and every shape is associated with its unique attributes, as stated in (A), (R), (L), (D) and (C), respectively.

In addition, the internal logic of NLI ensures that patterns like an arc or line will be checked first before proceeding to a more general shape like a cluster. ($P1$) indicates the rule for describing the location. ($P2$) provides other options for describing the component such as its pattern significance relative to the rest of the wafer, and if it only exhibits on a subset of wafer maps in the group.

Table 5.8: The Grammar for \mathcal{L}_0 .

Grammar Rules	
(G0)	$S \rightarrow \{C_1 C_2\} C^*$
(G1)	$C_1 \rightarrow Comp$
(G2)	$C_2 \rightarrow Comp_1 Comp_2$
(G3)	$Comp_1 \rightarrow Comp \text{ RELATION}$
(G4)	$Comp_2 \rightarrow Comp$
(G5)	$C^* \rightarrow null \mid \text{RELATION } C$
(G6)	$C \rightarrow MinorComp C^*$
(G7)	$MinorComp \rightarrow \text{TYPEMINOR } Descr_{Location} \langle opt \rangle$
(P0)	$Comp \rightarrow \{Check_{Arc} Check_{Ring} Check_{Line} Check_{Donut} \dots Check_{Cluster}\} Descr_{Location} \langle opt \rangle$ $\mid \text{TYPEMAJOR } Descr_{Location} \langle opt \rangle$
(P1)	$Descr_{Location} \rightarrow \text{SPREAD } \{\text{PREPOSITION DIRECTION}\} \dots$ $\dots \{\text{PREPOSITION REGION}\}$
(P2)	$\langle option \rangle \rightarrow \text{SIGNIFICANCE}$ $\mid \text{SUBSET}$
(A0)	$Check_{Arc} \rightarrow Descr_{Arc} \text{ TYPEARC}$
(A1)	$Descr_{Arc} \rightarrow \text{LENGTH THICKNESS}$
(R0)	$Check_{Ring} \rightarrow Descr_{Ring} \text{ TYPELINE}$
(R1)	$Descr_{Ring} \rightarrow \text{THICKNESS CONNECTIVITY COMPLETENESS}$
(L0)	$Check_{Line} \rightarrow Descr_{Line} \text{ TYPELINE}$
(L1)	$Descr_{Line} \rightarrow \text{WAVINESS LENGTH THICKNESS}$
(D0)	$Check_{Donut} \rightarrow Descr_{Donut} \text{ TYPEDONUT}$
(D1)	$Descr_{Donut} \rightarrow \text{SIZE THICKNESS DENSITY COMPLETENESS}$
(C0)	$Check_{Cluster} \rightarrow Descr_{Cluster} \text{ TYPECLUSTER}$
(C1)	$Descr_{Cluster} \rightarrow \text{SIZE DENSITY}$
Descriptions of above grammar rules	
(G0)	A pattern can have one or two major components and some minor ones
(G1)	The case where there is a single major component
(G2)	The case where there are two major components
(G3)	The first major component and its relationship to the second one
(G4)	The second major component
(G5)	The case where there is or is not other minor component and their relationship if any
(G6)	Minor components can be one or many
(G7)	A minor component is described only by its location
(P0)	The component is described by its shape and location with other options.
(P1)	Location is defined by the component's spread, direction and region
(P2)	How obvious the component is compared to the original wafer map
(A0)	An arc is defined by its length and thickness
(R0)	A ring is defined by its thickness, connectivity, and completeness
(L0)	A line is defined by its waviness, length, and thickness
(D0)	A donut is defined by its size, thickness, density, and completeness
(C0)	A cluster can be defined by its size and density

5.5.4 Implementing the grammar with software scripts

If we view the grammar as an interpretation workflow, it is not hard to see that we need two sets of software scripts to enable the interpretation. One is already mentioned above that we need a script for every lexicon rule. Another set of scripts are needed to implement other grammar rules in Table 5.8. For example, the attention mask can be used to decide how many sub-regions to focus on. This determination can be based on finding *density peaks* in the attention mask (which can be made as a contour plot). This can be a way to implement the (G0) rule in grammar, i.e. to decide whether we should follow C_1 or C_2 and whether C^* should be activated.

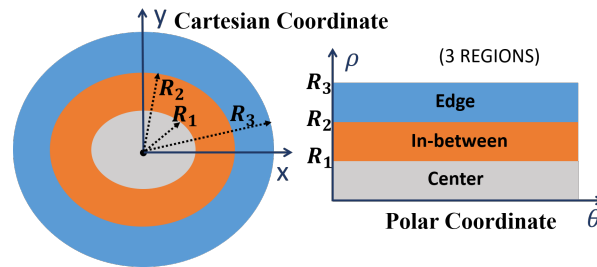


Figure 5.13: The definition of REGION on the wafer map

To simplify presentation, in this section we use two examples to illustrate how the scripts can be implemented. Recall in Figure 5.13, two sub-regions (i.e. two density peaks) are identified from the attention mask. To determine the location name of a sub-region, a wafer is divided into three areas: “edge”, “in-between”, and “center”. An *area checker* finds which area the sub-region is in. This determination is by transforming a map into polar coordinates and dividing its radius into three areas as shown in Figure 5.13. The area from the wafer map center to R_1 , i.e. $(\rho, \theta) = (R_1, 2\pi)$ is called the “center”, the next $(R_2 - R_1, 2\pi)$ is the “in-between”, and the last $(R_3 - R_2, 2\pi)$ is the “edge”.

Suppose the density peak falls in the “edge” sub-region, then the component might be

an “arc”. To check if it is actually an “arc”, we implement an *arc type checker* depicted in Algorithm 5.1. The input includes a threshold T_1 to determine if the component is an “arc”. First, a component must have its density peak fall inside the edge region in order to be called an “arc”. Then, we calculate two density estimates r, x within the component’s angular spread in the edge region. The ratio ($\frac{x}{r}$) of these density estimates is compared to the threshold T_1 to determine if it is an “arc”. Notice that the *arc type checker* relies on a fixed threshold.

One may raise the concern regarding the robustness of using a specific threshold, i.e. a person may see a pattern as an “arc” but the NLI fails to interpret it as an “arc”. Note that our NLI does not aim to optimize with respect to such an accuracy objective. Instead, it aims to find a describable set, and as stated in Section ?? this set has to satisfy two conditions on the Minions’ recognition graph. In this sense, the two conditions provide a check for the NLI’s result.

Algorithm 5.1: An arc type checker

```

Input: Threshold:  $T_1, R_2, R_3$ 
Output: Is it an Arc: True/False
Data: Wafer Map Matrix of the Component
1 edge region  $\leftarrow (\rho, \theta) = (R_3 - R_2, 2\pi)$  ; // Global
2  $T_1 = 66\%$  ; // Default
3 Assert the density peak is in edge region
4 Find the angular spread
5  $r \leftarrow$  the density sum within the angular spread in edge region
6  $R^* \leftarrow$  the region  $(\rho, \theta) = (R_3, 2\pi) - (R_2 + \frac{R_3 - R_2}{2}, 2\pi)$ 
7  $x \leftarrow$  the density sum within the angular spread in  $R^*$ 
8 if  $\frac{x}{r} < T_1$  then
9 |   return False ; // It is not an arc
10 end
11 return True ; // It is an arc

```

5.5.5 Examples of canonical utterances for describing patterns

Table 5.9 shows the canonical utterances generated by the NLI. Each row shows the salient wafer maps from two cores and their attention masks. The utterances interpreted by the NLI are listed (1: top-left, 2: top-right, 3: bottom-left, 4: bottom-right map). Notice that wafer maps in the same cluster core can have slightly different utterances.

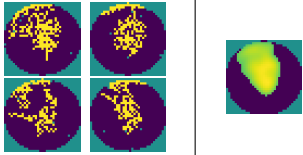

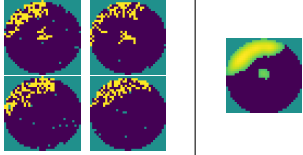

The high-level utterances for the two cluster cores are shown in Table 5.10. The table shows two stacked heatmaps for each core. The first is by stacking the four maps in the core. The second is by stacking the four maps and all the neighboring wafer maps in the describable set. A *neighboring wafer map* is directly connected to at least one wafer map in the core. This is one way we can use the MINION graph to further select a *subgroup* of wafers from a describable set.

The high-level descriptor “something” can be used to capture a set without detailed description of the pattern shape. For location, wafer maps in Table 5.9 include a variety of descriptions all related to the center region. Hence, they can be captured with another high-level descriptor “around”.

For the first core, the first component can be at either 11 or 12 o’clock directions. To capture all four wafer maps, the high-level description can become “11 to 12” o’clock. Note that a “wide spread” range of two clock values can be converted into a single clock value by taking their average. In this way, the first wafer map is included in the describable set (which is a requirement).





Another consideration for a high-level utterance is that, not all wafer maps in a set have the same number of components (This is captured in (*P2*) grammar rule). The second cluster core in Table 5.9 exhibits this situation. In this case, an optional phrase “only exhibit on some wafers” is appended to the utterance to indicate the fact that the utterance of the second component only applies to some but not all wafers in the

Table 5.9: Salient wafer maps from a cluster core and their canonical utterances

Sali. Wafer Maps (Core)	Atten. Mask	Canonical Utterance (1: top-left, 2: top-right, 3: bot-left, 4: bot-right)
		<ol style="list-style-type: none"> 1. null size more fails density cluster type wide spread from 9 o'clock direction to 1 o'clock direction along edge and extend to big size somewhat solid density cluster type upward direction at center. 2. null length thin thickness arc type at 12 o'clock direction null spread along edge and extend to big size some-what solid density cluster type upper right direction at center. 3. null length thin thickness arc type at 11 o'clock direction null spread along edge and extend to huge size more fails density cluster type upper left direction at center. 4. null length thin thickness arc type at 12 o'clock direction null spread along edge and extend to big size somewhat solid density cluster type upward direction at center.
		<ol style="list-style-type: none"> 1. null size more fails density cluster type wide spread from 9 o'clock direction to 12 o'clock direction along edge and small size cluster type upward direction at center not obvious. 2. null size some fails density cluster type wide spread from 9 o'clock direction to 1 o'clock direction on edge and small size cluster type left direction at center region not obvious. 3. null size more fails density cluster type wide spread from 9 o'clock to 1 o'clock touch edge. 4. null size more fails density cluster type wide spread from 9 o'clock direction to 2 o'clock direction on edge.

given set. In general, after individual canonical utterances are interpreted for wafer maps, incompatible phrases from individual wafers can be abstracted out to satisfy a given high-level utterance (e.g. “something” at the $Check_{Arc}$ and $Check_{Cluster}$ nodes to replace an “arc” phrase and a “cluster” phrase.). Note that while a high-level utterance can be used to describe a set of wafers, we make sure that a describable set always includes all wafers from the corresponding cluster core.

Table 5.10: Examples of wafer grouping

Cluster Core Stacked	Subgroup Stacked	Canonical Utterance
		Something at 11 to 12 o'clock direction along edge extend to something around center.
		Something at 11 o'clock direction wide spread along edge and something around center only exhibit on some wafers.

5.5.6 Paraphrasing with GPT-3

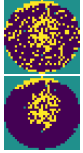
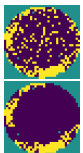


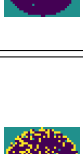
The work [96] also considers converting a canonical utterance into a natural language description. This functionality can be useful when a user asks IEA to provide a summary report of the wafermap data, for example in answering questions such as “What patterns are in the dataset” as mentioned in Section 4.4. This is in contrast to the work presented in [111] where the goal is to translate a natural language query into an internal representation.

In work [96], we can leverage the capability provided by a state-of-the-art *language model*, specifically, the GPT-3 model [173] by the time of 2022. We take advantage of the *in-context learning* capability provided by GPT-3 to achieve *few-shot learning*.

In the few-shot learning, we guided the GPT-3 by designing the input and output to follow two prefix words: “Sentence:” and “Paraphrase:”, respectively. The stop word is the same as the output prefix, so that the model will stop generating once it completes one paraphrase. The temperature parameter is set to be 0.9 which allows more flexibility to paraphrase a given utterance.

To demonstrate the paraphrasing capability, Table 5.11 shows some examples. For each example, two maps are shown: the original wafer map and its salient region map. In each case, a canonical utterance from the NLI is shown, followed by one paraphrase given by GPT-3.

Table 5.11: Examples of wafer pattern interpretation

Original / Salient	Canonical Utterance Interpreted by the Grammar / GPT-3 Paraphrasing Output
	<p>Utterance: null length thin thickness arc type at 12 o'clock direction null spread along edge and extend to big size somewhat solid density cluster type upper right direction at center.</p> <p>GPT-3: There is a thin arc at 12 o'clock along the edge and extended to a somewhat solid big cluster located upper right at the center.</p>
	<p>Utterance: thick thickness broken connectivity half completeness ring type wide spread from 3 o'clock direction to 12 o'clock direction on edge.</p> <p>GPT-3: There shows a thick half-ring, which is not fully connected, widespread from 3 o'clock to 12 o'clock on the edge.</p>
	<p>Utterance: null size more fails density cluster type wide spread from 9 o'clock direction to 12 o'clock direction along edge and small size cluster type upward direction at center not obvious.</p> <p>GPT-3: There exists a cluster with more fails wide spread from 9 o'clock to 12 o'clock along the edge and a small cluster located upward at the center, which is not an obvious pattern.</p>
	<p>Utterance: small size cluster type at 10 o'clock direction along edge and extend to small size cluster type upper left direction near center.</p> <p>GPT-3: There exists a small cluster at 10 o'clock along the edge, which extends into a small cluster located upper left near the center.</p>
	<p>Utterance: small size cluster type at 10 o'clock direction on edge and extend to small size cluster type upper left direction near center, and minor component at 6 o'clock direction along edge.</p> <p>GPT-3: There is a small cluster at 10 o'clock along the edge which extends into a small cluster located upper left near the center, and there is also a minor component located at 6 o'clock along the edge.</p>

5.5.7 NLI’s application results

In work [96], the NLI was applied to help analyze test data collected from a recent product line. The wafers were stamped with days in three months (call them Period 1,2,3). The three periods have 1448, 1383, and 74 wafers, respectively.

Figure 5.14 illustrates the flow of investigating a correlation between wafer probe and final test in [96]. From the MINION graphs, we obtain a set of cluster cores. Based on each core, results from the NLI are stored in a database table. Each row in the table corresponds to a wafer map containing the wafer attribute values determined by the NLI.

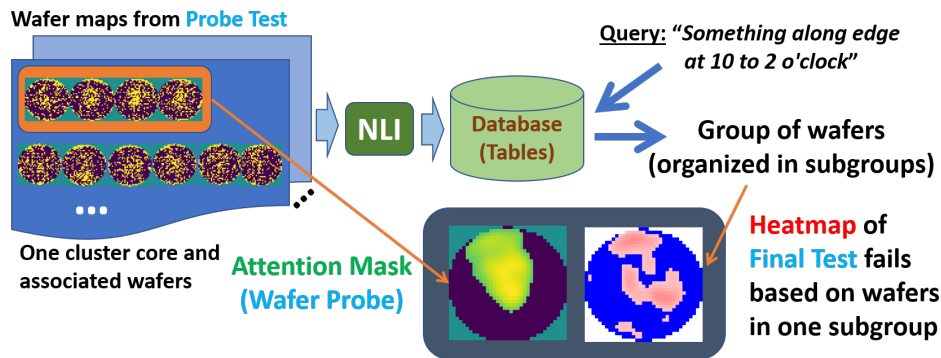


Figure 5.14: Flow to investigate a correlation between wafer probe and final test where each try starts with a simple query to find a group of wafers.

In a try, a query is specified to extract a group of wafers. With a query, the describable sets are extracted as the group of wafers for plotting. In plotting, we consider arranging the wafers in *subgroups* based on primitive patterns in the group (see discussion of Table 5.10 before). For each subgroup, two maps are shown. The first is the *attention mask* from the primitive pattern. The second is the heatmap from final test fails based on all wafers in the subgroup.

Figure 5.15 shows a summary result arranged by subgroups along the x-axis. Each bar shows the number of wafers in the subgroup (corresponding to a primitive pattern). Through the attention masks, we can see a pattern trend evolving over the three periods.

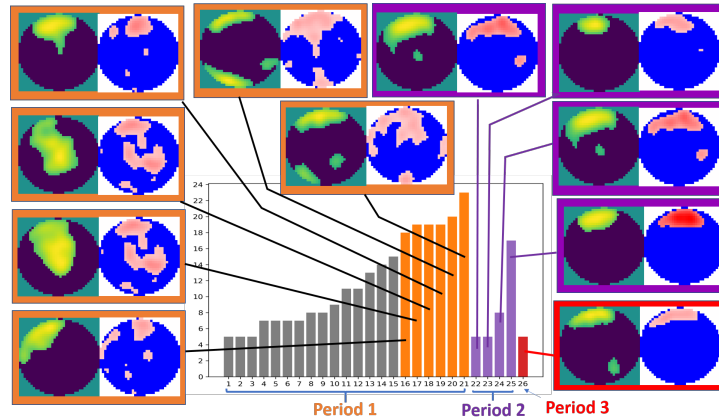


Figure 5.15: Finding of a pattern trend lasting over three periods

Then, the correlation to the final test can be seen by comparing each attention mask to the corresponding final test failing heatmap.

The work [96] also demonstrated the results from searching for a correlation between the described pattern class (from wafer probe) and an E-Test parameter. Each plot is based on two sites of an E-test parameter. To facilitate visualization, the plots are ranked according to their “bias” value, calculated as the distance between the average position of the selected wafers and the average position of the rest of the wafers. Note that in each case, we show one of the top-ranked plots. Figure 5.16 shows one finding. The 1st and 2nd sites are at the two closest locations to the pattern.

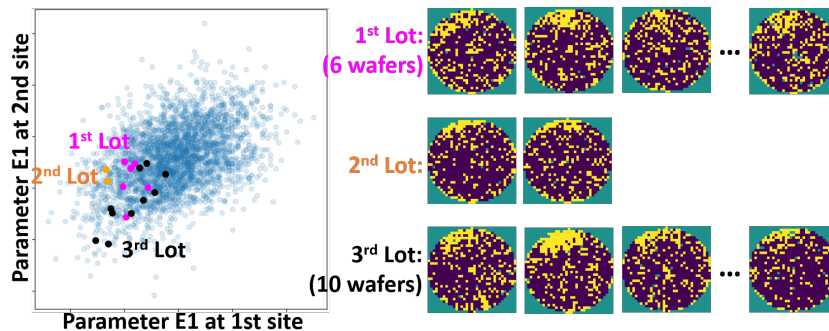


Figure 5.16: E-test correlation plot based on wafer maps describable by “cluster fails at direction from 11 o’clock to 12 o’clock along edge”

More application results can be found in work [96]. It is important to note that the final results represented in the work [96] were obtained through multiple iterations of querying IEA, visualizing the plots, refining the next search step accordingly, and finally reaching an interesting finding. In summary, IEA 2022, enabled by the MINION and NLI modules, was the first time the Language-driven analytics was realized to efficiently navigate the exploration in a Choose-and-Bound search space.

Chapter 6

IEA 2023

行路難！行路難！多岐路，今安在？

長風破浪會有時，直掛雲帆濟滄海。

(The road is hard! The road is hard! So many turnings, And now where am I?) So when a breeze breaks waves, bringing fair weather, I set a cloud for sails, cross the blue oceans!

— 《行路難》李白， A poem from Tang Dynasty

6.1 Introduction

The emergence of large Language Models (LLM) [174][173][175][176][19] changed our view for implementing a domain-specific AI Assistant like IEA in 2023. Among them, ChatGPT [19] has demonstrated remarkable performance for engaging in dialog on a wide variety of topics, including answering questions and generating code.

Figure 6.1 shows an example dialog with ChatGPT regarding how to correlate wafer map pattern to E-test parameter. The first question asks a *how-to* question, followed by a question asking to list the relevant statistical methods. The third question asks for a

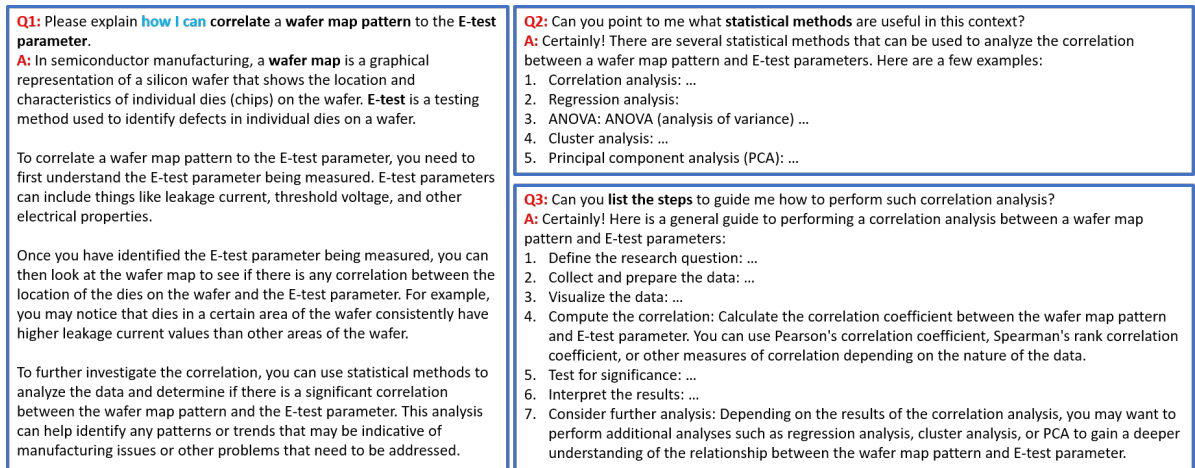


Figure 6.1: Dialog with ChatGPT (03/26/2023) based on questions for how to correlate wafer map patterns to E-test parameters

list of steps to perform the correlation. It can be seen from the responses that ChatGPT presents a very good general understanding of the topic.

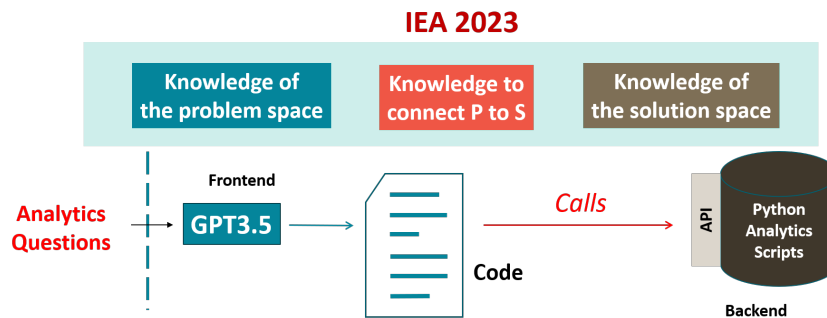


Figure 6.2: Overview of IEA 2023

In contrast to a general dialog like that shown in Figure 6.1, our use of an LLM is more specific. Figure 6.2 illustrates the *grounding* problem faced in IEA 2023, comparing to the semantic parsing problem faced in IEA-2022 depicted in Figure 5.4. In IEA 2023, we desire the LLM to generate code according to our question or instruction. Instead of generating unconstrained code, we desire the LLM to generate code that calls our specific API supported by the backend software to answer the question or to complete the task specified by our instruction.

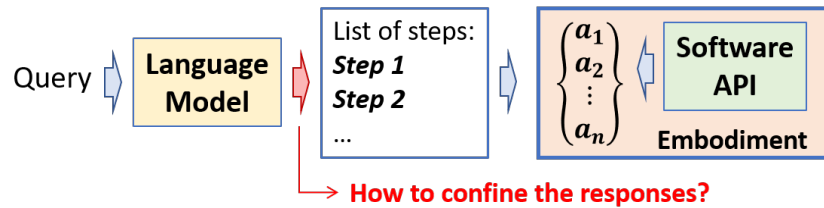


Figure 6.3: Task grounding problem: How to confine model responses within the scope of an embodiment with admissible actions $\{a_1, a_2, \dots, a_n\}$?

Abstractly, we formulate the problem and call it the *task grounding* problem. Figure 6.3 illustrates the task grounding problem. First there is a given *embodiment* for performing the tasks. In our application, the embodiment is based on a software API which defines a set of *admissible actions* $\{a_1, \dots, a_n\}$. While a model response has to be a list of steps, each step has to be *realizable* by a subset of the admissible actions.

6.1.1 The task grounding problem

Task grounding means that one desires to use an LLM to *act* in a specific environment [177][178]. Given an LLM learned with rich world knowledge, the goal is to ground high-level instructions expressed in natural language to a defined set of actions admissible for a given embodiment. Simply put, we desire to constrain the responses from the LM to be within the capability of the given embodiment.

Consider the response of question 3 in Figure 6.1. Step 4 suggests Pearson’s correlation and Spearman’s rank correlation. In our specific context, “*correlation*” can have a different meaning. For example, the “*correlate*” in “correlate a wafer map pattern to E-test parameter” means *to find an E-test parameter whose values can be used to indicate the possible occurrence of the wafer map pattern during wafer sort* (i.e. this is called “failure pattern feedback” in [96]). By grounding an LLM with this domain knowledge, we desire the LLM to interpret the term “*correlation*” as how we would interpret it (not how a common person would).

Steps like those listed with the question 3 also need to be specific enough to enable automatic mapping to some API calls for their realization. By grounding an LLM with a set of admissible actions, we ensure each step in LLM’s responses to be always realizable.

6.1.2 Approaches for solving a task grounding problem

One popular way to constrain LLM’s responses with a specific response structure is through *prompt* engineering [173][176]. Prompt engineering provides input-output examples with a task specifier (the “prompt”) for the model to emulate the desired response structure. However, as pointed out in [178], prompt engineering alone is not sufficient to fully constrain a LM to a set of admissible actions.

In addition to prompt engineering, there are two usual approaches to constrain LLM’s responses. One is to use *reinforcement learning* to align the responses to the desired outputs [177]. Another way is by treating it as a *constrained semantic parsing* problem [170] (as we did in [96]) and implementing a way to adjust/select responses such that only admissible actions are allowed [178]. It can be implemented as a postprocessing component separated from the LM, e.g. to rank/score responses from LLM and only admit those acceptable ones.

6.1.3 Feasibility of the common approaches

Prompt engineering would be effective if we can be sure that the LLM has the desired knowledge or skills in the model, and we just need to be more expressive in our queries so that the model knows to respond accordingly. However, verifying that a given LM indeed has the knowledge and skills we desire, by itself, can be a challenging research problem.

For example, we might desire the correlation task to start with our MINIONs analysis

method [14]. We are not sure to what extent a given LLM “understands” the proposed method (even though the LLM may have “read” the paper).

As proven in [176] and also in [177] for task grounding, reinforcement learning can be an effective approach to align an LLM to respond in a specific domain. However, reinforcement learning requires training data. In our application, if we have a way to generate a large dataset of acceptable user queries to the IEA, then we might consider reinforcement learning. To get to that point we need to first define what user queries are acceptable and develop a formal model that defines the scope of acceptable queries. If this is done, then the model can be used as a *query generator*.

Our prior work [179], as described in the previous chapter, follows the *constrained semantic parsing* approach [170]. However, comparing to the task grounding problem in Figure 6.3, the formulation in [179] is more restricted. In [179], the grounding is achieved through a *grammatical model* that limits the responses to be a set of *canonical utterances*. The approach enables implementation of a constrained parser using prompt engineering with GPT-3. However, the parser does not consider scenarios where one query might be contextually related to the next query. In other words, each query to IEA-2022 is treated independently.

Solving the connection problem between the LLM and backend API as constrained parsing was largely due to the limited capabilities of GPT-3. With GPT3.5, we no longer need a grammatical model to define a set of canonical utterances. Rather, we can use a *knowledge graph* (KG) as the target for the grounding. As it will be explained below, what made this new approach possible was in the dramatically-improved capability of GPT3.5 for paraphrasing, comparing to GPT-3. The paraphrasing capability of GPT-3, while it was there, was much less reliable than that of GPT3.5. Because GPT3.5 could do paraphrasing much more reliably, it enabled us to introduce a KG into IEA and redesigned IEA-2023 into the current IEA called IEA-Plot in 2023. The use of a KG

then enabled IEA-Plot to implement ways to deal with the problem of tracking contextual information of a dialog containing a sequence of instructions/questions.

6.1.4 The KG-based approach implemented in IEA-Plot

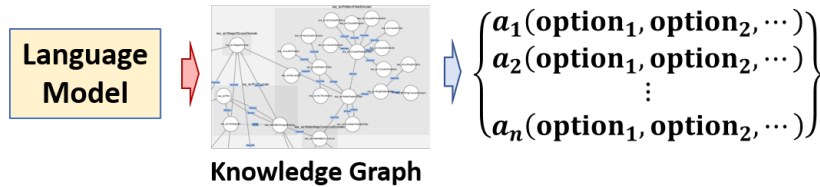


Figure 6.4: Using a knowledge graph to connect LLM and admissible actions

IEA-Plot implemented an innovative approach for solving the task grounding problem described above. Figure 6.4 depicts the main idea. The approach uses a *knowledge graph* (KG) [180] to connect an LLM and those admissible actions. For each action a_i , it has a set of acceptable options. Hence, for each user instruction, the responses are confined within the set of actions and their options. The KG serves three purposes:

- From the perspective of the LLM, the KG is used to constrain the LLM’s responses. Each response is represented as a subgraph of the KG.
- From the perspective of backend API, the KG captures our *analytic knowledge* on how to utilize API function calls to accomplish an analytic task.
- From the perspective of IEA implementation, the KG is a model to enable data management for keeping track of the current tool’s state during its execution.

The importance of domain knowledge in *domain-specific machine learning* has been emphasized in two previous theses from our lab [181][182]. Throughout our research, domain knowledge had been emphasized from the very beginning. For example, Section 1.4.8 in the Introduction chapter, emphasizes that domain knowledge is essential

for learning from data in general in our domain-specific application contexts. In view of the DSML big picture depicted in Figure 2.22, domain knowledge is essential to drive the Choose-and-Bound search for data exploration. Two of the four essential questions in DSML, highlighted in Section 2.7.4 concern about domain knowledge and then, the DSML equation highlights the necessity of domain knowledge (equation 2.1 first explained in Section 2.7.4 and later refined into equation 3.1 in Section 3.12.2 after the lessons learned with the Knowledge-Driven View discussed in Section 3.1). While we had all these discussions on the importance of domain knowledge, it was not until IEA-Plot ¹ that we could finally point to a tangible place, i.e. the KG, and refer to it as the “domain knowledge”.

