

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Mitigating Post-Training Effects on Generative Diversity in Language Models

Permalink

<https://escholarship.org/uc/item/9pd2k2vc>

ISBN

9798189272363

Author

Wong, Justin

Publication Date

2026-05-01

Peer reviewed|Thesis/dissertation

Mitigating Post-Training Effects on Generative Diversity in Language Models

By

Justin Yat-Fung Wong

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sanjit A. Seshia, Co-Chair
Professor Joseph E. Gonzalez, Co-Chair
Professor Alexandros G. Dimakis
Professor William D. Thompson

Spring 2026

Mitigating Post-Training Effects on Generative Diversity in Language Models

Copyright 2026

By

Justin Yat-Fung Wong

Abstract

Mitigating Post-Training Effects on Generative Diversity in Language Models

by

Justin Yat-Fung Wong

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sanjit A. Seshia, Co-Chair

Professor Joseph E. Gonzalez, Co-Chair

Large language models (LLMs) have become remarkably proficient at generating coherent, high-quality responses. However, they often struggle to produce diverse outputs, especially when multiple, equally plausible answers exist. This limitation becomes critical in areas like adversarial testing, search, and synthetic data generation, where generating distinct yet valid responses is essential. This thesis opens with a discussion of the timely need for addressing the diversity of language models. I'll then introduce SimpleStrat, a lightweight method that leverages the model itself to automatically stratify the solution space. By performing stratified sampling, we can improve diversity and coverage without sacrificing quality. I'll also describe how we measure resampling diversity using CoverageQA, and discuss how these ideas extend to creative and open-ended tasks. To extend these ideas into the weight space, I'll discuss Stylus, which navigates the quality-diversity tradeoff by leveraging adapter merging to produce higher-quality images while accessing meaningful diversity by randomizing subsets of adapters for merging. The thesis will conclude with a discussion on going beyond diversity to creativity and current challenges in applying LLMs to discover novel and creative work. Anyone can critique a masterpiece, but immortal is the artist who creates one.

To the friends I've made along the way,
whose emotional support was peer reviewed and statistically significant.

Contents

Contents	i
List of Figures	iii
List of Tables	vii
 I The Case for Diversity	 xi
1 The Case for Resampling Diversity	1
1.1 Settings That Require Diversity	1
1.2 Diversity for Test-Time Scaling	3
1.3 Diversity for Coverage	4
1.4 Reinforcement Learning: Learning by Exploring	5
2 Why Is It Not Solved?	8
2.1 Measuring Diversity	8
2.2 Why Is Mode Collapse So Severe in LLMs?	10
2.3 Approaches to Diversity	14
 II What Can We Do About It?	 17
3 SimpleStrat	18
3.1 Overview	18
3.2 Related Works	21
3.3 The SimpleStrat Method	23
3.4 Benchmark for Diversity: CoverageQA	25
3.5 Evaluation Setup	27
3.6 Measuring Diversity	30
3.7 Qualitative Example	31
3.8 Coverage Diversity on Proprietary Models	35
3.9 Distributional Diversity with Llama 3	37

3.10	Open-Ended Diversity	39
3.11	SimpleStrat in Production – Task Proposing	42
3.12	Discussion	43
3.13	Conclusion	44
4	Stylus	46
4.1	Overview	47
4.2	Related Works	48
4.3	Our Method: Stylus	49
4.4	Evaluation Setup	53
4.5	Human Evaluation	58
4.6	Automatic Benchmarks	58
4.7	VLM as a Judge	58
4.8	Diversity per Prompt	59
4.9	Ablations	61
4.10	Failure Modes and Limitations	62
4.11	Efficiency and Extensions	62
4.12	Discussion	70
4.13	Conclusion	70
III	Conclusion	71
5	Diversity Is Necessary but Not Sufficient for Creativity	72
5.1	Summary of Learnings	72
5.2	Future Work	73
5.3	Open Questions	75
	Bibliography	78
A	Ashera: Neural Optimization Modulo Theory	96
A.1	Introduction	96
A.2	Related works	98
A.3	Preliminaries	100
A.4	Method Overview	102
A.5	OMT Engine	103
A.6	Neural Diving	105
A.7	Experiment Setup	107
A.8	Results	107
A.9	Conclusions	110
A.10	Supplementary Details	111

List of Figures

2.1	Calibration plots from the GPT-4 technical report (OpenAI et al., 2024). Left: the pre-trained GPT-4 model on a subset of MMLU is well calibrated. Right: calibration worsens after post-training.	11
3.1	SimpleStrat recovers diversity when temperature scaling fails. For the prompt “Name a U.S. state,” language models exhibit a strong bias toward “California.” Our method, SimpleStrat, employs automatic stratification to uncover salient diversity dimensions (e.g., whether a state lies east or west of the Mississippi River) and applies stratified sampling to achieve balanced and diverse coverage across responses.	18
3.2	SimpleStrat workflow. SimpleStrat employs 3 phases: 1) auto-stratification to identify good dimensions of diversity that divide the solution space into equal partitions, 2) heuristic estimation to estimate the proportion of solutions in each stratum, and 3) probabilistic prompting where a concrete prompt is randomly sampled from the prompt distribution specified by the previous two phases. Critically, diverse resampling comes from both the random choice of prompt as well as the temperature of the LLM decoding.	20
3.3	Diversity scaled with temperature. We show 100 resamples of “Name one Great Lake in the United States.” On the left, we show the result of resampling GPT-4o 100 times per temperature. In contrast to SimpleStrat on the right, GPT-4o at temperature 1.5 still only samples Lake Huron once and never samples Lake Ontario. SimpleStrat improves the diversity across all temperatures.	32
3.4	Distributional Diversity Comparison. We show the response probability as defined by next-token-probabilities for the top 20 ground truth answers on Llama 3.1. For both 8B and 70B, SimpleStrat provides meaningful improvement to the response distribution both for values previously over-represented in the distribution and for those previously underrepresented.	33
3.5	Baseline vs SimpleStrat Probability Distributions This figure shows the answer distributions for 4 additional questions from CoverageQA curated. Each row represents a different question, showing distributions for Llama 3.1 8B and 70B.	34

3.6	Diversity measured with recall scaled with temperature. SimpleStrat applied to GPT-4o shows improved recall on CoverageQA compared to GPT-4o, AttrPrompt, and Claude 3.5 Sonnet. Recall indicates the percentage of ground truth answers observed after sampling 100 times. The benefit of SimpleStrat is especially pronounced at low temperatures, but remains visible across all temperatures. Prior work, AttrPrompt, directly asks for partitions instead of using the 20-questions-style prompting used here, resulting in lower-quality partitions.	35
3.7	Precision scaled with temperature. There is a fixed minor reduction of 10% to precision when using SimpleStrat. This indicates the improved diversity does not come at a significant cost to precision.	36
3.8	We see even stronger improvements on the manually curated CoverageQA set. .	37
3.9	KL divergence on individual question from CoverageQA Wikipedia. For each question in CoverageQA, we compare the KL divergence from reference distribution of Llama-3 with and without SimpleStrat. Lower divergence indicates closer alignment with the desired uniform distribution, arrow indicates direction of maximum improvement from baseline	38
3.10	Distributional Diversity Comparison. We show the a posterior probability as defined by next-token probabilities for ground truth answers on Llama 3.1 70B. SimpleStrat provides meaningful improvement to the sampling distribution both for values previously overrepresented in the distribution and those previously underrepresented.	38
3.11	KL divergence from uniform for Baseline vs SimpleStrat on CoverageQA Wikipedia. Lower divergence indicates closer alignment with the desired uniform distribution, arrow indicates direction of maximum improvement from baseline	39
3.12	KL divergence from uniform for Baseline vs SimpleStrat on CoverageQA Wikipedia. Additional plots for Llama 3 8B and 70B models	39
3.13	WritingPrompts Diversity. On creative writing prompts, we generate pairs of plot outlines. We measure diversity with embedding cosine distance and quality by checking for format adherence. SimpleStrat especially improves the diversity at low temperatures achieving at T=0 the same diversity as T=1 for base GPT-4o.	40
3.14	WritingPrompts Diversity. On creative writing prompts, we generate pairs of plot outlines. We measure diversity with embedding cosine distance and quality proxied by spelling. SimpleStrat especially improves the diversity at low temperatures achieving at T=0 the same diversity as T=1 for base GPT-4o.	41
3.15	Impact of temperature on formatting (including spelling). It's interesting to see that the impact of SimpleStrat is negligible compared to temperature. We see that SimpleStrat closely follows the expected quality degradation based on the temperature of the model generating the final outline.	41

3.16	SimpleStrat task proposing. We use SimpleStrat to generate plausible and grounded tasks for training browser web-agents. Starting with a randomly selected landing page, we ask SimpleStrat to propose a challenging goal. Then we deploy the agent under training to accomplish that task. Upon completing the task, we ask SimpleStrat to propose a new goal. A task is created by summarizing the full trajectory.	42
3.17	SimpleStrat proposes harder tasks. We show on the y-axis the percent of newly generated tasks that are solved with the NovaAct browser agent. Before we deployed the SimpleStrat task proposer, we saw only 20% of tasks are hard for our agent. We progressively roll out the SimpleStrat task proposer from July 6th to September 12th. By the end, we get as much as 60% hard tasks.	43
3.18	Training on the tasks created from SimpleStrat task proposer leads to better SFT performance (shown in yellow). There is a 5% improvement on internal benchmarks compared to previous data-collection approach (shown in brown).	44
4.1	Adapter Selection. Given a user-provided prompt, our method identifies highly relevant adapters (e.g. Low-Rank Adaptation, LoRA) that are closely aligned with the prompt’s context and at least one of the prompt’s keywords . Composing relevant adapters into Stable Diffusion improves visual fidelity, image diversity, and textual alignment. Note that these prompts are sampled from MS-COCO (T.-Y. Lin et al., 2015).	46
4.2	Stylus algorithm. Stylus consists of three stages. The <i>refiner</i> plugs an adapter’s model card through a VLM to generate textual descriptions of an adapter’s task and then through an encoder to produce the corresponding text embedding. The <i>retriever</i> fetches candidate adapters that are relevant to the entire user prompt. Finally, the <i>composer</i> prunes and jointly categorizes the remaining adapters based on the prompt’s tasks, which corresponds to a set of keywords.	47
4.3	Number of Adapters. Civit AI boasts 100K+ adapters for Stable Diffusion, outpacing that of Hugging Face. Low-Rank Adaptation (LoRA) is the dominant approach for finetuning.	49
4.4	Qualitative comparison between Stylus over realistic (left) and cartoon (right) style Stable Diffusion checkpoints. Stylus produces highly detailed images that correctly depicts keywords in the context of the prompt. For the prompt “A graffiti of a corgi on the wall”, our method correctly depicts a spray-painted corgi, whereas the checkpoint generates a realistic dog.	51
4.5	Characterization of Civit Adapter in StylusDocs. (a) Most adapters are categorized as characters or celebrities. (b) Adapter popularity exhibits a power-law distribution, with the top adapters receiving exponentially more downloads than the others.	56
4.6	Human Evaluation. Stylus achieves higher preference scores (2:1) over different datasets and Stable Diffusion checkpoints.	56

4.7	Automatic Evaluation Metrics. Figure (a) plots the CLIP/FID pareto curve. We observe Stylus shifts the curve down (improved visual fidelity, FID) and to the right (improved textual alignment, CLIP score) over a range of guidance values (CFG): [1, 1.5, 2, 3, 4, 6, 9, 12]. Table (b) evaluates Stylus against different retrieval methods. Stylus outperforms existing retrieval-based methods, attains the best FID score, and achieves similar CLIP score to Stable Diffusion.	57
4.8	Image Diversity. Given the same prompt, our method (left) generates more diverse and comprehensive sets of images than that of existing Stable Diffusion checkpoints (right). Stylus’s diversity comes from its masking scheme and the composer LLM’s temperature parameter.	57
4.9	Figure (a) and (b) evaluate the preference win rate using GPT-4V as a judge. Stylus achieves higher preference scores as judged by GPT-4V for visual quality and image diversity. Figure (c) shows that Stylus achieves higher diversity scores than Stable Diffusion when prompt length increases.	59
4.10	Different Retrieval Methods. Stylus outperforms all other retrieval methods, which choose adapters that either introduce foreign concepts to the image or override other concepts in the prompt, reducing textual alignment.	60
4.11	Comparison of Stylus’s inference overheads with Stable Diffusion’s inference time by batch size (BS). At BS=1, Stylus accounts for 75% of the image generation time, primarily due to the composer processing long context prompts from adapter descriptions. However, Stylus’s overhead decreases when batch size increases.	61
4.12	Stylus over different image-to-image tasks.	63
4.13	$\mathcal{d}$FID for top 100 keywords in PartiPrompts dataset. Stylus leads to consistently higher diversity when compared to Stable Diffusion checkpoints, especially for words describing concepts and attributes.	65
4.14	Categorization of Different Failure Modes.	68
A.1	Ashera workflow. The neural diver serves as a warm-starter providing an initial low cost feasible solution. After neural diving, the OMT engine in Ashera alternates between an <i>Optimization-aware Theory Solver</i> (TS*) for optimization and an SMT solver for verification. Each invocation of TS* returns a tighter blocking clause, restricting the SMT solver to search for strictly lower cost solutions than the current model.	102
A.2	Ashera: OMT Solver	104
A.3	Graph representation. We represent the OMT problem as a graph with edges connecting variable nodes and constraint/cost nodes. The logic structure is represented similarly to an abstract syntax tree (AST), with the AST leaf nodes coinciding with the atomic constraint nodes.	105

List of Tables

2.1	Exact prompt templates used for the rock-paper-scissors and Pokémon-type experiments.	15
2.2	A simple SFT intervention can partially rebalance an obviously skewed output distribution, though the effect is unstable across training.	15
2.3	A small balanced SFT run on GPT-4o-mini on rock-paper-scissors substantially changes the type distribution for Pokémon it produces.	16
3.1	CoverageQA domains.	27
3.2	Full prompt for auto-stratification.	28
3.3	Prompt for partition-specific heuristic estimation.	29
3.4	Prompt for final heuristic estimation.	30
3.5	Performance of Different Prompting Strategies across Temperature Settings (GPT-4o)	36
3.6	We assess distributional diversity as measured by KL Divergence. Smaller KL divergence is closer to uniform. We see improvement to distributional diversity for both 8B and 70B as well as Llama-3 and Llama-3.1.	38
3.7	Prompt for plot outlines.	41
4.1	Full prompt for the refiner VLM to generate better adapter descriptions.	54
4.2	Full prompt for the composer LLM.	55
4.3	Refiner’s impact on End2End performance. Without a refiner, Stylus performs worse than SD v1.5 due to the poor quality of author-provided descriptions. Annotating adapters with GPT-4o significantly improves adapter descriptions and achieves higher CLIP/FID scores than Stylus’s default refiner VLM, Gemini-Ultra.	60
4.4	Full prompt judging compositional quality (left) or textual alignment (right) using VLM.	66
4.5	Full prompt judging diversity using VLM.	67
A.1	OMT Solver Performance on Scheduling. We report PAR-2 scores of process time to account for timeouts and provide number of solved problems in parenthesis.	108
A.2	OMT Solver Performance on Multi-Agent TSP. We report PAR-2 scores of process time to account for timeouts and provide number of solved problems in parenthesis.	109

A.3 Ablation on Neural Diving. We both test performance transfer to larger problems trained on smaller problems (Curriculum) and applying neural diving on existing solvers (Neural Diver + Gurobi/OptiMathSAT). 110

A.4 Scheduling Benchmark Summary. 116

A.5 Multi-Agent TSP Benchmark Summary. 117

Acknowledgments

First, I want to express my deepest gratitude to my advisors Sanjit A. Seshia and Joseph E. Gonzalez. Sanjit has been an incredible influence on my research taste. His deep expertise in formal reasoning provided me stability in a field ever changing. I have always felt that if Sanjit is convinced a topic is worth pursuing it certainly will be. More than an expert, Sanjit can impressively follow along on just about any topic. I am grateful for his unwavering support through all my pivots. In Joey, I find a kindred spirit. Joey is an incredible storyteller and a never-ending source of new ideas. I recall watching Joey lead sponsor meetings and being blown away by how excited I felt.

Both advisors lead groups that span a wide range of research domains, exposing me to the deepest research insights coming from hardware side-channel attacks to designing medical rehab tasks with Sanjit’s group and improving exploration for classic RL algorithms to efficiently moving data between cloud regions and providers in Joey’s Group. I am grateful to have been exposed to so many brilliant ideas and to have learned how to communicate my own ideas in a way that resonates with a wide audience.

I recall asking Paras Jain for his advice on the merits of co-advising. His advice was having one advisor affords you the freedom to try crazier ideas less justification. Two advisors require you to convince two people a problem is worth pursuing, by which time someone else may have already solved the problem. I believe it’s paid off well to learn the art of convincing people my pursuits are worthwhile and perhaps even got better at it than was productive.