6.2 Use of Knowledge Graph

Our decision to utilize a KG to connect LLM and backend API was partly inspired by a recent trend in natural language research where LLM and KG are combined to improve natural language inference (NLI) and question-answer (QA), e.g. see [183][184][185][186][187]. A KG, such as ConceptNet [188], provides structural knowledge that can be used to *ground* the reasoning process through an LLM [185]. A KG represents knowledge in a *symbolic space* while an LLM represents knowledge in a *vector semantic space*. The major challenge in their work is how to effectively fuse the two representations in a unified manner [186][187], mostly by end-to-end training.

In our IEA design, we mean to use KG in a different way though. As illustrated in Figure 6.5, the KG provides a target output space for the frontend semantic parser. The parser’s job is to map a query to a subgraph of KG. This mapping is based on two aspects of the query: its intent and the implied steps required for the task.

¹A promotion video introducing IEA-Plot at the 2023 IEEE International Test Conference can be found through this [link to YouTube](#).



Figure 6.5: A user query corresponds to a subgraph in the KG

The use of KG was motivated by another objective. We desire to use KG as a central place to store *domain knowledge* and as discussed in Section 6.1.4 above, this domain knowledge has three perspectives:

1. The LLM perspective (knowledge about the steps involved in an analytic process)
2. The API perspective (knowledge about the analytic tools and their use)
3. The IEA tool perspective (knowledge about the IEA implementation itself)

This domain knowledge is expressively represented in the KG and sharing of the knowledge can be achieved by sharing the KG. Ideally, we also want to design the KG such that future scaling of the IEA tool’s capability (e.g. adding a new function or option) can be done by adding nodes and edges to the KG.

We postpone the detailed discussion of our KG design to Chapter 7. Below we will explain the capabilities of IEA-Plot, followed by explaining how IEA-Plot solves the task grounding problem mentioned at the beginning of this chapter.

6.3 Wafermap Analytics in IEA-Plot

IEA-Plot is designed to support wafermap analytics. The backend API of IEA-Plot is built upon the MINIONs approach presented in Chapter 4 (and published before in [115][96][14]). This section highlights some specific capabilities implemented in IEA-Plot.

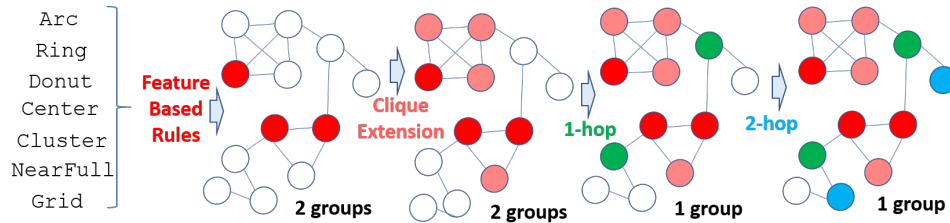


Figure 6.6: Identifying a pattern group satisfying a pattern Concept

As shown in Figure 6.6, IEA-Plot (the specific version along with the publication of [35]) included seven pattern concepts: `Arc`, `Ring`, `Donut`, `Center`, `Cluster`, `NearFull`, and `Grid`. Each pattern concept is modeled as a separate *resource domain* in our KG (see discussion in Section 7.1 for more detail about the domain partition in our KG design).

IEA-Plot’s backend supports various preprocessing steps to obtain a wafermap, e.g. salient map, masking, resizing, etc. In our KG, those options are modeled in the `WaferMapConstruct` resource domain. On a wafermap, the value of a die can also be determined in various ways. For example, it can be based on stacking wafers from the same lot, or based on a particular test bin. Those options are modeled in the `WaferMapsRepresentation` resource domain.

Given a set of wafermaps, each is checked to see if it satisfies a pattern concept. This check is based on hard-coded feature-based rules. For the features in use, we experimented and selected features reported from prior works (e.g. [124]) and implemented some of our own [96]. Suppose we want to check if a wafermap contains an `Arc` pattern, in our current approach we first apply the rule-based script for the `Arc` concept. Our rules are designed to be conservative so that if the script considers a wafermap as an `Arc`, the pattern would be obvious from our visual point of view. From the wafermaps picked by a rule, we can then expand the wafermap group based on the concepts and relations operated on the MINIONs graph, as explained in Section 4.6.3 before.

Figure 6.6 provides an example to show that after running a script, three nodes

are selected in the NINIONs graph. At this point, the three nodes are separated in two groups. Based on the MINIONs graph, we can extend the set of selected nodes in different ways. First, we extend the selection by including all nodes that belong to a *clique* based on an already-selected node. The result is shown in the second graph in Figure 6.6 (after the *clique extension*). The selected nodes still form two groups. The first group is a clique of size 4 and the second is a clique of size 3.

The next extension is based on including all nodes that have a direct connection with the current selected nodes. We call this operation the *1-hop extension*. Those newly-included nodes are marked as **green**. Then, repeating the idea of 1-hop extension, we can have *2-hop extension* which includes two additional nodes marked as **blue**.

6.3.1 Pattern group

In IEA-Plot, 1-hop extension is the current default to obtain a pattern group. For example, Figure 6.7 shows the IEA-Plot result based on the query to see the **Center** pattern. The display comprises three levels: after the rules, after the clique extension, and after the 1-hop extension.

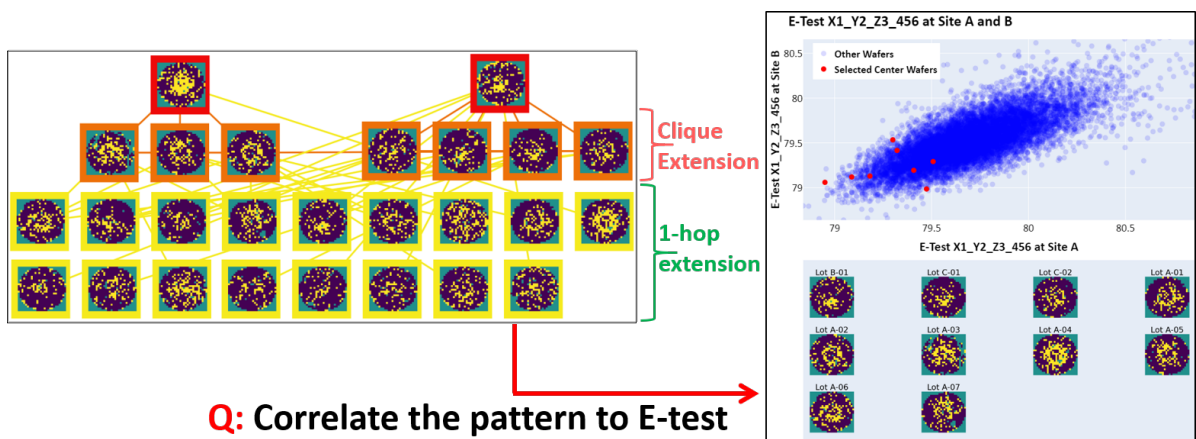


Figure 6.7: Results based on the **Center** pattern concept

When the next query “Correlate” is given, the pattern group is transferred from the wafer sort domain to the E-test correlation domain. IEA-plot generates correlation plots like the one shown in Figure 6.7. The plot shows 10 **Center** wafers from 3 lots. The two axes are two sites from the same E-test parameter where the two sites are closest to the pattern. Each dot shows the E-test values of the two sites. Red dots are those 10 wafers.

E-test correlation can result in multiple plots reported. A bias score is assigned to each plot and plots are ranked accordingly. For generating a correlation plot, IEA-plot searches for the subset of wafers from the pattern group, one lot at a time, which give the largest bias score. For detail of this E-test correlation method, please see [96].

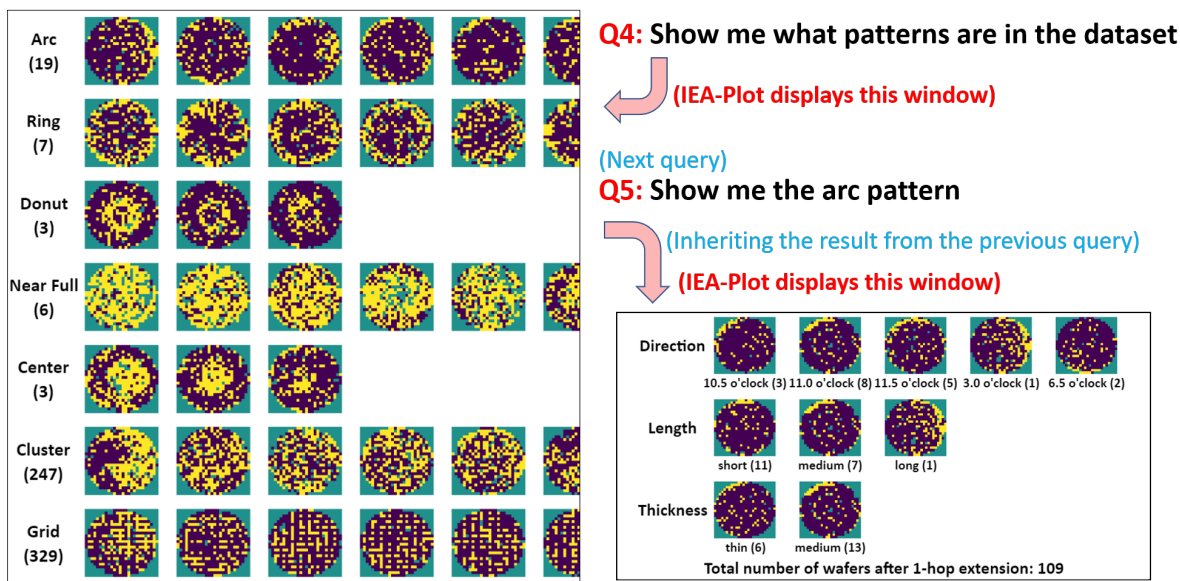


Figure 6.8: IEA-Plot outputs based on the two consecutive queries

Figure 6.7 shows a scenario where the user already knows what pattern to request. Initially, the user might want to see what patterns are available in the data and what options might be available for a particular pattern.

In Figure 6.8, **Q4** gives a list of available patterns. For the **Arc** pattern, **Q5** gives a list of available options. These results are generated based on the rule-based scripts with no extension. Result of **Q5** shows that the **Arc** pattern can point to five different

directions, have three length types, and two thickness types. The number of wafermaps satisfying each option is shown. From here, a user can select options to define a pattern group. If no option from a category is selected in a query, the default is “all”.

6.4 Analytics Driven by a Dialog

Figure 6.9 shows a dialog example. IEA-Plot’s output screenshots for those queries are shown in Figure 6.10 (except for **Q4** and **Q5** which are already shown in Figure 6.8 above). The dialog includes an intent switching at **Q4** and a domain switching at **Q12**. These two types of context switching have been discussed in detail above.

<u>Note:</u>	Q1: Show me the wafermaps
(Narrowing data scope)	Q2: Show me those only with yield loss greater than #%
(Selecting a new plot type)	Q3: Let me see them based on lot-to-lot arrangement
(Switching Intent; See Figure 15)	Q4: What patterns are in this dataset?
(See Figure 15)	Q5: Show me the options of the Arc pattern
(Pick a pattern group)	Q6: Let’s focus on the pattern at 11 o’clock direction
(Selecting a new plot type)	Q7: Let me see which lots they are in
(Narrowing data scope)	Q8: Please zoom-in to January
(Back to previous data scope)	Q9: Go back
(Pick a new pattern group)	Q10: Show me the Donut pattern only
(Back to previous data scope)	Q11: Go back
(Switching Domain)	Q12: Please correlate the pattern to E-test

Figure 6.9: A dialog example and IEA-Plot’s outputs shown in Figure 6.10

Q2 and **Q8** are two examples where the data scope is narrowed. In IEA-Plot, the data scope is inherited into the next query by default unless (1) the next query selects a new scope, (2) the next query is an intent switching or a domain switching. To restore to the previous data scope, a special query “Go Back” is used. For example, **Q9** resets the data scope back to **Q7** which has the same data scope of **Q6**.

Q10 asks to see a different pattern from **Q6**. This creates a new data scope, i.e. the set of wafers having the Donut pattern. **Q11** then resets the data scope back to the

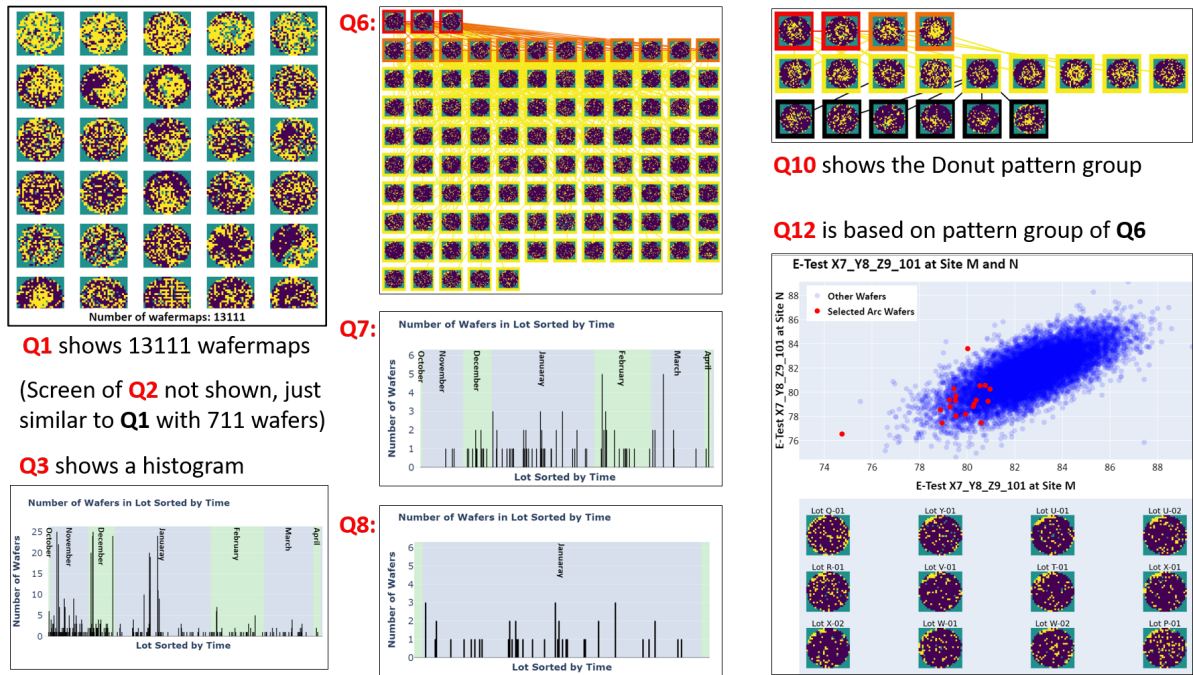


Figure 6.10: IEA-Plot output screenshots for queries listed in Figure 6.9, screenshots for **Q4** and **Q5** displayed in Figure 6.8 previously

previous scope which is the scope of **Q6**. Then, the Arc pattern group is transferred to the E-test domain when processing **Q12**, generating the correlation plot as shown.

6.5 Frontend Parser for Task Grounding

Our parser implementation is fundamentally different from that reported in [179] (and explained in the previous chapter). With the availability of GPT3.5 [16], we leverage its power by taking a *generative approach*. Figure 6.11 illustrates the approach. As shown in the figure, the capability of GPT3.5 we leverage is the *paraphrasing* capability, as discussed above in the last paragraph in Section 6.1.3.

By traversing the KG, we can extract a list of acceptable configurations (see Section 7.1 for detail about the KG). A configuration corresponds to a subgraph in the KG. For a configuration, we can generate a list of concatenated phrases (i.e. a sequence of

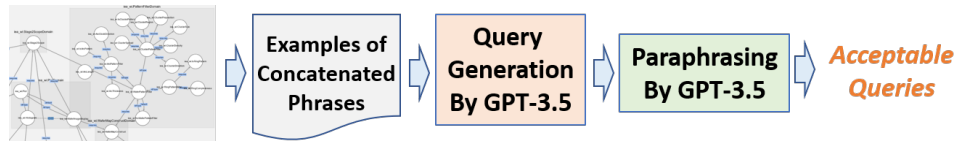


Figure 6.11: Generating acceptable queries using a GPT model [16]

phrases separated by comma) by enumerating combinations of phrases associated with those activated nodes (see Figure 6.12 for an example). For each concatenated phrase, we ask GPT [16] to make it a full English sentence. Then, we ask GPT to repeatedly paraphrase the sentence to obtain semantically-equivalent sentences. At the end, for each concatenated phrase we obtain a list of acceptable queries.

Concatenated Phrase: { Show, Wafermap display, For all time, In stage2.csv file, Group by lot, Sort by yield loss, With top 100 wafers }

Sentence Completion: **Show wafer map display for all time in stage2.csv, grouped by lot, sorted by yield loss, limited to top 100 wafers.**

Paraphrasing: **Organize wafer map data for all time in stage2.csv, group by lot, sort by yield loss, show top 100 wafers.**
Generate a wafermap display for all time periods in the stage2.csv file, grouping the data by lot and sorting by yield loss, with a focus on the top 100 wafers.
Show a wafermap display that groups all time data from stage2.csv file by lot, sorts it by yield loss, and showcases only the top 100 wafers.

Figure 6.12: An example to obtain a list of acceptable queries

6.5.1 Intent capture

Parsing of a query is divided into two stages: *intent capture* and *phrase matching* (see Section 7.1.3 for more detailed discussion in view of activating a subgraph in the KG). Intent capture determines if the given query requires a *switch of context*. Our KG supports two types of context switching: intent switching and domain switching (e.g. Figure 6.9). In addition, intent capture determines the current scope of the intent, i.e. the set of allowable nodes in the KG. Under an intent, those nodes are considered for phrase matching.

For intent capture, we use a sentence-BERT (SBERT) model [189]. Figure 6.13 illustrates our approach. First, we sample a set of acceptable queries where each option is covered at least once. The sampling is done in two phases. First, we sample acceptable configurations. In our experiment, we started with over 250K configurations. Then, we sample a subset of them for generating acceptable queries.

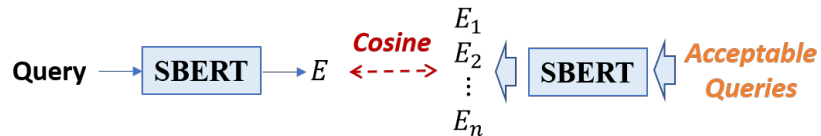


Figure 6.13: Intent capture by comparing pairwise SBERT embeddings

Given a query, we use SBERT to obtain an embedding, a 384-dimension vector. Given a list of n acceptable queries, we obtain a table of n embeddings (E_1, \dots, E_n in Figure 6.13). When a query is entered to IEA-Plot, SBERT generates an embedding E for the query. Then, this E is compared with E_1, \dots, E_n using cosine similarity. The most similar E_i is used to indicate the intent of the query.

While the approach is simple, it is important to note that its performance can largely depend on the set of acceptable queries stored in the table, i.e. the *coverage* of the set. This coverage depends on the paraphrasing power of the GPT model. IEA-Plot leverages such power provided by the model to simplify the parser implementation.

SBERT was trained to decide semantic similarity between two sentences [189]. While the original SBERT model performed reasonably well in our intent capture, we also found fine-tuning the model could improve the result.

Figure 6.14 shows a 2D projection (using t-SNE) of the embeddings based on 3623 acceptable queries we sample. The purple markers are queries with no context switching. The other three colors each represents switching to the particular intent. As seen, fine-tuning the model makes separation among different groups of queries more clear.

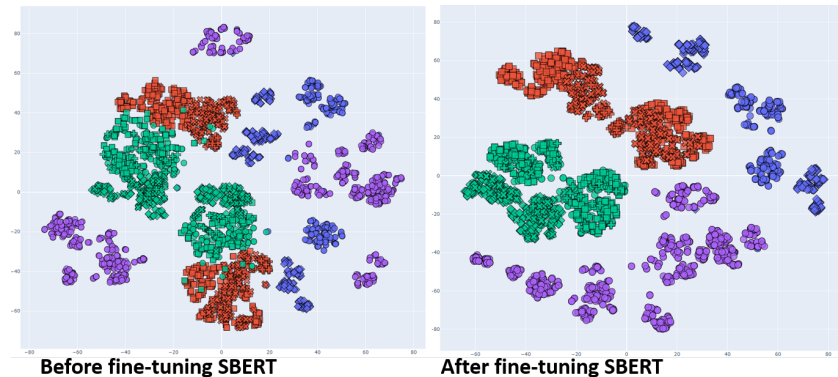


Figure 6.14: Fine-tuning SBERT improves our intent capture

Our fine-tuning follows the *retrofitting* idea suggested in [190]. Among the 3623 queries, we sample 382 to fine-tune the SBERT (following Figure 6.13, this means that $n = 382$ in E_n). Then, we test the model on the remaining 3241 queries and find only 1 query whose intent is captured wrong. The cause for the mistake is actually due to the fact that paraphrasing can generate a query that looks quite different from its original query. In Figure 6.13, this means that we have to include this one query into the set of acceptable queries on the right to cover the special case.

6.5.2 Phrase matching

Our approach to phrase matching also relies on checking the cosine similarity between two embeddings. The difference is that in phrase matching, the two embeddings are from two phrases (instead of two sentences). And instead of using SBERT, we use the original BERT model [191].

Figure 6.15 illustrates the approach. Similar to Figure 6.13, we first build a table of phrase embeddings q_1, \dots, q_m . Each q_j is labeled with the corresponding node name in the KG. The starting point is also a set of acceptable queries.

Given a query, we need to extract phrases stated in the query. We use the core-NLP

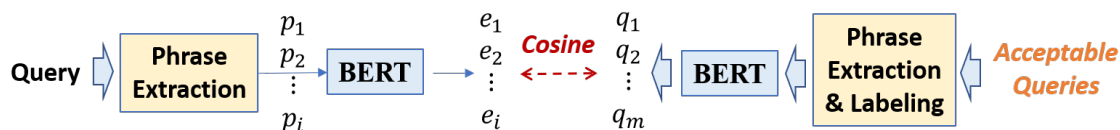


Figure 6.15: Phrase matching by comparing pairwise BERT embeddings

API [192] to obtain a constituency parsing tree. We then applied custom rules to extract potential phrases. If the query is the original query generated by sentence completion, the label of a phrase can be determined easily by checking the phrases stored with the nodes in the KG. If the query is obtained by paraphrasing, then the label is determined by matching its BERT embedding with the BERT embeddings of those phrases from the original query.

In the implementation we have grown the table to contain over 13K embeddings categorized with 116 labels. We have verified the performance with over 5K paraphrased queries and found no mistakes.

Given a user query, the process to obtain embeddings e_1, \dots, e_i in Figure 6.15 is similar. First we use core-NLP API and rules to obtain phrases p_1, \dots, p_i . Then, we apply BERT to get the embeddings. After that, for each e_i we search (using cosine similarity again) the embedding table to find the best-matched q_j and then we activate the corresponding node in the KG.

Figure 6.16 uses two query examples with two different intents to illustrate the parsing process. Note that these two queries each contains several phrases to make the parsing more difficult. In practice, most queries might be much simpler, involving simple action with one or no option (e.g. the dialog example in Figure 6.9).

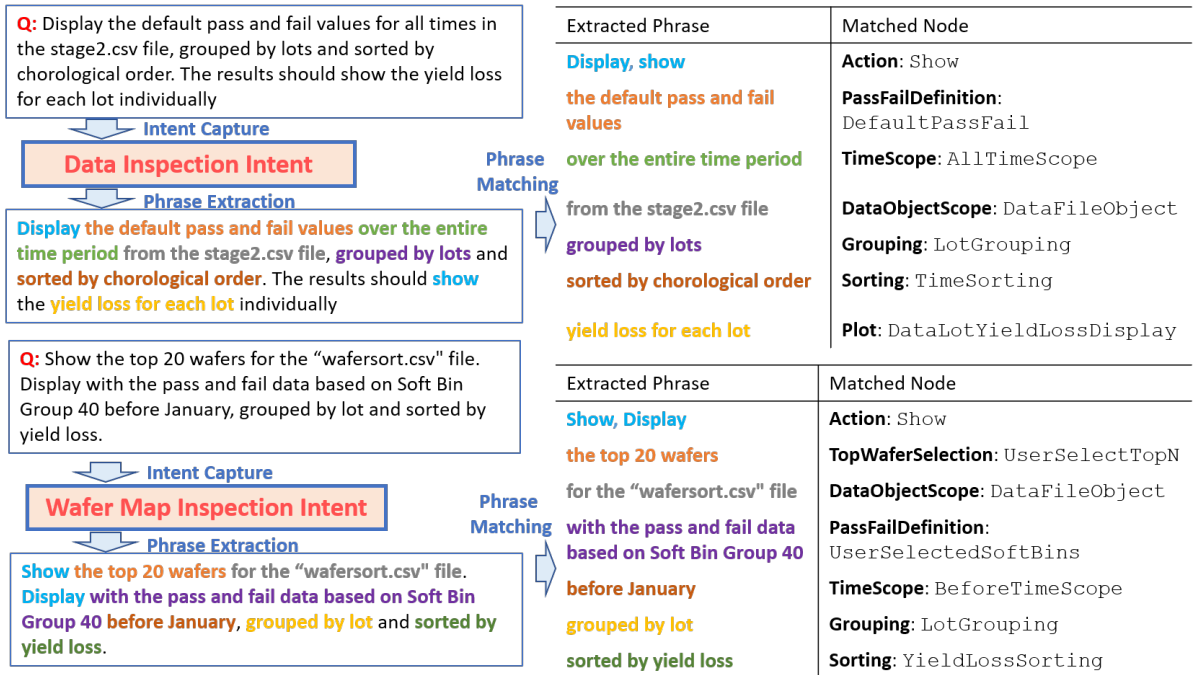


Figure 6.16: Examples of parsing a query into a KG configuration

6.5.3 Dialog representation of an analytic process

In typical ML, a given type of analysis (e.g. classification, clustering, etc.) is applied on a dataset to obtain a result. In contrast, we consider DSML as an iterative exploration process as depicted with Figure 2.22 before. Under our DSML view, in each step the user explores the data from a *perspective*. With IEA-Plot, this perspective corresponds to the configurations implied by the user queries.

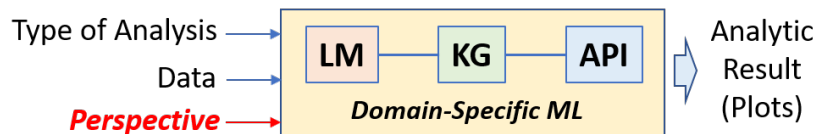


Figure 6.17: LLM enables data exploration through dialog

Figure 6.17 depicts this view. Each exploration process is represented through a dialog. Which analytic results (plots) are meaningful to a user is up to the user to

decide. The job of IEA-Plot is to facilitate the exploration process. LLMs bring two major benefits: (1) enabling dialog representation of an analytic process; (2) improving the efficiency of the exploration.

In IEA-Plot, the KG provides a specification for how a particular analytic tool is used within the application scope. For example, in our KG the concept of a pattern recognition is captured with a high-level **Concept** node called `WaferPatternFilter` (see Section 7.1). It involves the seven pattern concepts mentioned above. A rule-based script is associated with each pattern concept to filter-in wafermaps as a starting point to form a pattern group. These scripts can be replaced with better scripts later. The recognition relies on a MINIONs graph constructed based on our MINIONs-based wafer-to-wafer similarity measure. Others can add new implementations with their own similarity measures. Overall, the KG separates the API implementation from its usage and provides a clear contextual definition for how an analytic tool is used in the application domain.

In other words, the KG in IEA-Plot can be seen as providing a formal functional specification for each analytic tool in the backend, and the KG makes sure that once a tool satisfies the functional specification, the tool can be used in a meaningful way within the application scope defined by the KG.

6.6 A Remark about IEA-Plot

IEA-Plot is designed to enable analytics driven by a dialog. Inputs to IEA-Plot are queries forming a dialog. Outputs are plots. The essential idea for implementing IEA-Plot is the development and use of a KG. The main contribution of IEA-Plot is showing how the KG can be constructed, which captures the domain knowledge in the specific application domain. Our KG together with the front-end parser can be shared with others, providing a platform for customizing their own IEA tool by adding their own

backend API.

The performance of our frontend parser depends on the paraphrasing power of the LLM. How to make an LLM better understand the terms used in our domain and improve such power is a separate research issue. While the current parser is implemented as a constrained parser, as we collect more query examples over usage, it will become feasible to consider other approaches, such as reinforcement learning [178] or joint LLM+KG reasoning [185][186]. Those can be interesting future research directions.

Although IEA-Plot is designed for user to interact with the tool through a dialog, it is possible to add a separate GUI to also enable using the tool's functions through mouse clicks. In this case, the focus might not be on enabling dialog-driven analytics. Instead, it can be for automatic translation of a usage session (i.e. a sequence of mouse clicks) into a natural language description (e.g. a workflow description). In IEA-Plot, each analytic step is represented as a subgraph of the KG and with the generative approach, can be converted into a natural language description. Such automatic workflow capture can be an interesting future add-on feature spanned from our current IEA-Plot design.

Chapter 7

Knowledge Graph

盪胸生曾雲，決眚入歸鳥。會當凌絕頂，一覽衆山小。

Swelling clouds sweep by. Returning birds ruin my eyes vanishing. One day soon, at the summit, the other mountains will be small enough to hold, all in a single glance.

— 《望嶽》 杜甫， A poem from Tang Dynasty

As discussed in the previous chapter, the use of knowledge graph (KG) is the central idea to enable the implementation of IEA-Plot in 2023. This chapter provides more detail of the KG design in IEA-Plot.¹ The initial development of the KG was manual. In Chapter 8, we will then describe the initial work aiming to automate the KG construction process.

¹The KG design is specific for the IEA-Plot version published with the IEA-Plot paper [35] in 2023. Since the publication of the IEA-Plot, the software continues to evolve and be improved over time. The current KG has been enhanced from the original version. Hence, what is being described in this chapter was about the ideas in the original KG design in 2023. The first version of KG is essential to demonstrate the end-to-end capabilities of IEA-Plot. While the KG might not be optimal, it marks a milestone that for the first time, there is a formal representation of domain knowledge.

7.1 Development of the KG in IEA-Plot

As pointed out in [180], the term “knowledge graph” can have various meanings, and the announcement of the Google KG [193] separates its modern views from the historical views. KG is a rich field. Terms (e.g. the term “ontology”) used in the field can sometimes be confusing [180].

In our view, the field includes two distinct uses of KG. One is for representing and organizing the knowledge from vast amounts of data such as data available from the Internet. RDF [194] (for data representation), and RDF Schema (RDFS) and OWL [195] (the ontology languages) are standards for this purpose. The other is for representing and organizing people’s knowledge. ConceptNet [188] is a popular example. ConceptNet does not use a complicated ontology (like OWL). It includes 35 well-defined relations to connect common sense “concepts”. ConceptNet is suitable for representing the graph nodes with *distributional vector embeddings*, making it suitable for use with an LLM to perform joint reasoning [185][186][187].

Given our multiple purposes to use KG from the three perspectives described in Section 6.1.4, we need a hybrid model somewhat between RDF/OWL and ConceptNet. On one hand, we need a KG capable of modeling the data in our tool. On the other hand, we need the KG simple enough to enable LLM+KG joint reasoning in the future.

7.1.1 The KG design

To avoid confusion, Figure 7.1 clarifies the formalism of our KG design and the terminology in use. A graph is a collection of triples (*source, predicate, target*). The KG comprises two separate graphs: the *domain graph* and the *data graph*. When we refer to the term *ontology*, we mean the domain graph. The domain graph provides our interpretation to process the data graph, and this processing includes the three

perspectives mentioned before (Section 6.1.4).

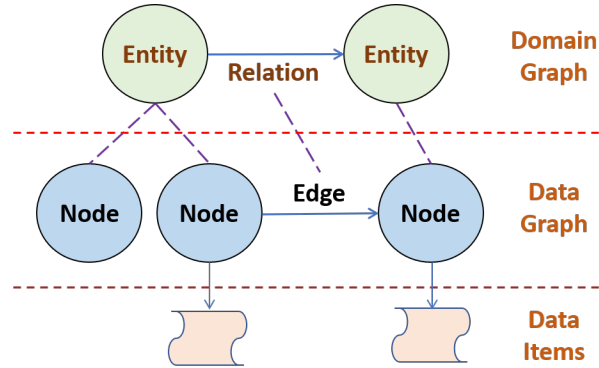


Figure 7.1: Hierarchy in our knowledge graph design

In the *domain graph*, sources and targets are called *entities*. Predicates are called *relations*, which are always directional. In the *data graph*, sources and targets are *nodes* and predicates are *edges*. A node is an instance of an entity and an edge is an instance of a relation. These usages of the terms are consistent with those described in [180].

A node can have a list of *data items*. There are two types of data items stored with a node. The first is to store data objects relevant to the execution of the tool. The second is to store example phrases/sentences that represent the semantic meaning of the node.

7.1.2 The ontology

In the field of KG, an *ontology* is a concrete, formal representation of what terms means in the given domain [180]. Figure 7.2 depicts our ontology as axioms in the domain graph. A triple $(Entity_1, Relation_a, Entity_2)$ means that a node of $Entity_1$ and a node of $Entity_2$ can have an edge of $Relation_a$. If a triple $(Entity_1, Relation_b, Entity_2)$ is not present in this graph, then it means the two nodes cannot have an edge of $Relation_b$. Hence, the graph provide the axioms that triples in the data graph have to satisfy.

The semantics of the domain graph can be explained as the following. We divide nodes

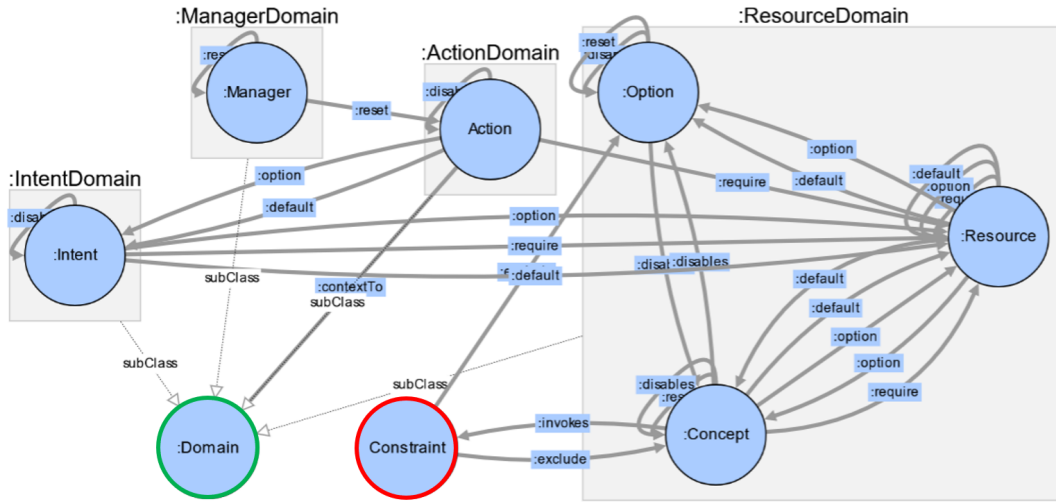


Figure 7.2: Axioms in the domain graph, representing the ontology we use

into four domains: **Manager**, **Action**, **Intent**, and **Resource**. These four entities are all subclasses of the parent entity **Domain**. Inside the resource domain, there are two types of nodes: **Concept** and **Option**. In addition, there is a special type of node called **Constraint** which is used to capture constraints between concept-concept and concept-option pairs, whenever needed. In total, our ontology defines 8 entities (types of nodes).

The ontology further defines eight types of relations. Below we use the term “activate” to mean that during subgraph extraction, a node is included in the current subgraph.

1. (source, **require**, target): When the source is activated, the target is required to be activated.
2. (source, **default**, target): When the source is activated and no option node is activated by the user query, the target is the default and is activated.
3. (source, **option**, target): The target can be activated as an option when the source is activated.

4. (source, **reset**, target): When the source is activated by the user query, all previously-activated options from the target are reset.
5. (source, **disable**, target): When the source is activated, the target is deactivated. This can be used to model mutually-exclusive activations among nodes.

The sixth relation is called **contextto**. Its usage is specific to the form (**Action**, **contextto**, **Domain**). This is used to model *domain switching* in a dialog. For example, when the analytic context switches from the context of analyzing wafer sort data to the context of correlating between wafer sort data and E-test data, it involves a domain switching.

The last two relations are **invoke** and **exclude**, specific for the use to model node-node constraints. (**Concept**, **invoke**, **Constraint**) means activation of the concept node will invoke the constraint. (**Constraint**, **exclude**, node) means the constraint excludes the activation of the node which can only be a node of either **Concept** or **Option**.

7.1.3 Key ideas for constructing the data graph

Figure 7.3 uses a conceptual example to illustrate the key ideas for constructing the data graph². Each box in Figure 7.3 represents a resource domain that contains a subgraph. There are two types of edges shown in the figure. A solid edge is a **require** edge. A dash edge is not a require edge. There are three example queries where for each, the activated subgraph is illustrated.

Consider the first query “*Show me the yield loss*”. This query activates the *Show* action node. It is determined (intent determination will be discussed later) that the intent is to *inspect* the data from the “yield loss” perspective. This inspection requires

²The data graph is evolving. Our current data graph contains 243 nodes with 1147 edges, which can be accessed from our IEA project page: <https://iea.ece.ucsb.edu/iea/project>

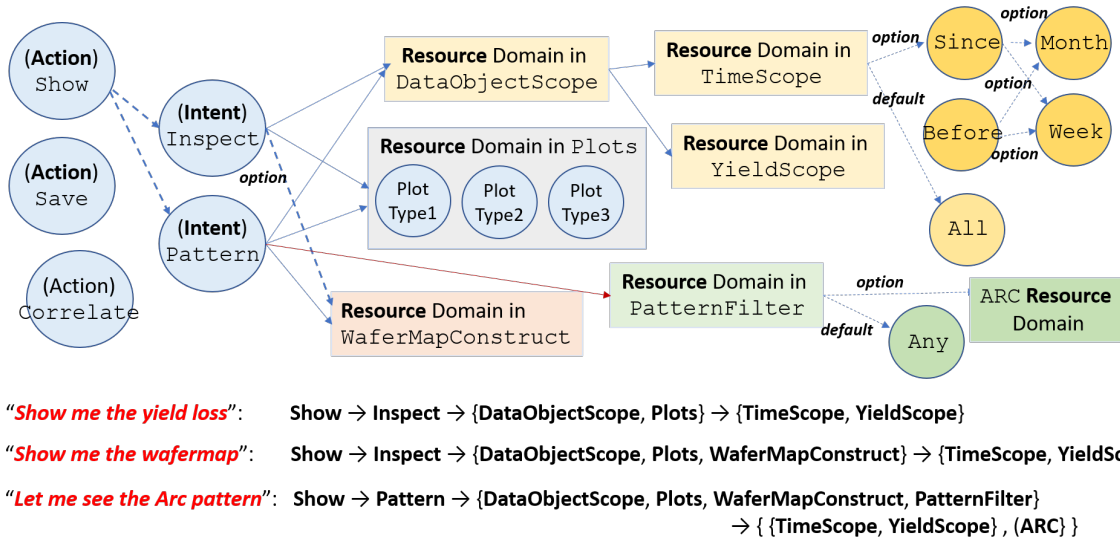


Figure 7.3: A conceptual example to illustrate our graph construction

performing two tasks: (1) Select the data scope to inspect. This is represented by the **require** edge pointing to the `DataObjectScope` resource domain. (2) Select a plot type for the display. This is represented by the **require** edge pointing to the `Plots` resource domain.

The data selection task further requires resources from two domains: `TimeScope` and `YieldScope`, where the first allows selecting a time interval based on month or week, while the second allows selecting data based on a yield threshold (not shown in the figure). Because the query does not specify a time scope, the default is `All` which is pointed by the `TimeScope` through a **default** edge.

Similarly, `Plots` domain contains a list of plot types and may include a default (e.g. `Type1`). Defaults and options are modeled in the subgraph of the `Plots` domain.

The second query requires an additional resource, `WaferMapConstruct`, used to determine how the wafer maps are constructed (e.g. a pass/fail wafermap, a stacked wafermap, a wafermap after some filter, etc.). The second query also requires all resources required by the first query. As a result, all data and options collected from the first query is

inherited by the second query as its starting point.

The third query asks to switch *context* from data inspection to do some wafermap pattern analysis. *Context switching* in our design means to switch from one intent to another. The third query requires one additional resource `PatternFilter` (Examples of using this resource domain have been discussed in Section 6.3 before in the context of wafermap analytics). Because the query asks to see specifically the “Arc” pattern, this triggers the option to request the `ARC` resource.

The API perspective

From the API perspective, **Action** nodes correspond to the steps in the “main” program. Each resource domain corresponds to a portion of the API functionality. For example, the API support various ways to select the data. How to call those functions with what options are organized in the knowledge subgraph within the `DataObjectScope` domain. In our KG, each node in a resource domain (**Concept** or **Option**) corresponds to an available function in the API. Their dependency structure is modeled in the knowledge subgraph in the domain.

The KG manager perspective

The KG manager is responsible for managing the extracted subgraphs from one query to the next. An extracted subgraph corresponds to a *configuration* telling how to call the backend API.

A configuration contains a list of admissible actions with their options as shown in Figure 6.4. An action can be thought of as a function call. From the API perspective, function calls are organized in hierarchy (e.g. calling a function a_i may involve calling other functions a_{i_1}, a_{i_2}, \dots). This hierarchy is reflected in the subgraph. In other words, our KG contains the knowledge of the API organization.

From the KG manager’s perspective, the three relations, **require**, **default**, **option**, are essential for modeling the functional dependency structure. The three relations, **reset**, **disable**, **contextto**, are used for managing the change of subgraph from one query to the next. Their semantic meanings are illustrated through examples in Figure 7.4.

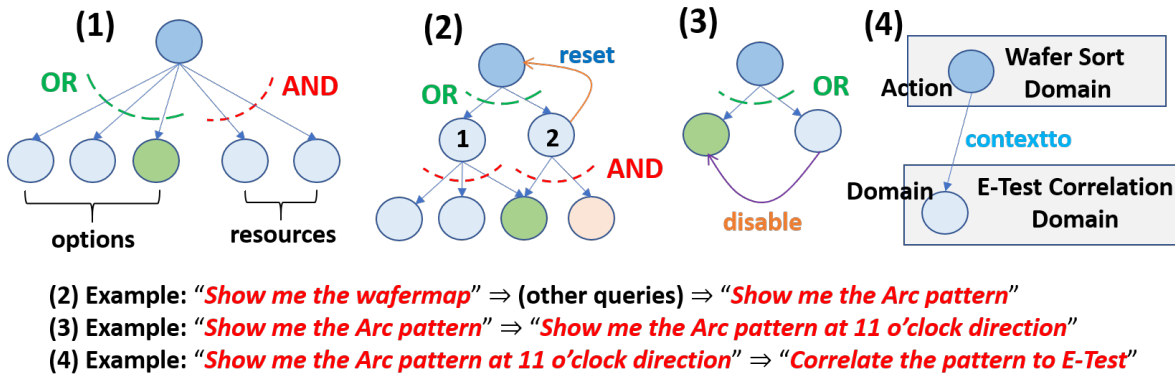


Figure 7.4: Semantic meanings of the six relations for subgraph management

Example (1) in Figure 7.4 illustrates that a node in an extracted subgraph in general can have two parts of child nodes: the OR part and the AND part. Edges in the OR parts are of **option** type where one of them is **default**. Edges in the AND part are of **require** type. For example, to execute the function represented by the parent, it requires certain specific resources and can have the various options. In this case, we can use **default/option** to model the options and use **require** to model the resources.

Examples (2)-(4) then illustrate how the KG manager handles subgraph change from one query to the next.

In example (2), the first query asks to see “wafermap”. This corresponds to the intent of “wafermap inspection”. Suppose node “1” models this intent. After that, there are other queries within the scope of this intent. Then the last query asks to see “Arc pattern”, triggering a new intent “pattern analytics”. Suppose node “2” models this intent.

Activation of node “2” triggers a *switch of intent*. This is modeled through a **reset** relation from node “2” to its parent. The parent previously maintains the current configuration resulting from queries before the intent switching. The activation of node “2” therefore **resets** the configuration of its parent, telling it to compute a new one from scratch.

Consider the **green node** in example (2). It is a shared resource between the two intents. Within the scope of the first intent, those queries may have set the available options under the **green node** (not shown). Without a reset, a subsequent query would inherit those options. The reset notifies the manager to restore everything back to its default. For the KG manager, options choices are handled cumulatively from one query to the next, until it encounters a **reset**.

Example (3) shows a situation where a previously-selected option is replaced with a newly-selected option. The first query asks to see “Arc pattern” but does not specify a direction. Hence, the default is to include *all* directions. This default is the **green node** in example (3). The second query then provides a specific direction “at 11 o’clock”. This option replaces the previous option, and is modeled as a **disable** relation. In general, the **disable** relation can be used to model a set of mutually-exclusive options.

Example (4) shows an example of *domain switching*. The first query is in one domain, operating on one dataset, the wafer sort dataset. The second query involves E-test data. In our KG, we consider the two queries belonging to two separate domains. The second query triggers a switch from one-dataset analytics to cross-dataset analytics.

Domain switching is modeled through a **contextto** relation. For example, the **Action** node in the wafer sort domain is labeled as the **Show** action. When the correlation action is recognized, it invokes switching to the E-test correlation domain that contains the action node **Correlate**.

The **contextto** relation tells the manager to (1) bring in the E-test dataset and (2)

transfer the *current analytic result* (e.g. pattern groups) from the wafer sort domain to the correlation domain. As discussed in [96], our interpretation of E-test correlation is based on a given set of pattern groups.

The query perspective

Figure 7.5 shows a sketch of the subgraph for the first query in example (4). As discussed in Section 7.1.1, example phrases can be attached to a node. Figure 7.5 shows four such nodes.

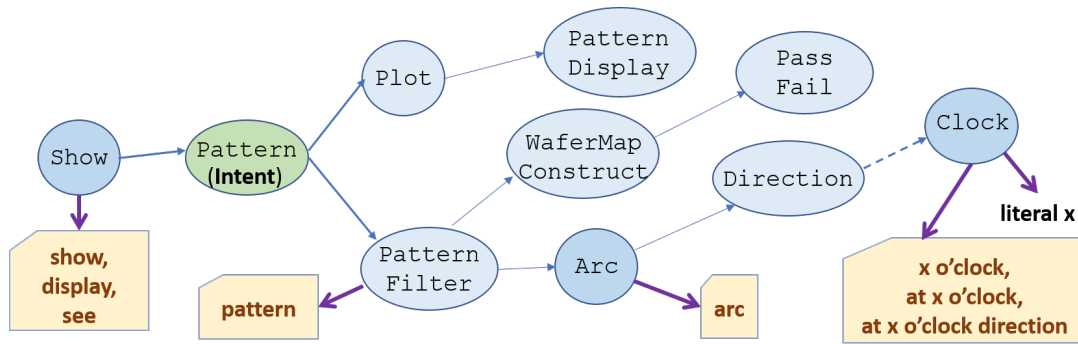


Figure 7.5: Subgraph for “*Show me the Arc pattern at 11 o'clock direction*”

Given the query, the parser’s job is to determine that these four nodes should be activated. Then, the KG manager can extend from the activated nodes to obtain a subgraph (by following the **require** edges).

In a simple way, we might think that the four nodes can be activated by matching texts in the query to the phrases attached with those nodes. For example, *Show* in the query matches “show” in the **Show** node. The text *at 11 o'clock* matches the “at x o'clock” in the **Clock** node. While text matching can be used, it can substantially limit the scope of the acceptable queries to our tool.

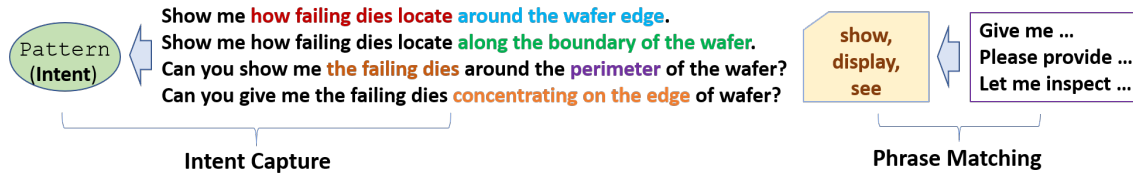


Figure 7.6: Intent Capture and Phrase Matching for parsing a query

Detail about intent capture and phrase matching

Figure 7.6 shows that the parser’s job includes two aspects: *intent capture* and *phrase matching*. Consider again the first query in example (4) in Figure 7.5. For intent capture, the parser needs to know to activate the **Pattern** node. This might be doable by matching the word *pattern* to the text “pattern” in the **PatternFilter** node. However, consider the four queries shown in Figure 7.6. Those queries imply to see failing patterns as well, but none of them mentions the word “pattern” or “arc”.

Furthermore, the four queries imply looking for a pattern along the wafer edge. If our tool is smart enough, it should know that this includes the **Arc** pattern as we have defined it in our KG. This means that we need to match “arc” with phrases like *wafer edge*, *boundary of the wafer*, *perimeter of the wafer*, etc. We call it the *phrase matching* problem.

As discussed in Section 6.5 already, intent capture and phrase matching are the two problems where we leverage the power of LLMs [173][19][191] for tackling the problems.

The constraint graph

In our KG, a separate *constraint graph* is maintained using **Constraint** nodes and **invoke/exclude** relations. Figure 7.7 illustrates their use.

In the KG, each resource domain comprises three types of nodes. An **Entry** node is an instance of the **Resource** entity. Inside a domain, an **Option** node is a node with an

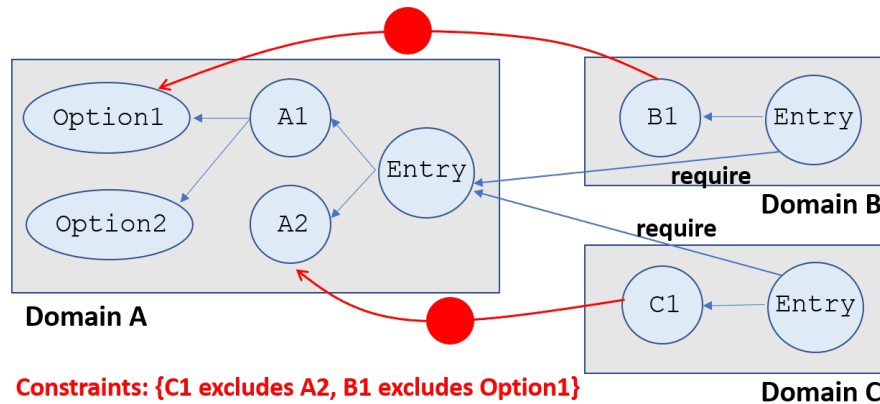


Figure 7.7: A constraint graph is separately maintained in our KG

incoming **option** edge and without any outgoing **option** edge. Other nodes in a domain are instances of the **Concept** entity.

Suppose resource A is shared by both B and C. This is modeled by the two **require** edges from the **Entry** node of B and from the **Entry** node of C, both to the entry node of A. Domain A contains two concepts, A1 and A2, where A1 can have two options, **Option1** and **Option2**. Suppose in our backend API, B1 can be used with A1, but when that happens, cannot have **Option1**. Also, C1 cannot be with A2 (if we think C1 and A2 as two functions, they cannot be called together due to our API design).

To model such constraints, we create constraint nodes with constraint edges as shown in the figure. The red circles are instances of the **Constraint** entity. Each red edge comprises an **invoke** and an **exclude** relation.

For example, domain A can be resources for making plots. Domain B is for inspecting patterns on wafermaps. Domain C is for inspecting wafermaps in general. A1 and A2 are two plot types. While A1 can be used to display wafermaps, **Option1** is not valid when the wafermaps are given as pattern groups. Also A2 is a plot that can only be used when wafermaps have already had a pattern group label assigned. To model this knowledge about the API usage structure, we put in the two constraints.

Comments on the scalability concern

An important consideration in our KG design is scalability. Figure 7.8 shows that extending the tool’s functionality can be achieved by adding nodes (and edges) to the KG. This extension can be considered at three levels: **Option**, **Concept**, and **Resource**.

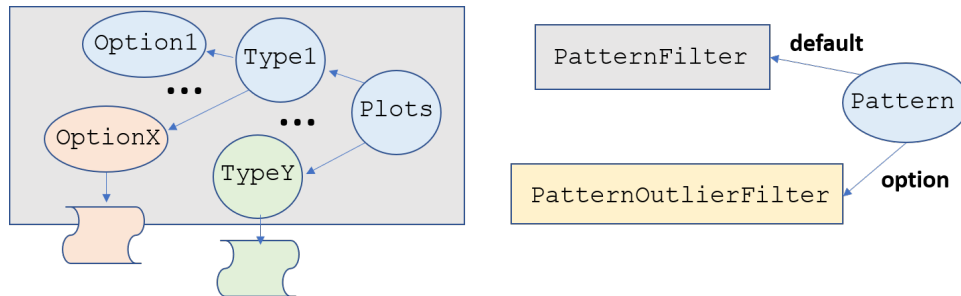


Figure 7.8: Extending tool functionality by adding nodes in KG

For example, suppose for a particular type of plot we desire to add a new option. With our IEA tool design, we can simply add the option to our API. Then, we add a node `OptionX` to represent this option, and we add a list of example phrases for activation of this option node.

Similarly, if we desire to add a new plot type, we can add a new **Concept** node, say `TypeY` (and its available options as **Option** nodes) in the `Plots` domain, with a list of example phrases for activation of the concept node.

In our KG design, the `Pattern` intent node requires resources from the `PatternFilter` domain. For pattern analytics, our current implementation looks for *systematic patterns*. Suppose we desire to include a new way that checks for “outlier” patterns, i.e. a pattern that is both significant and unique. We can add this functionality into the API. On the KG, we can create a new resource domain, say `PatternOutlierFilter`. Then, we can make the `PatternFilter` domain as default and `PatternOutlierFilter` as an option.

As seen in the above examples, extending the capabilities supported by IEA-Plot can be done by working within a focused scope of the KG. This locality feature provides a

major benefit which facilitates scaling of IEA-Plot’s capabilities over time.

7.2 The Importance of Having the First KG

The KG construction in the development of IEA-Plot is manual. As explained in Section 6.2 and in Sections 7.1.3 to 7.1.3 above, our KG is supposed to capture the knowledge from three perspective: the LLM (or query) perspective, the backend API perspective, and the perspective for managing the KG itself. The KG described above might look like a result from extensive engineering efforts. This was indeed the case. The development process took several months and went through many iterations to adjust the KG design (i.e. deciding on the ontology, the node types, the edge types). It was a tedious process that demanded much domain knowledge.

It should be noted that the manual process for developing the KG in IEA-Plot was a necessary step. Without seeing a first version of KG that was sufficient to enable a complete product like IEA-Plot, we had no reference to the scope of the KG construction problem. It would have been meaningless to consider automation of the KG construction when we did not even know what an acceptable KG should look like. Hence, while the KG described in Section 7.1 might look somewhat “ad hoc”, its contribution to the overall research is not. The KG not only enables the completion of IEA-Plot but also serves as the stepping stone for the next phase of the IEA development.

Once we have the first KG, we can then ask two follow-on questions:

1. From a formal language point of view, what type of domain knowledge is supposed to be captured in our KG?
2. If we are going to build a tool to alleviate the KG construction process, what should the tool look like?

The first question asks for a formalism regarding the domain knowledge that is supposed to be captured in our KG. The formalism provides clarity on the scope of the knowledge and helps make the KG development process become more formal. The second question asks for a tool that can provide *assistance* to the KG construction process. To develop such a tool, we need to understand what aspects the tool should be designed to assist on and on those aspects, to what extent the assistance should be offered.

7.3 A Formalism Regarding the KG in IEA

Abstractly, a KG in IEA comprises a collection of *concepts*. Each concept has a corresponding *DSML oracle* to decide on its existence. Formally, we can say that a concept is a Boolean variable and its DSML oracle makes an assignment of 0 or 1 to this Boolean variable based on a given input dataset. Figure 7.9 depicts this abstract view.



Figure 7.9: An DSML oracle decides on the value of a Boolean concept x

With this view, we can then think that our KG represents a collection of *knowledge statements*, where each statement can be stated as a Boolean formula based on k Boolean variables. Each variable corresponds to a concept. Note that in practice, k would not be a large number and hence, we can generally treat k as a constant. Once we use a Boolean formula to represent a knowledge statement, the the underlying question is, does this formula involve *quantifiers* (\exists, \forall) or not? If we do not allow quantifiers, then the Boolean formula essentially represents a statement in *propositional logic*. If we do allow quantifiers, then we basically allow our knowledge statements to be stated based on a language structurally similar to the language of polynomial hierarchy (PH) (see discussion in Section 3.12.1).

In Section 3.12.2 before, we have discussed that the analytic problem solved by IEA can be considered as the problem class $O(\text{poly}(n))^{DSML}$, in view of the $\Delta_2^P := P^{NP}$ optimization problem class. Note that the $O(\text{poly}(n))^{DSML}$ problem class is defined from the *solving* perspective. The $O(\text{poly}(n))$ is to denote that the number of search steps is bounded by $O(\text{poly}(n))$. This is reasonable as in practice, i.e. the number of search steps will not be exponential. And if consider each search step is based on a selected dataset and a question (which corresponds to a knowledge statement generated from the KG), then we are saying that the number of (dataset, question) pairs is not exponential in one Choose-and-Bound search session.

From the *representation* perspective, the $O(\text{poly}(n))^{DSML}$ class seems to limit the knowledge statement from using quantifier. However, as explained below this is not the case and we can allow a language similar to the PH (Section 3.12.1) to describe knowledge statements.

Analogous to the optimization hierarchy $\Delta_{i+1}^P := NP^{\Sigma_i^P}$ discussed in Section 3.12.1, we can therefore define the *DSML hierarchy* (DSML-H) as $\nabla_{i+1}^P := P^{\Xi_i^P}$ (we use the symbol Ξ as analogous to the symbol Σ in the PH definition), where $\Xi_{i+1}^P := P^{\Xi_i^P}$, i.e. Ξ_{i+1}^P has access to an Ξ_i^P oracle and with that access, the complexity from the verifier’s perspective remains P (instead of NP as that in PH). We have $P^{\Xi_2^P}$ to be $O(\text{poly}(n))^{DSML}$ which is the IEA problem class originally defined (DSML oracle includes ML^* and $Co-ML^*$, see equation 3.1 in Section 3.12.2).

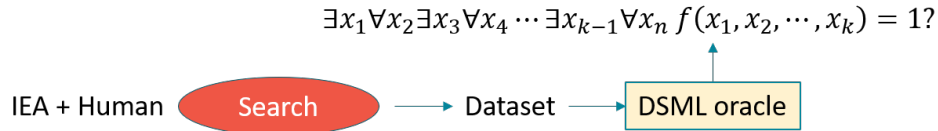


Figure 7.10: The DSML hierarchy where k can be treated as a constant

From the representation point of view, we are saying that a knowledge statement is a

Boolean formula $f(x_1, \dots, x_k)$ with k concepts where k is a constant (and without loss of generality, assume that k is an even number). Hence, we can consider 2^k as a constant. The evaluation of f can be based on a number of alternating quantifiers, as exemplified in Figure 7.10. The assignment of each variable x_i is based on its corresponding DSML oracle and the given dataset. In other words, the overall problem statement can be stated as: Find the dataset that satisfies f .

Because k is a constant, given the corresponding DSML oracles and a search space of a polynomial number (i.e. $O(\text{poly}(n))$) of possible choices for the dataset, deciding if $f() = 1$ is at the complexity level $P^{\Xi_2^P}$ when no quantifier is involved. Then, adding the quantifiers bring up the complexity into the DSML-P, which even though is more complex, is still $O(\text{poly}(n))^{DSML}$ because at each level of hierarchy we have $\Xi_{i+1}^P := P^{\Xi_i^P}$.

The DSML-H class can be seen as a formal language that limits the complexity of knowledge statements that can be derived from a KG in IEA. In practice, a KG can be designed to limit the knowledge statements in a lower class in DSML-H, e.g. allowing no \forall quantifier or allowing only one \forall quantifier. Also in a KG design, one can even limit the Boolean formula to a certain type, e.g. no negation, no disjunction, or limiting to monomials only. These are two sets of orthogonal choices to consider when developing a KG in IEA (i.e. one on the quantifiers and the other on the Boolean formula itself). Regardless of the choices, the KG design would not consider knowledge statements beyond DSML-H and from the solving perspective, the complexity is $O(\text{poly}(n))^{DSML}$.