I also would like to thank my dissertation committee members Alexandros G. Dimakis and William D. Thompson for their advice and insights during my PhD. The work in this thesis and during the PhD wouldn’t have been possible without my collaborators. I had my research start working on systems for self-driving with Peter Schafhalter, Eyal Sela, Ionel Gog, Krishna Agrawal, Simon Mo, Sukrit Kalra, Xin Wang, Ion Stoica, Kesav Viswanadha, Francis Indaheng, Eddie Kim, Ellen Kalvan, Yash Pant, Daniel Fremont; working on Ashera and solvers with Yuandong Tian, Pei-wei Chen, Tianjun Zhang, Jeffrey Ichnowski, Chirag Sharma, Mohamed Elgharbawy; on reinforcement learning with Fangchen Liu, Pieter Abbeel, Medhini Narasimhan, Benjamin Eysenbach; creative writing with Kevin Yang, Andrew Cohen, Xiaomeng Yang, Morteza Behrooz; VideoArena with Michael Luo, Brandon Trabucco, Yanping Huang, Zhifeng Chen, Ruslan Salakhutdinov, Brandon Huang, Alex Cao, Michelle Chen, Anastasios Angelopoulos, Wei-lin Chiang, Chris Donahue, Dacheng Li, Shiyi Cao, Yunhao Fang, Yukang Chen, Shuo Yang, Xiaolong Wang, Hongxu Yin, Song Han, Yao Lu; LLMs for Low resource languages with Federico Mora, Haley Lepe, Sahil Bhatia, Karim Elmaaroufi, George Varghese, Elizabeth Polgreen; Agentic Systems with Xiaoxiang Shi, Colin Cai, Yichuan Wang, Chi Wang; LLM forecasting with Lisa Dunlap, Jacob Steinhardt, Danny Halawi, Matthew Shen; LLMs for Lean with Lakshya Agarwal, Samion Suwito, Richard Zhuang, Shu Yuan, Darren Shen; agentic systems with Sijun Tan, William Tang, Manan Roongta, Erran Li, Raluca Popa; and diversifying LLM responses with Yury Orlovskiy, Alexander Shypula, Josh Kang, Suhong Moon, Anirudh Chakravarthy, Keagan

Long, and Xiangjun Wang. Although not all the ideas became papers, our learnings have influenced me deeply as a researcher.

I want to especially thank Kevin Yang and Yuandong Tian of RLCD for getting me interested in the storytelling domain and diverse generation. Susanne Johnston, Joseph Scherrer, Harry Anselmo, Dick Close, Camille Ragin, and Roland Dunbrack for inspiring me to do research and fall in love with science in primary and upper school. Finally, a very special thank you to Suman Jana, Ronghui Gu, and Chaolin Zhang for teaching me how to publish research as an undergrad.

Most importantly, I want to thank my apartmentmates, Bhaskar Roberts, Franklin Wang, and Julien Piet, for supporting me and allowing me to support them through the toughest years of our PhDs. Alvin Wan and Danijar Hafner for teaching me how to get the most out of conferences. Niklas Lauffer, Roshni Rawal, Altan Haan, and An Wang for letting me 5th wheel. To Laura and Conor Power, for being the ‘first mom’ of the group. To Charles Packer, Shishir Patil, and Vivian Fang for co-running 3DV, SkyLab’s best reading group. All the doctors who rehabilitated me after my concussion. To Shyam Chaudhry, Patrick Ledwith, Michael Tai, Kevin Gong, and Yi Wang, my best of friends, who have given me a bed to stay, a shoulder to cry on, and an occasional disapproving look, each when I needed it the most. To my partner, Valerie Huang, for her unwavering support, bringing so much joy to my life, and always knowing how to put a smile on my face. To my family for fulfilling their own dreams so that I can pursue mine, and especially my brother, Nathan, for letting me vicariously fulfill my dream of being a chemist.

Part I

The Case for Diversity

Chapter 1

The Case for Resampling Diversity

1.1 Settings That Require Diversity

Diversity is a luxury. In a sense, it speaks to how far language models have come that we can begin by asking the question of diversity at all. When I started my studies, the most (in)famous example was Tay, Microsoft’s Twitter chatbot experiment (Kraft, 2016). Tay was launched as a chatbot meant to develop “conversational understanding” by interacting with people online and learning from users’ posts (Kraft, 2016). Within a day, it had been manipulated into producing racist and genocidal outputs (Kraft, 2016).

“literally anybody i know could’ve told you this would happen. ONLY tech company people could be so clueless about humanity”

— Leigh Alexander, quoted in Kraft (2016)

This reaction captures a widely held intuition: if you train a model to be fluent based on the messy reality of what people post on the internet, it will reply as uncontrollably and unhinged as the cacophony of voices on the internet. The challenge of building a chatbot is not encouraging interesting and surprising replies, but preventing the model from parroting the worst of the nonsense its creators have exposed it to. When ChatGPT, and before it InstructGPT, reached the public, what was remarkable was not that the model hidden deep within it knew that *fromage* is the French translation for cheese. Of course that fact is written somewhere on the internet. What was remarkable was that the model was aligned enough to ship without a Tay-style disaster (Ouyang et al., 2022a).

Notably, the main technical breakthrough behind InstructGPT was reinforcement learning, used to encourage desirable behavior and discourage undesirable behavior (Ouyang et al., 2022a). And even so, benchmarks such as TruthfulQA (S. Lin et al., 2021), closed-book QA evaluation on datasets such as TriviaQA (Joshi et al., 2017), and hallucination benchmarks for image captioning such as ALOHa (Petryk et al., 2024) showed how prone models remained to plausible but factually incorrect responses. How could we ever find productive use for such an unreliable stochastic parrot (Bender et al., 2021)?

To the time-travelers from the early 2020s, no, you should not care about diversity. Fluency is picking from 1 million words (for English) the one or two that are reasonable, and factuality is not picking the one of the two that is false. In that regime, sacrifice as much variance and diversity as necessary in the name of ruling out the minefield of definitely wrong answers. Even today, document retrieval, medical diagnosis, and many forms of work automation are not necessarily settings where surfacing different solutions is preferable to consistently giving the same correct one. If the model you are working on is still almost always wrong, you can safely put this thesis down.

For those of us fortunate enough to build on the shoulders of giants who made models more reliable one data label at a time, the question becomes more interesting. We begin to ask not only whether a model can answer a question, but whether it can do something more useful than that. Three core capabilities are unlocked with diversity.

1. **Verification is easier than solving.** In many settings, especially math and coding, it is much easier to spot an error and check a proposed solution than it is to derive a solution in the first place. Test-time scaling, and more generally guess-and-check strategies dating back to Euler’s Characteristic Equation Method for differentiable equations (Euler, 1743), Cook’s formalization of NP-completeness (Cook, 1971), Counterexample-guided inductive synthesis (CEGIS) from program synthesis (Solar-Lezama et al., 2006), and generative adversarial networks (Goodfellow et al., 2014), exploit this characteristic. As long as the generator has some nontrivial probability of proposing a correct solution, a verifier that can certify correctness solves many practical problems. Without diversity, however, an LLM guesser simply tries the same answer over and over again, making test-time scaling moot.
2. **Models can point out flaws.** Language models are increasingly used to stress test other systems. One of the most constrained but productive examples is Rainbow Teaming, where models are judged by how well they cover the space of attacks needed to surface vulnerabilities in a target model (Samvelyan et al., 2024). More generally, models are used to generate synthetic training data (H. Dai et al., 2023; Dubey et al., 2024). But recursive training on low-diversity synthetic data is precisely the setting in which model collapse emerges: the tails of the original distribution disappear, and future generations become narrower and less capable (Shumailov et al., 2023).
3. **Training on diverse data and tasks leads to generalization.** Current training practice can be viewed as a corollary of the previous two points. In reinforcement learning with verifiable rewards, diverse model behavior can be paired with verifiers not only to rejection-sample solutions, but also to update the model so that it becomes better at producing correct solutions. This relies crucially on the model producing diverse outputs in the first place. More broadly, recent work has shown that diversity in training data and tasks encourages models to learn more generalizable solutions rather than merely memorizing narrow ones (Raventós et al., 2024). To close the loop, newer systems have begun to let models propose tasks for themselves, allowing the

model to participate in constructing the training curriculum it learns from (Trabucco et al., 2025; Pahuja et al., 2025; Novikov et al., 2025).

We now dive deeper into each of these settings to appreciate just how much the current lack of diversity is holding us back.

1.2 Diversity for Test-Time Scaling

In 2022, the simple suffix “Let’s think step by step” was shown to dramatically improve model problem solving (Wei et al., 2022). The “chain-of-thought” phenomenon is that a model coaxed to output a rationale before committing to an answer responds more accurately. This was the first instance of test-time scaling which has dominated the 2024 zeitgeist. Put simply test-time scaling is a general approach to solving harder problems with next-token models by getting the model to produce more tokens at inference time accessing more compute. As complexity theory would predict, harder problems naturally require more compute.¹ Theoretical work makes this intuition more precise. For the transformer architecture, Merrill and Sabharwal characterize the class of problems it can solve on inputs of length n with $s(n)$ sequential reasoning steps, showing that this class is upper-bounded by roughly $\tilde{O}(n^2 + s(n)^2)$ time and $O(s(n) + \log n)$ space (Merrill et al., 2023). Consequently, with $s(n) = O(n)$ or a linear number of sequential steps, the model has as much expressivity as regular expressions. With $s(n) = O(n^2)$, it can solve reachability problems. The challenge of test-time scaling is how to teach or encourage a model to use more tokens when faced with harder problems.

Test-time scaling led to significant breakthroughs with several popular ways to scale inference: sample multiple chains and vote among them via self-consistency (Xuezhi Wang et al., 2022), search over explicit trees of candidate thoughts via tree-of-thoughts (Yao et al., 2024), interleave reasoning with actions and environment feedback via ReAct (Yao et al., 2022), or use more agentic search procedures such as Language Agent Tree Search (LATs) (Zhou et al., 2023). On the mathematics problems, the returns can be dramatic. Minerva was among the first to show scaling language models and allowing them to produce longer mathematical derivations already led to strong quantitative math performance (Lewkowycz et al., 2022). More recently, AlphaProof demonstrated that problem-specific Monte Carlo tree search at test time can push all the way to olympiad-level formal reasoning (Hubert et al., 2026).

Within “test-time scaling,” note that there are two axes of test-time scaling. We call the two types: horizontal (resampling) and vertical (reacting). Horizontal diversity means getting different candidate solutions by resampling: different completions, different plans, different proofs, different variations of the prompt. By vertical diversity, we mean allowing a single trajectory to explore with the knowledge of prior failures. This creates opportunities to revise the plan in light of new evidence, including approaches that have already failed.

¹Even if an LLM someday proves $P \neq NP$, the models themselves certainly can’t solve hard classes of problems without exponentially more compute.

Horizontal and vertical diversity are both important for realizing the benefits of diversity in multi-turn LLM systems. Vertical diversity matters because letting a single trajectory to go deeper, deliberate longer, or interact with tools can uncover solutions when the initial attempt fails. This is emblematic of approaches like Reflexion, Self-Refine, and ReAct, where models are encouraged to reflect on intermediate outcomes before committing to an answer (Shinn et al., 2023; Madaan et al., 2023; Yao et al., 2022). Vertical diversity is difficult to decouple from improved self-reflection, better self-correction, and failure recovery. Horizontal diversity matters because many prompts admit multiple valid outputs, and repeated sampling should cover that space rather than collapsing onto a single favored mode. In this dissertation, we focus primarily on horizontal diversity because models show a dramatic loss in horizontal diversity the more they are trained.²

1.3 Diversity for Coverage

Multi-answer settings are exactly where diversity matters most: if a question has only one acceptable response, repeated sampling adds little beyond additional attempts at the same target. By contrast, when many distinct answers are equally valid, the goal is not merely to find one good solution, but to cover the solution space well. In optimization, symmetric or multi-solution problems are classically difficult because, in the general case, the search procedure cannot reason about alternative near-optimal solutions after observing only one. The effort required to be confident you are not in a local minimum was what made researchers write off gradient descent. We discuss these challenges in more detail in Appendix A.

Coverage is one of the clearest settings where current model limitations appear. Included are applications that probe the boundary of what a system or a program can and cannot do. Program testing is one such example. If we ask a model to generate unit tests for a piece of code, one plausible-looking test is rarely enough. The whole point is to cover different behaviors: ordinary cases, edge cases, or adversarial inputs that may not have the correct intended behavior. Systems such as EvalPlus (Jiawei Liu et al., 2023), R2E (Jain et al., 2024b), and LiveCodeBench (Jain et al., 2024a) use LLMs to generate test cases to verify the correctness of generated programs. This perspective appears directly also in work such as CodaMOSA, which uses pretrained code models to help test generation escape coverage plateaus. More importantly, it works: CodaMOSA improves coverage over a pure search-based baseline on many benchmarks, showing that LLM-proposed tests can productively expand the explored behavior space (Lemieux et al., 2023). EvalPlus makes the same point from the evaluation side: standard coding benchmarks often miss substantial functional errors because their tests do not sufficiently cover corner cases, and augmenting HumanEval with roughly 80 times more generated and mutation-based tests catches many previously undetected wrong programs, reduces measured pass rates by as much as 19.3–28.9%, and even changes model rankings (Jiawei Liu et al., 2023). The popular LiveCodeBench uses this

²As Eric Wallace once advised, focus on problems that become severe with scale rather than become obsolete with scale.

approach as well to generate test coverage. In these settings, the code coverage and testing validity are a direct function of the diversity of the LLM.

More broadly, popular evolutionary algorithms such as AlphaEvolve (O. Team, 2025), AlphaProof (Hubert et al., 2026), and GEPA (Agrawal et al., 2025) use language models as explicit search procedures over candidate programs, prompts, or policies, where progress depends on exploring a broad space of possibilities rather than repeatedly mutating the same narrow solution family. Unfortunately, current evolution methods are limited as LLMs tend to generate the same mutations over and over.

The same logic extends naturally from programs to LLM systems themselves. Once a language model is embedded inside a larger application with tools, retrieval, memory, or untrusted external text, it too becomes something vulnerable to attacks. Here the role of diversity is to generate a broad attack surface. What matters is whether it can cover many semantically distinct failure modes: direct jailbreaks, indirect prompt injections hidden in retrieved documents, role confusion, instruction collisions, and other attacks that exploit the fact that LLMs read everything as text. Rainbow Teaming is an explicit example of this coverage mindset, treating the generator as a search procedure over the space of attacks rather than as a source of one-off anecdotes (Samvelyan et al., 2024).

Prompt injection makes the point especially sharply. In a prompt injection attack, the vulnerability is often not that the model fails on one canonical malicious string, but that it fails to robustly separate trusted instructions from untrusted text across many variants. The Instruction Hierarchy paper formalizes precisely this issue: models need to learn which instructions should take precedence when different sources conflict (Wallace et al., 2024). From a testing perspective, this is again a coverage problem. We want models, or automated evaluators built from models, to generate many different ways of hiding, rephrasing, or embedding malicious instructions so that we can discover where the target system breaks. In other words, once LLMs become both the system under test and the tool for generating tests, diversity stops being a luxury and becomes the mechanism by which we find the failures that matter.

1.4 Reinforcement Learning: Learning by Exploring

Reinforcement learning turns diversity into data. Unlike supervised learning, it does not begin with a fixed dataset of correct behaviors to imitate. Instead, the learner must generate candidate actions, trajectories, or solutions for itself, and only then does reward reveal which of them were useful. In this sense, diversity determines what the learner gets to learn: without sufficient variation in behavior, there is no signal to distinguish better from worse strategies.

A growing body of work in large language models provides concrete empirical support for this view. Y. Yu et al. (2023b) show that when LLMs are used to generate training data, the diversity of attributes in the prompts directly controls both performance and bias: richer attribute coverage leads to more robust models, while low-diversity prompts produce

datasets with systematic skews (e.g., geographic or demographic bias). Importantly, this effect arises even when the total volume of data is held constant, suggesting that diversity, not just scale, is a primary driver of learning.

Complementing this, H. Chen et al. (2024) introduce explicit diversity metrics for synthetic datasets and demonstrate that these metrics correlate with downstream performance across tasks. Their results show that models trained on more diverse data not only perform better on average, but are also more stable under distribution shift. In this way, diversity acts as a form of implicit regularization, preventing overfitting to narrow patterns.

Finally, Havrilla et al. (2024) highlight a critical failure mode of data curation: selecting only high-quality examples can degrade performance if it reduces diversity. They show that datasets optimized for correctness or cleanliness alone can become overly homogeneous, limiting the model’s ability to generalize to novel inputs. Together, these results establish that diversity is not merely complementary to quality; it is a necessary component of it.

This perspective directly mirrors the exploration–exploitation tradeoff in reinforcement learning. A learner that always repeats the action that currently appears best may achieve reasonable short-term reward, but it sacrifices the opportunity to discover better alternatives. Exploration is therefore a demand for behavioral diversity: the agent must allocate probability mass to actions whose value is uncertain, because without sampling them it cannot improve its estimate. Diversity is what keeps the hypothesis space open long enough for learning to progress.

Paradoxically, RL training dynamics tend to collapse the very diversity they depend on. Once a particular strategy begins to outperform others, optimization shifts probability mass toward it, reducing exploration. In the context of RL for LLMs, especially in on-policy settings where models learn from their own generations, this creates a feedback loop: the model trains on an increasingly narrow distribution of its own outputs, progressively losing the ability to encounter novel situations. The result is a form of mode collapse, where both behavior and learning opportunities shrink over time.

One way to counteract this collapse is to inject diversity externally, for example by exposing models to a wide range of tasks and environments. Recent work suggests that the most useful tasks are often those in a “Goldilocks zone” of difficulty: tasks that are too easy provide little learning signal, while tasks that are too hard rarely yield successful trajectories from which the model can improve (Mahrooghi et al., 2026). Recent work goes further by using LLMs themselves to generate this diversity. In InSTA, the model constructs tasks across a broad set of web environments, effectively shaping its own training distribution (Trabucco et al., 2025). In mathematical reasoning, related approaches reward models for generating problems in a Goldilocks zone of difficulty, ensuring that self-generated data remains informative (A. Zhao et al., 2025).