To summarize Figure 7.10, the only places we need to deal with the “exponential” complexity are inside those DSML oracles. The knowledge statement is represented as a Boolean formula. And because the formula has a constant size, regardless of using quantifiers or not, it adds a constant (could be a large constant) complexity to the solving. The search space has a polynomial complexity, i.e. a polynomial number of choices for selecting a dataset. Therefore, the overall solving complexity remains to

be $O(\text{poly}(n))^{DSML}$, regardless of the complexity of the Boolean formula. And if we consider the *DSML* complexity as a constant (because they are treated as oracles), then the overall complexity from the checker perspective is $O(\text{poly}(n))$. This is an important observation because if this is true, then it provides a hope for later pursuing a generative AI approach where we can then ingest the knowledge stored in KG into an LLM and train the LLM to become our solver.

The formalism provided in this section is not trying to provide a definition for the language to construct a KG in IEA. Rather, we provide the formalism as a way of thinking about the KG in IEA. To construct a KG in IEA, we need to think about the knowledge statements intended to be supported by IEA and develop the concepts and the corresponding DSML oracles accordingly. Most importantly, as discussed above the KG specification language can be complex (in DSML-H) and yet, because each formula is bounded by using a constant number of concepts, the specification complexity does not make solving complexity more complex from a theoretical complexity sense. Hence, the language for KG can be more flexibly defined without affecting the solving complexity which will remain to be $O(\text{poly}(n))^{DSML}$.

The discussion also shows that the complexity that really matters for solving an analytic problem, is inside those DSML oracles and in a sense, those DSML oracles “hide” the real complexity from the rest of the solving process. With those DSML oracles in place, since the rest of the solving has a polynomial complexity we have hope to train an LLM as the solver (because this solver only needs to deal with polynomial complexity). From this angle, we see that in order to train an LLM to be the solver for solving an analytic problem, we need to define the set of concepts and their DSML oracles to begin with. In other words, we need to provide an assigned interpretation for all the domain-specific terms used in an input statement to an LLM, in order for the LLM to solve the problem.

7.4 Toward Automating KG Construction

In this section, we describe an experiment trying to use GPT3.5 for automatic knowledge graph construction. The study was documented in an arXiv paper [196]. We used RISC-V unprivileged ISA specification (“RISC-V Spec”) [197] as an example. We tried to study how GPT3.5 performed in the specific KG construction task. Most of the analyses were done manually. In this study, our goal was to understand the problem, rather than proposing a way to solve the problem.

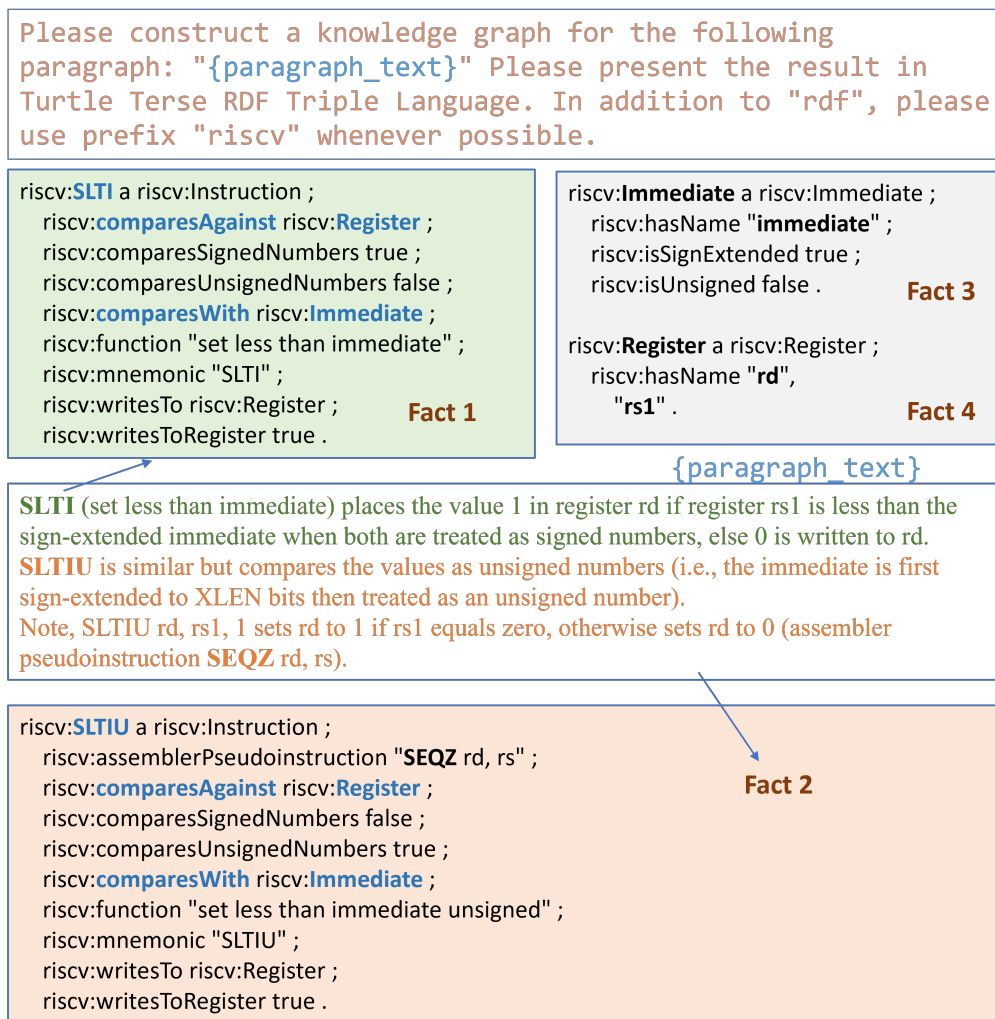


Figure 7.11: An example of KGC using a prompt to GPT3.5. The input is a paragraph from the RISC-V Spec and the output is in RDF TTL format.

7.4.1 The experiment setup

We consider KG Construction (KGC) as the following: Given an ordered sequence of *text items* T_1, \dots, T_n , KGC processes one T_i at a time from 1 to n and generates an individual knowledge graph (KG): $g_i = KGC(T_i)$. Let G_{i-1} be the KG after *merging* all g_1, \dots, g_{i-1} . We obtain $G_i = MERGE(G_{i-1}, g_i)$. From the RISC-V Spec, we consider each paragraph as a text item. We choose GPT3.5 [16] as our LLM to use. For implementing the *KGC* step, we rely only on prompting to the LLM.

Figure 7.11 shows an example of the KGC. The KG is represented in the Turtle Terse RDF Triple language (RDF TTL) [198]. The specific prompt in use is shown in the figure. The input paragraph can be seen in two parts: (1) definition of the SLTI instruction, and (2) definition of the SLTIU instruction.

Notice that the second sentence starts with “SLTIU is similar” by referencing to the first sentence. The RDF output is shown as four “Facts”. The most interesting aspect of the result is shown in “Fact 2” for SLTIU where the RDF duplicates all the predicates used in “Fact 1”, i.e. `compareAgainst`, `comparesSignedNumbers`, `comparesUnsignedNumbers`, and `comparesWith`. This indicates that the LLM does understand the phrase “is similar” and reflects its understanding in copying the RDF representation of SLTI.

Terminology

We will use several terms in this section to help the discussion. Refer back to Figure 7.11. An RDF output is given as multiple *rdf blocks*. We call each rdf block a *Fact*. Each Fact starts with a *subject entity*. For example, `SLTI`, `SLTIU`, `Immediate`, and `Register`, are subject entities. A Fact represents a set of *triples* each in the form $(subject, predicate, object)$. An *object entity* is the one that appears as the object of

a triple and is *not* a subject entity. We also differentiate two types of predicate. For a triple, if both of its subject and object are subject entities, we call the predicate a *relation*. Otherwise, we call it a *feature*. For example, in Figure 7.11 `compareAgainst` is a relation and `comparesSignedNumbers` is a feature.

For simplicity, we use the term “RDF” to refer to the RDF output given by GPT3.5.

Background facts (BFs)

While Figure 7.11 shows some encouraging result, we notice that the RDF misses some detail in the original paragraph. For example, it does not differentiate the usages of the two registers “rd” and “rs1”, i.e. which is the source and which is the destination.

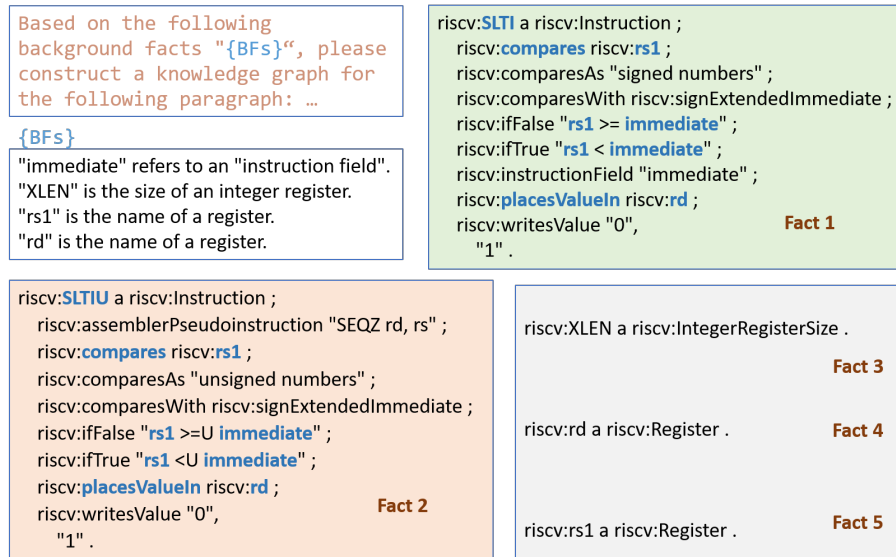


Figure 7.12: Improved RDF by repeating the example in Figure 7.11 and supplying the *background facts* (BFs).

Figure 7.12 shows another RDF by using a different prompt. This prompt adds a list of *background facts* (BFs) we manually created. The RDF shows an improvement. In particular, the RDF shows that the instructions `compares` register “rs1” with the `signExtendedImmediate`, and `placesValueIn` “rd”. It even includes the detail that the comparison is “<” (less than).

Focus of the study

The examples above show that GPT3.5 alone *can* be used for KGC and produce reasonably good result. Adding BFs can help improve the result further. It looks promising. However, it turned out that without adding BFs, the example was one of the few easy cases we encountered. Others were not as easy. Without BFs, an output RDF could be unsatisfactory due to two reasons: (1) the RDF failed the syntactic check, and (2) the RDF passed the syntactic check but either got some fact(s) wrong or entirely missed the main fact(s) described in the paragraph. In our study, we observed at least 70% of the cases were in these two categories.

Figure 7.12 shows that adding BFs can influence the behavior of GPT3.5 for KGC. Then, it is interesting to see whether we can rely solely on adding BFs to reach a satisfactory RDF for every paragraph or not. Ideally, we would like to turn the KGC process for every hard case into an easy case (like Figure 7.12).

It is important to note that, in our study we did not concern where those BFs come from or what BFs to use. The study focused on one question: If GPT3.5 could not produce a satisfactory KGC result for a text item, does there exist a list of BFs to be added in the prompt such that the KGC result would become satisfactory?

7.4.2 Main finding from the study

Based on the paragraphs from the RISV-V Spec, we concluded from our study that the simple approach of adding BFs would be sufficient to enable GPT3.5 to produce satisfactory KGC result for every paragraph we had tried. This finding essentially implies that the core of the KGC problem is the problem of finding and applying the proper BFs for a given paragraph.

7.4.3 Related works in KGC

As far as we know, we are the first to tackle the problem of KGC for unstructured text data in the semiconductor chip design domain. Documents in a semiconductor company often use terminologies not known to the outside world. Hence, it is intuitive to think that KGC for those texts requires a domain person to provide some background knowledge to at least cover those terminologies. This thinking motivated us to take the view of supplying BFs.

Knowledge graph construction is a rich field with many techniques having been proposed to *solve* the problem [199, 200]. Conventional methods to constructing knowledge graphs follow a pipeline of NLP sub-tasks [201] such as entity recognition [202], entity linking [203], relation extraction [204], and coreference resolution [205] etc. Among the various tasks, named entity recognition (NER) provides a fundamental first step for domain knowledge acquisition. Standard off-the-shelf toolkits for NER [206, 207, 208, 209] combines machine learning models and rule-based components to label entities. Recent works solve the problem in an end-to-end fashion using deep learning models [210, 211, 200]. However, pre-trained models often incur low accuracy since training data from general public domain rarely covers our domain specific patterns. It is hard to iteratively ingest domain experts' knowledge to further improve the accuracy without dedicated retraining. Finetuning or retraining is often not a desirable option for many hardware companies because of the tremendous efforts required to create curated databases for the training tasks. Another potential route to our KGC problem is to implement customized rule-based extractors with features provided by existing constituency and/or dependency parsers [212, 213]. However, to our experience the set of rules can quickly grow overly-complicated and it is difficult for the approach to scale.

Restricting the output of a generative LM into a formal representation is related to the

problem of constrained semantic parsing [214, 215]. A structured meaning representation is often chosen as the output format [216]. However, converting the meaning representation into a KG can be another potential barrier. In summary, prompting an LLM to directly generate an RDF, if practical, can bypass all the complications mentioned above. This would not be feasible without the latest developments of the LLMs.

Despite that automatic knowledge graph construction in specific domain still remains an open challenge problem [199], a major difference of our work (and our objective) is that we are not trying to propose another KGC solver. Rather, we focus on verifying the result given by such a solver, in our case the GPT3.5. In other words, our work is *not* about being a technology provider as those surveyed in [199, 200]. Instead, we take the perspective of being a technology consumer.

If we look at it from the computational complexity perspective discussed in Section 3.12.1, specifically the perspective of the *oracle machine* [114], basically we treat GPT3.5 as an oracle for KGC. However, this oracle is not perfect. Hence, given the oracle, we are interested in knowing what can we do to make the oracle generate satisfactory results for us. And our answer is that, we need to provide the oracle with the proper BFs in order for it to do the job.

7.4.4 Taking an oracle-checker view

Our Oracle-Checker (OC) view was inspired by the theoretical model of Interactive Proofs (IP) [217]. An IP system comprises a *prover* and a *verifier*. The prover is assumed to be an all-powerful Turing machine. The verifier is a probabilistic Turing machine, with limited power like a solver in P (polynomial time). The IP approach was developed to characterize computational complexity classes.

In an IP system, the verifier interrogates the prover through a sequence of commu-

nications. At the end the verifier either is convinced that the answer provided by the prover is correct or reject it. An important aspect in the communications is to keep the prover honest. Because verifier’s computational power is limited, it is mostly prover’s job to make the verification task as easy as possible.

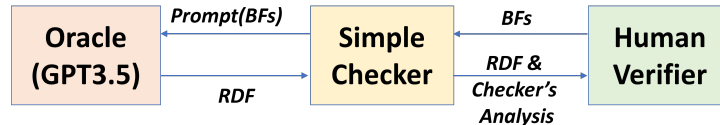


Figure 7.13: The KGC study is based on the oracle-checker view

Our OC view is different from the IP approach though. This is because what we consider the oracle (corresponding to the prover) has a limited power in practice. In addition, our oracle is probabilistic. On the other hand, our checker still needs a way to keep the oracle “honest”, i.e. a way to verify the answer provided by the oracle. However, because the oracle is not all powerful, we can no longer expect the oracle to give a form of answer that is always easily verifiable. This makes the verification harder than that in the theoretical IP model.

Figure 7.13 depicts our OC scheme used in this initial study. In our scheme we introduce a *human verifier* because we do not know yet how to make an automatic checker. Our “verifier” comprises two parts: the simple checker and the human verifier. The ultimate decision to accept or reject an answer (RDF) stays with the human verifier. The job of the simple checker is to analyze the answer and provide feedback to help the human verifier.

In this initial study, we keep the checker as simple as possible. Hence, much of the verification job is done by the human verifier, e.g. manually. Our goal for the study is to gain a good picture for what does it take for the verifier to complete the job. After getting this picture, we can then think about how to automate the work performed by the

human verifier as much as possible. This aspect will be discussed in detail in Chapter 8.

Instead of asking the oracle to make the verification task easier for the verifier, if needed we require the verifier to make the task easier for the oracle. In our OC scheme, there are only two ways the verifier can do this: (1) by providing BFs and (2) occasionally by splitting a paragraph into multiple sub-paragraphs.

Note that making the KGC task easier is opposite to that in the theoretical IP model where the task is made easier for the verifier. Therefore, we can think that in our OC scheme, the oracle is powerful but not all powerful, and some of the power still resides with the verifier.

7.4.5 When an LLM can be used as an oracle

We impose two requirements for an LLM to be used as an oracle in our OC scheme. First, the LLM needs to have the ability to support the performance of some *validity check* on the answer it provides. In this initial study, we simply ask the LLM to perform the check entirely. Later in Chapter 8, we will show how a checker can construct “more intelligent” questions and use the LLM to perform a check.

For each RDF Fact, our simple checker asks the LLM to perform an *entailment check*, asking whether or not the Fact can be logically entailed from the paragraph (and if BFs are provided, with the BFs as well). If this check passes for every Fact in the RDF, the checker accepts. Otherwise, it rejects. Then, the checker reports the result to the human verifier for review.

The second requirement is straightforward, asking for a consistent answer. The oracle must be able to demonstrate a *consistent* behavior in N repeated runs of a prompt. Because of the probabilistic nature of an LLM, it is possible that in repeated runs, no two answers are exactly the same. In this case, we consider the LLM failing the

consistency requirement. In our experiment, if the LLM could produce at least two exact same answers in 10 repeated runs, we consider it satisfying the consistency requirement for the given KGC task.

7.4.6 Consistency check

We focus our discussion with paragraphs from the first two chapters of the RISC-V Spec. The first chapter provides a general introduction. The second chapter provides specification of the instructions from the RV32I integer instruction set. The rest of the chapters are similar to the second chapter, providing specification for a particular instruction set defined in RISC-V. The example in Figure 7.11 is from chapter 2 of the RISC-V Spec. Because the descriptions from chapter 1 are more high-level, we expected that KGC would be more difficult for those paragraphs. However, as our analysis will show later, we find no significant difference on the GPT3.5’s performance for paragraphs from the two chapters.

For checking the consistency requirement, included in our simple checker is a *consistency check*. We repeat the same prompt 10 times and check to see if at least two RDFs are exactly the same. Before checking consistency, we also implement an RDF syntactic check using a publicly-available RDF parser [218]. If an RDF fails the syntactic check, it is excluded for the consistency check.

For paragraphs in the two chapters, Figure 7.14 shows the results of consistency check as two bar charts, for chapters 1 and 2 respectively. These results were obtained without BFs. The result of each paragraph may comprise multiple colored bars. Each color represents a group of RDFs that are exactly the same. A dark bar (■) shows those runs failing the syntactic check. Each orange bar (■) denotes the largest group of RDFs that are exactly the same (i.e. the most consistent group).

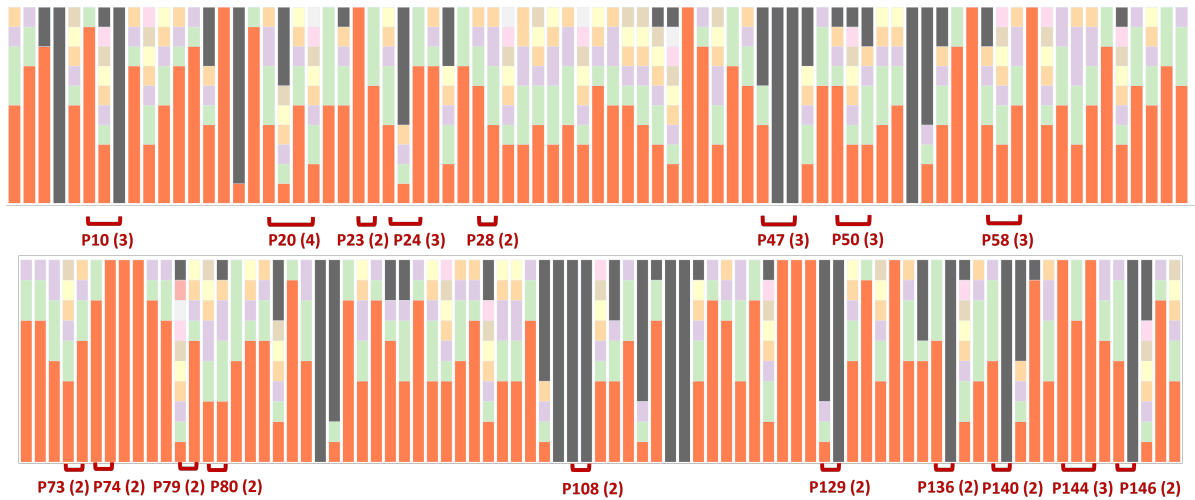


Figure 7.14: Results of consistency check without BFs provided; ■: Show the most consistent group in 10 repeated runs; ■: # of runs that failed

Below some of the bars, there are text notes. Each note means that for the original paragraph, multiple KGC trials failed in all 10 runs. Then, we split the paragraph into multiple sub-paragraphs to be processed separately. For example the first note is “P10(3)” indicating that paragraph 10 was split into 3 sub-paragraphs in the experiment. We will discuss this splitting strategy later. However, notice that some of the sub-paragraphs still fail the syntactic check even after the splitting.

Figure 7.14 demonstrates that in general GPT3.5 does have a systematic behavior for KGC. For those failing cases, we then rely on using BFs to resolve them. It is important to note that this consistency check says nothing about the quality of the resulting RDFs. This assessment is done afterwards.

7.4.7 The effect of using BFs

Figure 7.15 then shows the result of consistency check after BFs are provided. Note that in each case, the BFs were manually selected. In some cases, it took us many trials to find the proper BFs. However, once we found the proper BFs, as shown in the figure

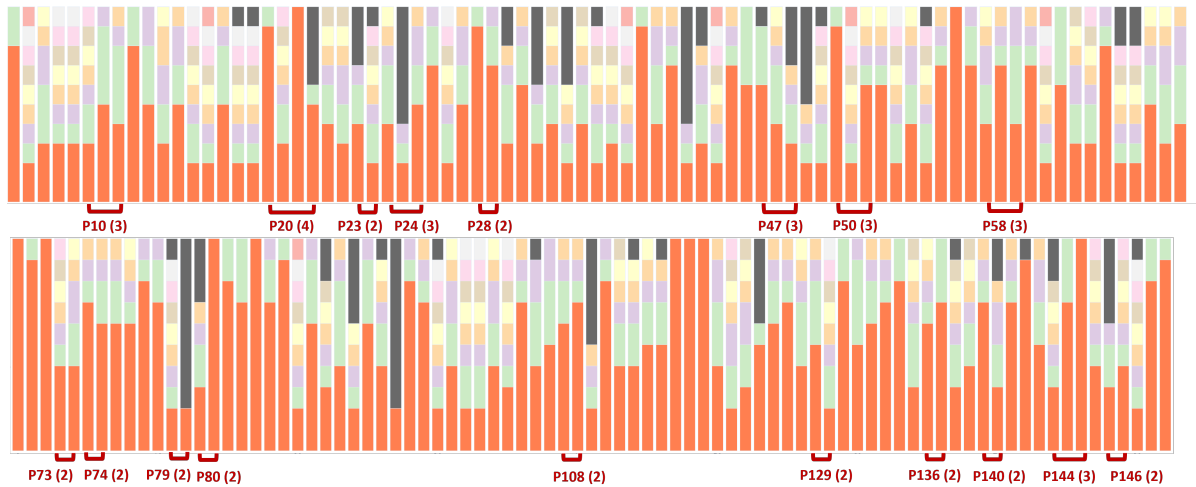


Figure 7.15: Results of consistency check with BFs provided; ■: Show the most consistent group in 10 repeated runs; ■: # of runs that failed

we could make the result much better.

For the two chapters, we had used in total 204 BFs. In Figure 7.15, there is no paragraph with a complete fail any more. In the worst case, we obtained two RDFs that are exactly the same. Based on the results, we can choose the RDF from the most consistent group (the ■ group) to perform the entailment check.

7.4.8 Entailment check

Each entailment check is carried out with two prompts. The first prompt (Prompt A in Figure 7.16) asks GPT3.5 to convert a Fact into a sentence. The second prompt (Prompt B) then asks GPT3.5 whether or not the given paragraph (and background facts if available) logically entails the statement of fact. In Figure 7.16, Prompt B is combined with query 1 or query 2 to form two different prompts, one without BFs and the other with BFs. It should be noted that given an RDF, the entailment check is applied to each Fact individually. Recall that a Fact is an RDF block that may include multiple triples.

Figure 7.17 summarizes the entailment check results for paragraphs from the two

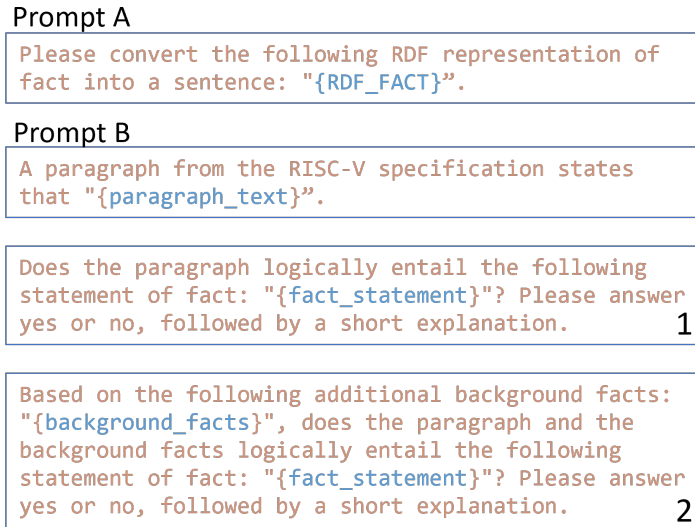


Figure 7.16: Two prompts used in the entailment check

chapters separately. The vertical axis shows the *entailment score*, a value between 0 and 1: Assuming an RDF contains N Facts. The ratio between the number of passing Facts and N is used as the score.

Each bar in Figure 7.17 corresponds to the result from one paragraph and can include three colors. The ■ color bars are based on the RDFs obtained without BFs. If a bar has only this color, it means that adding BFs is not necessary for passing the check. We may consider them as the “easy” cases.

For those cases where the ■ color bars do not reach the score 1.0, we then rely on BFs for bringing the entailment check score to 1.0. For those that do not show up with a ■ color bar at all, they are the “hard” cases. Without BFs, there is no Fact passing the entailment check.

The ■ bars then correspond to the entailment scores based on the RDFs obtained with BFs provided. Note that these bars are shown behind the ■ bars. Consequently, if an original ■ color bar already reaches the score 1.0 or if the ■ bar is shorter than the ■ bar, then the ■ bar cannot be seen.

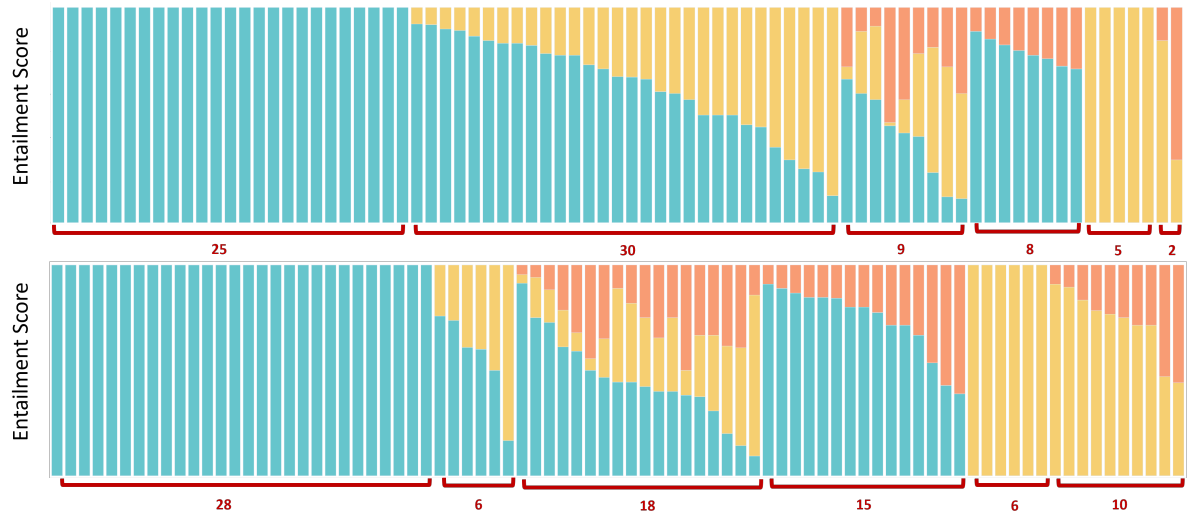


Figure 7.17: Results of entailment check for the two chapters of paragraphs; Each chart shows overlapping of two results from the runs without and with BFs provided; ■: showing % of RDF Facts passing the check where those Facts are obtained with no BFs provided; ■: showing % of RDF Facts passing with BFs provided; ■: With BFs provided, some Facts fail the check (mostly because of including the auxiliary entities not given in the paragraph) and are bypassed after manual review.

For some paragraphs, we need the ■ bars to bring the score to 1.0. They represent Facts reported by the simple checker as failing the entailment check after BFs are provided. However, after manual review these failures are bypassed. Those failures can be divided into three categories where the first one happens most frequently and the third happens only on very few cases.

The first category is the creation and use of *auxiliary entities* in the RDF. An auxiliary entity is the one that does not appear in the paragraph (nor the BFs) and is created to facilitate describing other entities. In a sense, we can consider those auxiliary entities as BFs automatically supplied by GPT3.5. Because they are not mentioned in the paragraph and the BFs, Facts involving them would fail the entailment check. However, the GPT3.5's ability to add auxiliary entities (i.e. its own BFs) can be quite desirable, because it helps provide BFs possibly missed by our manual preparation of BFs.

The second category involves an entity or a predicate that has nothing wrong by itself.

However, in the RDF the entity/predicate is specified within a particular namespace (e.g. “riscv:”, “rdf”). Because the original paragraph (and the BFs) do not explicitly state their use in the namespace, this can cause the entailment check to fail.

The third category, happening only on few cases, involves the use of namespaces other than rdf or risvc namespaces. In the Spec, there are some descriptions about other ISA architectures (MIPS, SPARC, etc.). Those descriptions may result in the creation of their respective namespaces. In an RDF, entities from two namespaces might be connected through a predicate. This can cause a problem for the entailment check as the original paragraph (and the BFs) simply considers those terms as entities rather than different namespaces.

7.4.9 Summary of the study

In summary, Figures 7.14-7.17 shows that it is feasible to use GPT3.5 as an oracle. Further, providing BFs can improve consistency and also help reach a satisfactory RDF for every paragraph.

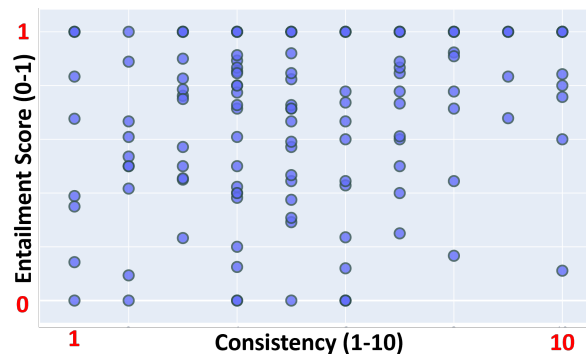


Figure 7.18: No correlation between the size of largest group from consistency check and the entailment check score

It should be noted that more consistency does not mean more likely to pass the entailment check. Figure 7.18 illustrates this point. The x-axis is the size of largest consistent group. The y-axis is the entailment score. Every dot is a paragraph. The

RDF is the one obtained without BFs provided (i.e. Figure 7.15). As seen, the results of consistency check and of entailment check have no obvious correlation.

With the consistency check and entailment check, we were able to obtain satisfactory KG for about 200 paragraphs in the first two chapters from RISC-V Spec. With BFs, a total of 597 and 577 subject entities were extracted for chapters 1 and 2, respectively.

To show how many paragraphs can be connected through the subject entities, we first used a simple method to group subject entities into high-level subject concepts. If subject entities shared the same suffix word, we grouped them together. For example, “CSRInstruction” and “StoreInstruction” were grouped with the high-level concept “Instruction”. Suffix can be easily split from the entity phrase since RDF already formatted the phrases into Camel or Snake case. We further used a stemming tool [219] to remove morphological affixes from the suffix words so that words with the same stem would be grouped together. For example, “encodings” and “encodes” belonged to the same group.

For chapter 1, we collected a total of 168 and 178 subject concepts from $BF\phi$ and BFA , respectively. These subject concepts provided in total 421 connections for $BF\phi$ and 557 connections for BFA , respectively, to all the paragraphs.

Figure 7.19 shows a bipartite graph between subject concepts and paragraphs. An edge means the subject concept appears in the paragraph. Only those subject concepts that connect at least two paragraphs are shown in the graphs. The graph has 75 subject concepts with 454 edges.

For chapter 2, we collected a total of 191 subject concepts with BFs injected. These subject concepts provided in total 497 connections to all the paragraphs. Figure 7.20 shows a similar bipartite graph. The graph has 82 subject concepts with 388 edges.

More results on the study can be found in [196].

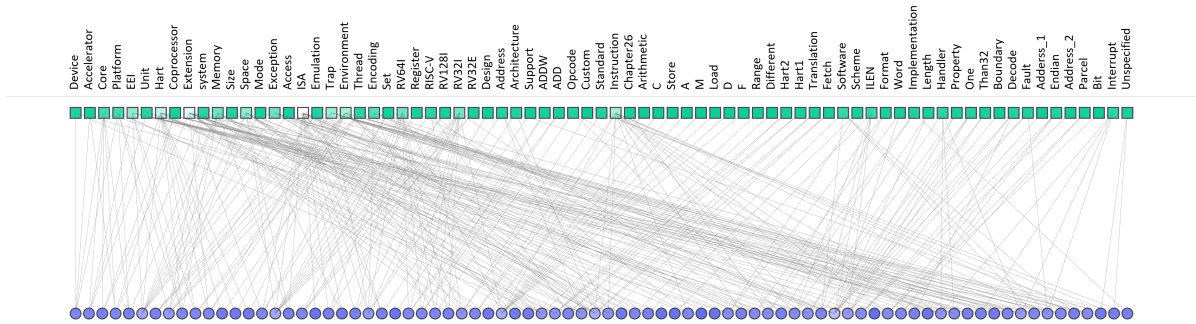


Figure 7.19: Chapter 1 (With BFs): there are 75 concepts shared by at least two paragraph and the total number of edges is 454; In the bipartite graph, the bottom dots each represents a paragraph from chapter 1 and the upper squares each represents a subject concept. More transparent the color indicates more edges are connected between the concepts and paragraphs.

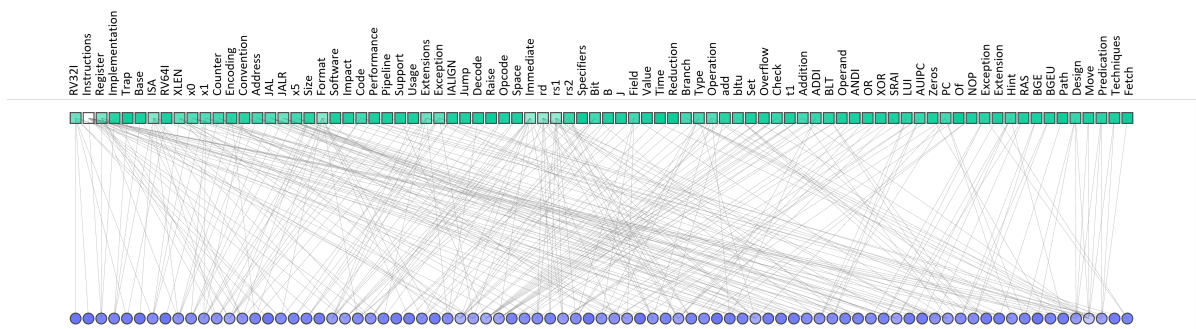


Figure 7.20: Chapter 2 (With BFs): there are 82 concepts shared by at least two paragraph and the total number of edges is 388; In the bipartite graph, the bottom dots each represents a paragraph from chapter 2 and the upper squares each represents a subject concept.

Chapter 8

Oracle-Checker Scheme

泉眼無聲惜細流，樹陰照水愛晴柔。

小荷才露尖尖角，早有蜻蜓立上頭。

The flow from the mouth of the fountain is quiet and delicate. The tree's shade creates a gentle light. In the early morning only the angled tip of a small lotus flower is revealed, and a dragonfly sits on the head.

— 《小池》 楊萬里， A poem from Song Dynasty

The initial study for knowledge graph construction (KGC) (Section 7.4) provides several lessons for us to move onto the next stage of the research:

- While an LLM (GPT3.5) can be treated as an oracle for a complex task like KGC, there is a need to implement a checker for checking the LLM's responses. The checking has to be on a per-sample basis that on each sample, decides whether the particular response should be accepted or not.
- To process a domain-specific document, background knowledge is important. The LLM might already have some background knowledge on the domain. However, it

is important to supply proper background knowledge in the prompts to guide (or constrain) the LLM to provide the desirable responses.

- While GPT3.5 seems to be able to perform KGC well to an extent, its responses for a given sample can still vary significantly. Hence, to pursue a methodology for automatic checking, KGC might not be the most appropriate problem to start. We need to look for other simpler tasks as the focus for the automatic checker development.
- Two fundamental NLP tasks seem to be highly related to KGC: *entity extraction* and checking for semantic equivalence (*paraphrase decision*). KGC includes the extraction of nodes and then the determination of their relations. Extraction of the nodes can be seen as the process of entity extraction. Entailment check is to determine if one sentence s_a entails the a fact f . If we treat $s_b = s_a + f$, then entailment check can be seen as the problem for determining the semantic equivalence between s_a and s_b .

Based on the above lessons learned, we therefore decided to focus on entity extraction and paraphrase decision as the two NLP tasks in the pursuit to develop an automatic Oracle-Checker (OC) scheme. The initial effort is reported in this chapter.

8.1 Per-Sample Examination of LLM’s Responses

Named entity recognition and paraphrase detection are two basic NLP tasks that are considered as textbook topics by now [167]. Given that LLMs have demonstrated remarkable success across a variety of much more complex NLP tasks [220, 221, 222, 223], it is anticipated that their performance on such basic tasks should be generally acceptable. For example, given the query to an LLM “*Please perform entity recognition for entities in*

the following sentence: ‘{text}’” or the query “Decide whether the following two sentences are semantically equivalent: ‘{sentence1}’, ‘{sentence2}’”, we expect the LLM to be able to provide acceptable responses on most of the queries.

Depending on the application context, acceptable performance in general can mean that we still need a way to locate individual queries with unacceptable responses. For example, within a semiconductor chip design company one desires to use an LLM to perform the two basic tasks on a design document or a technical note. Entity extraction might be used to extract a list of hardware entities and paraphrase might be used to re-write a document to make it more readable. In this context, it is desirable to have an automatic way that can examine the LLM’s responses on a per-sample basis, accepting some responses and rejecting others. Even if only 50% of the responses are trustworthy, knowing which 50% can still render the LLM quite useful, as one is certain it has already completed half of the job.

In this chapter, we introduce a novel per-sample examination strategy called *oracle-checker* (OC) scheme to address the needs in practice when per-sample acceptance is a crucial concern. In an OC scheme, the LLM works off-the-self and the scheme requires no human interaction. A *checker* is designed to *interact* with the LLM instead. To check a response, the checker asks the LLM with additional queries and based on the answers, decide if the response should be accepted or rejected.

Under our OC scheme, the LLM is treated as an *oracle* for the task. The idea of oracle has existed for a long time in theoretical studies of computational complexity [224, 225]. In *interactive proofs* (IP), the interaction is between a *prover* and a *verifier*. The prover (i.e. the oracle) is assumed to have unlimited power with respect to computing a given function $f()$ and the verifier (i.e. the checker), on an instance x , interacts with the prover to decide if its answer $f(x)$ is acceptable or not.

Two earliest theoretical ideas are drawn upon for this work. The first is about checking

for *group homomorphism* (GH-check) [226] and the second is about checking for *graph isomorphism* (GI-check) [227]. The GH-check is also called a *linearity check* [226] which is checking how far a given $f()$ is from being a homomorphism. In theory, the GI-check is divided into two parts. When the answer is “yes” (isomorphic), the checker asks the oracle for an efficiently-verifiable proof and accepts the answer after the proof is checked. When the answer is “no” (non-isomorphic), the checker verifies the claim probabilistically by asking the oracle a sequence of carefully-constructed graph isomorphism questions. Then, depending on the answers the checker decides to accept the “no” answer or not.

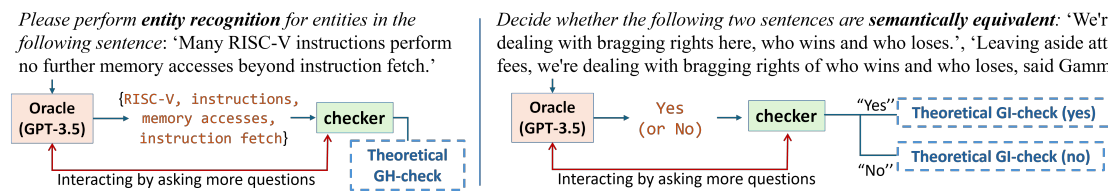


Figure 8.1: Realizing the two theoretical checks in practice by treating GPT-3.5 as an oracle.

Figure 8.1 summarizes the work. For entity extraction, we apply the idea of GH-check. In this case, the $f()$ mentioned above can be considered as the unknown LLM’s entity extraction process. We are interested in checking how far $f(x)$ is from a “linear” extraction process for each sentence instance x . For paraphrase decision, we apply the idea of GI-check. The $f(s_a, s_b)$ corresponds to LLM’s internal process to decide if the two sentences s_a and s_b are semantically equivalent or not. Instead of checking a property of $f()$, the check is about the acceptance of the yes/no response. The check is split into two parts, one for “yes” answer (equivalent) and the other for “no” answer (non-equivalent).

The main challenge for realizing the GH-check in the context of entity extraction is that we need to find a way to reduce the problem of checking the extracted entities to the problem of checking the homomorphism between two algebraic groups. Devising the two algebraic groups in the context of entity extraction is the key to enable a practical implementation of the GH-check idea.

The main challenge for realizing the GI-check in the context of paraphrase decision includes two aspects. While an efficiently-computational proof for isomorphism of two graphs is simply a permutation, it is not obvious what such a proof for semantically equivalence should be. While a sequence of isomorphism questions can be intuitively constructed in the graph context [227], it is not obvious how such questions can be constructed in the sentence context.

In this work, we therefore devise novel techniques to overcome these challenges and achieve practical realization of the ideas from both GH-check and GI-check in entity extraction and paraphrase decision, respectively. It is important to emphasize that due to the interactive nature of our OC scheme, the check is *not* based on the assumption that there is a correct answer for the question to LLM. The focus is on the acceptability of an LLM’s answer, not on its correctness. Consequently, most experiment results reported in this work are relative to the checkers. Hence, when we say that an LLM’s answer can be trusted, this trust is established with respect to the particular checker. Although a checker cannot assert the correctness of an answer, it can be used to separate LLM’s responses into two categories: trustable and untrustable. Hence, a checker can be used as a *characterization* tool to perform per-sample examination on a given dataset.

The main contributions of the work includes the following: (1) We propose the novel techniques for realizing the OC scheme in practice by treating GPT3.5 [16] as an oracle. (2) We demonstrate how the theoretical ideas of GH-check and GI-check can be implemented in the two NLP tasks. (3) With our checker, by comparing results from RISC-V ISA Spec [197] and from the DOCRED dataset [228], for the first time we have a way to quantify the following intuition—extracting *hardware* entities from a hardware design document is harder than extracting *named* entities as commonly studied in NLP research. (4) Also for the first time, we have a way to quantify individual label ambiguity in the MSR Paraphrase corpus (MSRP) [229].

The limitation of this work is that our techniques are specific to realizing the two checks on the two NLP tasks. Extending to other task-check combinations might require a different technique. However, given a rich set of theoretical checks already there [226] and a variety of NLP tasks to explore [167], this work can provide a guide for further studies to realize other task-check combinations.

8.2 Realizing GH-check In Entity Extraction

For detail of the GH-check, please refer to [226]. Given two groups $G_1 : (S_1, \circ)$ and $G_2 : (S_2, +)$ where S_1, S_2 are closed under the two binary operations $\circ, +$, respectively. Let h be a function $h : S_1 \rightarrow S_2$. h is a homomorphism if $\forall x, z \in S_1, h(x \circ z) = h(x) + h(z)$. Assume h is a homomorphism and call it a *reference function*. To check if another function $f : S_1 \rightarrow S_2$ is also a homomorphism, the GH-check consists of running n tests, each by uniformly sampling x, z from S_1 and checking if $f(x \circ z) = f(x) + f(z)$ holds.

Passing each test increases our confidence that f is close to h . This closeness is measured by a distance between f and h as $\delta(f, h) = Pr_x[f(x) \neq h(x)]$ for an x uniformly sampled from S_1 . We say f is ϵ -far from h if $\delta(f, h) > \epsilon$. In theory [226], if f is ϵ -far from h , then the test would fail with a probability at least $3\epsilon - 6\epsilon^2$ when $\epsilon \leq \frac{1}{4}$, and at least $\frac{2}{9}$ otherwise. In Section 8.2.3, we provide more detailed discussion on this theoretical bound in view of our practical GH-check implementation.

8.2.1 Constructing the reference function h

Given a sentence s , let g be the entity extraction and $g(s)$ produces an entity set: $E_s = \{e_1, \dots, e_m\}$. To realize GH-check, we take two steps: (1) define the reference function h by defining the two groups G_1, G_2 , (2) define the corresponding function f under check where f is based on g .

Let 2^{E_s} denote the power set of E_s . 2^{E_s} can be represented by 2^m bit vectors (denoted as \mathcal{B}^m), each element as (b_1, \dots, b_m) where $b_i = 0$ or 1 . To define h , we let $G_1 = (S_1, \oplus)$ and $G_2 = (S_2, \ominus)$ where \oplus, \ominus are bit-wise XOR and XNOR, respectively, and h, S_1, S_2 are formulated as follows:

$$h : S_1 \rightarrow S_2, h(x) = \bar{x}, \text{ where } S_1 = \{x | x \in \mathcal{B}^m\} \text{ and } S_2 = \{y | y \in \mathcal{B}^m\} \quad (8.1)$$

It is easy to verify that h is a homomorphism because $\forall x, z$ we have $h(x \oplus z) = \overline{x \oplus z} = \bar{x} \ominus \bar{z} = h(x) \ominus h(z)$. In fact, h is a isomorphism because it is a bijective/invertible homomorphism.

8.2.2 Homomorphism test in entity extraction

Next, we define the function-under-check f . Given h , f must be also a mapping from S_1 to S_2 . Further, f also needs to involve g , the entity extraction. To define f , we assume that there exists another entity set $E'_s = \{e'_1, \dots, e'_m\}$ where each e'_i is a *synonym* of the e_i in the original set E_s .

For a vector $x = (b_1, \dots, b_m) \in S_1$, f takes x as input and first does the following: For each $b_i = 1$, f replaces e_i in the original sentence s with e'_i . This step produces a modified sentence s_x . Then, f calls the entity extraction g on s_x to obtain the entity set $L_x = g(s_x)$. Then, L_x is vectorized into a bit vector (c_1, \dots, c_m) in S_2 , which is the output $f(x)$, where $c_i = 1$ means the original entity e_i is extracted and $c_i = 0$ means the synonym entity e'_i is extracted. Since our reference function is $h(x) = \bar{x}$, ideally, we desire $f(x) = \bar{x}$, i.e. $\forall i, c_i = \bar{b}_i$. Figure 8.2 illustrates the mapping of f .

To run a test, two vectors x, z are sampled from S_1 . A third vector $x \oplus z$ is calculated. With the synonym replacements, they result in three modified sentences: $s_x, s_z, s_{x \oplus z}$. Applying entity extraction g on them obtains three entity sets: $L_x, L_z, L_{x \oplus z}$. They are

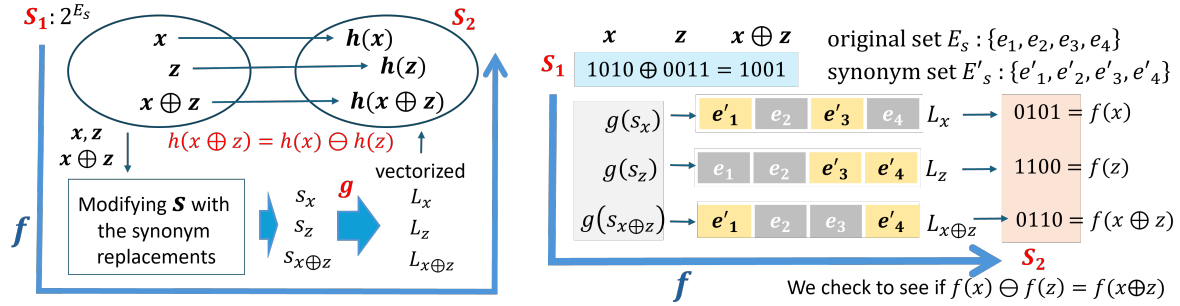


Figure 8.2: Illustration of the reference function h , the entity extraction g , and the function-under-check f , with an example starting with E_s containing four extracted entities from original sentence s

then vectorized to obtain the corresponding bit vectors in S_2 as described above. The test is then to check if $f(x) \oplus f(z) = f(x \oplus z)$. In Figure 8.2, an example starting with four entities $\{e_1, e_2, e_3, e_4\}$ is provided to illustrate this process.

To understand the meaning of our GH-check in entity extraction, assume we are able to exhaustively check that $f(x) = h(x), \forall x \in S_1$. This means that for a given entity e_i , its extraction is independent of replacing any combination of other entities with their synonyms. It is important to note two points: (1) For a large m , exhaustive check can be costly. Hence, we use a few homomorphism tests to check if $f = h$. (2) Each GH-check is applied to the entity extraction on a given sentence s . Acceptance is decided on s , not on individual entities. Hence, the check is about entity extraction on each s as a whole. In our check, we do not accept any individual entities if s fails.

To run the GH-check on an s , we need to obtain the synonym set E'_s . Later in Sections 8.4.2, we will discuss how to do this by querying the oracle, i.e. GPT3.5. With our GH-check, the acceptance of an entity extraction response depends on the E'_s in use. Hence, a more precise way to interpret an accepted response on s is that: From the perspective of E'_s , entity extraction on s is more trustable than on other sentences that are rejected. As mentioned before, we use the checker for characterization of per-sample responses, not for verifying their correctness by assuming there are correct answers.

8.2.3 A remark on the theoretical GH-check

In property testing [226], checking if function f (mapping from one group to another) has a property P is based on the idea of measuring a distance between f and a reference set of functions \mathcal{F} . All functions in \mathcal{F} is assumed to have the same domain as f and have the property P .

The distance between two functions f and g is defined as $\delta(f, g) = \Pr_x[f(x) \neq g(x)]$ where the random x is drawn uniformly from the domain. Then, the distance between f and \mathcal{F} , denoted as $\delta(f, \mathcal{F})$, can be defined as $\min_{g \in \mathcal{F}} \{\delta(f, g)\}$. f is said to be ϵ -close to \mathcal{F} if $\delta(f, \mathcal{F}) \leq \epsilon$ and equivalently, f is ϵ -far if $\delta(f, \mathcal{F}) > \epsilon$.

Suppose we desire to check group homomorphism on $f : S_1 \rightarrow S_2$ for two groups $G_1 = (S_1, \circ)$ and $G_2 = (S_2, +)$. In this case, the group homomorphism is our property P . We apply n tests where each test is simple: Uniformly sample x and z from S_1 and verify that $f(x \circ z) = f(x) + f(z)$ holds.

The theoretical question centers on the value of each test brings. To characterize this value, we hypothesize \mathcal{F} containing functions that have the property P . Then, the theoretical question can be stated as: How one test can bound the distance from f to \mathcal{F} (how close f is to be a homomorphism if the test passes)? The main theoretical result can be summarized as the following theorem.

Theorem 8.2.1 *A group homomorphism test provides the following result: If f is a homomorphism, the test will pass with probability 1. If f is ϵ -far from \mathcal{F} , then the test will fail with a probability at least $3\epsilon - 6\epsilon^2$ when $\epsilon \leq \frac{1}{4}$ and at least $\frac{2}{9}$ otherwise.*

Note that the “ $\frac{1}{4}$ ” is picked arbitrarily here (see [226] for more detail). For example, suppose f is not a homomorphism and f is 0.2-far from \mathcal{F} (from being a homomorphism). Because $0.2 < \frac{1}{4}$, running one test has at least $3 * 0.2 - 6 * (0.2)^2 = 0.36$ probability to reveal that f is not a homomorphism.

To see the theoretical bound, $3\epsilon - 6\epsilon^2$, assume that $h \in \mathcal{F}$, h is the homomorphism closest to f and we have $\Pr_{x \in S_1}(f(x) \neq h(x)) > \epsilon$, i.e. f is ϵ -far from h . Then, the rejection probability $\Pr_{x, z \in S_1}(f(x) + f(z) \neq f(x \circ z))$ for a test, is lower-bounded by the following [226]:

$$\begin{aligned} & \Pr_{x, z \in S_1}[f(x) \neq h(x) \wedge f(z) = h(z) \wedge f(x \circ z) = h(x \circ z)] \\ & + \Pr_{x, z \in S_1}[f(x) = h(x) \wedge f(z) \neq h(z) \wedge f(x \circ z) = h(x \circ z)] \\ & + \Pr_{x, z \in S_1}[f(x) = h(x) \wedge f(z) = h(z) \wedge f(x \circ z) \neq h(x \circ z)] \end{aligned} \quad (8.2)$$

It is a lower bound because the test requires $f(x) + f(z) = f(x \circ z)$, whereas between f and h on $x, z, x \circ z$, the reason for this equality not holding can be due to one disagreement (disagree on one point only) and possibly due to more disagreements as well, and Equation 8.2 ignores the events where f and h have more than one disagreements. The three events are disjoint and hence, each of the three probability terms can be lower-bounded by $\epsilon - 2\epsilon^2$, resulting in the overall lower bound $3\epsilon - 6\epsilon^2$. To see this, consider the first term in equation 8.2:

$$\begin{aligned} & \Pr_{x, z \in S_1}[f(x) \neq h(x) \wedge f(z) = h(z) \wedge f(x \circ z) = h(x \circ z)] \\ & = \Pr_{x, z \in S_1}[f(x) \neq h(x)] - \Pr_{x, z \in S_1}[f(x) \neq h(x) \vee (f(z) \neq h(z) \wedge f(x \circ z) \neq h(x \circ z))] \\ & \geq \epsilon - (\Pr_{x, z \in S_1}[f(x) \neq h(x) \wedge f(z) \neq h(z)] + \Pr_{x, z \in S_1}[f(x) \neq h(x) \wedge f(x \circ z) \neq h(x \circ z)]) \\ & = \epsilon - 2\epsilon^2 \end{aligned} \quad (8.3)$$

It should be noted that our practical GH-check as discussed in Section 8.2.1 is a simplified version of the theoretical GH-check because our GH-check does not test against the assumed \mathcal{F} . Instead, we define one specific reference function h in our GH-check,

which is a bijective/invertible homomorphism, i.e. an isomorphism. And we are interested in checking how close f is to the particular h . In our GH-check, to verify $f(x \circ z) = f(x) + f(z)$ the checker proceeds by first verifying $f(x) = h(x)$ and $f(z) = h(z)$, as h is fixed. If any of them fails, the test is considered failed, i.e. $f(x \circ z) \neq f(x) + f(z)$. If both passes, then the checker further verifies $f(x \circ z) = f(x) + f(z)$.

Hence, the rejection probability of the first step is bounded by:

$$\begin{aligned} & \Pr_{x,z \in S_1}(f(x) \neq h(x) \vee f(z) \neq h(z)) \\ &= \Pr_{x \in S_1}[f(x) \neq h(x)] + \Pr_{z \in S_1}[f(z) \neq h(z)] - \Pr_{x,z \in S_1}[f(x) \neq h(x) \wedge f(z) \neq h(z)] \quad (8.4) \\ &= 2\epsilon - \epsilon^2 \end{aligned}$$

And the rejection probability of the second step is bounded by (this is the third term in Equation 8.2)

$$\begin{aligned} & \Pr_{x,z \in S_1}[f(x) = h(x) \wedge f(z) = h(z) \wedge f(x \circ z) \neq h(x \circ z)] \\ &= \epsilon - 2\epsilon^2 \end{aligned} \quad (8.5)$$

The two events are disjoint. Hence, the total rejection probability of our GH-check implementation is lower bounded by $3\epsilon - 3\epsilon^2$ which is larger than the $3\epsilon - 6\epsilon^2$ as stated in Theorem 8.2.1. Therefore, our practical GH-check draws upon the idea of the theoretical GH-check, but is different from the theoretical GH-check. The theoretical idea inspired us to define the reference function h and devised the f to be checked in view of h . In our restricted setting, we can check the homomorphism $f(x \circ z) = f(x) + f(z)$ in two separate steps with a larger lower bound on the rejection probability.

In our current implementation, the interpretation of the bit vectors in G_2 is that for each bit c_i , $c_i = 1$ means the original entity e_i is extracted, and $c_i = 0$ means the synonym entity e'_j is extracted (Section 8.2.2). Based on this interpretation, the checker

also checks the extraction of the synonym entities. Alternatively, the checker can be modified to bypass checking on the synonym entities, i.e. $c_i = 0$ means the extraction of the particular entity can be ignored. This relaxation can potentially make more sentences pass the GH-check because the check only cares about the original set of entities. In the current work, we desire to show the more stringent version of the GH-check so we can observe a lower bound on the passing rate, comparing to the more relaxed version.

8.3 Realizing GI-check In Paraphrase Decision

Similar to the GH-check above, in paraphrase decision our OC scheme also requires the checker to interact with the LLM. This interaction is separated in two: “yes” case when the response is semantically equivalence and “no” case when the response is non-equivalence. These two cases correspond to the two cases in theoretical GI-check [227]—verifying graph isomorphism and non-isomorphism, respectively. Figure 8.3 illustrates the high-level ideas of our GI-check.

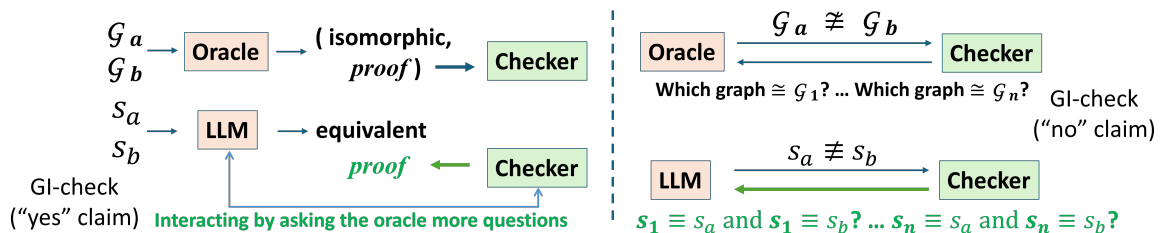


Figure 8.3: Two ways for realizing GI-check (yes and no claims separately) in paraphrase decision

Given two graphs $\mathcal{G}_a, \mathcal{G}_b$ and asking the oracle if they are isomorphic, when the answer is “yes” (isomorphic), the theoretical check simply asks the oracle for a *proof*. The proof is a permutation of nodes that maps \mathcal{G}_a to \mathcal{G}_b . With such a proof, the checker can easily verify the correctness of the proof [224]. The challenge to translate this idea into the paraphrase context is that, our checker cannot simply asks the LLM for an “easily

verifiable” proof. As a result, the proof is constructed by the checker by asking the LLM more questions. The trick is in what questions to ask (see Section 8.3.2).