However, once models participate in generating their own training data, the diversity problem becomes recursive. A narrow task generator induces a narrow curriculum, which in turn produces a model with narrower capabilities, further constraining future data generation. Conversely, maintaining diversity at the level of task generation encourages models to move beyond memorizing specific patterns and toward learning generalizable structure

(Raventós et al., 2024).

By closing the loop between exploration, data generation, and learning, these systems amplify both the benefits and risks of diversity. If we want models to discover novel strategies rather than merely reinforce existing ones, we must not only optimize for reward, but actively preserve and expand the diversity of the experiences from which they learn.

Chapter 2

Why Is It Not Solved?

2.1 Measuring Diversity

If I had two boxes that I could repeatedly send a set of prompts to and get responses, how would I quantify which box is more diverse? Certainly, if one box always replies with the same response for a prompt, it's less diverse. More generally, I would want a measure that captures not just whether responses are different at the surface level, but whether they vary in meaningful ways, such as phrasing, structure, or underlying ideas. This suggests looking at both the frequency of distinct outputs and the distribution over responses, for example using entropy, as well as measuring semantic differences between responses using embeddings or similarity metrics. Ideally, such a measure would distinguish between trivial variation (e.g., rewording the same answer) and substantive diversity (e.g., offering genuinely different perspectives or approaches), allowing for a principled comparison between the two boxes.

It is tempting to define diversity as simply how stochastic or random a model is. If we set out to measure how random or uncertain the model is, *generation entropy* provides a reasonable estimate of model randomness. This was discussed as a proxy for trainability in DAPO where it was observed that, under naive PPO or GRPO training, generation entropy decreases rapidly as training progresses (Q. Yu et al., 2025). To be precise, for a collection of tokens x generated part of the sequence or not,

$$H_t = - \sum_x p(x \mid \text{context}) \log p(x \mid \text{context})$$

This is reasonable for estimating how confident the model is on the set of tokens generated, and as Shannon originally intended, it's maximal when the distribution is uniform (Shannon, 1948). In the context of diversity, this tells us how random the token generator is. Using entropy as a proxy for diversity conflates epistemic uncertainty – uncertainty due to the model's lack of knowledge or imperfect training – with aleatoric uncertainty – inherent ambiguity or diversity where multiple outcomes are genuinely plausible. Unfortunately, decoupling these sources of entropy is known to be challenging (Wimmer et al., 2023). Put

simply, it's no different if I'm not sure how to solve a math proof versus knowing multiple ways to prove a theorem.

When we complain that recent models lack diversity, we usually mean I have a set of solutions I consider reasonable and the model outputs a certain choice over and over again. Can we at least define diversity once we get a set of responses? For the case where we are measuring the randomness of a set of outputs, Self-BLEU is a standard proxy (Y. Zhu et al., 2018). We simply ask how much n-gram overlap does a randomly selected response from the set have with the remainder as a reference.

$$\text{Self-BLEU} = \frac{1}{N} \sum_{i=1}^N \text{BLEU}(y_i, \{y_j\}_{j \neq i}) \quad (2.1)$$

Not only is this approach coarse in terms of n-grams, but this approach makes it hard to compare the diversity across prompts as there may only be three primary colors but five even digits (in base 10). As we discussed in Chapter 1.1, diversity is only meaningful as far as it maintains good quality. We usually have some contextual sense of the set of acceptable solutions and its size.

This view parallels the formal framework of Control Improvisation (Fremont et al., 2017), where the goal is not unconstrained randomness, but randomized generation subject to hard, soft, and randomness constraints (requirements that outputs have enough variety or unpredictability). In that sense, diversity is meaningful only when defined over a constrained set of acceptable outputs, rather than over arbitrary variations. Control Improvisation laid the groundwork for Scenic (Fremont et al., 2023) with the goal of creating diverse behaviors for testing autonomous systems.

Computer vision has historically also faced this challenge: how diverse is a set of images of cats? It is subjective whether a cartoon, cat-shaped cloud counts. To quantify the diversity of image generation models, the Fréchet Inception Distance (FID) is classically used (Heusel et al., 2017). FID is a metric that compares the distribution of features from generated images to those from real images.

More concretely, both real and generated images are passed through a pretrained Inception network to obtain feature embeddings. These embeddings are modeled as multivariate Gaussians with means μ_r, μ_g and covariances Σ_r, Σ_g . The Fréchet distance between these Gaussians is then computed as:

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

Lower FID indicates that the generated images are closer to the real data distribution in feature space. Applying similar ideas in the text space, MAUVE (Pillutla et al., 2021) is a metric based on distributional difference between text embeddings, defined by embedding real and generated text into a shared space, constructing empirical distributions P and Q , and measuring the divergence frontier $\{(D_{\text{KL}}(P \parallel R_\lambda), D_{\text{KL}}(Q \parallel R_\lambda)) : \lambda \in [0, 1]\}$ where $R_\lambda = \lambda P + (1 - \lambda)Q$; the MAUVE score is then a scalar summary (e.g., normalized area under this curve) that quantifies the similarity between the two distributions, with higher

values indicating closer alignment. Both approaches rely on the quality of the embedding model and an agreed-upon reference corpus. However, this is the best we can do when we want to quantify how “natural” (diverse and realistic) a set of generations is.

We introduce these metrics to emphasize how uncertainty, intra-set variation, or naturalness are closely related to but different from diversity. As alluded to earlier, diversity in most settings requires knowledge of what is considered the set of reasonable candidates. Complicating further, language is inherently ambiguous. It’s not clear when asking “Name a state” whether states of matter (solid, liquid, gas) are equally preferable as answers. Additionally, it is subjective whether New York City and Philadelphia are more similar or New York City and San Francisco. Geographically, NYC and Philadelphia are closer, but in skyline, density, and cultural role, NYC and San Francisco may feel more alike. This highlights that the diversity of a response generator also depends on the context and can be subjective. In chapter 3, we restrict to settings where distinct responses are equally preferred, and note measuring diversity in general remains an open question. We note survey response modeling, for instance, as a setting that targets modeling complex preference distributions (Yong Cao et al., 2025; Moon et al., 2024).

What Makes Diversity Hard to Measure?

Stepping back, we ask what makes for a good evaluation. A good evaluation needs to A) align with the real objective, B) resist gaming and superficial optimization, and C) be measurable and operationally tractable. Measuring diversity is intrinsically challenging on all three fronts. First, diversity is subjective. It’s easy to spot repetition but it’s hard to compare two differently diverse generators from each other. Second, it’s easy to game. As diversity often comes at the cost of quality, it leaves room for creative solutions that are marginally incorrect. Finally, the set of valid responses can be impractically large to realize, but the coverage of a set of responses ought to be judged against the size and shape of the full diversity of the set.

2.2 Why Is Mode Collapse So Severe in LLMs?

As described in the original GPT-4 technical report (OpenAI et al., 2024), the base model trained on perplexity maintains good calibration, as shown in Figure 2.1. Among questions for which the model is 40% confident, it answers correctly about 40% of the time. This is likely because log loss is a proper scoring rule.

Log loss is a proper scoring function. In expectation, a forecaster minimizes log loss by reporting its true beliefs.

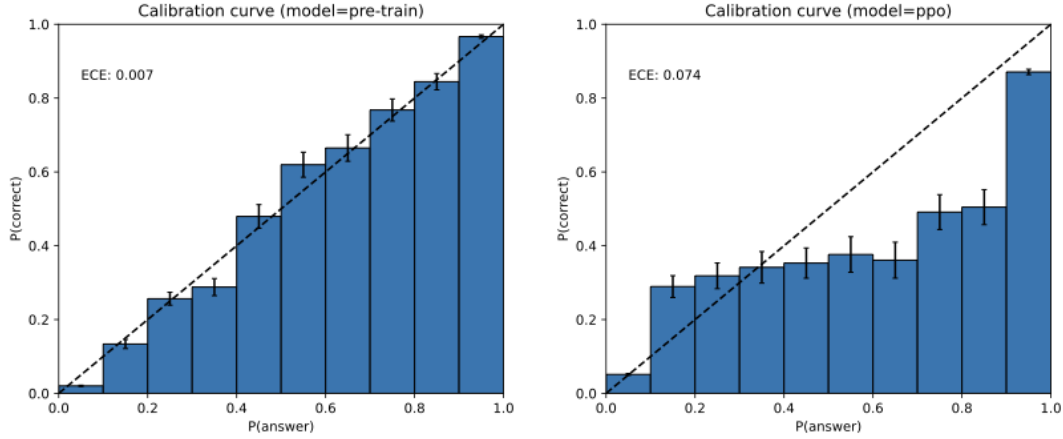


Figure 2.1: Calibration plots from the GPT-4 technical report (OpenAI et al., 2024). Left: the pre-trained GPT-4 model on a subset of MMLU is well calibrated. Right: calibration worsens after post-training.

Derivation. Let $Y \in \{1, \dots, K\}$ be the true outcome, and let $p = (p_1, \dots, p_K)$ be a reported distribution with $p_i \geq 0$ and $\sum_i p_i = 1$. Define the log loss for realized outcome $Y = y$ by

$$L(p, y) = -\log p_y.$$

If the true distribution is $q = (q_1, \dots, q_K)$, then the expected log loss is uniquely minimized by reporting $p = q$:

$$\mathbb{E}_{Y \sim q}[L(p, Y)] \geq \mathbb{E}_{Y \sim q}[L(q, Y)],$$

with equality if and only if $p = q$ on the support of q . Under the true distribution q , the expected log loss of reporting p is

$$\mathbb{E}_{Y \sim q}[L(p, Y)] = \sum_{i=1}^K q_i (-\log p_i) = -\sum_{i=1}^K q_i \log p_i.$$

If instead we report the truth, $p = q$, then

$$\mathbb{E}_{Y \sim q}[L(q, Y)] = -\sum_{i=1}^K q_i \log q_i.$$

Subtracting gives

$$\mathbb{E}_{Y \sim q}[L(p, Y)] - \mathbb{E}_{Y \sim q}[L(q, Y)] = \sum_{i=1}^K q_i \log \frac{q_i}{p_i} = D_{\text{KL}}(q \parallel p).$$

Since $D_{\text{KL}}(q \| p) \geq 0$ for all distributions p, q , with equality if and only if $p = q$ on the support of q , it follows that

$$\mathbb{E}_{Y \sim q}[L(p, Y)] \geq \mathbb{E}_{Y \sim q}[L(q, Y)],$$

with equality if and only if $p = q$. Equivalently,

$$\mathbb{E}_{Y \sim q}[-\log p_Y] = H(q) + D_{\text{KL}}(q \| p),$$

where $H(q)$ is the entropy of q . Because $H(q)$ does not depend on p , the expectation is minimized exactly at $p = q$.

In the binary case, if the true event probability is q and the reported probability is p , then

$$\mathbb{E}[L(p, Y)] = -q \log p - (1 - q) \log(1 - p),$$

which is likewise minimized at $p = q$.

Corollary. If a model is trained to minimize log loss, it is incentivized to maintain calibrated next-token probabilities rather than systematically overstate or understate its confidence in the limit.

This suggests next-token training will allow bias to creep in proportional to the occurrence rate in the training data. DS-GRPO calls this selection bias, as the initial starting policy of the RL training may have a bias towards selecting certain solutions already.

However, as shown in the other panel, reinforcement learning does not have this property. The core intuition can be seen through a simple two-coin decision problem, a classic example from decision theory.

A two-coin decision problem. Suppose an agent must choose between two coins, A and B , and receives reward 1 if the chosen coin lands heads and 0 otherwise. Let

$$\Pr(\text{heads} \mid A) = p_A, \quad \Pr(\text{heads} \mid B) = p_B,$$

with $p_B > p_A$. Let $q \in [0, 1]$ denote the probability that the agent chooses coin B , so that it chooses coin A with probability $1 - q$. The expected reward of this mixed strategy is

$$\mathbb{E}[R] = qp_B + (1 - q)p_A = p_A + q(p_B - p_A).$$

Because $p_B - p_A > 0$, the expected reward is strictly increasing in q . Therefore the maximizing strategy is

$$q^* = 1,$$

that is, to choose coin B all of the time. No mixed strategy can do better:

$$qp_B + (1 - q)p_A \leq p_B$$

with equality only when $q = 1$. The conclusion does not depend on the size of the edge. Even if coin B has only a 51% chance of heads while coin A has 49%, the reward-maximizing policy is still to choose B on every round.

This is the logic of exploitation. Once one action is believed to be even slightly better, calibrated hedging is no longer optimal for maximizing expected value. The optimizer does not care that the edge is small. It only cares about which option is larger.

From coins to candidate completions. Reinforcement learning for language models induces an analogous decision problem. For a given prompt, the model may assign nonzero probability to many candidate continuations or reasoning traces. If training rewards only whether a sampled trace succeeds, then the learning rule does not directly ask the model to preserve a faithful distribution over all plausible solutions. Instead, it asks the model to increase expected reward. If one candidate solution appears even slightly more promising than another, the gradient signal favors increasing its probability mass. Over time, small preference differences are converted into deterministic choice.

This helps explain why RL can make a model more confident than it ought to be. If the model’s initial preferences are biased, incomplete, or merely noisy, RL will still amplify them. High-probability successful trajectories are sampled more often, so they are more likely to be reinforced; after reinforcement, they become even more likely to be sampled again. A mild asymmetry becomes extreme. DS-GRPO distinguishes this as reinforcement bias.¹

The different nature of these two phases of LLM training is made clear when we think about next-token training as akin to optimizing the forward KL divergence and reinforcement learning as optimizing the reverse KL divergence.² Let P denote a target distribution over outputs and let Q_θ denote the model distribution. The forward KL is

$$D_{\text{KL}}(P \parallel Q_\theta) = \mathbb{E}_{x \sim P} \left[\log \frac{P(x)}{Q_\theta(x)} \right] = \mathbb{E}_{x \sim P} [-\log Q_\theta(x)] - H(P).$$

Since $H(P)$ does not depend on θ , minimizing forward KL is equivalent to maximizing

$$\mathbb{E}_{x \sim P} [\log Q_\theta(x)],$$

which is exactly maximum likelihood. Thus, classifying the next token with cross-entropy is optimizing forward KL: samples come from the data distribution, and the model is pushed to assign mass wherever the data places mass.

In contrast, maximum-entropy RL starts from the objective

$$\mathbb{E}_{x \sim Q_\theta} [r(x)] + H(Q_\theta).$$

¹In a sense, we perhaps should not be surprised that *RL* emerged from a quant shop: what is more quintessentially quant than exploiting the smallest detectable alpha for profit? Reinforcement learning behaves similarly. Give it even a slight estimated edge, and absent some countervailing regularizer, it will lever that edge into a concentrated position.

²This derivation and framing were inspired by Dibya Ghosh’s blog post “KL Divergence for Machine Learning” (Ghosh, 2018).

Now define a target distribution in energy form by

$$P(x) \propto \exp(r(x)).$$

Then

$$\log P(x) = r(x) - \log Z$$

for normalizing constant Z , so

$$\mathbb{E}_{x \sim Q_\theta}[r(x)] + H(Q_\theta) = \mathbb{E}_{x \sim Q_\theta}[\log P(x)] + \log Z + H(Q_\theta).$$

Using $H(Q_\theta) = -\mathbb{E}_{x \sim Q_\theta}[\log Q_\theta(x)]$, this becomes

$$\mathbb{E}_{x \sim Q_\theta}[\log P(x) - \log Q_\theta(x)] + \log Z = -D_{\text{KL}}(Q_\theta \parallel P) + \log Z.$$

Therefore, maximizing reward plus entropy is equivalent, up to an additive constant, to minimizing the reverse KL

$$D_{\text{KL}}(Q_\theta \parallel P).$$

This is the key asymmetry: the policy samples from itself, is rewarded for concentrating mass on high-reward outputs, and is only secondarily encouraged to remain diffuse through the entropy term. That is why reverse KL is mode-seeking in a way that forward KL is not.

In existing RLVR training, the main remaining buffer against collapse is KL divergence to the reference model. Intuitively, this term penalizes the policy for drifting too far from the reference distribution, so it acts as a tether against aggressively concentrating all probability mass on a narrow set of trajectories. This, however, slows down training, since the point of RL training is to change the model meaningfully so it acquires a capability the reference model does not have. As such, it is increasingly common to reduce or remove the reference model altogether (Q. Yu et al., 2025). Further, the reference model itself may already contain these biases.

The broader lesson is that diversity collapse is not a sign that the model is training poorly. On the contrary, it is the rational consequence of optimizing the wrong objective. If success is measured only by single-sample reward, then concentration is exactly what one should expect. The challenge, and the motivation for the rest of this dissertation, is to design evaluation methods and generation procedures that optimize simultaneously for correctness and coverage over valid alternatives.

2.3 Approaches to Diversity

There is a large and important space of training-based approaches. One can target diversity during supervised fine-tuning or mid-training. With knowledge of the exact mode collapse, explicitly training the model on the underrepresented mode will reintroduce valid continuations. In domains where the desired answer distribution is known, one can also shape

Table 2.1: Exact prompt templates used for the rock-paper-scissors and Pokémon-type experiments.

Setting	Prompt template
RPS inference	System: You are playing rock paper scissors. Respond with exactly one word: rock, paper, or scissors. Nothing else. User: Round {i}: Let’s play! What do you choose?
RPS SFT data	System: same as above User: same format as above, with randomized round numbers across choices Assistant: one of rock, paper, or scissors
Pokémon	System: Pick a Pokémon type. Respond with exactly one word: the type name. Nothing else. User: Round {i}: Pick a Pokémon type!

Table 2.2: A simple SFT intervention can partially rebalance an obviously skewed output distribution, though the effect is unstable across training.