When the answer is “no” (non-isomorphic), the theoretical check is probabilistic. The checker asks a series of n questions as shown in Figure 8.3. Each of $\mathcal{G}_1, \dots, \mathcal{G}_n$ is based on randomly picking either \mathcal{G}_a or \mathcal{G}_b and randomly picking a permutation π to transform the selected graph. These two random selections ensure that the oracle can answer those questions correctly only when its claim $\mathcal{G}_a \not\cong \mathcal{G}_b$ is true to begin with. If the truth was $\mathcal{G}_a \cong \mathcal{G}_b$, then the oracle would not be able to correctly answer those questions other than random guessing on each question. In this case, the probability of the oracle answering all n questions correctly would be only $\frac{1}{2^n}$. As a result, asking n such questions allows the checker to accept the $\mathcal{G}_a \not\cong \mathcal{G}_b$ claim with confidence $(1 - \frac{1}{2^n})$ [224]. The trick to translate this probabilistic check into a check in the paraphrase context is also in the series of questions to ask, where the questions are about the semantic equivalence of two sentences s_a, s_b to each constructed s_i . In our GI-check, the two random selections correspond to (1) randomly selecting either s_a or s_b , and (2) asking the LLM to generate a set of paraphrases for the selected sentence and pick one of them as the s_i in the i th question. Then, the checker accepts the original “no” claim only if the LLM can correctly answer all the questions. Section 8.3.3 explains how to pick the paraphrase s_i .

8.3.1 A remark on the theoretical GI-check

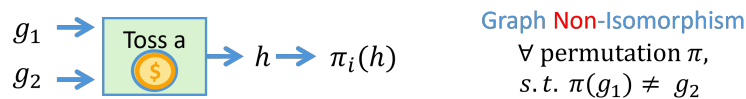
Given two graphs g_1, g_2 of N vertices labeled by $\{1, \dots, N\}$. The two graphs is isomorphic if there exists a permutation π on the labels such that π transforms g_1 into g_2 , i.e. $\pi : g_1 \rightarrow g_2$.

Suppose an oracle claims that g_1, g_2 are isomorphic. In theory, checking this “yes” answer is straightforward by asking the oracle to provide a permutation π that transforms

g_1 into g_2 . This π serves as a *proof* that g_1, g_2 are isomorphic.

In contrast, when the oracle's answer is that the g_1, g_2 are not isomorphic, this means that there does not exist any π that can transform g_1 into g_2 . Then, getting a straightforward proof is difficult because a proof needs to show that for all permutations, the two graphs cannot be isomorphic. Therefore, checking for non-isomorphism requires a different strategy.

The strategy [224] involves two random choices as illustrated in Figure 8.4. The first is to randomly choose between g_1 and g_2 . Let this result be h . The second is to randomly choose a permutation π_i and apply it to h . Let the resulting graph be denoted as $\pi_i(h)$. The checker then asks the oracle from which of the g_1, g_2 is the graph $\pi_i(h)$ generated. If g_1, g_2 are indeed non-isomorphic, the oracle will be able to answer a series of such questions correctly. Otherwise, if g_1 is actually isomorphic to g_2 , then the oracle can only guess the answer randomly and if n such tests are run, the probability that the oracle correctly guesses the answers for all n questions is only $\frac{1}{2^n}$.



Query the oracle: $\pi_i(h) = g_1$ or $\pi_i(h) = g_2$?

Figure 8.4: Probabilistic check by querying the oracle: $\pi_i(h) = g_1$ or $\pi_i(h) = g_2$?

Note that the essence of checking for graph isomorphism is based on a proof for the isomorphism. Our implementation of the GI-check is therefore to construct a proof in the context of semantic equivalence. The essence of checking for graph non-isomorphism is based on finding an *indifferentiable graph* $\pi_i(h)$, indifferentiable in the sense that if g_1, g_2 are actually isomorphic, then the oracle cannot differentiate where $\pi_i(h)$ comes from. Our implementation of the GI-check on the semantic non-equivalence therefore focuses on capturing this idea of indifferentiability and applying the idea to find a so-called

“indifferentiable” paraphrase in the semantic non-equivalence context (Section 8.3.3).

8.3.2 Establishing a proof for semantic equivalence

In this work, we use the term *phrase* to mean ≥ 2 consecutive words in a sentence and not the sentence itself. We use letters μ and ν to denote phrases. Given a sentence s , a *decomposition* of s is a set of phrases $\{\mu_1, \dots, \mu_k\}$ such that their union *covers every word* in s (except prepositions).

Our proof scheme is inspired by the idea of *compositional phrase alignment* (CPA) [230, 231]. Given two sentences s_a, s_b deemed as semantically equivalent (denoted as “ \equiv ”) by LLM, a *weak proof* $WP_{s_a \rightarrow s_b}$ is established by finding a decomposition $U = \{\mu_1, \dots, \mu_k\}$ of s_a and a set of phrases $V = \{\nu_1, \dots, \nu_k\}$ of s_b such that for all $1 \leq i \leq k$, we have $\mu_i \equiv \nu_i$, i.e. sentence-level semantic equivalence is proved by a decomposition with two or more phrase-level semantic equivalences. Note that for two sentences s_a, s_b , a weak proof can be in either “ $s_a \rightarrow s_b$ ” or “ $s_b \rightarrow s_a$ ” direction. Hence, we also define a *strong proof* (SP) as containing weak proofs in both directions.

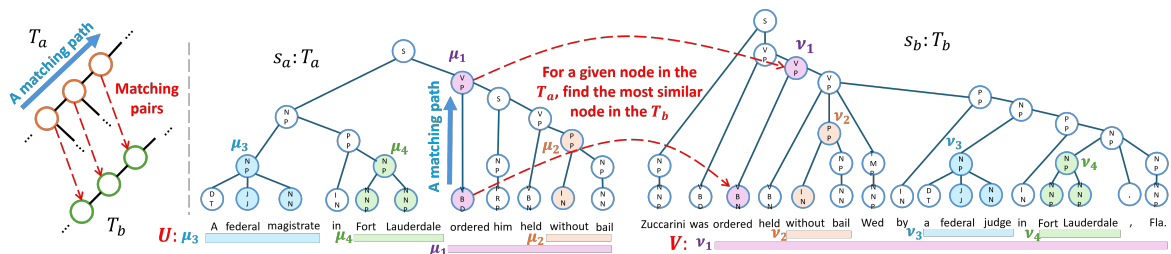


Figure 8.5: Finding candidates (U, V) . μ_1 to μ_4 form a decomposition of s_a , and $(\mu_i \rightarrow \nu_i, i = 1 \dots 4)$ are the matching pairs, so the checker can ask the LLM the four questions: if $\mu_i \equiv \nu_i, i = 1 \dots 4$.

In the OC scheme, the checker’s job is to find potential candidates for (U, V) and if a candidate, say (U_1, V_1) is found, the checker then asks the LLM the corresponding phrase-level questions (e.g. “ \forall pair $\mu_i \in U_1, \nu_i \in V_1, \mu_i \equiv \nu_i$?”) to establish a weak proof. To find

(U, V) candidates, we implement a method summarized in Algorithm 8.1. Figure 8.5 uses an example to illustrate this method.

Algorithm 8.1: Find candidates: (U, V) , $U \ni \mu$ as a decomposition of s_a and $V \ni \nu$, s.t. $(\mu \rightarrow \nu)$.

Input: Two sentences s_a, s_b
 $T_a \leftarrow \text{parse_tree}(s_a)$; $T_b \leftarrow \text{parse_tree}(s_b)$; $S_U \leftarrow \phi$; $D \leftarrow \phi$
for each node x_i **in** T_a **do**
 $D \leftarrow D \cup (x_i \rightarrow y_j)$, where $y_j \in T_b$ is the most similar node to x_i
for each leaf node l **in** T_a **do**
 while $(l \rightarrow l') \in D$ **and** $l \neq \text{root}$ **and** $l' \neq \text{root}$ **do**
 $\text{temp} \leftarrow (l \rightarrow l')$
 if (l is a leaf **or** $\text{children}(l) > 1$): $l \leftarrow \text{parent}(l)$; $l' \leftarrow \text{parent}(l')$
 if (temp is not at the leaf level): $S_U \leftarrow S_U \cup \text{temp}$;
 $(U, V) \leftarrow \text{output}(S_U, s_a)$

Given s_a, s_b , the method first uses the core-NLP API [232] to find their constituency parsing trees, T_a, T_b where every node corresponds to a word or a phrase (see Figure 8.5). Then, the central idea is by finding a collection of *matching paths* in T_a . A matching path starts from a leaf in T_a toward its root but excluding the root. It also has to move beyond the leaf level. Every node on a matching path in T_a has a *matching node* in T_b . For a node in T_a (as phrase μ), its matching node in T_b (as phrase ν) is the *most similar* phrase to μ . This is denoted as a *matching pair* $(\mu \rightarrow \nu)$. The similarity score between two words/phrases can be calculated using the word/phrase embeddings provided by a language model like BERT [233]. In Algorithm 8.1, only the last matching pair (recorded by temp) is used to represent a matching path in S_U , which corresponds to a phrase in s_a and a phrase in s_b .

In Algorithm 8.1, the first for-loop collects the the matching pairs into D . The second for-loop then relies on D to find matching paths. Suppose it finds $S_U = \{(l_1 \rightarrow l'_1), \dots, (l_n \rightarrow l'_n)\}, n > 1$. Each l_i (l'_i) corresponds to a phrase μ_i (ν_i) in s_a (s_b). Then, the $\text{output}()$ step is to check if collectively $\{\mu_1, \dots, \mu_n\}$ cover the entire s_a (i.e. a decom-

position). If it does, then Algorithm 8.1 returns $U = \{\mu_1, \dots, \mu_n\}$ and $V = \{\nu_1, \dots, \nu_n\}$. Based on (U, V) , the checker generates n questions, i.e. “ $\mu_i \equiv \nu_i?$ ” for every i , and asks the LLM those questions. Without loss of generality, suppose LLM confirms that indeed $\mu_j \equiv \nu_j$ for $1 \leq j \leq k \leq n$. Then, the checker verifies again if $\{\mu_1, \dots, \mu_k\}$ cover the entire s_a . If it does, then $\{(\mu_1 \equiv \nu_1), \dots, (\mu_k \equiv \nu_k)\}$ becomes a weak proof $WP_{s_a \rightarrow s_b}$.

Recall that a weak proof can be in either direction: “ $s_a \rightarrow s_b$ ” ($WP_{s_a \rightarrow s_b}$) or “ $s_b \rightarrow s_a$ ” ($WP_{s_b \rightarrow s_a}$). Algorithm 8.1 is shown to find candidates for establishing $WP_{s_a \rightarrow s_b}$. To find candidates for $WP_{s_b \rightarrow s_a}$, we can simply swap the two inputs. In addition, the last step *output* is default to verify coverage of s_a , and as an alternative, this can be changed to verify coverage of s_b . Doing so provides a second chance to find candidates for $WP_{s_b \rightarrow s_a}$. It is important to note that Algorithm 8.1 is only for finding potential (U, V) candidates, i.e. each as a collection of phrase pairs, and our checker only reports a proof *after* the LLM confirms the semantic equivalence of enough individual phrase pairs.

8.3.3 Picking a paraphrase in GI-check for a “no” claim

When the LLM says that $s_a \not\equiv s_b$, as explained with Figure 8.3 before, the checker needs to find an s_i and asks the LLM if “ $s_i \equiv s_a?$ ” and if “ $s_i \equiv s_b?$ ”. In our checker, s_i is a paraphrase of the sentence s randomly chosen from s_a or s_b . Without loss of generality, let s_a be the chosen s and hence, the checker already expects $s_i \equiv s_a$ because of the paraphrasing. Then, the really interesting question is “ $s_i \equiv s_b?$ ”. If LLM’s answer is “yes”, then the original claim “ $s_a \not\equiv s_b$ ” can be rejected based on the *triangular evidence*: $(s_a \not\equiv s_b, s_i \equiv s_b, s_i \equiv s_a)$ which forms a contradiction.

Hence, for a “no” claim, the checker is looking for such a triangular evidence. To improve checker’s chance to find a triangular evidence, the paraphrase to be selected as the s_i can be a factor. Our checker gives priority to an s_i such that a weak or strong proof

can be found between s_i and s_b . In this case, we call s_i an *indifferentiable paraphrase*, i.e. from the checker’s point of view, s_i is indifferentiable from both s_a and s_b . Later in Section 8.4.3, experimentally we will show that for up to 46.8% ($= \frac{858}{1832}$) of the “no” claims, such an indifferentiable paraphrase can be found (Table 8.7).

8.4 Experiments

For entity extraction, the findings are based on 5000 sentences randomly sampled from the DOCRED dataset [228]. In addition, we use 500 sentences randomly sampled from the RISC-V unprivileged ISA specification [197] for comparison. For paraphrase decision, the experiments are based on 5000 samples from the MSR Paraphrase corpus [229]. In all experiments, we treat GPT3.5 [16] as our oracle.

8.4.1 Prompts

Prompts for GH-check

In our work, we did not focus on optimizing the prompts to GPT3.5. Prompt engineering was only pursued to the point of getting enough consistent and valid responses. To follow common convention in NER, our prompt also asks GPT3.5 to provide a *class* (or type) for each entity. Classes in the DOCRED dataset includes such as “people”, “location”, “organization”, and so on. As a contrasting example, the chemical NER dataset CHEMDNER [234] (which we did not use), includes classes of “abbreviation”, “family”, and “formula” (of chemicals), and so on. It might typically be expected that an entity is formed only by consecutive words from the text. However, we did not enforce this constraint. For example, consider that for the phrase “William and Annie Washington”. GPT3.5 would extract both “William Washington” and “Annie Washington” as entities.

In contrast, the DOCRED labels only have “William” for the first entity.

Prompts are used for two purposes: to perform the entity extraction task and to help obtain the synonym entity set E'_s discussed in section 8.2.2.

Prompt 8.2 shows the prompt for entity extraction. We gave an example entity list as the output in the prompt, but we did not provide an example input nor explicitly define classes. This is because we did not want to over-constrain the responses, even though adding a full in-context example might increase the number of valid responses.

Prompts 8.3, 8.4, and 8.5 are used for obtaining the synonym entity set E'_s . Prompt 8.3 is a decision-making prompt to determine if an extracted entity is name based. Prompts 8.4 and 8.5 are generation prompts, and produce the actual text to replace entities in our sentence transformations. For these prompts we include a suggestion for formatting.

Algorithm 8.2: Entity extraction

```

1 Please perform entity recognition for all entities in the following
  sentence: "{text}"
2 Present the result as a strictly formatted numbered list e.g.
3 "1.  movie:  Wizard of Oz
4 2.  animal:  tiger"

```

Algorithm 8.3: Named/non-named classification

```

1 Is the entity "{entity}" in the sentence "{text}" a "named entity"
  or a "normal entity"? Please explain in 20 words or less, and
  then place your answer in double brackets [[ ]]

```

Algorithm 8.4: Replacement of named entity

- 1 Please list five random entities (type: "{entity_type}") that could replace "{entity}" in the sentence "{text}".
 - 2 Format your answer with a numbered list of the synonyms. e.g.
 - 3 "1. Synonym1
 - 4 2. Synonym2"
-

Algorithm 8.5: Synonym list of non-named entity

- 1 Please list synonyms for "{entity}" in the sentence "{text}".
 - 2 Format your answer with a numbered list of the synonyms. e.g.
 - 3 "1. Synonym1
 - 4 2. Synonym2"
-

Prompts for GI-check

The experiments involve three prompts. Prompt 8.6 shows the main prompt which queries GPT3.5 for the semantic equivalence on a pair of sentences. The “System Message” and “Human Message” are not part of the prompt. They are for indicating the different portions of the prompt.

Algorithm 8.6: Query for sentence-level semantic equivalence: yes or no

- 1 System Message:
 - 2 "You are a helpful assistant that decides if two sentences are paraphrases designed to output JSON."
 - 3 Human Message:
 - 4 "Decide whether the following two sentences are semantically equivalent:
 - 5 1. '{sentence1}' 2. '{sentence2}'
 - 6 Answer yes or no and provide a short explanation. Output with keys 'answer' and 'explanation'. "
-

Like that in the entity extraction experiments, we did not focus on optimization of

the prompts. As discussed with the Algorithm 8.1 before, our GI-check involves asking the oracle whether two candidate sets of phrases (set of matching pairs) are semantically equivalent or not. Prompt 8.7 shows the prompt used for this purpose. The “text1” and “text2” in the prompt refer to the two phrases in a matching pair found by the algorithm. Furthermore, our GI-check requires the capability of generating paraphrases from a sentence. Prompt 8.8 shows the prompt for this purpose.

Note that for the checker to check on a GPT3.5’s “Yes” claim, the number of queries depend on the number of matching pairs found. This number is usually not large and on the order of 10. To check on a GPT3.5’s “No” claim, in addition to the queried used to find a proof, the checker needs to ask 3 questions to GPT3.5, two for generating the 5 paraphrases for each sentence and one for asking the semantic equivalence between the selected paraphrase and one of the original sentences. In our experiments, the efficiency is mostly dominated by the response time from GPT3.5.

Algorithm 8.7: Query for phrase-level semantic equivalence: yes or no

```

1 System Message:
2 "You are a helpful assistant that decides if two texts could be
  paraphrases designed to output JSON."
3 Human Message:
4 "Decide whether the following two texts could be considered
  semantically equivalent:
5 1. '{text1}' 2. '{text2}'
6 Answer yes or no and provide a short explanation. Output with keys
  'answer' and 'explanation'. "
```

Algorithm 8.8: Query for generating paraphrases

```

1 System Message:
2 "You are a helpful paraphrases generator designed to output JSON."
3 Human Message:
4 "Generate five paraphrases that are semantically equivalent to the
  following sentence: '{sentence}'
5 Output with key 'paraphrases'."
```

8.4.2 Results on entity extraction

To perform GH-check, five tests are run for a given sentence s . Initially, the entity set E_s is obtained by repeating the entity extraction request 11 times and taking the most consistent entity set as the answer. Then, to perform each test, 3 modified sentences (corresponding to the x, z and $x \oplus z$) are obtained with the synonym replacements (see Section 8.2.2). This modification relies on the set of synonym E'_s obtained also by querying GPT3.5. To check $f(x) \ominus f(z) = f(x \oplus z)$, the entity extraction requests on the three modified sentences each is also repeated 11 times. Hence, the checker generates 33 queries to GPT3.5 for one test. If any of the 11^3 answer combinations passes the GH-check, the checker considers the test passed. If all five tests pass, the checker accepts the answer, i.e. accepts the entity set as a whole and otherwise, rejects it.

Table 8.1 summarizes our findings by showing three main results: (1) the accept/reject % on the 5000 DOCRED sentences, (2) the accept/reject % on the 500 RISC-V sentences, and (3) the correlations across a measure on the in-sample acceptance rate (A_{rate}), a measure on the in-sample consistency of GPT3.5 response (Con_o, Con_{rs}), and the # of entities under each GH-check ($|E_s| = |E'_s| = m$).

Table 8.1: (1) GH-check on RISC-V sentences fails more than DOCRED sentences (51.6% vs. 20.46%). (2) GPT3.5 extracted about 70% ($= \frac{8830+2349}{15931}$) of the human-labeled entities and also extracted 7457 not labeled as entities in DOCRED.

%	Accept	Reject
DOCRED	79.54	20.46
RISC-V	48.4	51.6
# of labeled entities in the GPT3.5's extracted set (4752 not extracted)		
	∈ Accept	∈ Reject
Labeled	8830	2349

Specifically, an in-sample consistency is measured as a % based on the number of

Table 8.2: (1) In-sample acceptance rate (A_{rate}) is correlated to the in-sample consistency measure (Con_{rs}), and not as correlated to $|E_s|$. (2) Cross-sample correlations: Con_o is the % of consistency on the original sentence. Con_{rs} is the % of smallest consistency on the 15 modified sentences. A_{rate} is the % of passing out of the 11^3 combinations.

DOCRED /RISC-V	Con_o	Con_{rs}	$ E_s $
A_{rate}	0.80/0.65	0.86/0.72	-0.41/-0.65
$\#E$	-0.26/-0.43	-0.30/-0.45	—

consistent responses in 11 repeated requests. Con_o is based on the original s when requesting GPT3.5 for obtaining E_s . Con_{rs} is the smallest such measure across 15 modified sentences during the five tests. While we consider passing any of the 11^3 combinations as passed, A_{rate} measures the number of 11^3 combinations passing the GH-check as a %. Since A_{rate} , Con_o , Con_{rs} , and $|E_s|$ are measured on each sentence, across a collection of sentences we can therefore calculate the Spearman correlation between any pair of them. The result is shown in the Table 8.1.

From the experiment results, we summarize three points: (1) Entity extractions on sentences from RISC-V are much less trustable (31% less) than entity extraction on sentences from DOCRED. This result quantifies our intuition that extracting domain-specific entities (e.g. “hardware entities”) is different from and can be harder than entity recognition as commonly studied in NLP research. (2) Even for entity extraction on DOCRED, GPT3.5 can disagree with human labels on a significant number of sentences, indicating that it is difficult, if not impossible, to define a so-called correct answer for every entity extraction per sample. (3) The high correlations between A_{rate} and Con_{rs} reveal that for per-sample examination, our GH-check is somewhat similar to and yet not the same as measuring the consistency from repeated runs. Interestingly, this correlation is lower for RISC-V sentences, indicating again the GPT3.5’s behavior on them is different from that on the DOCRED sentences.

8.4.3 Results on paraphrase decision

As noted in the MSR Paraphrase (MSRP) corpus dataset [229], “semantic equivalence” can really mean “semantic near-equivalence” that a pair of sentences ideally entail each other but often might have some minor mismatches in their content. Deciding whether or not the difference in content is significant enough to make two sentences not semantically equivalent, can be a personal judgment call. In MSRP corpus, the label annotators made these judgment calls. When we ask GPT3.5 with paraphrase decisions on sentence pairs from MSRP, the GPT3.5 makes its own judgment calls.

Finding proofs for “Yes” (semantic equivalence) claims

The MSRP dataset contains sentence pairs (s_a, s_b) with a label to indicate if they are semantically equivalent. Below, we use “Yes” to denote semantic equivalence and “No” to denote non-equivalence. 5000 MSRP sentence pairs are given to GPT3.5 to ask the semantic equivalence question. Each pair has two results: the “Yes/No” result by the label and the “Yes/No” answer by GPT3.5. We ran our checker according to the answers given by GPT3.5.

The 5000 pairs are divided into two categories, those answered “Yes” and those answered “No” by GPT3.5. As mentioned before, they are treated differently by our checker. In this section, we focus on those “Yes” pairs. Table 8.3 summarizes the experiment results.

The first result compares GPT3.5’s answers to the given labels. Overall, GPT3.5 disagrees on 25.16% of the labels. This was our initial evidence to confirm that indeed, for a significant number of pairs there might not be a clear judgment to decide their semantic equivalence. As a result, we anticipated that it might be difficult for our checker’s acceptance rate on GPT3.5’s answers to surpass 75%, because for at least 25% of the

pairs there might not be a correct answer. Then, when we apply our checker on those 3168 “Yes” cases given by GPT3.5, we found that the checker was able to find weak proof (WP) on 57% of them and strong proof (SP) on 32.7% of them.

Table 8.3: GPT3.5 claims compared with MSRP labels. * indicates GPT disagreements with the labels. GPT3.5 disagrees on 25.16% of the human labels in MSRP.

Label	GPT	%
Yes (3454)	Yes	77.7
	No*	22.4
No (1546)	Yes*	31.4
	No	68.6

Table 8.4: % of provable “Yes” cases is up to 57%.

	WP (%)	SP (%)
GPT (3168)	57.0	32.7
Label (3454)	54.0	31.8

Table 8.5: On 500 GPT3.5’s own generated paraphrases, GPT3.5 answers “Yes” on 495 and their provable % is shown below.

WP (%)	SP (%)
62.5	45.3

Out of curiosity, we were also interested in seeing on how many labeled “Yes” pairs for which our checker could also find a proof. The result is that 54% has a WP and 31.8% has a SP, as shown in Table 8.4. These two numbers are comparable to the previous two numbers reported on GPT3.5. This shows that the GI-check used by our checker has its own judgment on semantic equivalence, which is somewhat independent from both the human annotators’ and GPT3.5’s judgment.

Then, instead of using pairs from MSRP, we were interested in assessing GPT3.5 based on its own generated-paraphrases. Hence, we randomly selected 500 sentences from MSRP and asked GPT3.5 to generate a paraphrase for each to make a pair. On these 500 pairs, GPT3.5 answers “Yes” on 495. On those 495 pairs, our checker has a better chance to find a proof: WP for 62.5% and SP for 45.34%, as shown in the Table 8.5.

Note that in the above experiments, we only asked GPT3.5 with each question once and took the answer as it. There were no repeated runs as that in Section 8.4.2. However, we did a sanity check by repeating the semantic equivalence question five times on 500 randomly sampled pairs from MSRP. We found that for only 59 pairs, there is an inconsistency in the five answers. If we did this for the 500 pairs of GPT3.5 own paraphrases, the inconsistency is observed on only four pairs. We see that checking for consistency is different from our GI-check as they produce very different outcomes.

Checking “No” (semantic non-equivalence) claims

Table 8.6: Show % of those “No” cases, where our checker found a proof for their semantic equivalence.

	WP (%)	SP (%)
GPT (1832)	24.5	11.2
Label (1546)	25.9	9.7

Following the results in Table 8.3, in this section we consider those “No” cases. There are two categories: “No” from the GPT3.5 and “No” from the label. Out of 5000 pairs, there are 1832 GPT3.5 “No” cases and 1546 label “No” cases, shown in Table 8.6. First, for a sentence pair (s_a, s_b) , we tried our checker to find a proof to see if showing $s_a \equiv s_b$ is possible. The results are reported in Table 8.6 for both categories. Notice that in

Table 8.7: On 1832 GPT3.5 claimed “No” cases, show % rejected by the checker in two scenarios: *with* (“w/”) or *without* (“w/o”) using an indifferentiable paraphrase p (using WP or SP)

using WP		using SP	
w/ (858)	w/o (974)	w/ (385)	w/o (1447)
61.5%	18.8%	70.4%	30.3%
Total rejects: 711		Total rejects: 709	

both categories, our checker can find a WP to show $s_a \equiv s_b$ for about 25% of the cases, indicating many “No” cases might not be trustable.

Then, we applied our checker on the 1832 GPT3.5’s “No” cases. On each case, we focus on finding an indifferentiable paraphrase p (see Section 8.3.3). Given a pair s_a, s_b , the checker first asks GPT3.5 to generate 5 paraphrases for s_a (or s_b) and then for each paraphrase p tries to prove $p \equiv s_b$ (or $p \equiv s_a$). This proof can be a WP or a SP. If succeeded, p is an indifferentiable paraphrase. In the checking, our checker gave priority to pick an indifferentiable paraphrase p if it could be found. Otherwise, the paraphrase p was randomly picked from the 10 paraphrases generated.

Suppose p is a paraphrase of s_a , when GPT3.5 answers “Yes” to the question “ $p \equiv s_b$?”, the checker obtains the triangular evidence (s_a, s_b, p) to reject the original “ $s_a \neq s_b$ ” claim. Table 8.7 reports the % of those rejected cases. We see that by using a WP, an indifferentiable p can be found for 858 out of 1832 cases (i.e. 46.8%). Then, by using one paraphrase (i.e. $n = 1$ in Figure 8.3), if it is an indifferentiable p , the checker has higher chance (61.5%, 70.4%) to reject the original “No” claim than that using a random p (18.8%, 30.3%). This shows that using an indifferentiable p indeed has a significantly better chance to to make GPT3.5 come back with a contradictory answer. Note that overall, from the checker’s perspective, about 40% ($= \frac{(3454 \cdot (1-0.54) + 1546 \cdot 0.259)}{5000}$) of the MSRP labels can be ambiguous, when combining results from Tables 8.4 and 8.6.

8.5 Related Work

There are many works on characterization and trustworthiness of LLMs. They can be viewed in three contexts: (1) Fine tuning and evaluating on a collection of datasets is a popular approach for improving LLMs performance [235, 236, 237, 238]. Example benchmarks include those to evaluate human-GPT differences, broadly [239] and specifically in education [240, 241] and those on overall hallucination tendencies [242, 243, 244]. (2) There are works by curating a dataset specifically for the purpose of improving the models in the context of enabling preference or ranking to the output results or flagging potentially harmful results. Often the product is an auxiliary model, for example, as mathematical reward models [245], for improvement on code vulnerability [246], and for enhancement on illegal topic detection [247]. (3) Self-consistency can be an important method for evaluating a LLM. A recent work uses self-consistency [248] to improve performance, where a followup expands into a universal consistency check [249]. Auxiliary models can also be reintroduced to self-consistency as alternatives to voting [250]. In addition to the above works, there is also work focusing on the effect of the prompts by evaluating if the prompts are at fault for failure cases [251].

Our work takes a direction different from all those reviewed above, and focuses on per-sample examination. To verify an LLM’s response, our checker interacts with the LLM by asking more questions to collect evidences from the LLM to support an accept or reject decision of the response. This interactive checking does not require a benchmark to define correct answers to begin with.

8.6 Limitations

For entity extraction, our GH-check is limited to checking how the extraction of each entity depends on the extraction of others in the same sentence. Hence, it does not address the aspect that extraction of an entity can depend on the semantic meaning of the entire sentence. Also, GH-check depends on the synonym set E'_s and optimizing this set is a separate issue. For paraphrase decision, GI-check relies on phrase-level semantic equivalence to find a proof. Algorithm 8.1 is to find candidate phrase pairs for GPT3.5 to verify their semantic equivalence. Our implementation is limited in two aspects: (1) There can exist a pair not found by Algorithm 8.1, which can lead to a proof; (2) It is possible that the sentence-level equivalence claim cannot be shown at the phrase level. In both cases, our GI-check will not find a proof even though the GPT3.5’s “Yes” claim is not incorrect.

8.7 Summary

The work summarized in this chapter demonstrates how the ideas of two theoretical checks, GH-check and GI-check, can be implemented in practical setting of entity extraction and paraphrase decision, respectively, by treating an LLM as the oracle. Our OC scheme mimics the essential idea of Interactive Proofs, where the checker relies on the oracle to be convinced that an answer is acceptable. This work only scratches the surface of the rich results from the theoretical field [226]. Exploring the possibilities to apply other theoretical checks in different NLP tasks can be an interesting future research direction.

Chapter 9

Journey from IEA - The Next Decade

浩浩乎如憑虛御風，而不知其所止；

飄飄乎如遺世獨立，羽化而登僊。

Vast and grand, like a dragon soaring amidst the wind, Yet unaware of its final destination, it is pinned. Drifting and floating, like a solitary figure beyond earthly bounds, Transforming into celestial beings, ascends beyond mortal mounds.

— 《前赤壁賦》 蘇軾, A poem from Song Dynasty

In Chapter 2 to Chapter 7, we have described the two-decade journey to IEA and presented IEA-Plot as our answer to the initial question regarding “applying ML and AI in design and test” brought up at the beginning of Chapter 1. IEA-Plot presented in Chapter 6 marks the end of the current IEA journey and also the beginning of a new one. In particular, the Knowledge Graph (KG) view discussed in Chapter 7 and the Oracle-Checker (OC) scheme discussed in Chapter 8 open the door for the next decade

of the IEA journey. The KG view is the start to pursue a Generative AI (GAI) version of IEA. The OC scheme is the start to pursue a version of *checker-based* IEA that treats an LLM as an oracle. In this chapter, we will present our ideas on these two future versions of IEA and a list of important questions to be answered in the next decade.

9.1 The Generative AI View

As shown in Figure 2.1 in Section 2, our journey to IEA ends with the Generative AI (GAI) view. After the completion of the IEA-Plot in 2023, we began to see that IEA-Plot can be a stepping stone to attain GAI in the semiconductor domain.

With the growing popularity of the GPT models [176][19][16] and the stable diffusion model [252], GAI has become a mainstream trend of AI. The IEA-Plot design discussed in Chapter 6 and Chapter 7 relies on the manual construction of a KG. With the KG in place, the implementation of IEA-Plot solves a *task grounding* problem (Section 6.1.1), which is similar to the *constrained parsing* problem discussed in Section 5.4.3. It seems that our current use of the GPT model is limited to “parsing” rather than “generating”. And the next natural question to ask is: Can we design an IEA that utilizes the code generation power of an LLM?

9.1.1 IEA-Plot as a query generator

To answer this question, we should start by noting that the frontend parser for solving the task grounding problem already followed a generative approach, as discussed with Figure 6.11 in Section 6.5. From the KG, a large number of (canonical) acceptable user inputs (queries) are generated. Then, by leveraging the paraphrasing power of GPT3.5, this initial query set can be expanded into a larger set.

Figure 9.1 depicts this view by seeing IEA-Plot as a query generator. Essentially,

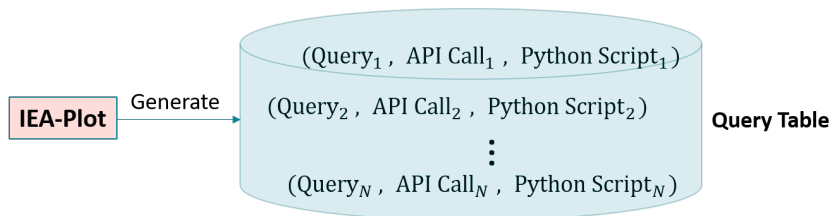


Figure 9.1: IEA-Plot is a query generator

IEA-Plot is capable of generating a very large *query table* where each entry in this table is a 3-tuple, containing the query, the API calls to the backend for executing the query, and the actual Python script. This query table can then be used as the training dataset to retrain an LLM (e.g. fine-tuning).

For example, each training sample can be the query followed by the sequence of the API calls represented as a sequence of symbolic steps. The the trained LLM will be able to take a user query as input and generate a sequence of steps which can be used to call the backend API. This LLM is still *grounded* by the backend API.

As another example, each training sample can be the query followed by the Python code. Then, the trained LLM will be able to behave like a code generator where a user provides an instruction for the LLM to generate directly a piece of Python code according to the instruction. The difference between this LLM and the current LLM like the GPT model is that with our domain specific KG in place, such an LLM will be able to truly understand domain-specific terms and generate code specific to the domain-specific application.

9.1.2 Toward a wafermap generator

On a smaller scale, a GAI approach can also be implemented within the scope of wafermap analytics. Using a Natural Language Interpreter (NLI) such as the one presented in Section 5.5.2, we have shown a way to treat the MINIONS+NLI as a separate

component for generating a wafermap database. An example of a table in such a database is already presented in Table 5.11 in Section 5.5.

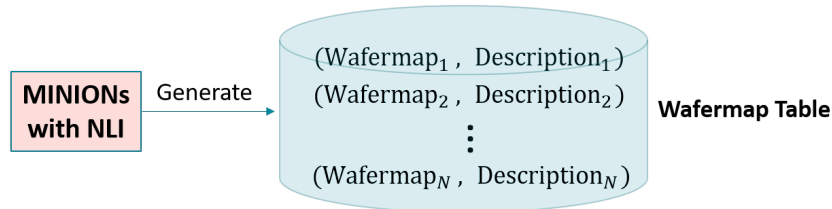


Figure 9.2: MINIONS with a Natural Language Interpreter (NLI) as a wafermap generator

Figure 9.2 presents the idea, in parallel to the idea present in Figure 9.1 above. Each sample in the wafermap table is a 2-tuple, containing a wafermap image and a description of the wafermap. With such a dataset available, we can then train a diffusion model [252] as a wafermap generator. This model should have the capability that on a user description, it will be able to generate a very large number of wafermaps according to a user description.

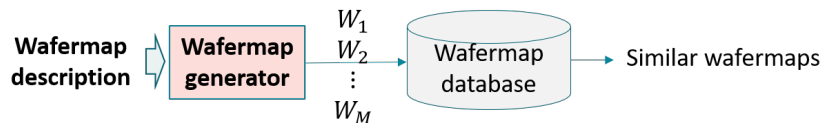


Figure 9.3: With a wafermap generator, retrieval of wafermaps according to a user description can be made simpler — similarity search achieved (and simplified) by a generative approach

Suppose such a wafermap generator is there. Then, Figure 9.3 shows that essentially the generator can replace our current MINIONS based approach for wafermap analytics. Given a user description of what pattern to look for, the generator can produce a very large number of wafermaps $\mathcal{W} = W_1, \dots, W_M$. These generated wafermaps can be compared to the wafermaps in the database and find those wafermaps *very close* to the generated ones. This similarity search can be made simple, e.g. just based on the cosine similarity between two images, as long as M is large enough. In a sense, with a wafermap

generator in place we no longer require to use a MINIONs graph to determine the similarity between a pair of wafermaps. The similarity between two wafermaps W_a and W_b in the database can be determined through their closeness to wafermaps in the group \mathcal{W} . If W_a is very close to $W_i \in \mathcal{W}$ and W_b is very close to $W_j \in \mathcal{W}$, then W_a is similar W_b (they belong to the same pattern group) in view of the given user description. Note this retrieval base on cosine similarity is akin to what we proposed in Section 6.5.1 and 6.5.2, where the former is based on image similarity and the latter on text embeddings.

9.1.3 Questions to be answered in the next decade

If the future journey of IEA takes on the GAI path as described above, then there will be several fundamental questions to be answered. Figure 9.4 depicts one of the fundamental questions.

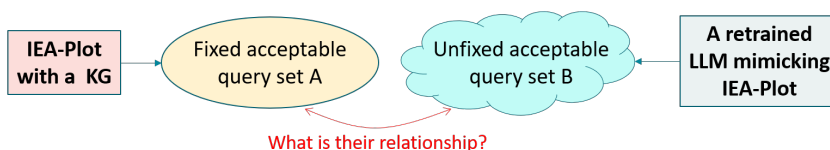


Figure 9.4: Does a retrained LLM provide generalization from A to B?

In the current IEA, KG is a formal representation of the domain knowledge. The KG implies a set of *canonical queries* acceptable by IEA. The variations of these queries come from the paraphrasing power of the LLM. In a sense, the set of acceptable queries is *fixed* by the KG. If we train an LLM following the GAI approach discussed in Section 9.1.1 above, essentially this LLM is mimicking the behavior of IEA-Plot. In a sense, the domain knowledge in the KG is assimilated into the LLM and after that, the set of acceptable queries is no longer fixed. Given the current fixed acceptable query set A and the future unfixed acceptable query set B, the fundamental question is to ask about the relationship between these two sets. In other words, we will be interested in knowing

what *generalization* the LLM can provide to go beyond the current acceptable set A.

In view of the possible generalization through retraining an LLM, below we list a number of fundamental questions to drive the IEA journey in the next decade.

- Can the LLM generalize well when its responses are limited within the scope of API calls? In other words, can the LLM find a novel way to utilize our backend API to accomplish an analytic task?
- Can the LLM generalize well when its responses are Python codes? In other words, can the LLM write new Python code beyond the scope of our backend API to accomplish an analytic task?
- For the LLM, how much benefit can prompt engineering bring? In other words, to what extent does prompts have a effect on the quality of the responses?
- With the LLM, do we still require some sort of *grounding* on its responses? In other words, with the LLM can we forget about the KG and constrained parsing entirely when building the GAI version of the IEA?
- Suppose in the new IEA, we no longer need to solve a grounding problem. With a proper prompt, there is a way to get to a response (a sequence of API calls or a piece of code) that can be acceptable for a given analytic task. Then, this means that the new IEA only needs to be based on a set of *checkers* to verify the trustworthiness of the generated responses. Then, the fundamental question is: What types of checkers are needed to make an IEA?

In bringing up the last question, we are essentially seeking the direction of shifting the focus of IEA from one based on the GAI paradigm into a checker version of IEA based on the oracle-checker scheme presented in Chapter 8.

9.2 The Oracle-Checker View

Property testing [226] and Interactive Proofs [224] are two rich fields in theoretical computer science. Our OC scheme presented in Chapter 8 makes the first step that draws upon those theoretical ideas and bring them into practical realization (in the context of per-sample examination of LLM’s responses by treating the LLM as an oracle). Our OC scheme starts two new research areas along the IEA journey:

- How to apply the OC scheme to assist in the IEA development.
- How to make a checker version of IEA in the future by treating an LLM as a domain-specific oracle.

In view of these two new research areas, we can then list a number of fundamental questions to be answered in the next decade.

9.2.1 Questions to be answered in the next decade

In our initial work presented in Chapter 8, two basic NLP tasks are selected: entity extraction and paraphrase decision. This is not by accident. We selected them not only because they are relevant to our objective in view of achieving automatic KG construction, but also because the LLM has already shown reliable performance on those tasks in general. In other words, the LLM *can* be treated as an oracle for the tasks.

The assumption to apply our OC scheme is that the LLM can be treated as an oracle. This means that the LLM has to generally show good performance on the given task to begin with and on top of that, has the ability to support a session of interactive checking on its response. If the LLM’s responses are not above a certain quality level to begin with, then it will put too much burden on the checker to verify the responses. As a

result, the checker can become rather complex, violating the original goal of keeping the checker simple and efficient.

If the LLM is less reliable on a given task, the checker needs to do more work on the checking. There is a tradeoff between the two. At a certain point, the checker can become too complex to be practical. In view of this LLM-checker relationship, we can then ask a number of fundamental questions as below.

- On what criteria can an LLM be used as an oracle on a task?
- On what criteria can an LLM be used as a domain-specific oracle on a domain-specific task?
- Given a rich set of property testers covered in [226], is it possible to realize other testers in some other tasks relevant to IEA?
- On what criteria should an LLM not be used as an oracle on a task?
- On what criteria should an LLM not be used as a domain-specific oracle on a domain-specific task?

Note that the last two questions are different from the first two and can be harder questions to answer. The first two ask for criteria for the acceptance of an LLM as an oracle. The last two ask for criteria for the rejection of an LLM as an oracle. The rejection kind of the questions can be more difficult to find a solution. From a research point of view, most of the proposed works always prefers a positive result than a negative one. Nevertheless, negative results are crucial because they can guide us to avoid the realm of no solution and prevent us from being trapped in the world of no-free-lunch.

Chapter 10

Conclusion

飛來峰上千尋塔，聞說雞鳴見日升。

不畏浮雲遮望眼，只緣身在最高層。

Upon the peak, a tower stands tall, a thousand feet it soars. They say the cock crow heralds the sunrise, seen from its lofty floors. Unafraid of clouds that may obscure the sight. For only from the highest perch, the truest view alights.

— 《登飛來峰》 王安石， A poem from Song Dynasty

Figure 2.1 in Chapter 2 depicts the evolution of views through the two-decade journey of IEA. This thesis provides detail on this evolution and Figure 10.1 is a recap of Figure 2.1. The OC scheme presented in Chapter 8 is a detour on the journey. While it is closely related to the IEA development, the work presented in Chapter 8 can be seen as the start for a new research area by itself.

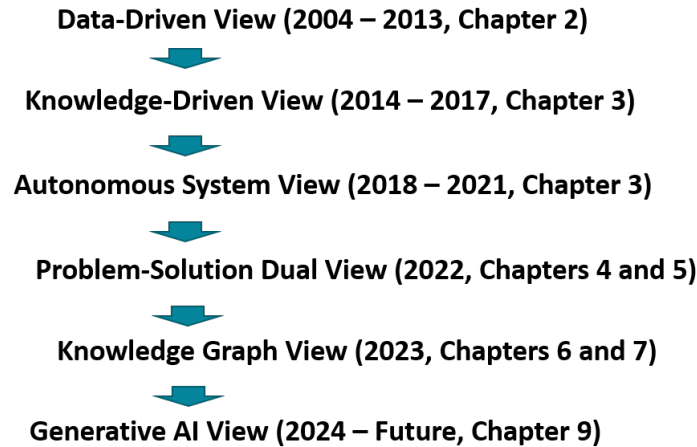


Figure 10.1: Evolution of views along the journey to IEA

10.1 Takeaways

Data-Driven View Figure 2.22 in Section 2.7 is a summary of the lessons learned during the first decade. It was due to the studies carried out in the first decade that we learned the implications of the *data wall* (Section 2.7.1, Section 2.7.2). One of the implications is that domain-specific ML means Decision-Support ML (DSML) (Section 2.7.4) which demands solving a Co-ML problem (Section 2.7.4). It was also during the first decade of studies that the necessity of domain knowledge in domain-specific learning became apparent.

Knowledge-Driven View Lessons learned from this period include the following: (1) In an attempt to model domain knowledge, it was realized that the end results of analytics could be thought of as plots. This led to the idea of *plot-based analytics* (Section 3.3). (2) In view of constructing a plot, the domain knowledge could be modeled as a sequence of *primitive steps*. The idea of primitive step motivated the later idea of *concept recognition* in the Autonomous System view (Section 3.10). (3) It was the three challenges (Section 3.5) faced in this period motivated the later idea of *language-driven analytics* emerging under the Problem-Solution Dual view (Chapter 5).

It was also during this period of studies that the idea of local no-free-lunch (L-NFL) was formed (Section 3.9), the meaning of Occam learning in DSML became clear (Section 3.9.3), and the essence of DSML was uncovered (Section 3.12) where a DSML oracle was supposed to solve a decision problem rather than an optimization problem.

Autonomous-System View This view enabled the construction of the first IEA in 2018 (Section 3.10). The most important discovery from the development of IEA 2018 was that domain knowledge was modeled in terms of *concepts*. At the time, DSML oracles were thought to be *concept recognizers*. The most important lesson learned from IEA 2018 was the infeasibility to capture all domain knowledge in an AI Assistant. Some domain knowledge was hard to capture and should be left outside of IEA. The lesson turned the use of the NLP interface from the purpose of information retrieval to the purpose of driving the analytics (and hence, became language-drive analytics).

Problem-Solution Dual view The MINIONs approach discussed in Chapter 4 provides a concrete example for the Problem-Solution Dual view discussed in Chapter 5. Realizing the Problem-Solution Dual view was the most important turning point of the entire IEA journey. It substantiated the idea of language-driven analytics (Section 5.2) and led to the implementation of IEA 2022 (Section 5.2.2).

Knowledge Graph view The completion of IEA-Plot in 2023 marks the end of the two-decade journey and also the beginning of a new journey. IEA-Plot (Section 6) demonstrates how to leverage the power of a latest LLM to build an AI Assistant in a domain-specific application. At the core, it solves a *task grounding* problem (Section 6.1.1). And in IEA-Plot, task grounding is achieved through a knowledge graph (Chapter 7).

The manual construction of KG in IEA-Plot was necessary, even though it was tedious. The KG view enabled us to see the difference between problem formulation and problem solving (Section 7.3) in IEA, where in practice a complex problem statement described

with a language in polynomial hierarchy does not mean solving the problem requires the same level of complexity. This is an important observation leading to the hope to pursue a Generative AI approach for implementing a future IEA.

Generative AI view In IEA-Plot, the task grounding problem was solved by taking a generative approach (Section 6.5). Section 9.1 therefore discusses the next step to achieve a Generative AI (GAI) version of IEA and posts several fundamental questions to be answered in the next decade of the IEA journey. The core question is on how much generalization a GAI version of IEA can achieve based on learning from a KG version of IEA (e.g. IEA-Plot), or if there is any meaningful generalization at all (Section 9.1.3).

The Oracle-Checker (OC) scheme (Chapter 8) is novel and yet, leaves several fundamental questions unanswered. In an OC scheme, the checker is supposed to be much simpler than the oracle, in view of the complexity for completing a task. The checker relies on interactions with the oracle to verify oracle's responses. Our work presented in Chapter 8 opens the door to start a new direction of research. And the most essential question to be answered through the research is (Section 9.2.1): On what criteria can an LLM be treated as an oracle?

10.2 Ten Questions And Their Answer

The two-decade journey provides answers to some of the important questions arising from the original motivation to apply ML in semiconductor chip design and test. Below we summarize ten such questions that can be useful for practitioners to consider when trying to apply ML in their applications. Based on the materials presented in this thesis, we provide a short answer to each question.

Q: I have many ML-based analytic tools already in place in my company.

Which one is the best for my application?

A: According to the Problem-Solution Dual View, that is the wrong question to ask.

Q: I heard the new ML model (or technology) XYZ published last month? Should I take a look? Maybe it can help solve my problem.

A: Unlikely, especially if you have not fully understood the essence of your problem yet.

Q: Then, how can I apply ML in semiconductor chip design and test?

A: Build an IEA.

Q: What is the difference between ML and domain-specific ML?

A: Domain-specific ML means Decision-Support ML (DSML). In DSML, a solver solves a decision problem. In ML, a solver (a model) solves an optimization problem.

Q: What is domain knowledge?

A: It is a collection of statements that involve domain-specific concepts whose interpretation is subject to the domain expert.

Q: How to incorporate my domain knowledge into analytics?

A: Build a knowledge graph in IEA.

Q: How to leverage the power of LLM in semiconductor chip design and test?

A: Build a knowledge graph and solve a task grounding problem as the first step.

Q: How to build a KG?

A: Start with a document of descriptions for what you desire to be represented in the KG and use an LLM under the OC scheme to generate a draft of KG.

Q: If I help build an IEA, will IEA replace my job?

A: No. Some of your domain knowledge is left with you. IEA is only your assistant.

Q: Can I train an LLM to do what I want in my application?

A: Yes, but you need to build a KG version of IEA first before you can get to a GAI version of IEA.

10.3 Philosophical Remarks

This thesis is concluded with the following philosophical remarks.

When solving a problem,
think about the problem and solution as a pair.

If you believe in free lunch, apply ML.

If you concern about no-free-lunch, apply DSML.

If trying to be an oracle is too hard, try to be an effective checker.

The fastest way to get to a destination is
to know giving up on a difficult route and see an alternative,
even though at the time the alternative might look like a big detour.

In practice, there is no real difficulty in a problem,
only difficulty in yourself.

Bibliography

- [1] L.-C. Wang, “Data learning based diagnosis,” *ACM/IEEE ASP Design Automation Conference*, pp. 247–254, 2010.
- [2] L.-C. Wang, “Experience of data analytics in EDA and test - principles, promises, and challenges,” *IEEE Transactions on CAD*, vol. 36, no. 6, pp. 885–898, 2017.
- [3] L.-C. Wang, “Data mining in functional test content optimization,” *ACM/IEEE Asian South Pacific Design Automation Conference*, 2015.
- [4] W. Chen, L.-C. Wang, and J. Bhadra, “Simulation knowledge extraction and reuse in constrained random processor verification,” *ACM/IEEE Design Automation Conference*, pp. 1–6, 2013.
- [5] K.-K. Hsieh, W. Chen, L.-C. Wang, and J. Bhadra, “On application of data mining in functional debug,” *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 670–675, 2014.
- [6] N. Sumikawa, J. Tikkanen, L.-C. Wang, L. Winemberg, and M. S. Abadir, “Screening customer returns with multivariate test analysis,” *IEEE International Test Conference*, 2012.
- [7] jeff Tikkanen, N. Sumikawa, L.-C. Wang, and M. S. Abadir, “Multivariate outlier modeling for capturing customer returns — how simple it can be,” *IEEE 20th International On-Line Testing Symposium (IOLTS)*, pp. 164–169, 2014.
- [8] S. Siatkowski, L.-C. Wang, N. Sumikawa, and L. Winemberg, “Learning the process for correlation analysis,” *IEEE VLSI Test Symposium*, 2017.
- [9] J. Tikkanen, S. Siatkowski, N. Sumikawa, L.-C. Wang, and M. S. Abadir, “Yield optimization using advanced statistical correlation methods,” *IEEE International Test Conference*, 2014.
- [10] K.-K. Hsieh, L.-C. Wang, W. Chen, and J. Bhadra, “Learning to produce direct tests for security verification using constrained process discovery,” *ACM/IEEE Design Automation Conference*, pp. 1–6, 2017.

- [11] a. e. Sebastian Siatkowski, “Consistency in wafer based outlier screening,” in *IEEE VLSI Test Symposium*, 2016.
- [12] L.-C. Wang, *Learning from Limited Data in VLSI CAD*, pp. 375–399. Springer International Publishing, 2019.
- [13] N. Sumikawa, M. Nero, and L.-C. Wang, “Kernel based clustering for quality improvement and excursion detection,” *IEEE International Test Conference*, 2017.
- [14] Y. J. Zeng, L.-C. Wang, and C. J. Shan, “Miniature interactive offset networks (minions) for wafer map classification,” in *IEEE International Test Conference*, pp. 190–199, 2021.
- [15] Y. J. Zeng, L.-C. Wang, C. J. Shan, and N. Sumikawa, “Learning a wafer feature with one training sample,” in *IEEE International Test Conference*, pp. 1–10, 2020.
- [16] OpenAI, “Gpt-3.5-turbo,” 2022.
- [17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.
- [18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States* (P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1106–1114, 2012.
- [19] OpenAI, “Introducing ChatGPT,”
- [20] S. Bengesi, H. El-Sayed, M. K. Sarker, Y. Houkpati, J. Irungu, and T. Oladunni, “Advancements in generative AI: A comprehensive review of gans, gpt, autoencoders, diffusion model, and transformers,” *IEEE Access*, vol. 12, pp. 69812–69837, 2024.
- [21] R. Miikkulainen, “Generative AI: an AI paradigm shift in the making?,” *AI Mag.*, vol. 45, no. 1, pp. 165–167, 2024.
- [22] A. Pandey, “The generative ai boom in 6 charts,” 2024.
- [23] N. Scoble-Williams, D. Sinti, and G. Vert, “Generative ai and the future of work,” 2024.
- [24] 2024.
- [25] N. Arya, “The growth behind llm-based autonomous agents,” 2023.

- [26] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, A. Jindal, B. Khailany, G. Kokai, K. Kunal, X. Li, C. Lind, H. Liu, S. Oberman, S. Omar, G. Pasandi, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, W. Turner, K. Xu, and H. Ren, “Chipnemo: Domain-adapted llms for chip design,” 2024.
- [27] K. HEYMAN, “Eda pushes deeper into ai,” 2023.
- [28] F. Persia, “Labview in the age of ai-driven programming,” 2023.
- [29] N. Callegari, D. G. Drmanac, L.-C. Wang, and M. S. Abadir, “Classification rule learning using subgroup discovery of cross-domain attributes responsible for design-silicon mismatch,” *ACM/IEEE Design Automation Conference*, pp. 374–379, 2010.
- [30] N. Callegari, L.-C. Wang, and P. Bastani, “Speedpath analysis based on hypothesis pruning and ranking,” *ACM/IEEE Design Automation Conference*, pp. 346–351, 2009.
- [31] L.-C. Wang and et al., “Some considerations on choosing an outlier method for automotive product lines,” *IEEE International Test Conference*, 2017.
- [32] N. Sumikawa, L.-C. Wang, and M. S. Abadir, “A pattern mining framework for inter-wafer abnormality analysis,” *IEEE International Test Conference*, 2013.
- [33] D. H. Wolpert, “The lack of a priori distinctions between learning algorithms,” *Neural Compt.*, vol. 8, no. 7, pp. 1341–1390, 1996.
- [34] D. H. Wolpert, “The Relationship Between PAC, the Statistical Physics framework, the Bayesian framework, and the VC framework,” *Technical Report, SFI-TR-03-123*, 2003.
- [35] M. Dupree, M. J. Yang, Y. J. Zeng, and L.-C. Wang, “Iea-plot: Conducting wafer-based data analytics through chat,” in *IEEE International Test Conference*, IEEE, 2023.
- [36] L.-C. Wang, “Regression simulation: applying path-based learning in delay test and post-silicon validation,” in *Proceedings Design, Automation and Test in Europe Conference*, pp. 692 – 693, IEEE, 2004.
- [37] L.-C. Wang, T. Mak, K.-T. Cheng, and M. S. Abadir, “On path-based learning and its applications in delay test and diagnosis,” in *Proceedings ACM/IEEE Design Automation Conference*, pp. 492 – 497, ACM/IEEE, 2004.
- [38] B. Schölkopf and et al., *Learning with Kernels:Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

- [39] L.-C. Wang, J.-J. Liou, and K.-T. Cheng, “Critical path selection for delay fault testing based upon a statistical timing model,” *IEEE Transactions on CAD*, vol. 23, no. 11, pp. 1550 – 1565, 2004.
- [40] L.-C. Wang, P. Bastani, and M. S. Abadir, “Design-silicon timing correlation — a data mining perspective,” *ACM/IEEE Design Automation Conference*, pp. 384–389, 2007.
- [41] B. Lee, L.-C. Wang, and M. S. Abadir, “Refined statistical static timing analysis through learning spatial delay correlations,” *ACM/IEEE Design Automation Conference*, pp. 149–154, 2006.
- [42] P. Bastani, N. Callegari, L.-C. Wang, and M. S. Abadir, “Statistical diagnosis of unmodeled systematic timing effects,” *ACM/IEEE Design Automation Conference*, pp. 355–360, 2008.
- [43] P. Bastani, N. Callegari, L.-C. Wang, and M. S. Abadir, “Diagnosis of design-silicon timing mismatch with feature encoding and importance ranking – the methodology explained,” *IEEE International Test Conference*, pp. 1–10, 2008.
- [44] N. Callegari, P. Bastani, L.-C. Wang, and M. S. Abadir, “A statistical diagnosis approach analyzing design-silicon timing mismatch,” *IEEE Transactions on CAD*, vol. 28, no. 11, pp. 1728 – 1541, 2009.
- [45] P. Bastani, K. Killpack, L.-C. Wang, and E. Chiprout, “Speedpath prediction based on learning from a small set of examples,” *ACM/IEEE Design Automation Conference*, pp. 217–222, 2008.
- [46] V. Vapnik, *The Nature of Statistical Learning Theory*. 2000.
- [47] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. 1994.
- [48] J. Chen, B. Bolin, L.-C. Wang, J. Zeng, D. G. Drmanac, , and M. Mateja, “Mining ac delay measurements for understanding speed-limiting paths,” *IEEE International Test Conference*, 2010.
- [49] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski, “Rule induction for subgroup discovery with cn2-sd,” *Journal of Machine Learning Research*, vol. 5, pp. 153–188, 2004.
- [50] N. Callegari, L.-C. Wang, and P. Bastani, “Feature based similarity search with application to speedpath analysis,” *IEEE International Test Conference*, pp. 1–10, 2009.

- [51] O. Guzey, L.-C. Wang, J. Levitt, and H. Foster, “Functional test selection based on unsupervised support vector analysis,” *ACM/IEEE Design Automation Conference*, pp. 262–267, 2008.
- [52] P.-H. Chang, L.-C. Wang, and J. Bhadra, “A kernel-based approach for functional test program generation,” *IEEE International Test Conference*, pp. 1–10, 2010.
- [53] W. Chen, N. Sumikawa, L.-C. Wang, J. Bhadra, X. Feng, and M. S. Abadir, “Novel test detection to improve simulation efficiency — a commercial experiment,” *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 101–108, 2012.
- [54] O. Guzey, L.-C. Wang, and J. Bhadra, “Enhancing signal controllability in functional test-benches through automatic constraint extraction,” *IEEE International Test Conference*, pp. 1–10, 2007.
- [55] O. Guzey, C. Wen, L.-C. Wang, T. Feng, H. Miller, and M. S. Abadir, “Extracting a simplified view of design functionality based on vector simulation,” in *Hardware and Software, Verification and Testing*, pp. 34–49, Springer Berlin Heidelberg, 2007.
- [56] M. J. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*. The MIT Press, 1994.
- [57] C. Zhang and S. Zhang, “Association rule mining, models and algorithms,” *Lecture Notes in CS*, vol. 2307, 2002.
- [58] N. Sumikawa, D. G. Drmanac, L.-C. Wang, L. Winemberg, and M. S. Abadir, “Forward prediction based on wafer sort data — a case study,” *IEEE International Test Conference*, 2011.
- [59] D. G. Drmanac, L.-C. Wang, and M. Laisne, “Wafer probe test cost reduction of an rf/a device by automatic testset minimization: A case study,” *IEEE International Test Conference*, 2011.
- [60] N. Sumikawa, Li-C.Wang, and M. S. Abadir, “An experiment of burn-in time reduction based on parametric test analysis,” *IEEE International Test Conference*, 2010.
- [61] L.-C. Wang and M. S. Abadir, “Data mining in eda - basic principles, promises, and constraints,” *ACM/IEEE Design Automation Conference*, pp. 1–6, 2014.
- [62] J. Zeng, M. S. Abadir, G. Vandling, L.-C. Wang, A. Kolhatkar, and J. Abraham, “On correlating structural tests with functional tests for speed binning of high performance design,” *IEEE International Test Conference*, pp. 31 – 37, 2004.