Checkpoint	Rock	Paper	Scissors	Training loss
Base model	18%	10%	72%	—
Step 300 (epoch 1)	2%	23%	75%	0.05
Step 600 (epoch 2)	1%	71%	28%	0.33
Step 900 (epoch 3)	27%	33%	40%	0.43

the logits more directly by training on data whose target distribution is intentionally balanced or uniform over valid outcomes. More broadly, if RL training sharpens preferences too aggressively, then part of the solution may be to redesign the training objective itself so that coverage over valid answers is rewarded rather than ignored (Lanchantin et al., 2025). At the same time, RL-style post-training can sometimes preserve general capabilities better in practice when paired with explicit regularization or pretraining-gradient mixing, as in InstructGPT (Ouyang et al., 2022a).

As a quick experiment for rock-paper-scissors, we find prompting GPT-4o mini gives 18% rock, 10% paper, and 72% scissors. Since the target proportions should be uniform, we use the simple SFT approach of training on one third of each outcome. One such run is summarized in Table 2.2.

This result simply shows SFT distribution can dramatically impact diversity and is a clear place to directly fix known diversity issues. The experiment was done with batch size 1 and 300 samples with a 100:100:100 rock:paper:scissors. However, catastrophic forgetting is well established in LLM fine-tuning (H. Li et al., 2024), and training on a toy distribution

Table 2.3: A small balanced SFT run on GPT-4o-mini on rock-paper-scissors substantially changes the type distribution for Pokémon it produces.

Rank	Base GPT-4o-mini	Balanced SFT (bs=1, 3 ep)
1	Fire (40)	Rock (13)
2	Dragon (38)	Psychic (11)
3	Electric (10)	Fighting (11)
4	Water (6)	Water (10)
5	Psychic (3)	Bug (9)

like rock-paper-scissors can easily hurt unrelated capabilities. We see this bleed over in the Pokémon³ (Game Freak, 1996) responses in Table 2.3, where increasing the model’s preference for rock in rock-paper-scissors also changes its preferences over Pokémon types.

It is difficult, but a worthwhile pursuit to systematize approaches to untraining diversity collapse or training models to not collapse in the first place. However, there are others better positioned to study these problems, so we focus our efforts on measuring and building diverse systems with non-diverse models.

The next two chapters of the dissertation focus on two methods for improving generative diversity without modifying model training. We focus on training-free methods for convenience and to make measurable progress. However, this focus should not be mistaken for a claim that diversity is only or best solved as an inference-time problem. The first, *SimpleStrat* (Wong et al., 2025), operates at the prompt and sampling level: it uses the model’s own world knowledge to partition the solution space and then resamples from those partitions in a more balanced way. The second, *Stylus* (Luo et al., 2024), operates at inference time in image generation by retrieving, composing, and randomizing adapters so as to improve both quality and diversity. Although these systems are quite different in mechanism, they share a common philosophy: diversity can often be improved substantially at deployment time without trading off quality.

³Originally marketed in Japan as *Pocket Monsters*, but later marketed to Western audiences under the now-household name Pokémon.

Part II

What Can We Do About It?

Chapter 3

SimpleStrat

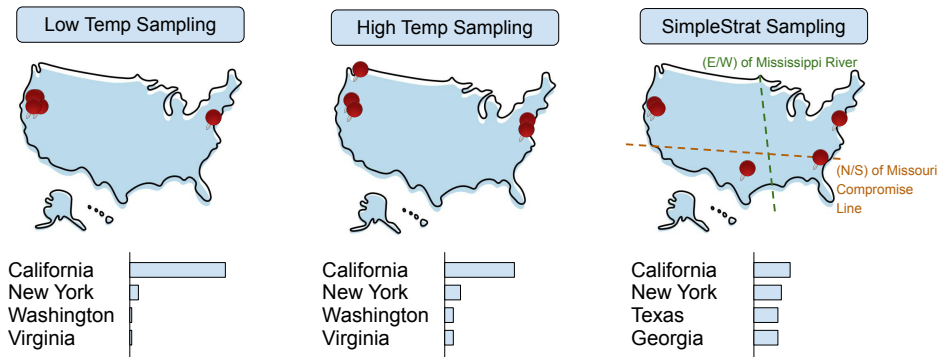


Figure 3.1: **SimpleStrat recovers diversity when temperature scaling fails.** For the prompt “Name a U.S. state,” language models exhibit a strong bias toward “California.” Our method, SimpleStrat, employs automatic stratification to uncover salient diversity dimensions (e.g., whether a state lies east or west of the Mississippi River) and applies stratified sampling to achieve balanced and diverse coverage across responses.

3.1 Overview

Large language models (LLMs) are routinely resampled in order to get a wide set of plausible generations. Three key settings where this is important are: 1) improving exploration for planning and search (e.g. Tree-of-thought (Yao et al., 2024), AgentQ (Putta et al., 2024)), 2) generating inputs for integration tests, especially for LLM systems (Samvelyan et al., 2024), and 3) generating diverse datasets for post-training (Dubey et al., 2024) and fine-tuning (H. Dai et al., 2023). These use cases rely on resampling outputs to generate multiple plausible utterances in hopes of capturing multiple answers and candidate solutions.

Temperature scaling, which controllably flattens an LLM’s softmax distribution, is commonly used to increase response diversity. However, this approach assumes that next-token probabilities accurately represent the set of valid answers, which is not true under mode collapse. In such cases, increasing temperature does not lead to meaningful diversity. Moreover, temperature scaling introduces a second limitation: higher temperatures degrade generation quality. Recent evidence suggests that removing temperature scaling is desirable for multi-step reasoning to reduce error compounding (E. Zhang et al., 2024). This is especially critical in syntax-sensitive settings such as code generation, where low temperatures (≤ 0.15) are often used. We illustrate this severe mode collapse in Figure 3.1. When asked to “Name a US State,” the model heavily skews towards answering “California.” High temperature only marginally softens the skew while surfacing incorrect answers and hurting instruction following.

Our goal is to improve response diversity when resampling LLMs, even in cases of severe mode collapse in next-token probabilities, without manual intervention (Lanchantin et al., 2025). Our analysis reveals that GPT-4 assigns 87% of its logit weight to “California” when prompted to name a US state. This observed bias can be attributed to the worsening of calibration due to post-training as reported in the GPT-4 tech report (OpenAI et al., 2024). This stark bias mirrors human cognitive bias, exemplified by the blue-seven phenomenon—where individuals disproportionately select blue and seven when asked to choose a random color and number (Towse et al., 2014). To counteract similar biases in human populations, social scientists, particularly in political polling, employ stratified sampling techniques (Simpson, 1951; Howell, 1992; Morris, 2022). We propose adapting this method to address mode collapse in LLMs.

We propose SimpleStrat, a training-free sampling approach to increase diversity of responses. SimpleStrat improves LLM generation diversity without compromising quality, yielding outputs that better align with the task’s objective answer distribution. SimpleStrat consists of three stages: auto-stratification, heuristic estimation, and probabilistic prompting. Even if a language model cannot generate diverse solutions, we find that it can be prompted to identify useful partitions of the solution space based on the user request. We call this process *auto-stratification*. In Fig. 3.1, SimpleStrat identifies two semantically significant strata from the user request, “Name a US State”: “(East/West) of the Mississippi River” and “(North/South) of the Missouri Compromise Line.”

Next, the heuristic estimation computes the joint probabilities across all strata. Returning to Fig. 3.1, SimpleStrat estimates the proportion of solutions in each of the four possible regions of the US. Finally, SimpleStrat samples a stratum from the joint probability distribution and augments the original user prompt with constraints based on the selected stratum. This prompt can then be used to sample fully-formed answers to the user query. We note that this approach to response diversity is orthogonal to increasing temperature which also introduces variance to the generation.

Response diversity is notoriously difficult to measure as it requires a notion of quality specific to the application. Otherwise, random tokens present a trivial solution to achieving diverse yet meaningless responses. On the extreme of measuring quality, question answering

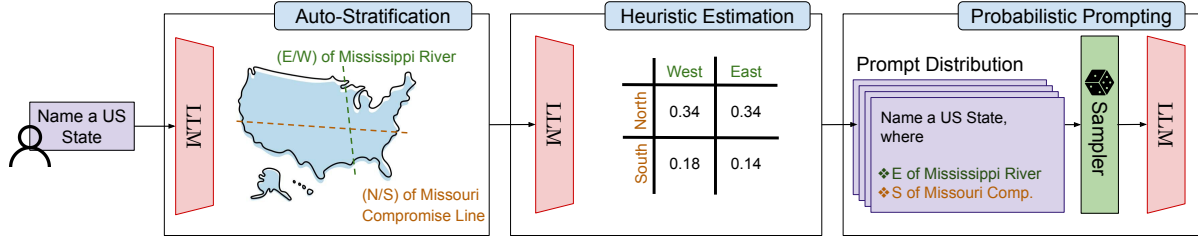


Figure 3.2: **SimpleStrat workflow.** SimpleStrat employs 3 phases: 1) auto-stratification to identify good dimensions of diversity that divide the solution space into equal partitions, 2) heuristic estimation to estimate the proportion of solutions in each stratum, and 3) probabilistic prompting where a concrete prompt is randomly sampled from the prompt distribution specified by the previous two phases. Critically, diverse resampling comes from both the random choice of prompt as well as the temperature of the LLM decoding.

benchmarks capture the quality of the responses but are carefully designed to only admit one valid solution. As such, a fitting setting to measure diversity is required to have underspecified questions for which more than one plausible answer exists. Distinct from ambiguous question answering where the goal is to measure the model’s ability to ask clarifying questions, we seek questions where the quality of the response can be independently verified without additional context. Based on these requirements, we introduce CoverageQA, a benchmark of underspecified questions with on average 28.7 equally plausible answers.

We adopt three measures of response diversity based on the setting: 1) In settings where the target answer distribution and logits are available, we measure *distributional diversity* by computing the Kullback-Leibler (KL) Divergence from the response distribution to the target answer distribution over valid answers. By measuring diversity this way, we can both penalize models for producing nonsense responses and missing valid responses. 2) In settings where we have only black-box access to the model, the response distribution can be estimated by sampling. Since the tail of the distribution is inherently difficult to capture, recall and precision serve as a natural proxy for diversity and quality. We call this *coverage diversity*. 3) Finally, in settings where exhaustively enumerating valid solutions is infeasible, we cannot assess if valid responses are missing. As such, we propose text embedding distance as a proxy for diversity. We assess SimpleStrat across these three settings: measuring *distributional diversity* with Llama 3 and *coverage diversity* over Claude and GPT-4o with prompts from CoverageQA as well as *open-ended diversity* with embedding distance on creative writing prompts from WritingPrompts (A. Fan et al., 2018).

When assessing *distributional diversity*, we show SimpleStrat samples from a less biased distribution by computing the response distribution of our method based on next-token probabilities. On CoverageQA, SimpleStrat implemented with Llama 3 models demonstrate a substantial 0.36 average reduction in KL Divergence, signifying a substantial decrease

in dissimilarity between the model’s response distribution and the target distribution. In the blackbox setting, we see as much as $4\times$ increase in recall, especially evident at low temperatures. These gains lead to a consistent increase in diversity on top of temperature scaling, leading to improved diversity at all temperatures. Finally, we consider the open-ended setting of creative writing. We measure diversity by resampling pairs of creative writing story plots from WritingPrompts (A. Fan et al., 2018) and see similar embedding distances at temperature 0 sampling with SimpleStrat comparable to temperature 1 with standard decoding. Crucially, these gains do not come at the expense of generation quality.

Taken together, these results motivate SimpleStrat as a training-free approach for improving diversity with *auto-stratification* and *probabilistic prompting*. The method improves diversity across several metrics and settings. On CoverageQA, it achieves an average reduction of 0.36 in KL divergence on Llama 3 models and as much as a $4\times$ increase in recall for GPT-4o. On WritingPrompts (A. Fan et al., 2018), SimpleStrat yields pairwise embedding distances at temperature 0 comparable to standard sampling at temperature 1.

3.2 Related Works

Temperature Scaling

Going back as far as Platt scaling (Platt, 2000) and later applied to neural networks (Hinton, 2015; Guo et al., 2017), temperature scaling controls the randomness of probability distributions¹. For dataset generation with LLMs, Chung et al. (2023) extends temperature-based diversity by additionally downsampling previously sampled tokens. To address the decrease in quality, they advocate for human intervention to manually filter out irrelevant diversity and manually fix wrong answers in QA tasks. We show in our work that temperature scaling leaves much to be desired.

Improving Language Model Diversity with Search

In autoregressive generation, choices over early tokens tend to have more impact on the eventual completion. Beam search ameliorates this bias by allowing for multiple candidates in searching for the probability maximizing completion, Maximum a Posteriori (Lowerre et al., 1976). At the end of the search, beam search will have multiple candidate solutions encountered during search. Diverse Beam Search (DBS) proposes introducing an auxiliary dissimilarity objective quantifying the diversity among candidates in the beam (Vijayakumar et al., 2016). Especially on the task of image captioning, DBS shows improvement for discovering higher probability completions and discovering diverse continuations. Our improvements are orthogonal to beam search and our in-context approach corrects for inaccuracies in the modeled likelihoods of candidate solutions.

¹Use of a temperature parameter goes back at least to Verhulst’s development of logistic regression in response to Malthus’ *An Essay on Principle of Population* (Malthus, 1798; Verhulst, 1838).

Other approaches (Samvelyan et al., 2024; Bradley et al., 2023; Novikov et al., 2025) based on MAP-Elites (Mouret et al., 2015) require manually determined dimensions of relevant diversity and discretization of the solution space into equally-sized bins. Diversity is then achieved by mutations and evolutionary methods to cover adjacent bins. This search is potentially slow if the seed set of solutions does not already provide coverage over the solution space. Our approach does not need seed solutions and avoids manually identifying dimensions of diversity. Instead, we rely solely on capabilities within the model.

In-context Methods to Increase Diversity

When LLMs were first introduced, LMs were used to augment existing datasets with more diversity (Wei et al., 2019; Ng et al., 2020; H. Dai et al., 2023). As natural language is difficult to guarantee correctness, the space of augmentations is conservatively limited to thesaurus-based synonym replacement. More recently, Language Model Crossover proposes presenting a random subset of existing data points to an LLM and asking it to hallucinate more data points that likely came from the same distribution (Meyerson et al., 2023). This is limited to combining aspects of existing data points into new generations. Although these methods address the limitations of using the model’s token probabilities by in-context learning, they are ineffective at generating meaningful diversity. They are limited to either human-identified domains of interest or trivial variations sourced from synonyms or random subsampling of the existing data.

Applications of Diversity

As shown by (Raventós et al., 2024), dataset diversity is crucial for model generalization. Below sufficient coverage of the desired task, the model will resort to memorization, but when sufficient diversity is presented it will learn to generalize. As LLMs are increasingly used for generating synthetic data (Dubey et al., 2024), methods for diversity will be critical. This insight follows from extensive work demonstrating the benefits of data augmentation for bias mitigation (Sharmanska et al., 2020) and domain adaptation (S.-W. Huang et al., 2018; Dunlap et al., 2023; Trabucco et al., 2023b).

In code and math applications, checking validity efficiently enables more aggressive augmentations. One such augmentation for diversifying the languages supported by the model, data is translated to different natural or programming language (N. Chen et al., 2023; Casano et al., 2023). In other domains such as images, text-to-image models have been used to diversify data into uncommon settings. In the setting of diversifying an accumulating dataset, these methods can take advantage of an existing source of variance (for translation) or a set of previously generated data points. Our primary focus is on settings where SimpleStrat is unaware of past data samples to support a wider set of applications.

Ambiguous or Underspecified Datasets

ClariQ (Aliannejadi et al., 2020), CLAUQA (J. Xu et al., 2019), and AmbigQA (Min et al., 2020) focus on assessing the LM’s ability to formulate clarifying questions. These questions tend to have only two candidate solutions, as there exists a ground truth clarifying question whose answer fully specifies the question. Ambiguous Trivia QA (Kuhn et al., 2022) also looks at under-specified questions but assumes a user has contextual information that’s hidden. For instance, “Where in England was she born?” or “Who was the first woman to make a solo flight across this ocean?” We distinguish our underspecified question setting in this paper as one where the user is indifferent. In this setting, given an answer it should be easy to verify the answer is correct without additional hidden context.

Coding datasets like Description2Code (Caballero et al., 2016), Wiki2SQL (V. Zhong et al., 2017), SPIDER (T. Yu et al., 2019), code-contest (Y. Li et al., 2022), Apps (Hendrycks et al., 2021), and Leetcode Hard (Shinn et al., 2023) admit multiple valid answers. However, the space of valid implementations is infinite, making diversity difficult to measure, and good coding practices enforce preferences among valid implementations. We additionally construct CoverageQA to have an exhaustive list of ground-truth answers in order to measure the impact of diversity on coverage.

3.3 The SimpleStrat Method

Workflow overview

As illustrated in Figure 3.2, SimpleStrat consists of three stages, 1) auto-stratification, 2) heuristic estimation, and 3) probabilistic prompting. The outputs of the first two stages can be cached per prompt to avoid recomputing partitions.

Auto-Stratification

For a given user request, r_{user} , we call S , the space of valid solutions. In many settings, the space of potential solutions, S may be naturally partitioned based on geography, parity, or demographics. The partition function, $P : S \rightarrow L$, assigns any solution s from S to a partition label l_j in L the set of partition labels. Partition functions are most useful if they’re as balanced as possible. A balanced partition function minimizes

$$imbalance(P, L) = \max_{l \in L} (|\{s \mid P(s) = l\}|) - \min_{l \in L} (|\{s \mid P(s) = l\}|)$$

The goal of auto-stratification is to search for a set of partition functions $\mathbf{P} = \{P_1, P_2, \dots, P_n\}$, that are balanced. In settings where valid solutions are easy to miss or the solution space is large (or even infinite), stratified sampling ensures that a limited number of samples provides broad and even coverage.

Based on this insight, we prompt the language model to identify promising dimensions of diversity. Concretely, the language model proposes clarifying questions that can eliminate

roughly half of the potential solutions conditioned on the user request. These clarifying questions tend to align with semantically significant differences. In the running example, when asked, “Name a US State,” the states can be partitioned based on whether they lie east or west of the Mississippi River.

Heuristic Estimation

As previously observed in (Zou et al., 2022; Yan et al., 2023; Halawi et al., 2024), LLMs can be used in forecasting to estimate well-calibrated probabilities of events that have not yet occurred. For forecasting, the model’s success benefits substantially from having updated news through web search. Although unnecessary for the offline benchmarks we consider, this may be helpful for accurate estimation depending on the application. However, as our goal is diversity, we stand to benefit even from coarse-grain approximate proportions. We employ a similar reasoning template as Halawi et al. (2024) to estimate the proportion of valid solutions that lie within each stratum.

In heuristic estimation, we look to estimate the joint distribution for each stratum, $\vec{l} = [l_1, l_2, l_3, \dots]$. Formally, we define the weighted-stratification as $\mathcal{W} = (\mathbf{P}, \rho)$, where $\rho(\vec{l}) = Pr_{s \sim S}[P_1(s) = l_{1,j}, P_2(s) = l_{2,j}, P_3(s) = l_{3,j}, \dots]$ for $\mathbf{P}$ identified in auto-stratification. To improve scalability, we assume the partition functions are independent and multiply the marginal probabilities to get the joint probabilities associated with each stratum.

$$\rho(l_1, l_2, \dots, l_m) = \prod_i \rho_i(l_i) \quad (3.1)$$

We ask the LLM for each l_j to estimate the marginal proportion of solutions for which it holds. As this may not add up to 1, we normalize the estimates to form a proper probability distribution. For simplicity, this chapter focuses on settings where all solutions in the solution space are equally likely. As noted in Sec. 3.3, we encourage the LLM to propose balanced partitions. However, heuristic estimation allows imbalanced partitions by reweighting the sampling process to favor strata with more potential solutions. In Fig. 3.2, the LLM determines the joint probabilities across two strata, the Mississippi River and the Missouri Compromise Line.

Probabilistic Prompting

After heuristic estimation, a set of strata are sampled from the joint probability distribution in Eqn. 3.1. This implicitly forms a *probabilistic prompt*, which specifies a distribution over concrete language model prompts. After a prompt is sampled, the LLM is then used to sample from within the stratum. Back to Fig. 3.2, East and South are sampled from the Mississippi and Missouri strata respectively, augmenting the final prompt with diverse specifications.

Formally, call $\vec{l}$ a stratum defined by choices of $l_{i,j}$ for each P_i across all i . Call *Prompt* a function that maps the stratum, $\vec{l}$, to a concrete prompt, $Prompt(\vec{l})$. The probabilities of the

prompt distribution are defined by $Pr[\text{Prompt}(\vec{l})] = \rho(\vec{l})$. We then compute the probability of a solution as follows:

$$\begin{aligned} Pr[s] &= \sum_{\vec{l}} Pr[\text{Prompt}(\vec{l})] Pr[s \mid \text{Prompt}(\vec{l})] \\ &= \sum_{\vec{l}} \rho(\vec{l}) Pr[s \mid \text{Prompt}(\vec{l})]. \end{aligned} \tag{3.2}$$

The specific language model’s next-token probabilities define $Pr[s \mid \text{Prompt}(\vec{l})]$.

As the probabilistic prompt is in a human-readable form, the user can inspect the properties and the proportions and modify it to adjust for unwanted bias or remove unwanted factors. For instance, when proposing English baby names, we may want the model to propose male vs female names equally often, even though there are more unique female than male baby names ². This interpretability and controllability is a major advantage of SimpleStrat in practice.

3.4 Benchmark for Diversity: CoverageQA

Overview

We wish to evaluate generation diversity in settings where 1) user requests have more than one distinct correct answer, 2) answers are equally likely, and 3) answers do not require hidden or implicit context to verify. These three features allow diversity to be measured quantitatively in terms of a language model’s ability to represent the target distribution when resampled. The second condition makes the evaluation easier, reducing the problem to a measure of coverage over the solution space. Existing benchmarks are not designed for this purpose. As such, this chapter introduces CoverageQA for assessing language model generation diversity. The dataset consists of two splits: CoverageQA-Curated, a manually curated set of naturally underspecified questions, and CoverageQA-Wikipedia, an automatically generated dataset of underspecified questions.

CoverageQA-Wikipedia Approach

To generate CoverageQA-Wikipedia, we leverage the Wikidata knowledge base which contains all relational mappings between entities and properties in Wikipedia. Our generation process starts with an initial item-property pairing and a constraint on the number of correct answers. We then perform a recursive search through Wikidata to find all sets of item-property constraints and their corresponding answers that meet our criteria. These constraints are subsequently transformed into natural language questions using GPT-4.

²As reported in (Wilson, 2016), there are 18,993 unique names for girls and 13,959 for boys in 2015 report by Social Security Administration.

Consider an initial pairing of the Wikidata item “country” with the property “instance of”. We might specify that we want between 20 and 40 valid answers. Our search would then yield a set of all constraints from the knowledge base that fit the initial conditions, such as “instance of country, located in Europe, uses Euro as currency”. GPT-4 would convert this into a natural language question like “Name a country located in Europe that uses the Euro as its currency.”

This approach has several advantages: 1) it creates a diverse and extensible benchmark that can track weekly Wikidata updates, 2) it allows the size of the solution space to be specified by adding or removing constraints, and 3) it can in principle curate a large dataset with relatively little manual effort or supervision. In the initial instantiation of CoverageQA-Wikipedia, we use 145 questions across 6 domains, each corresponding to a different initial seed item-property pair. To ensure quality, we employ both automatic filters, such as excluding overly generic properties, and manual curation to remove redundant or unsuitable questions.

Dataset Details

Generation Procedure

To generate the questions, we manually came up with initial item and property pairings to run the recursive search. We constrain the recursive search to yield 20–40 possible answers to keep the questions within common and relevant categories. We found that with fewer than 20 answers, the questions become too obvious, while with more than 40, they tend to get too specific and stray from general knowledge. The recursive search first finds all items that satisfy the initial conditions, then iteratively adds properties in steps until either the maximum depth (number of constraints) is reached or the number of answers falls outside the desired range. We blacklist properties that are detrimental to high-quality question generation, such as an item’s presence in a specific database, numeric properties like population, and properties that introduce high ambiguity. We then manually evaluate the generated conditions and answers to ensure they meet our criteria. With an appropriate initial condition, one query can generate hundreds of valid constraints that can later be turned into questions. Finally, we use GPT-4 to convert these constraints into natural language.

3.5 Evaluation Setup

For our evaluation, we use the proprietary models GPT-4o (`gpt-4o-2024-08-06`) and Claude 3.5 Sonnet (`claude-3.5-sonnet-20240620`). We use open-source models from the Llama 3 and 3.1 families. Inference for these models was run on 8 A100-80GB GPUs. CoverageQA-Wikipedia is based on the snapshot from 07-03-2024. For text embeddings, we use third-generation embeddings from OpenAI.

Table 3.1: CoverageQA domains.

Domain	Question Count	Average Number of Answers
General Knowledge (Curated)	10	64.1
US National Parks	5	11
Geography Questions	74	27.5
Periodic Table Elements	11	24.2
Physics Nobel Laureates	31	16.8
Famous Athletes	18	9
Musical Instruments	6	10
Total	155	24.1

Auto-stratification Prompt. We provide the full prompt in Table 3.2. To improve prompt adherence, we provide one in-context example in the form of one simulated round of multi-turn conversation, i.e. we provide an example set of reasoning following the template.

System Prompt:
You are a helpful brainstorming assistant that is careful to consider all factors to a problem.

User:
I am tasked with the following request:
<i>% User Request</i>
Help me brainstorm how to respond to the user request by providing a list of True/False properties the solution may or may not have. Use the following step-by-step to come up with good properties:
1. If you were playing 20 questions, what’s a good first question to ask that would split the possibilities in half?
List at least 5 questions and their corresponding properties.
Question: <Description>
2. Rewrite each question as a True/False property that’s true for one half and false for the other.
Question: <Description>
True/False Property: <Property Description>
3. For each property, come up with an example that would satisfy the property.
Property: <Description>
Example: <Description>
Is it a valid answer to the user’s request? <Yes/No>
4. For each property, come up with an example that would not satisfy the property.
Property: <Description>
Example: <Description>
Is it a valid answer to the user’s request? <Yes/No>
5. Does the property mention a candidate answer in it?
Property: <Description>
Does the property mention a candidate answer in it? <Yes/No>
6. For each property, list whether we should include it or not in the final list of properties. Do not include ones where an example from above is not valid or if it mentions a candidate answer in it.
Property: <Description>
Include in final list? <Yes/No>
Final List of True/False Properties:
1. <Property Description 1>
2. <Property Description 2>
Ensure all properties listed are sentences that are either True or False

Table 3.2: Full prompt for auto-stratification.

Heuristic Estimation Prompt. We first take each partition function from auto-stratification and estimate a starting probability with the prompt in Table 3.3. This prompt is heavily inspired by Halawi et al. (2024). We then collect all the proportions and pass them through a final heuristic estimation prompt to remove redundant properties, such as negations, and give the model a chance to correct inaccurate probabilities. See Table 3.4 for the full prompt. Finally, we ask the model to select at most three properties.

Note that for performance reasons, we estimate the marginal probabilities and make a simplifying assumption of independence. This is not strictly true if one partition function is the negation of the other. This leads to potential strata assigned positive probability even though the stratum has no solutions. Otherwise, there would be $2^{\# \text{ of partition functions}}$ strata whose probabilities must be estimated. Further, LLMs seem less reliable when asked to

System Prompt:

You are an expert superforecaster, familiar with the work of Tetlock and others. Your mission is to generate accurate predictions for forecasting questions. Aggregate the information provided by the user. Make sure to give detailed reasoning.

User:

I am tasked to estimate the probability that a random solution to “*User Request*” has the following property “*Partitioning Property*”

Instructions:

1. Provide at least 3 reasons why the answer might be no.
 { Insert your thoughts }
2. Provide at least 3 reasons why the answer might be yes.
 { Insert your thoughts }
3. Rate the strength of each of the reasons given in the last two responses. Think like a superforecaster (e.g. Nate Silver).
 { Insert your rating of the strength of each reason }
4. Aggregate your considerations.
 { Insert your aggregated considerations }
5. Output your answer (a number between 0 and 1) with an asterisk at the beginning and end of the decimal.
 { Insert your answer }

Table 3.3: Prompt for partition-specific heuristic estimation.

estimate fine-grained probabilities, whereas most marginal probabilities are by design close to 0.5.

Formally, if $P = \neg Q$, then the stratum $P \wedge Q$ has zero probability, even though we assumed it to be $\Pr[P] \Pr[Q]$. We handle approximation error in estimating the true prompt distribution by allowing the model to reply “Invalid” to trigger a resample. With this adjustment, the probabilistic prompt distribution is maintained for this extreme case.

System Prompt:

You are an expert superforecaster, familiar with the work of Tetlock and others. Your mission is to generate accurate predictions for forecasting questions. Aggregate the information provided by the user. Make sure to give detailed reasoning.

User:

I'm playing a game where my friend has been tasked to:

“*User Request*”

I have the following Y/N statements I can ask my friend. I have probabilities that I think are true: % Insert numbered list of partitions and proportions.

Instructions:

1. For each Y/N statement, is it redundant with another statement?

Y/N statement: <description>

Is redundant? <Y/N: Explanation>

2. Are any of the probabilities inaccurate? If it is sufficiently accurate just report back the same value.

Y/N statement: <Description>

Is accurate? <Y/N: Explanation>

Probability: <Probability>

3. Pick at most three statements that are least redundant and pair well together. Prefer ones that are closest to 50% for most information.

Final List of True/False Properties:

1. <Y/N Properties>:: <Probability>

2. <Y/N Properties>:: <Probability>

Table 3.4: Prompt for final heuristic estimation.

3.6 Measuring Diversity

We consider three measures of diversity. In the setting where we have access to all ground truth answers, we can measure *distributional diversity* when the model weights are available and *coverage diversity* when blackbox. Without a comprehensive list of valid solutions, we rely on embedding distance to measure *open-ended* diversity.

Distributional diversity. For models with accessible softmax next-token probabilities, we can compute the probability of each solution in the solution space. We then define distributional diversity as the distributional distance between the response distribution implied by the sampling process and logits and the ground-truth distribution derived from these probabilities. For our baseline, we prompt the models and directly compute $Pr[s|Prompt(\vec{l})]$ for each solution, s . This is simply the product of the individual next-token probabilities. For SimpleStrat, the probability involves the next-token probability conditioned on the probability the prompt is selected. Formally, the probability an answer is sampled by SimpleStrat can be computed based on Eqn.3.2. The next-token-probability-based response distribution

$Pr[s|Prompt(\vec{l})]$ computed just as in the baseline, and we do a sum weighted by the joint probabilities assigned in heuristic estimation. We assign the remaining probability density to an “Invalid” category to form a proper distribution. The probabilistic formulation allows us to easily compute the response distribution of SimpleStrat. Note that by design the ground-truth distribution for CoverageQA is uniform over valid solutions and zero elsewhere.

Coverage diversity. In settings where we do not have access to the next-token distribution, we evaluate diversity by resampling responses to CoverageQA 100 times per question. This allows us to empirically observe the diversity in the form of coverage. We call this coverage diversity. To measure coverage, we report the recall: unique valid solutions divided by total valid solutions on the reference solutions. This is not to be confused with a notion of recall where we might measure how many valid solutions a classifier recognizes as valid. To ensure this does not come at the cost of quality, we also show precision is not reduced. We show an ablation of just asking the model to propose criteria without 20 questions formulation in Fig. 3.6. We see little improvement from only allowing the model to propose a set of criteria and then applying the uniform sampling like AttrPrompt (Y. Yu et al., 2023a)

Open-ended diversity. For many compelling applications of diversity, a goal of diverse generation is to identify solutions not previously known to the user. In these open-ended settings, the space of valid solutions often cannot be exhaustively enumerated. Here, creative writing prompts serve as an example. We ask the model to provide plot outlines in order to decouple high-level creative choices from stylistic choices. We measure diversity by sampling a pair of story plot outlines from the same prompt and measuring the cosine distance of their embeddings. Unlike CoverageQA, writing a high-quality plot is a more challenging, longer-context task. Accordingly, we assess quality by spell-checking for non-standard English words and adherence to the requested three-act outline. These checks do not capture every aspect of quality, but they are sufficient to capture the quality degradation characteristic of high temperature. Claude does not allow temperatures above 1, and GPT-5 was not studied because it does not expose user-controlled temperature.

3.7 Qualitative Example

Consider the question “Name one Great Lake in the United States.” as shown in Fig. 3.3. We see that temperature scaling with GPT-4o results in a strong preference/bias for Lake Erie. This is certainly a correct continuation and under the language modeling objective should be incentivized. Increasing the temperature helps sample the next most likely candidate solutions more often. However, even when increasing the temperature past 1 there is still incomplete coverage over the solution space. Specifically, Huron is only seen once out of 100 samples at 1.5 temperature, and Lake Ontario is never observed. This is undesirable if the data is used to propose candidate plans, generate test cases, or generate training data. Further, the model has a strong persistent preference for Lake Erie potentially leading to undesired biases in downstream use cases.

In Fig. 3.3, we observe a much more uniform distribution over valid solutions when using

SimpleStrat. Notably, we observe full coverage over all five Great Lakes. At lower temperatures, there is still a preference for a single lake over the others, in this case Lake Huron. However, this is less pronounced at higher temperature showing the orthogonal benefit of SimpleStrat. We provide additional examples in Fig. 3.5 and further top-20 response distributions in Fig. 3.4. These visualizations show the full resolution response probabilities as computed by log probs of each solution family, emphasizing just how strongly the models prefer certain solutions.

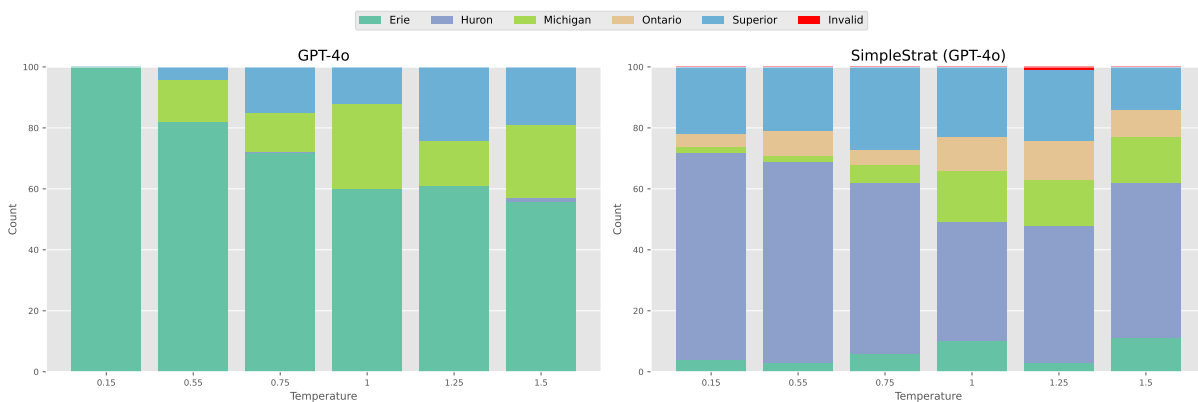


Figure 3.3: **Diversity scaled with temperature.** We show 100 resamples of "Name one Great Lake in the United States." On the left, we show the result of resampling GPT-4o 100 times per temperature. In contrast to SimpleStrat on the right, GPT-4o at temperature 1.5 still only samples Lake Huron once and never samples Lake Ontario. SimpleStrat improves the diversity across all temperatures.