- [63] J. Chen, L.-C. Wang, P.-H. Chang, J. Zeng, S. Yu, and M. Mateja, "Data learning techniques and methodology for fmax prediction," *IEEE International Test Conference*, 2009.
- [64] J. Chen, J. Zeng, L.-C. Wang, and M. Mateja, "Correlating system test fmax with structural test fmax and process monitoring measurements," *IEEE ASP Design Automation Conference*, pp. 419–424, 2010.
- [65] J. Chen, J. Zeng, L.-C. Wang, M. Mateja, and J. Rearick, "Predicting multi-core system fmax by data-learning methodology," *IEEE International Symposium on VLSI Design, Automation and Test*, 2010.
- [66] T. Hastie and et al., *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2001.
- [67] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- [68] G. Drmanac, F. Liu, and L.-C. Wang, "Predicting variability in nanoscale lithography processes," *ACM/IEEE Design Automation Conference*, pp. 545–550, 2009.
- [69] D. Ding, J. A. Torres, and D. Z. Pan, "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE Trans. on CAD*, vol. 30, no. 11, pp. 1621–1634, 2011.
- [70] B. N. Lee, L.-C. Wang, and M. S. Abadir, "Reducing pattern delay variations for screening frequency dependent defects," in *23rd IEEE VLSI Test Symposium (VTS)*, pp. 153–160, IEEE Computer Society, 2005.
- [71] B. N. Lee, L.-C. Wang, and M. S. Abadir, "Issues on test optimization with known good dies and known defective dies - A statistical perspective," in *IEEE International Test Conference*, pp. 1–10, IEEE Computer Society, 2006.
- [72] S. H. Y. Wu, B. N. Lee, L. Wang, and M. S. Abadir, "Statistical analysis and optimization of parametric delay test," in *IEEE International Test Conference*, pp. 1–10, IEEE Computer Society, 2007.
- [73] L. Breiman, "Random forests," *Machine Learning Journal*, vol. 45, pp. 5–32, 2001.
- [74] S. H. Wu, D. G. Drmanac, and L. Wang, "A study of outlier analysis techniques for delay testing," in *IEEE International Test Conference*, pp. 1–10, IEEE Computer Society, 2008.
- [75] I. Jolliffe, *Principal Component Analysis*. Springer, 1986.

- [76] D. G. Drmanac, B. Bolin, L. Wang, and M. S. Abadir, “Minimizing outlier delay test cost in the presence of systematic variability,” in *IEEE International Test Conference*, pp. 1–10, IEEE Computer Society, 2009.
- [77] D. G. Drmanac, N. Sumikawa, L. Winemberg, L. Wang, and M. S. Abadir, “Multidimensional parametric test set optimization of wafer probe data for predicting in field failures and setting tighter test limits,” in *Design, Automation and Test in Europe, DATE*, pp. 794–799, IEEE, 2011.
- [78] H. Li, M. Mansour, S. Maturi, and L. Wang, “A new sampling method for analog behavioral modeling,” in *International Symposium on Circuits and Systems (ISCAS)*, pp. 2908–2911, IEEE, 2010.
- [79] H. Li, M. Mansour, S. Maturi, and L. Wang, “Analog behavioral modeling flow using statistical learning method,” in *11th International Symposium on Quality of Electronic Design (ISQED)*, pp. 872–878, IEEE, 2010.
- [80] D. G. Drmanac, B. Bolin, and L. Wang, “A non-parametric approach to behavioral device modeling,” in *11th International Symposium on Quality of Electronic Design (ISQED)*, pp. 284–290, IEEE, 2010.
- [81] L. Wang and M. Marek-Sadowska, “Machine learning in simulation-based analysis,” in *ACM Symposium on International Symposium on Physical Design, ISPD* (A. Davoodi and E. F. Y. Young, eds.), pp. 57–64, ACM, 2015.
- [82] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [83] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor, “Canonical correlation analysis; an overview with application to learning methods,” *Neural Computation*, pp. 2639–2664, 2004.
- [84] L.-C. Wang, “An autonomous system view to apply machine learning,” in *IEEE International Test Conference*, 2018.
- [85] M. Nero, J. Shan, L.-C. Wang, and N. Sumikawa, “Concept recognition in production yield data analytics,” *IEEE International Test Conference*, 2018.
- [86] L.-C. Wang, “Keynote address: Machine learning in test data analytics,” *IEEE International Test Conference in Asia (ITC-Asia)*, 2020.
- [87] W. van der Aalst, A. Weijters, and L. Maruster, “Workflow mining: discovering process models from event logs,” *IEEE Transactions in Knowledge Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

- [88] W. M. Van der Aalst, V. Rubin, H. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, “Process mining: a two-step approach to balance between underfitting and overfitting,” *Software & Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010.
- [89] F. C. C. (FCC), “Operation of radar services in the 76-81 ghz band,” *FCC*, 2015.
- [90] J. E. Hopcroft, *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, 1979.
- [91] C. De La Higuera, “A bibliographical study of grammatical inference,” *Pattern recognition*, vol. 38, no. 9, pp. 1332–1348, 2005.
- [92] E. M. Gold, “Language identification in the limit,” *Information and control*, vol. 10, no. 5, pp. 447–474, 1967.
- [93] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, pp. 87–106, 1987.
- [94] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, 1st ed., 2011.
- [95] Y. S. Mahajan, Z. Fu, and S. Malik, “Zchaff2004: An efficient sat solver,” in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 360–375, Springer, 2004.
- [96] M. J. Yang, Y. J. Zeng, and L.-C. Wang, “Language driven analytics for failure pattern feedforward and feedback,” in *IEEE International Test Conference*, 2022.
- [97] N. C. Shawe-Taylor, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- [98] e. a. A. Gretton, “Kernel methods for measuring independence,” *Journal of Machine Learning Research*, vol. 6, pp. 2075–2129, 2005.
- [99] M. Kuss and T. Graepel, “The geometry of kernel canonical correlation analysis,” *Max Planck Institute for Biological Cybernetics, Technical Report, May*, vol. 108, 2003.
- [100] A. E. Council, “Guidelines for part average testing,” *AEC-Q001 Rev-D, December 9, 2011*, 2011.
- [101] A. Daniely and S. Shalev-Shwartz, “Complexity theoretic limitations on learning dnf’s,” in *29th Annual Conference on Learning Theory*, vol. 49 of *Proceedings of Machine Learning Research*, pp. 815–830, PMLR, 23–26 Jun 2016.

- [102] R. Motwani and P. Raghavani, “Randomized Algorithms,” *Cambridge University Press*, 1995.
- [103] D. Haussler, “Quantifying inductive bias: Ai learning algorithms and valiant’s learning framework,” *Artif. Intell.*, vol. 36, p. 177–221, sep 1988.
- [104] K.-K. Hsieh and L.-C. Wang, “A concept learning tool based on calculating version space cardinality,” *Tech. Report, March 23 2018*, 2018.
- [105] J. Pearl, “On the connection between the complexity and credibility of inferred models,” *International Journal of General Systems*, vol. 4, pp. 255–264, 1978.
- [106] R. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.
- [107] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [108] ed. by Azim Eskandarian, “Section 10 fully autonomous driving,” *Handbook of Intelligent Vehicles*, 2012.
- [109] I. Goodfellow, Y. Benjio, and A. Courville, *Deep Learning*. The MIT Press, 2016.
- [110] L.-C. Wang, “Ai at itc — promotion video,” *IEEE International Test Conference Youtube Channel*, 2018.
- [111] Y. J. Zeng, M. J. Yang, and L.-C. Wang, “Wafer map pattern analytics driven by natural language queries,” in *IEEE International Test Conferencel in Asia, 2022*.
- [112] M. W. Krentel, “The complexity of optimization problems,” *Journal of Computer and System Science*, vol. 36, pp. 490–509, 1988.
- [113] M. W. Krentel, “Completeness in the polynomial-time hierarchy a compendium,” *Technical Report*, 2008.
- [114] S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- [115] L.-C. Wang and J. Zeng, “Machine learning support for wafer-level pattern analytics,” *Chapter 9 in Machine Learning Support for Fault Diagnosis of System-on-Chip, Springer Nature*, 2023.
- [116] S. Illyes and D. Baglee, “Statistical bin limits: an approach to wafer dispositioning in ic fabrication,” pp. 95–98, 1990.
- [117] M. J. Moreno-Lizaranzu and F. Cuesta, “Improving electronic sensor reliability by robust outlier screening.,” *Sensors (Basel, Switzerland)*, vol. 13, no. 10, pp. 13521–13542, 2013.

- [118] R. Miller and W. C. Riordan, “Unit level predicted yield: a method of identifying high defect density die at wafer sort,” *International Test Conference*, 2001.
- [119] W. Riordan, R. Miller, and E. St Pierre, “Reliability improvement and burn in optimization through the use of die level predictive modeling,” *Reliability Physics Symposium, 2005. Proceedings. 43rd Annual. 2005 IEEE International*, pp. 435–445, 2005.
- [120] K. P. White, B. Kundu, and C. M. Mastrangelo, “Classification of defect clusters on semiconductor wafers via the hough transformation,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 2, pp. 272–278, 2008.
- [121] Y.-S. Jeong, S.-J. Kim, and M. K. Jeong, “Automatic identification of defect patterns in semiconductor wafer maps using spatial correlogram and dynamic time warping,” *IEEE Transactions on Semiconductor manufacturing*, vol. 21, no. 4, pp. 625–637, 2008.
- [122] M. Rosenblatt *et al.*, “Remarks on some nonparametric estimates of a density function,” *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.
- [123] F. Pedregosa and et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [124] M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, “Wafer map failure pattern recognition and similarity ranking for large-scale data sets,” *IEEE Tran. on Semi. Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015.
- [125] M. Fan, Q. Wang, and B. van der Waal, “Wafer defect patterns recognition based on optics and multi-label classification,” *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016.
- [126] J. Yu and X. Lu, “Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis,” *IEEE Tran. on Semi. Manufacturing*, vol. 29, no. 1, pp. 33–43, 2016.
- [127] a. a. Minghao Piao, “Decision tree ensemble-based wafer map failure pattern recognition based on radon transform-based features,” *IEEE Tran. on Semi. Manufacturing*, vol. 31, no. 2, pp. 250–257, 2018.
- [128] N. Yu, Q. Xu, and H. Wang, “Wafer defect pattern recognition and analysis based on convolutional neural network,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 566–573, 2019.
- [129] J. Wang, Z. Yang, J. Zhang, Q. Zhang, and W.-T. K. Chien, “Adabalgan: An improved generative adversarial network with imbalanced learning for wafer defective pattern recognition,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 3, pp. 310–319, 2019.

- [130] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and J. Bengio, “Generative adversarial networks,” *arXiv:1406.2661*, 2014.
- [131] T.-H. Tsai and Y.-C. Lee, “A light-weight neural network for wafer map classification based on data augmentation,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 4, pp. 663–672, 2020.
- [132] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.
- [133] M. Saqlain, Q. Abbas, and J. Y. Lee, “A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 3, pp. 436–444, 2020.
- [134] M. B. Alawieh, D. Boning, and D. Z. Pan, “Wafer map defect patterns classification using deep selective learning,” *ACM/IEEE Design Automation Conference*, 2020.
- [135] H. Hu, C. He, and P. Li, “Semi-supervised wafer map pattern recognition using domain-specific data augmentation and contrastive learning,” in *2021 IEEE International Test Conference (ITC)*, pp. 113–122, 2021.
- [136] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.
- [137] C. Shan, A. Wahba, L.-C. Wang, and N. Sumikawa, “Deploying a machine learning solution as a surrogate,” in *IEEE International Test Conference*, pp. 1–10, IEEE, 2019.
- [138] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” *arXiv:1701.07875v3*, 2017.
- [139] D. Berthelot, T. Schumm, and L. Metz, “Began: Boundary equilibrium generative adversarial networks,” *arXiv:1703.10717v4*, 2017.
- [140] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv:1511.06434v2*, 2016.
- [141] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.
- [142] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training GANs,” *arXiv:1606.03498v1*, 2016.

- [143] M. Abadi and et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.
- [144] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.
- [145] L.-C. Wang, J. Shan, and A. Wahba, “Facilitating deployment of a wafer-based analytic software using tensor methods: Invited paper,” in *International Conference on Computer-Aided Design (ICCAD)*, IEEE/ACM, 2019.
- [146] A. Wahba, L.-C. Wang, Z. Zhang, and N. Sumikawa, “Wafer pattern recognition using tucker decomposition,” in *VLSI Test Symposium (VTS), 2019 IEEE 37th*, pp. 1–6, IEEE, 2019.
- [147] A. Wahba, C. Shan, L.-C. Wang, and N. Sumikawa, “Measuring the complexity of learning in concept recognition,” in *Int. Symposium on VLSI Design, Automation and Test*, pp. 1–4, IEEE, 2019.
- [148] A. Wahba, J. Shan, L.-C. Wang, and N. Sumikawa, “Wafer plot classification using neural networks and tensor methods,” in *ITC-Asia*, pp. 79–84, IEEE, 2019.
- [149] F.-F. Li, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.
- [150] M. Fink, “Object classification from a single example utilizing class relevance metrics,” *Advances in Neural Information Processing Systems*, pp. 449–456, 2005.
- [151] E. G. Miller, N. E. Matsakis, and P. A. Viola, “Learning from one example through shared densities on transforms,” *Conference on Computer Vision and Pattern Recognition*, pp. 464–471, 2000.
- [152] E. Schwartz, L. Karlinsky, J. Shtok, and e. a. Harary, “Deltaencoder: An effective sample synthesis method for few-shot object recognition,” *Advances in NIPS*, pp. 2850–2860, 2018.
- [153] B. Hariharan and R. Girshick, “Low-shot visual recognition by shrinking and hallucinating features,” *International Conference on Computer Vision*, 2017.
- [154] B. Liu, X. Wang, M. Dixit, R. Kwitt, and N. Vasconcelos, “Feature space transfer for data augmentation,” *Conference on Computer Vision and Pattern Recognition*, p. 9090–9098, 2018.

- [155] e. a. H. Gao, “Low-shot learning via covariance-preserving adversarial augmentation networks,” *Advances in Neural Information Processing Systems*, pp. 983–993, 2018.
- [156] Z. Chen, Y. Fuy, Y. Zhang, and e. a. Jiang, “Multi-level semantic feature augmentation for one-shot learning,” *IEEE Transactions on Image Processing*, vol. 28, no. 9, pp. 4594–4605, 2019.
- [157] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” *Advances in Neural Information Processing Systems*, vol. 2, pp. 3320–3328, 2014.
- [158] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, “Analysis of representations for domain adaptation,” *Advances in Neural Information Processing Systems*, no. 22, pp. 137–144, 2007.
- [159] Y. Ganin, E. Ustinova, H. Ajakan, and e. a. Germain, “Domain-adversarial training of neural networks,” *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1–35, 2016.
- [160] S. Hochreiter, A. S. Younger, and P. R. Conwell., “Learning to learn using gradient descent,” *International Conference on Artificial Neural Networks*, pp. 87–94, 2001.
- [161] Y. Guo, N. C. Codella, L. Karlinsky, and e. a. Codella, “A broader study of cross-domain few-shot learning,” *A. Vedaldi et al. (Eds.): ECCV 2020, LNCS, Springer Nature Switzerland AG 2020*, vol. 12372, pp. 124–141, 2020.
- [162] X. Dong and J. Shen, “Triplet loss in siamese network for object tracking,” in *European Conference on Computer Vision*, Sep 2018.
- [163] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.
- [164] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2013.
- [165] S. Zhao, J. Song, and S. Ermon, “Infovae: Information maximizing variational autoencoders,” 2017.
- [166] L. van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.
- [167] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3 draft ed., 2023.

- [168] Y. Wang and et al., “Building a semantic parser overnight,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, 2015.
- [169] J. Berant and P. Liang, “Semantic parsing via paraphrasing,” in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1415–1425, Association for Computational Linguistics, 2014.
- [170] R. Shin and et al., “Constrained language models yield few-shot semantic parsers,” *CoRR*, vol. abs/2104.08768, 2021.
- [171] T. Wolfson and et al., “Break it down: A question understanding benchmark,” *CoRR*, vol. abs/2001.11770, 2020.
- [172] T. Brown and et al., “Language models are few-shot learners,” *CoRR*, vol. abs/2005.14165, 2020.
- [173] T. B. B. et al., “Language models are few-shot learners,” *CoRR (also in NeurIPS Proceedings)*, vol. abs/2005.14165, 2020.
- [174] R. Bommasani and et al., “On the opportunities and risks of foundation models,” *arXiv:2108.07258*, 2021.
- [175] M. C. et al., “Evaluating large language models trained on code,” *CoRR*, vol. abs/2107.03374, 2021.
- [176] L. O. et al., “Training language models to follow instructions with human feedback,” *CoRR*, vol. abs/2203.02155, 2022.
- [177] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents,” *CoRR*, vol. abs/2201.07207, 2022.
- [178] M. A. et al., “Do as i can, not as i say: Grounding language in robotic affordances,” *CoRR*, vol. abs/2204.01691, 2022.
- [179] Y. J. Zeng, M. J. Yang, and L.-C. Wang, “Wafer map pattern analytics driven by natural language queries,” in *IEEE International Test Conference in Asia*, 2022.
- [180] A. H. et al., “Knowledge graphs,” *CoRR*, vol. abs/2003.02320, 2020.
- [181] M. Nero, *Domain-Specific Machine Learning - A No-Free-Lunch Perspective*. UCSB PhD Thesis, March 2022.
- [182] J. Shan, *Domain-Specific Machine Learning - A Human Learning Perspective*. UCSB PhD Thesis, March 2022.

- [183] D. W. et al., “Dynamic integration of background knowledge in neural nlu systems,” *CoRR*, vol. abs/1706.02596, 2017.
- [184] B. Y. Lin, X. Chen, J. Chen, and X. Ren, “KagNet: Knowledge-aware graph networks for commonsense reasoning,” *CoRR*, 2019.
- [185] Y. e. a. Feng, “Scalable multi-hop relational reasoning for knowledge-aware question answering,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1295–1309, nov 2020.
- [186] M. e. a. Yasunaga, “QA-GNN: Reasoning with language models and knowledge graphs for question answering,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 535–546, jun 2021.
- [187] X. Z. et al., “GreaseLM: Graph reasoning enhanced language models for question answering,” *CoRR*, vol. abs/2201.08860, 2022.
- [188] R. Speer, J. Chin, and C. Havasi, “ConceptNet 5.5: An open multilingual graph of general knowledge,” in *AAAI Conference on Artificial Intelligence*, pp. 4444–4451, 2017.
- [189] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” *CoRR (also in EMNLP)*, 2019.
- [190] M. e. a. Faruqui, “Retrofitting word vectors to semantic lexicons,” in *Proc NAACL: Human Language Technologies*, pp. 1606–1615, 2015.
- [191] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, 2018.
- [192] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, “Stanza: A Python natural language processing toolkit for many human languages,” in *Proc. 58th ACL: System Demonstrations*, 2020.
- [193] A. Singhal, “Introducing the knowledge graph: things, not strings,” *Google Blog*, 2012.
- [194] G. Klyne, J. J. Carroll, and B. McBride, “RDF 1.1 Concepts and Abstract Syntax,” *W3C*, 2014.
- [195] P. H. et al., “OWL 2 Web Ontology Language primer (second edition),” *W3C*, 2012.
- [196] Y. Zeng and L.-C. Wang, “Domain knowledge graph construction via a simple checker,” 2023.

- [197] A. Waterman and K. Asanovi'c, "The risc-v instruction set manual, volume i: User-level isa, document version 20191213," 2019.
- [198] D. Beckett and T. Berners-Lee, "Turtle - terse rdf triple language," 2011.
- [199] J. Yan, C. Wang, W. Cheng, M. Gao, and A. Zhou, "A retrospective of knowledge graphs," *Frontiers of Computer Science*, vol. 12, pp. 55–74, 2018.
- [200] H. Ye, N. Zhang, H. Chen, and H. Chen, "Generative knowledge graph construction: A review," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, (Abu Dhabi, United Arab Emirates), pp. 1–17, Association for Computational Linguistics, Dec. 2022.
- [201] Y. Luan, L. He, M. Ostendorf, and H. Hajishirzi, "Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 3219–3232, Association for Computational Linguistics, Oct.-Nov. 2018.
- [202] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pp. 142–147, 2003.
- [203] D. Milne and I. H. Witten, "Learning to link with wikipedia," in *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, (New York, NY, USA), p. 509–518, Association for Computing Machinery, 2008.
- [204] D. Zelenko, C. Aone, and A. Richardella, "Kernel methods for relation extraction," in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pp. 71–78, Association for Computational Linguistics, July 2002.
- [205] D. Zelenko, C. Aone, and J. Tibbetts, "Coreference resolution for information extraction," in *Proceedings of the Conference on Reference Resolution and Its Applications*, (Barcelona, Spain), pp. 24–31, Association for Computational Linguistics, July 2004.
- [206] S. Bird, "NLTK: The Natural Language Toolkit," in *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, (Sydney, Australia), pp. 69–72, Association for Computational Linguistics, July 2006.
- [207] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System*

- Demonstrations*, (Baltimore, Maryland), pp. 55–60, Association for Computational Linguistics, June 2014.
- [208] J. R. Finkel, T. Grenager, and C. Manning, “Incorporating non-local information into information extraction systems by Gibbs sampling,” in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, (Ann Arbor, Michigan), pp. 363–370, Association for Computational Linguistics, June 2005.
- [209] W. Liao and S. Veeramachaneni, “A simple semi-supervised algorithm for named entity recognition,” in *Proceedings of the NAACL HLT 2009 Workshop on Semi-supervised Learning for Natural Language Processing*, (Boulder, Colorado), pp. 58–65, Association for Computational Linguistics, June 2009.
- [210] I. Mondal, Y. Hou, and C. Jochim, “End-to-end construction of NLP knowledge graph,” in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, (Online), pp. 1885–1895, Association for Computational Linguistics, Aug. 2021.
- [211] A. Harnoune, M. Rhanoui, M. Mikram, S. Yousfi, Z. Elkaimbillah, and B. E. Asri, “BERT based clinical knowledge extraction for biomedical knowledge graph construction and analysis,” *Computer Methods and Programs in Biomedicine Update*, vol. 1, p. 100042, 2021.
- [212] D. Chen and C. Manning, “A fast and accurate dependency parser using neural networks,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 740–750, Association for Computational Linguistics, Oct. 2014.
- [213] M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu, “Fast and accurate shift-reduce constituent parsing,” in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Sofia, Bulgaria), pp. 434–443, Association for Computational Linguistics, Aug. 2013.
- [214] R. Shin, C. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. Van Durme, “Constrained language models yield few-shot semantic parsers,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, (Online and Punta Cana, Dominican Republic), pp. 7699–7715, Association for Computational Linguistics, Nov. 2021.
- [215] Y. Lu, H. Lin, J. Xu, X. Han, J. Tang, A. Li, L. Sun, M. Liao, and S. Chen, “Text2Event: Controllable sequence-to-structure generation for end-to-end event extraction,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Online), pp. 2795–2806, Association for Computational Linguistics, Aug. 2021.

- [216] T. Wolfson, M. Geva, A. Gupta, M. Gardner, Y. Goldberg, D. Deutch, and J. Berant, “Break it down: A question understanding benchmark,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 183–198, 2020.
- [217] S. Arora and B. Barak, *Chapter 8 Interactive Proofs in Computational Complexity: A Modern Approach*. Cambridge University Press, June 2012.
- [218] C. Boettiger, *rdflib: A high level wrapper around the redland package for common rdf applications*, 2018.
- [219] C. D. Paice, “Another stemmer,” *SIGIR Forum*, vol. 24, p. 56–61, nov 1990.
- [220] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.
- [221] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, “Palm: Scaling language modeling with pathways,” *J. Mach. Learn. Res.*, vol. 24, pp. 240:1–240:113, 2023.
- [222] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” *CoRR*, vol. abs/2302.13971, 2023.
- [223] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. T. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, “OPT: open pre-trained transformer language models,” *CoRR*, vol. abs/2205.01068, 2022.

- [224] S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- [225] O. Goldreich, *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.
- [226] O. Goldreich, *Introduction to Property Testing*. Cambridge University Press, 2017.
- [227] M. Blum and S. Kannan, “Designing programs that check their work,” *J. ACM*, vol. 42, no. 1, pp. 269–291, 1995.
- [228] Y. Yao, D. Ye, P. Li, X. Han, Y. Lin, Z. Liu, Z. Liu, L. Huang, J. Zhou, and M. Sun, “DocRED: A large-scale document-level relation extraction dataset,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 764–777, Association for Computational Linguistics, July 2019.
- [229] W. B. Dolan and C. Brockett, “Automatically constructing a corpus of sentential paraphrases,” in *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.
- [230] Y. Arase and J. Tsujii, “Compositional phrase alignment and beyond,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Online), pp. 1611–1623, Association for Computational Linguistics, Nov. 2020.
- [231] M. Wang and C. Manning, “Probabilistic tree-edit models with structured latent variables for textual entailment and question answering,” in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, (Beijing, China), pp. 1164–1172, Coling 2010 Organizing Committee, Aug. 2010.
- [232] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, “Stanza: A python natural language processing toolkit for many human languages,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2020, Online, July 5-10, 2020* (A. Celikyilmaz and T. Wen, eds.), pp. 101–108, Association for Computational Linguistics, 2020.
- [233] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

- [234] M. Krallinger, O. Rabal, F. Leitner, M. Vazquez, D. Salgado, Z. Lu, R. Leaman, Y. Lu, D. Ji, D. M. Lowe, R. A. Sayle, R. T. Batista-Navarro, R. Rak, T. Huber, T. Rocktäschel, S. Matos, D. Campos, B. Tang, H. Xu, T. Munkhdalai, K. H. Ryu, S. V. Ramanan, P. S. Nathan, S. Zitnik, M. Bajec, L. Weber, M. Irmer, S. A. Akhondi, J. A. Kors, S. Xu, X. An, U. K. Sikdar, A. Ekbal, M. Yoshioka, T. M. Dieb, M. Choi, K. Verspoor, M. Khabsa, C. L. Giles, H. Liu, R. K. Elayavilli, A. Lamurias, F. M. Couto, H. Dai, R. T. Tsai, C. Ata, T. Can, A. Usie, R. Alves, I. Segura-Bedmar, P. Martínez, J. Oyarzabal, and A. Valencia, “The CHEMDNER corpus of chemicals and drugs and its annotation principles,” *J. Cheminformatics*, vol. 7, no. S-1, p. S2, 2015.
- [235] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, “Finetuned language models are zero-shot learners,” in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.
- [236] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe, “Training language models to follow instructions with human feedback,” in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), 2022.
- [237] OpenAI, “GPT-4 technical report,” *CoRR*, vol. abs/2303.08774, 2023.
- [238] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open foundation and fine-tuned chat models,” *CoRR*, vol. abs/2307.09288, 2023.
- [239] B. Guo, X. Zhang, Z. Wang, M. Jiang, J. Nie, Y. Ding, J. Yue, and Y. Wu, “How close is chatgpt to human experts? comparison corpus, evaluation, and detection,” *CoRR*, vol. abs/2301.07597, 2023.

- [240] S. Herbold, A. Hautli-Janisz, U. Heuer, Z. Kikteva, and A. Trautsch, “Ai, write an essay for me: A large-scale comparison of human-written versus chatgpt-generated essays,” *CoRR*, vol. abs/2304.14276, 2023.
- [241] G. Kortemeyer, “Could an artificial-intelligence agent pass an introductory physics course?,” *Phys. Rev. Phys. Educ. Res.*, vol. 19, p. 010132, May 2023.
- [242] T. R. McIntosh, T. Liu, T. Susnjak, P. Watters, A. Ng, and M. N. Halgamuge, “A culturally sensitive test to evaluate nuanced gpt hallucination,” *IEEE Transactions on Artificial Intelligence*, vol. 1, pp. 1–13, nov 5555.
- [243] V. Rawte, S. Chakraborty, A. Pathak, A. Sarkar, S. M. T. I. Tonmoy, A. Chadha, A. P. Sheth, and A. Das, “The troubling emergence of hallucination in large language models - an extensive definition, quantification, and prescriptive remediations,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023* (H. Bouamor, J. Pino, and K. Bali, eds.), pp. 2541–2573, Association for Computational Linguistics, 2023.
- [244] D. Johnson, R. Goodman, J. Patrinely, C. Stone, E. Zimmerman, R. Donald, S. Chang, S. Berkowitz, A. Finn, E. Jahangir, E. Scoville, T. Reese, D. Friedman, J. Bastarache, Y. van der Heijden, J. Wright, N. Carter, M. Alexander, J. Choe, C. Chastain, J. Zic, S. Horst, I. Turker, R. Agarwal, E. Osmundson, K. Idrees, C. Kieman, C. Padmanabhan, C. Bailey, C. Schlegel, L. Chambless, M. Gibson, T. Osterman, and L. Wheless, “Assessing the accuracy and reliability of AI-generated medical responses: An evaluation of the chat-GPT model,” *Res. Sq.*, Feb. 2023.
- [245] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, “Let’s verify step by step,” *CoRR*, vol. abs/2305.20050, 2023.
- [246] M. Bhatt, S. Chennabasappa, C. Nikolaidis, S. Wan, I. Evtimov, D. Gabi, D. Song, F. Ahmad, C. Aschermann, L. Fontana, S. Frolov, R. P. Giri, D. Kapil, Y. Kozyrakis, D. LeBlanc, J. Milazzo, A. Straumann, G. Synnaeve, V. Vontimitta, S. Whitman, and J. Saxe, “Purple llama cyberseceval: A secure coding benchmark for language models,” *CoRR*, vol. abs/2312.04724, 2023.
- [247] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine, and M. Khabsa, “Llama guard: Llm-based input-output safeguard for human-ai conversations,” *CoRR*, vol. abs/2312.06674, 2023.
- [248] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, “Self-consistency improves chain of thought reasoning in language

- models,” in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, OpenReview.net, 2023.
- [249] X. Chen, R. Aksitov, U. Alon, J. Ren, K. Xiao, P. Yin, S. Prakash, C. Sutton, X. Wang, and D. Zhou, “Universal self-consistency for large language model generation,” *CoRR*, vol. abs/2311.17311, 2023.
- [250] P. Manakul, A. Liusie, and M. J. F. Gales, “Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023* (H. Bouamor, J. Pino, and K. Bali, eds.), pp. 9004–9017, Association for Computational Linguistics, 2023.
- [251] S. Lin, J. Hilton, and O. Evans, “TruthfulQA: Measuring how models mimic human falsehoods,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Dublin, Ireland), pp. 3214–3252, Association for Computational Linguistics, May 2022.
- [252] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach, “SDXL: improving latent diffusion models for high-resolution image synthesis,” *CoRR*, vol. abs/2307.01952, 2023.