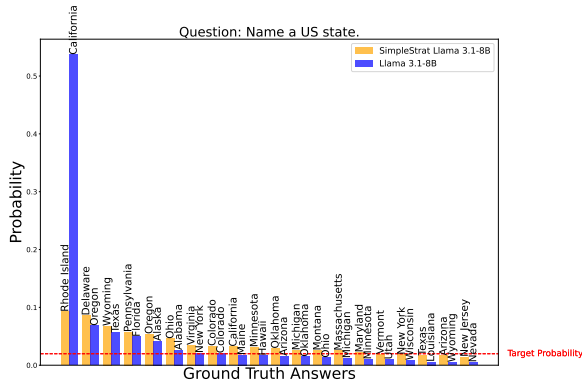
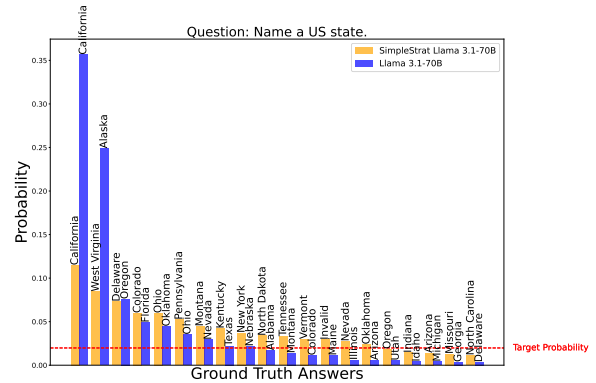
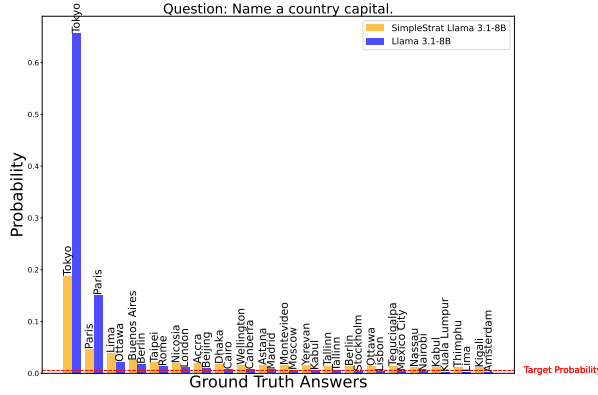
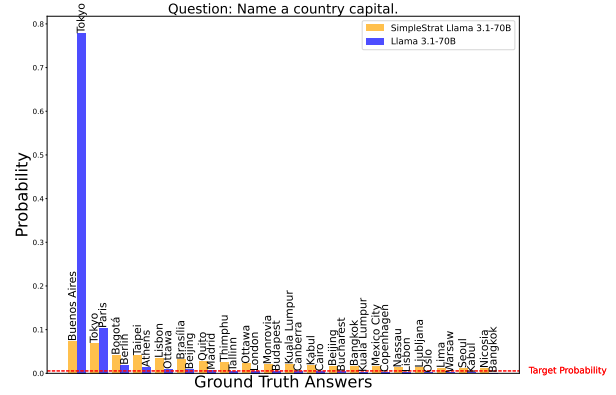
(a) **Llama 3.1 8B.**(b) **Llama 3.1 70B.**

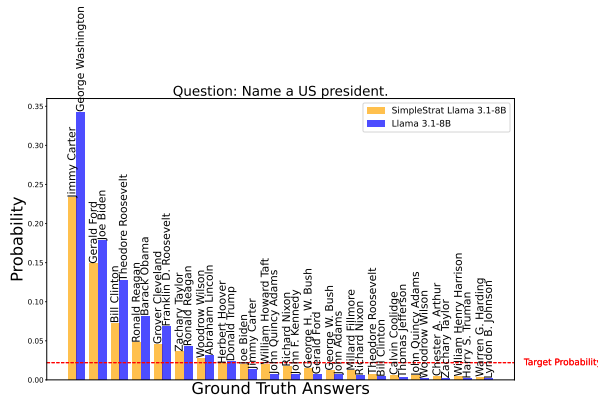
Figure 3.4: **Distributional Diversity Comparison.** We show the response probability as defined by next-token-probabilities for the top 20 ground truth answers on Llama 3.1. For both 8B and 70B, SimpleStrat provides meaningful improvement to the response distribution both for values previously over-represented in the distribution and for those previously underrepresented.



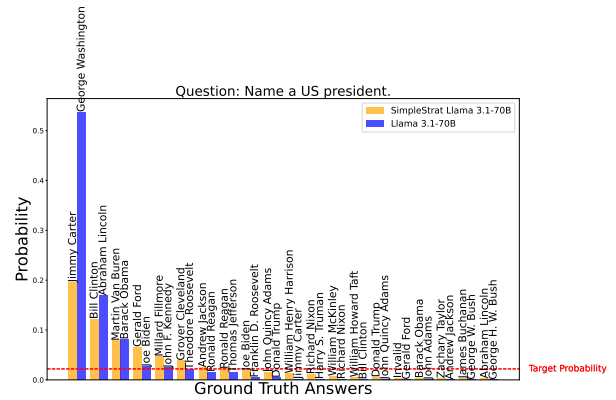
(a) Llama 3.1 8B



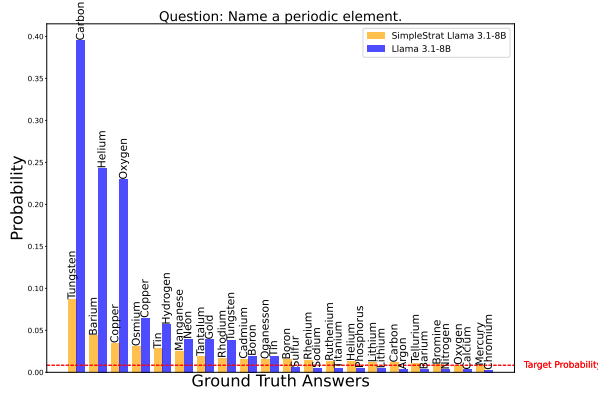
(b) Llama 3.1 70B



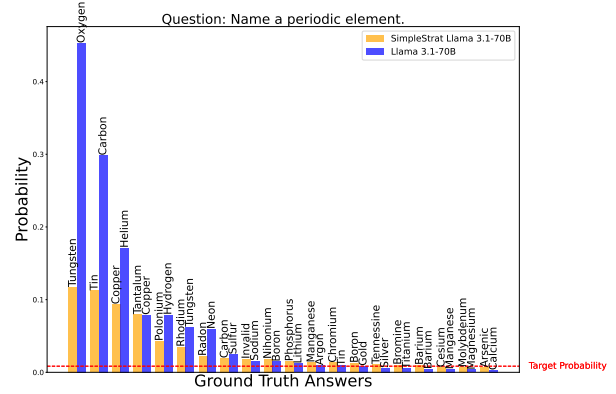
(c) Llama 3.1 8B



(d) Llama 3.1 70B



(e) Llama 3.1 8B



(f) Llama 3.1 70B

Figure 3.5: Baseline vs SimpleStrat Probability Distributions This figure shows the answer distributions for 4 additional questions from CoverageQA curated. Each row represents a different question, showing distributions for Llama 3.1 8B and 70B.

3.8 Coverage Diversity on Proprietary Models

We first assess coverage diversity, specifically, the model’s ability to recall all the valid solutions upon resampling. This measure is clearly impacted by temperature as temperature zero or greedy decoding of LLMs leads to a single deterministic result. We compare the coverage diversity (recall) of SimpleStrat, GPT-4o, and Claude 3.5 Sonnet as a function of temperature. We sweep over temperatures from 0 to 1.5. SimpleStrat with GPT-4o leads to an improvement in recall across all temperatures as shown in Fig. 3.6. Compared to prior work such as AttrPrompt, SimpleStrat provides substantially more diversity when applied to the same underlying model (Y. Yu et al., 2023a). SimpleStrat also scales well with temperature, showing gains across the full sweep. These recall improvements do not appear to come at the expense of quality.

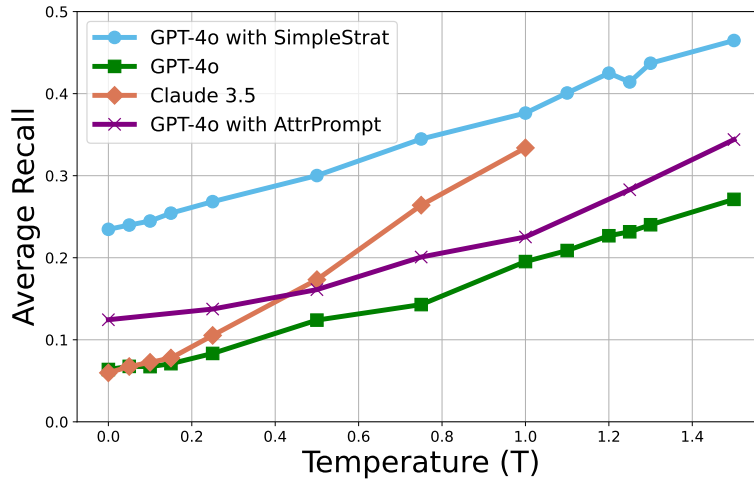


Figure 3.6: **Diversity measured with recall scaled with temperature.** SimpleStrat applied to GPT-4o shows improved recall on CoverageQA compared to GPT-4o, AttrPrompt, and Claude 3.5 Sonnet. Recall indicates the percentage of ground truth answers observed after sampling 100 times. The benefit of SimpleStrat is especially pronounced at low temperatures, but remains visible across all temperatures. Prior work, AttrPrompt, directly asks for partitions instead of using the 20-questions-style prompting used here, resulting in lower-quality partitions.

Precision. Fig. 3.7 shows that these gains do not come at a substantial cost to precision. Precision is calculated over the set of 100 attempts as the fraction that are in the ground truth. Recall, as mentioned earlier, is calculated as how many unique ground-truth solutions were observed in the 100 attempts. The modest drop in precision can be attributed to cases where heuristic estimation is ineffective. This can create strata with no valid solutions. Because the constraints are added as additional conditioning, the model then has a competing

Table 3.5: Performance of Different Prompting Strategies across Temperature Settings (GPT-4o)

Temp.	GPT-4o (std)	SimpleStrat (std)	20Q Abl. (std)	Single Prompt Abl.(std)
0	0.0646 (0.0011)	0.2423 (0.0050)	0.1215 (0.0021)	0.0019 (0.0008)
0.25	0.0849 (0.0016)	0.2669 (0.0022)	0.1405 (0.0028)	—
0.5	0.1204 (0.0036)	0.3017 (0.0043)	0.1660 (0.0025)	—
0.75	0.1497 (0.0046)	0.3433 (0.0067)	0.1821 (0.0336)	—
1.0	0.1871 (0.0071)	0.3884 (0.0094)	0.2381 (0.0049)	0.0272 (0.0014)
1.25	0.2280 (0.0053)	0.4250 (0.0062)	0.2872 (0.0065)	—
1.5	0.2676 (0.0059)	0.4634 (0.0085)	0.3304 (0.0104)	—

objective: obey the constraints versus obey the original instruction. As such, infeasible requests lead to best-effort solutions that are incorrect.

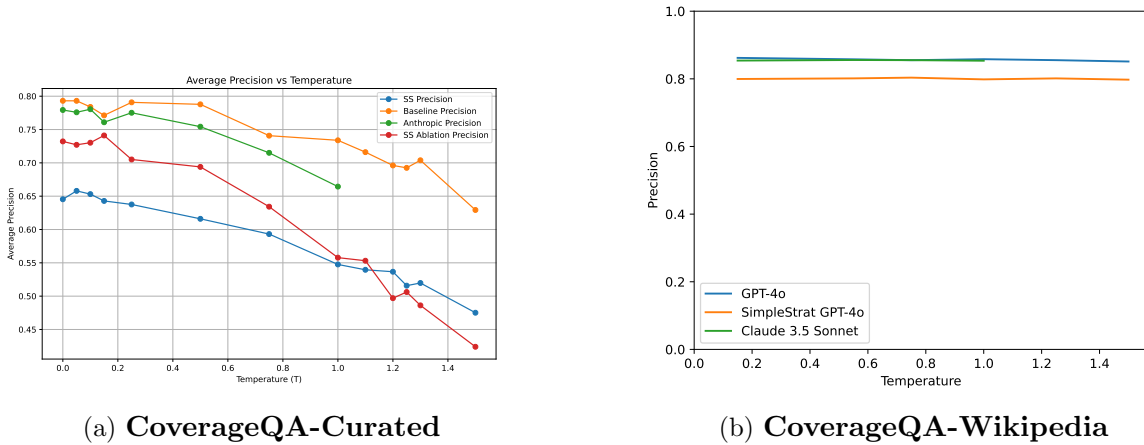


Figure 3.7: **Precision scaled with temperature.** There is a fixed minor reduction of 10% to precision when using SimpleStrat. This indicates the improved diversity does not come at a significant cost to precision.

We also manually curate questions with known solution sets such as NFL teams and U.S. state capitals. This bank serves as a split to validate results on less synthetic sources that are more common knowledge. We see even stronger performance improvement on this set in Fig. 3.8.

Ablations. We conduct two ablations to evaluate key design choices. First, we remove the **20 Questions** framing (**20Q Abl.**) and directly ask the model for partitions. Second, we test a **single-stage** version (**Single Prompt Abl.**) that combines all instructions into one prompt using the 20 Questions framing. Results in **Table 3.5** show that the single-prompt setup performs significantly worse than the baseline GPT-4o, suggesting that more tokens do not necessarily improve diversity and that our gains are not due to prompt complexity alone. This supports the intuition that probabilistic prompting—via “coin flips”—introduces ben-

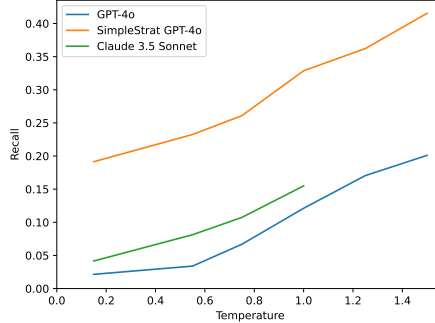


Figure 3.8: We see even stronger improvements on the manually curated CoverageQA set.

eficial randomness. The **20 Questions ablation** further confirms our 20 question framing provides better partitions.

3.9 Distributional Diversity with Llama 3

We use Llama 3 models to measure distributional diversity by analyzing logits for all valid answers. This is not feasible with GPT-4o or Claude 3.5 Sonnet, since estimating true probabilities would extensive heavy sampling. Although GPT models can output log probabilities, they report these only for the tokens in the observed generation trace, rather than for all possible continuations.

Across both 8B and 70B Llama models, SimpleStrat achieves an average reduction in KL divergence from uniform of 1.14 compared to the baseline on the curated CoverageQA dataset. For the general CoverageQA dataset, the reduction is 0.36. These results indicate that SimpleStrat produces a response distribution closer to the ground truth distribution than the baseline method.

Additionally, we analyze per-question KL divergence with the scatter plot in Fig 3.9. It shows KL divergence values for SimpleStrat (y-axis) versus the baseline (x-axis) for each question in the CoverageQA Wikipedia dataset. Points above the diagonal line represent questions where SimpleStrat outperforms the baseline by yielding a lower KL divergence. Points tend to fall near or above this line, indicating SimpleStrat produces more uniform samples on CoverageQA.

As shown in Fig. 3.10, Llama 3.1’s base distribution is heavily biased toward its preferred answer, such as “California” in Fig. 3.1. It is therefore unsurprising that simply increasing temperature yields only limited diversity. In contrast, SimpleStrat produces a much more uniform distribution. Overrepresented solutions are pushed downward and underrepresented solutions are lifted upward.

Table 3.6: We assess distributional diversity as measured by KL Divergence. Smaller KL divergence is closer to uniform. We see improvement to distributional diversity for both 8B and 70B as well as Llama-3 and Llama-3.1.

Model	CoverageQA-Curated		CoverageQA-Wiki	
	Baseline	SimpleStrat	Baseline	SimpleStrat
Llama-3-8B-Instruct	2.78	1.74	2.75	2.47
Llama-3.1-8B-Instruct	2.47	1.19	2.60	2.39
Llama-3-70B-Instruct	3.24	2.17	3.28	2.73
Llama-3.1-70B-Instruct	2.70	1.54	2.78	2.38

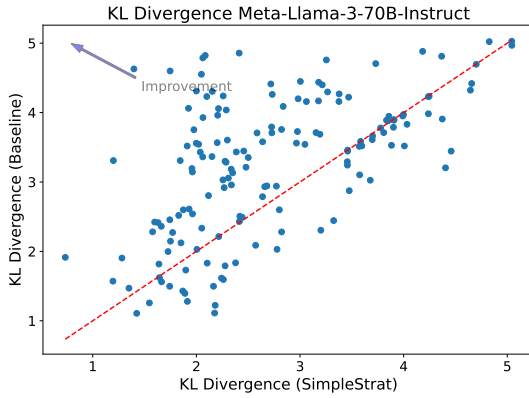


Figure 3.9: **KL divergence on individual question from CoverageQA Wikipedia.** For each question in CoverageQA, we compare the KL divergence from reference distribution of Llama-3 with and without SimpleStrat. Lower divergence indicates closer alignment with the desired uniform distribution, arrow indicates direction of maximum improvement from baseline

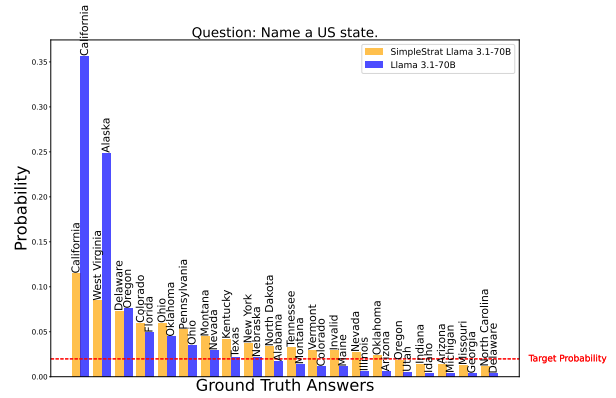


Figure 3.10: **Distributional Diversity Comparison.** We show the a posteriori probabilities as defined by next-token probabilities for ground truth answers on Llama 3.1 70B. SimpleStrat provides meaningful improvement to the sampling distribution both for values previously overrepresented in the distribution and those previously underrepresented.

We also provide scatter plots for Llama 3 in Fig. 3.11a, Fig. 3.11b, and Fig. 3.12.

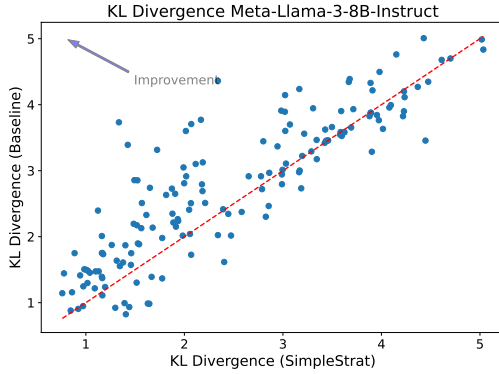
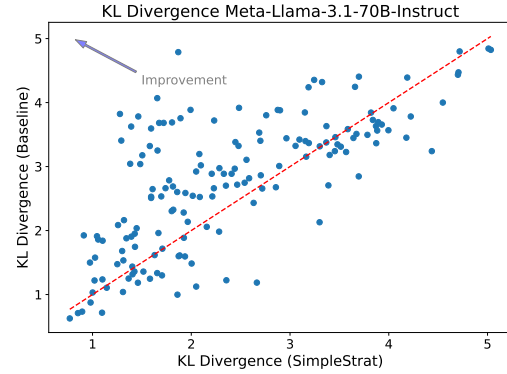
(a) **Llama 3.1 8B**(b) **Llama 3.1 70B**

Figure 3.11: KL divergence from uniform for Baseline vs SimpleStrat on CoverageQA Wikipedia. Lower divergence indicates closer alignment with the desired uniform distribution, arrow indicates direction of maximum improvement from baseline

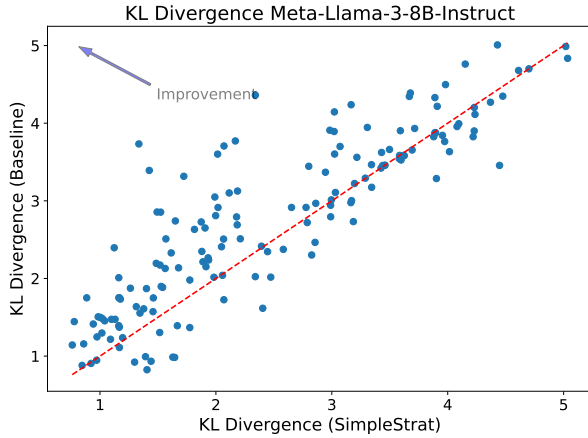
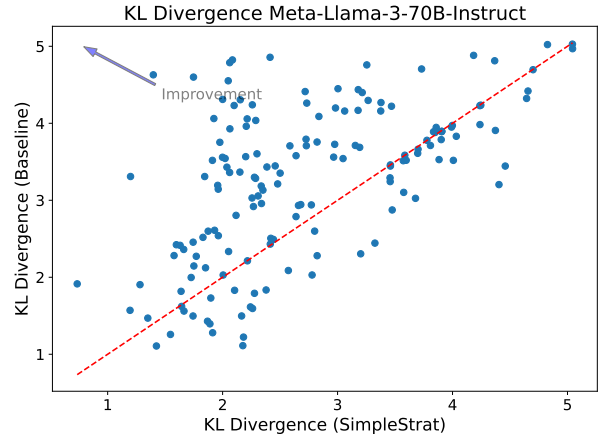
(a) **Llama 3 8B**(b) **Llama 3 70B**

Figure 3.12: KL divergence from uniform for Baseline vs SimpleStrat on CoverageQA Wikipedia. Additional plots for Llama 3 8B and 70B models

3.10 Open-Ended Diversity

Fig. 3.13 shows SimpleStrat shifts the curve to the right, affording diversity while maintaining the same quality. We provide a coarse grain measure of diversity by measuring the proportion of plots from the 100 random prompts taken from WritingPrompts that do not have formatting errors or more than 20% of words outside of the English dictionary. As we are asking for an outline, format errors we check for include proper monotonic numbering and providing the number of requested chapters to the story. Notably, at temperature zero,

we achieve the same diversity as temperature scaling to temperature 1. This suggests we get diversity for free without sacrificing quality.

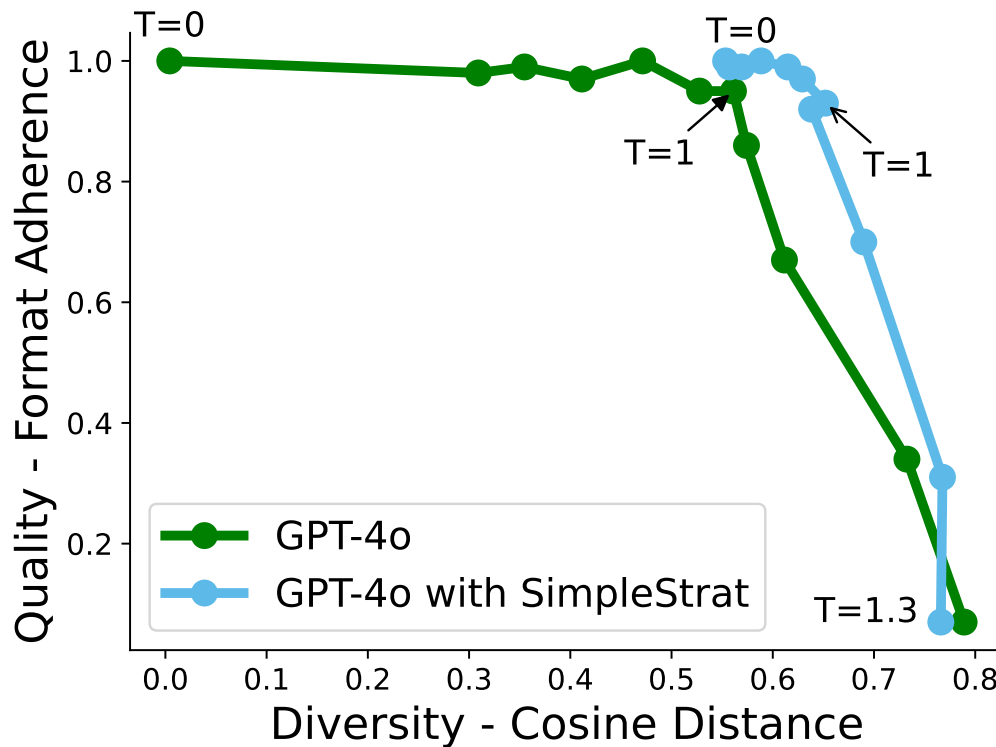


Figure 3.13: **WritingPrompts Diversity.** On creative writing prompts, we generate pairs of plot outlines. We measure diversity with embedding cosine distance and quality by checking for format adherence. SimpleStrat especially improves the diversity at low temperatures achieving at $T=0$ the same diversity as $T=1$ for base GPT-4o.

Judging Plot Outlines. To judge plot outlines, we ask the model to generate outlines guided by the format shown in Table 3.7. Notably, we ask for three acts to prevent the outlines from getting too long. To check for validity, we use `pyspellcheck` to assess whether there are over 20% words not in the English dictionary and ensure that there are three acts and a `THE END` to finish the story. The last condition protects against the case where the model rambles on incoherently and produces what is definitely not a sensible outline. With spelling alone, we can already see the degradation of the model due to temperature (Fig. 3.14). Finally, we see formatting is not affected by SimpleStrat and largely tracks the temperature used to sample from the language model (Fig. 3.15).

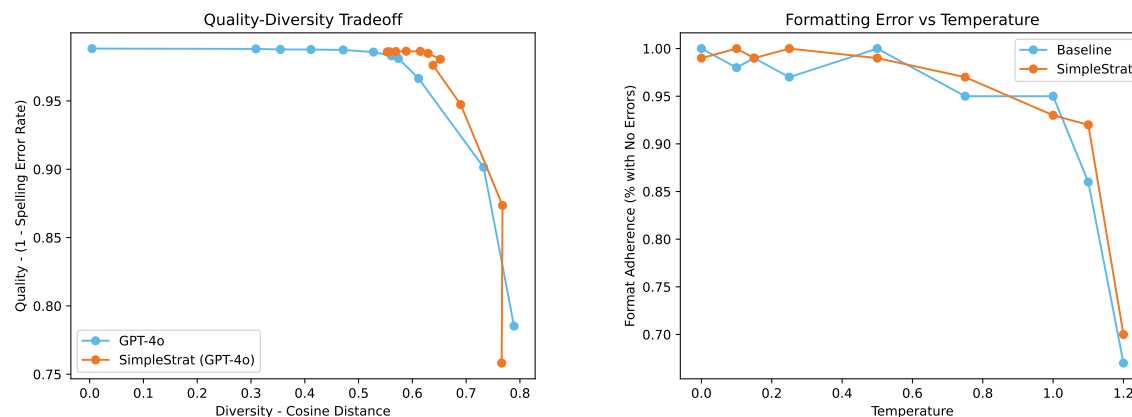


Figure 3.14: **WritingPrompts Diversity**. On Figure 3.15: **Impact of temperature on for-**
creative writing prompts, we generate pairs of **matting (including spelling)**. It’s interest-
plot outlines. We measure diversity with em-
bedding cosine distance and quality proxied by ligible compared to temperature. We see that
spelling. SimpleStrat especially improves the SimpleStrat closely follows the expected qual-
diversity at low temperatures achieving at T=0 ity degradation based on the temperature of the
the same diversity as T=1 for base GPT-4o. model generating the final outline.

System Prompt:

Format:

****Title**:** <TITLE>

****Setting**:** <SETTING>

****Characters**:** <CHARACTERS>

****Act 1**:** <ACT 1 TITLE>

1. <Content>

2. <Content>

3. <Content>

...

****Act 2**:** <ACT 2 TITLE>

1. <Content>

2. <Content>

3. <Content>

...

****Act 3**:** <ACT 3 TITLE>

1. <Content>

2. <Content>

3. <Content>

...

THE END

User:

Write a 3 part story outline based on the following prompt:

“*User Request*”

Table 3.7: Prompt for plot outlines.

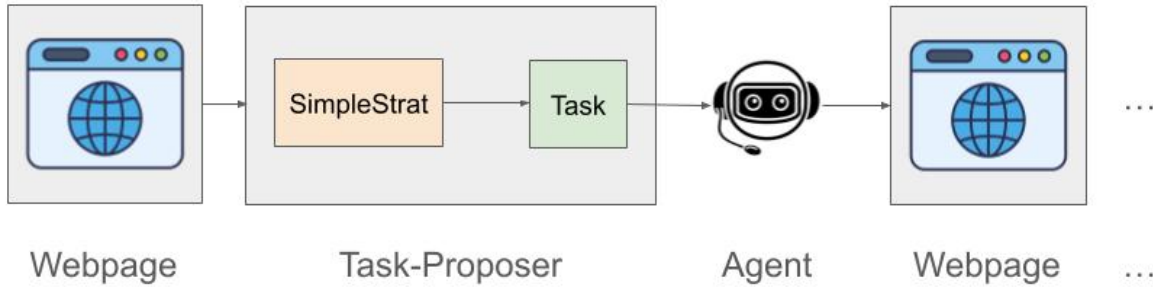


Figure 3.16: SimpleStrat task proposing. We use SimpleStrat to generate plausible and grounded tasks for training browser web-agents. Starting with a randomly selected landing page, we ask SimpleStrat to propose a challenging goal. Then we deploy the agent under training to accomplish that task. Upon completing the task, we ask SimpleStrat to propose a new goal. A task is created by summarizing the full trajectory.

3.11 SimpleStrat in Production – Task Proposing

We implement a task-proposing with SimpleStrat in combination InSTA and Explorer. Before development of the SimpleStrat task-proposing agent, the existing task proposer was trained by fine-tuning a vision language model based on human proposed tasks. This approach was an effective starting point; however, the model tends to propose canonical tasks given a website. For instance, purchasing coffee makers is a canonical task on amazon marketplace and booking an uber to the SFO airport is a canonical Uber task. Since we then train on this task distribution, the model eventually saturates on "common" task. As shown in Figure 3.17, this approach could only produce 20% of tasks not solvable by the model in training. Naturally, this is a good setting to deploy SimpleStrat.

As shown in 3.16, we start with a webpage chosen from a curated list of landing page. In this setting, we formulate a SimpleStrat prompt asking the model to pose Yes/No questions to guess a hidden *challenging* goal that effectively tests the model capability. Simply by labeling this hypothetical goal as challenging, the Yes/No questions are now partitions of challenging goals (as the model understands it). By sampling goals with SimpleStrat, we can revisit the same site and generate unique/diverse tasks that cover the space of challenging task. For the examples above, we find tasks like find me the cheapest shampoo on sale or rebook an uber to my most recent destination. Critical for making SimpleStrat practical, we are able to cache the stratification based on the url. This allows for efficiently generating a large set of tasks per website. The proposer now proposes a wider distribution of tasks but has a critical limitation. The proposed tasks are ungrounded. There may not be shampoo on sale or recent destinations to pick from.

To address this, we combine SimpleStrat’s diversity with InSTA/Explorer which propose

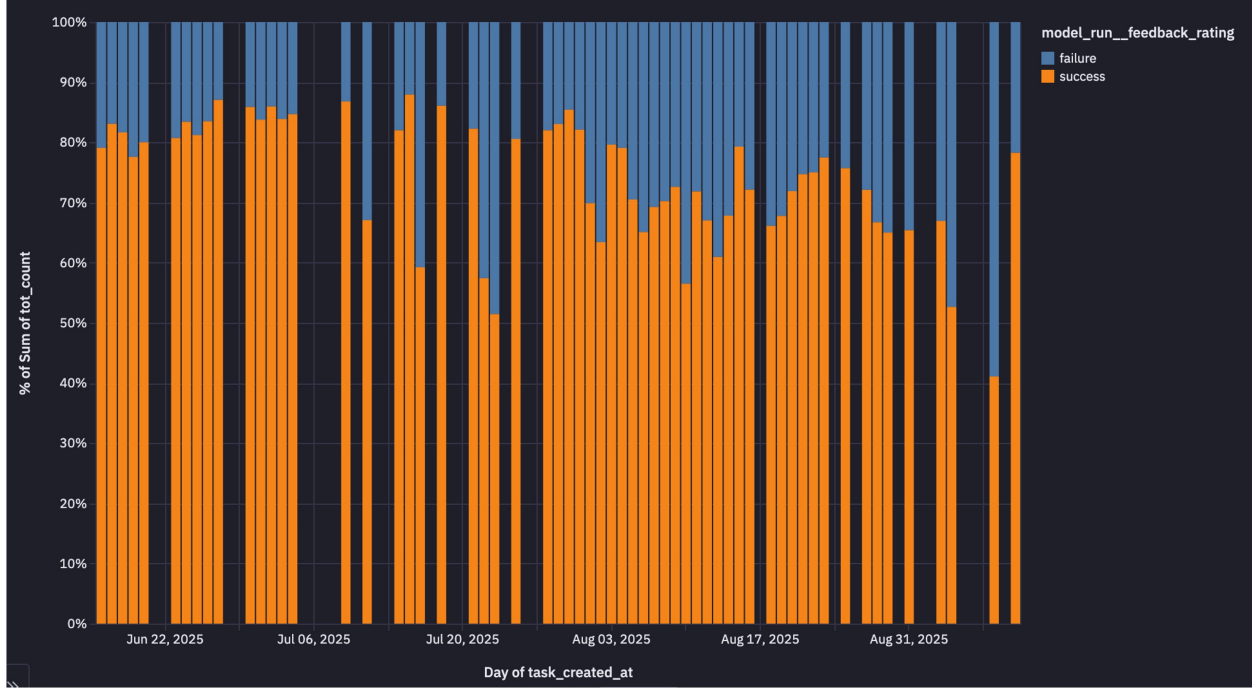


Figure 3.17: **SimpleStrat proposes harder tasks.** We show on the y-axis the percent of newly generated tasks that are solved with the NovaAct browser agent. Before we deployed the SimpleStrat task proposer, we saw only 20% of tasks are hard for our agent. We progressively roll out the SimpleStrat task proposer from July 6th to September 12th. By the end, we get as much as 60% hard tasks.

to run the agent being trained on the environment itself. By running the agent itself to accomplish a series of tasks, we can collect an exploratory web trajectory. Using this trajectory, we can synthesize a grounded task based on what exists on the webpage. Additionally, longer tasks tend to be more challenging and more rounds of SimpleStrat partitioning lead to more diversity. Figure 3.17 shows a progressive roll out of SimpleStrat to the synthetic task generation pipeline. By the time it is fully roll out, only 40% of tasks are solvable by the existing agent. With subsequent DAGGER-style intervention to correct failed trajectories, the agent fine-tuned on the new data performs 5% better on internal benchmarks than on data with the existing task proposer (see Figure 3.18). Our experience underscores how straightforward yet widely applicable SimpleStrat is.

3.12 Discussion

While SimpleStrat demonstrates empirical gains, its effectiveness depends on the model’s ability to identify meaningful axes in auto-stratification and estimate accurate joint probabilities. As LLMs improve in forecasting and external data integration, we expect these

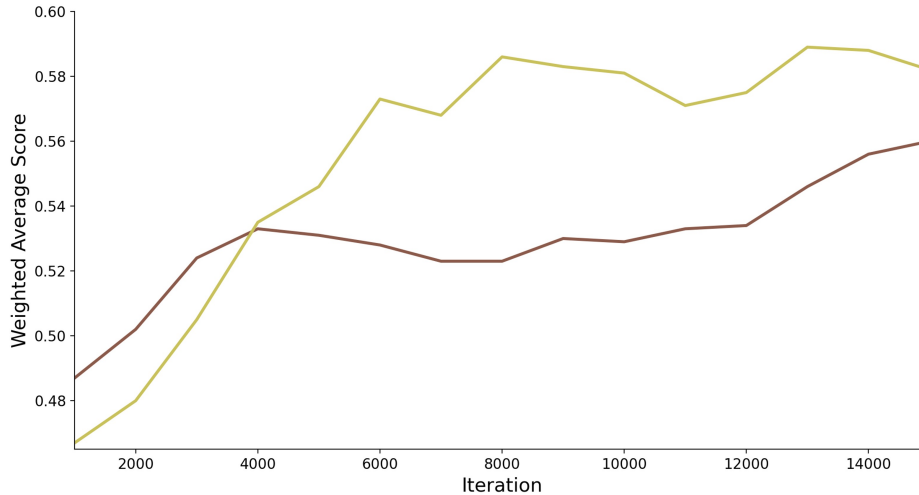


Figure 3.18: Training on the tasks created from SimpleStrat task proposer leads to better SFT performance (shown in yellow). There is a 5% improvement on internal benchmarks compared to previous data-collection approach (shown in brown).

estimates to become more reliable. Our prototype focuses on the model’s intrinsic capabilities, but potential biases—such as those related to race or gender—may influence stratification and estimation. For critical applications, the probabilistic prompt distribution should therefore be carefully reviewed. Finally, because CoverageQA consists of short responses, evaluation is simplified; however, we anticipate SimpleStrat will have the greatest impact in low-temperature, multi-step reasoning tasks (E. Zhang et al., 2024).

As research on learning beyond demonstrations and reinforcement learning accelerates, methods that promote diversity—such as SimpleStrat—are poised to become central to discovering novel solutions, strategies, and ideas. Wherever diversity is currently achieved through temperature scaling, SimpleStrat provides a more semantically grounded alternative.

3.13 Conclusion

This chapter introduced SimpleStrat, which leverages the language model itself to partition the solution space into distinct strata. We call this process *auto-stratification*. Specifically, the stratification problem is reframed as the imperfect-information game of *20 questions*, yielding strata that are both balanced and orthogonal. At inference time, a random stratum is selected and a sample is drawn from within that stratum. This achieves greater diversity while maintaining quality, unlike simply increasing temperature.

To quantitatively measure diversity, we introduced the CoverageQA dataset, which consists of underspecified questions with multiple equally valid answers. We measure diver-

sity with three metrics: for open-source models, we measure distributional difference with KL Divergence and for proprietary models, we measure coverage over the set of ground-truth solutions. In the open-ended setting without access to the ground truth distribution, we rely on distance in embedding space to measure diversity. Our rigorous evaluation on both proprietary and open-source LLMs demonstrated that SimpleStrat achieves significantly higher recall and produces answer distributions closer to uniform compared to traditional temperature-based sampling methods.

Chapter 4

Stylus: Automatic Adapter Selection for Diffusion Models

In the following chapter, we look at another approach to navigating the diversity-quality tradeoff. Similar to SimpleStrat described in the last chapter, we appeal to exploiting a model’s world knowledge about the structure and intent of each specific problem instance. Stylus leverages adapter merging at a model weight level to produce higher-quality images while accessing meaningful diversity by randomizing subsets of adapters for merging. We’ll introduce the problem Stylus sets out to solve and the full system. Then we’ll highlight how Stylus achieves an improved quality-diversity tradeoff at the weight level.

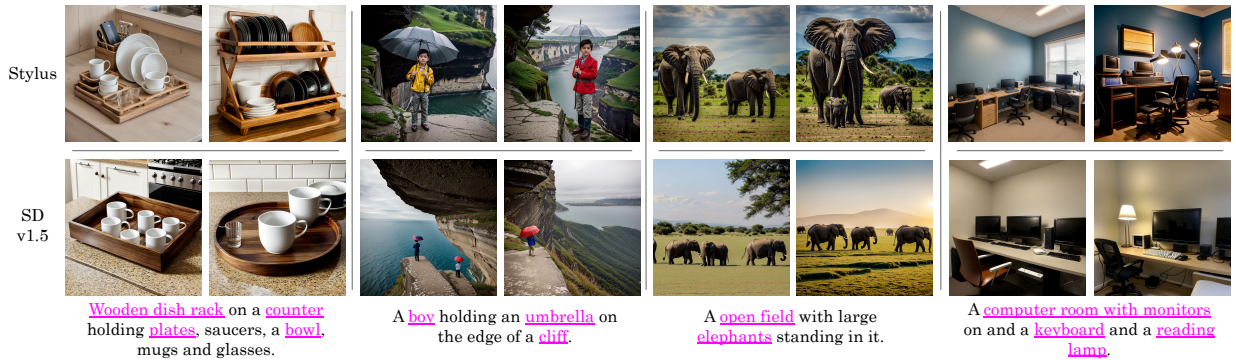


Figure 4.1: **Adapter Selection.** Given a user-provided prompt, our method identifies highly relevant adapters (e.g. Low-Rank Adaptation, LoRA) that are closely aligned with the prompt’s context and at least one of the prompt’s keywords. Composing relevant adapters into Stable Diffusion improves visual fidelity, image diversity, and textual alignment. Note that these prompts are sampled from MS-COCO (T.-Y. Lin et al., 2015).

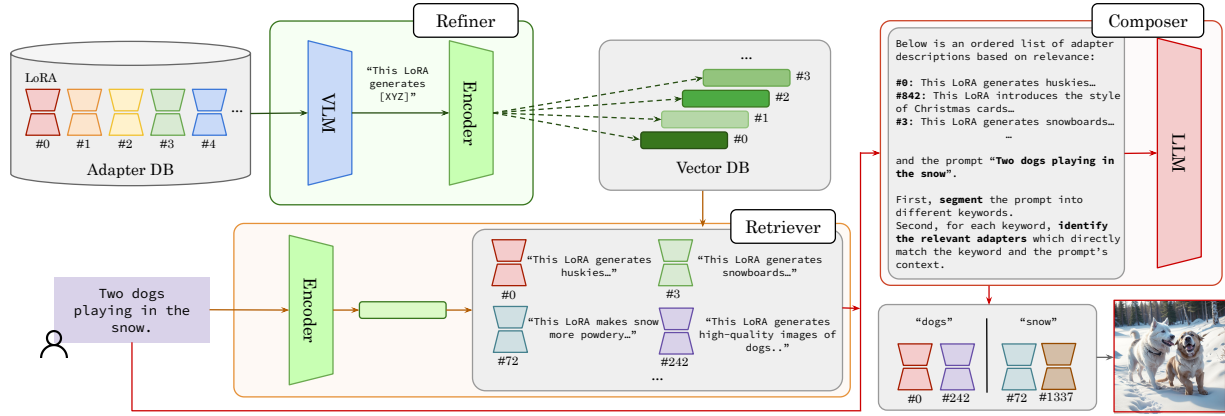


Figure 4.2: **Stylus algorithm.** Stylus consists of three stages. The *refiner* plugs an adapter’s model card through a VLM to generate textual descriptions of an adapter’s task and then through an encoder to produce the corresponding text embedding. The *retriever* fetches candidate adapters that are relevant to the entire user prompt. Finally, the *composer* prunes and jointly categorizes the remaining adapters based on the prompt’s tasks, which corresponds to a set of keywords.

4.1 Overview

In the evolving field of generative image models, finetuned adapters (Ha et al., 2016; Gal et al., 2022) have become the standard, enabling custom image creation with reduced storage requirements. This shift has spurred the growth of extensive open-source platforms that encourage communities to develop and share different adapters and model checkpoints, fueling the proliferation of creative AI art (Wolf et al., 2020; Maier, 2022). As the ecosystem expands, the number of adapters has grown to over 100K, with Low-Rank Adaptation (LoRA) (Hu et al., 2021) emerging as the dominant fine-tuning approach (see Fig. 4.3). A new paradigm has emerged where users manually select and creatively compose multiple adapters, on top of existing checkpoints, to generate high-fidelity images, moving beyond the standard approach of improving model class or scale.

In light of performance gains, our paper explores the automatic selection of adapters based on user-provided prompts (see Fig. 4.1). However, selecting relevant adapters presents unique challenges compared to existing retrieval-based systems, which rank relevant texts via lookup embeddings (Lewis et al., 2021). Specifically, efficiently retrieving adapters requires converting adapters into lookup embeddings, a step made difficult with low-quality documentation or no direct access to training data—a common issue on open-source platforms. Furthermore, in the context of image generation, user prompts often imply multiple highly-specific tasks. For instance, the prompt “two dogs playing in the snow” suggests that there are two tasks: generating images of “dogs” and “snow”. This necessitates segmenting the prompt into various tasks (i.e. keywords) and selecting relevant adapters for each task, a requirement beyond the scope of existing retrieval-based systems (Gao et al., 2024). Finally, composing multiple adapters can degrade image quality, override existing concepts,

and introduce unwanted biases into the model (see Sec. 4.11).

We propose Stylus, a system that efficiently assesses user prompts to retrieve and compose sets of highly-relevant adapters, automatically augmenting generative models to produce diverse sets of high quality images. Stylus employs a three-stage framework to address the above challenges. As shown in Fig. 4.2, the *refiner* plugs in an adapter’s model card, including generated images and prompts, through a multi-modal vision-language model (VLM) and a text encoder to pre-compute concise adapter descriptions as lookup embeddings. Similar to prior retrieval methods (Lewis et al., 2021), the *retriever* scores the relevance of each embedding against the user’s entire prompt to retrieve a set of candidate adapters. Finally, the *composer* segments the prompt into disjoint tasks, further prunes irrelevant candidate adapters, and assigns the remaining adapters to each task. We show that the composer identifies highly-relevant adapters and avoids conceptually-similar adapters that introduce biases detrimental to image generation (§ 4.9). Finally, Stylus applies a binary mask to control the number of adapters per task, ensuring high image diversity by using different adapters for each image and mitigating challenges with composing many adapters.

To evaluate our system, we introduce **StylusDocs**, an adapter dataset consisting of 75K LoRAs¹, that contains pre-computed adapter documentations and embeddings from Stylus’s *refiner*. Our results demonstrate that Stylus improves visual fidelity, textual alignment, and image diversity over popular Stable Diffusion (SD 1.5) checkpoints—shifting the CLIP-FID Pareto curve towards greater efficiency and achieving up to 2x higher preference scores with humans and vision-language models (VLMs) as evaluators. As a system, Stylus is practical and does not present large overheads to the batch image generation process. Finally, Stylus can extend to different image-to-image application domains, such as image inpainting and translation.

4.2 Related Works

Adapters. Adapters efficiently fine-tune models on specific tasks with minimal parameter changes, reducing computational and storage requirements while maintaining similar performance to full fine-tuning (Hu et al., 2021; Gal et al., 2022; Ha et al., 2016).

Our study focuses on retrieving and merging multiple Low-Rank adapters (LoRA), the popular approach within existing open-source communities (Wolf et al., 2020; Maier, 2022; Mangrulkar et al., 2022).

Adapter composition has emerged as a crucial mechanism for enhancing the capabilities of foundational models across various applications (Touvron et al., 2023a; Touvron et al., 2023b; D. Li et al., 2024; Podell et al., 2023; Rombach et al., 2022). For large language models (LLM), the linear combination of multiple adapters improves in-domain performance and cross-task generalization (C. Huang et al., 2024; Jang et al., 2023; H. Wang et al., 2024; Xinyi Wang et al., 2021; Chronopoulou et al., 2023; Z. Zhao et al., 2024). In the image domain, merging LoRAs effectively composes different tasks—concepts, characters, poses,

¹Sourced from <https://civitai.com/> (Maier, 2022).

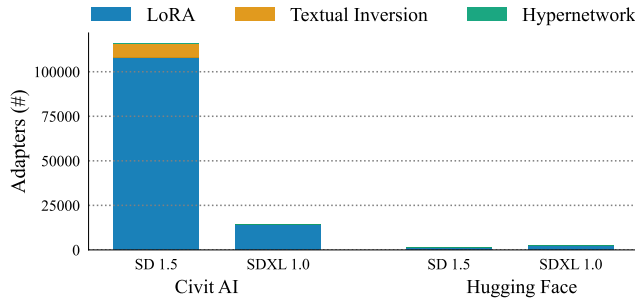


Figure 4.3: **Number of Adapters.** Civit AI boasts 100K+ adapters for Stable Diffusion, outpacing that of Hugging Face. Low-Rank Adaptation (LoRA) is the dominant approach for finetuning.

actions, and styles—together, yielding images of high fidelity that closely align with user specifications (M. Zhong et al., 2024; Nan Liu et al., 2023). Adapters also play a key role in synthetic data methods in few-shot computer vision (Trabucco et al., 2023a). Our approach advances this further by actively segmenting user prompts into distinct tasks and merging the appropriate adapters for each task.

Retrieval-based Methods. Retrieval-based methods, such as retrieval-augmented generation (RAG), significantly improve model responses by adding semantically similar texts from a vast external database (Lewis et al., 2021). These methods convert text to vector embeddings using text encoders, which are then ranked against a user prompt based on similarity metrics (Devlin et al., 2019; Gao et al., 2024; Reimers et al., 2019; Ma, 2023; J. Lin et al., 2023; Radford et al., 2021). Similarly, our work draws inspiration from RAG to encode adapters as vector embeddings: leveraging vision-language foundational models (VLM) to generate semantic descriptions of adapters, which are then translated into embeddings.

A core limitation to RAG is limited precision, retrieving semi-relevant documents that do not exactly answer the prompt. This leads to a “needle-in-the-haystack” problem, where more relevant documents are buried further down the list (Gao et al., 2024). Recent work introduces a *reranking* step; this technique uses cross-encoders to assess both the raw user prompt and the ranked set of raw texts individually, thereby discovering texts based on actual relevance (Reimers, 2023; Ma, 2023). Rerankers have been successfully integrated with various LLM-application frameworks (Jerry Liu, 2022; Chase, 2022; Pietsch et al., 2019).

4.3 Our Method: Stylus

Adapter selection presents three distinct challenges compared to existing methods for retrieving text documents, as outlined in Section 4.2. First, computing embeddings for adapters is a novel task, made more difficult without access to training datasets. Furthermore, in the

context of image generation, user prompts often specify multiple highly fine-grained tasks. This challenge extends beyond retrieving relevant adapters relative to the entire user prompt, but also matching them with specific tasks within the prompt. Finally, composing multiple adapters can degrade image quality and inject foreign biases into the model. Our three-stage framework below—**R**efine, **R**etrieve, and **C**ompose—addresses the above challenges (Fig. 4.2).

Refiner

The *refiner* is a two-stage pipeline designed to generate textual descriptions of an adapter’s task and the corresponding text embeddings for retrieval purposes. This approach is analogous to pre-computed embeddings over an external database of texts in retrieval-based methods (Lewis et al., 2021).

Given an adapter A_i , the first stage is a vision-language model (VLM) that takes in the adapter’s model card—a set of randomly sampled example images from the model card $\mathcal{I}_i \in \{I_{i1}, I_{i2}, \dots\}$, the corresponding prompts $\mathcal{P}_i \in \{p_{i1}, p_{i2}, \dots\}$, and an author-provided description,² D_i —and returns an improved description D_i^* . Optionally, the VLM also recommends the weight for LoRA-based adapters, as the adapter weight is usually specified either in the author’s description D_i or the set of prompts P_i , a feature present in popular image generation software (AUTOMATIC1111, 2022). We denote this weight/coefficient as α_i . If information cannot be found, the LoRA’s weight is set to $\alpha_i = 0.8$. In our experiments, these improved descriptions were generated by Gemini Ultra (G. Team, 2023) (see Sec. 4.3 for the prompt). We chose the Gemini class of models since it has mature safety guardrailings. Specifically, Google’s VertexAI API provides stringent safety settings to block explicit content for the input prompt. Safety filters helped us filter out around 30% of original adapters that were tagged as non-explicit by other model repositories.

The second stage uses an embedding model ($\mathcal{E}$) to embed the text description D_i^* for each adapters to yield embeddings, $e_i = \mathcal{E}(D_i^*)$. In our experiments, we create embeddings from OpenAI’s `text-embedding-3-large` model (J. Lin et al., 2023; Meng et al., 2024). We store pre-computed embeddings in a vector database, formally notated by the matrix, V .

Retriever

The *retriever* fetches the most relevant adapters over the entirety of the user’s prompt using cosine similarity. Precisely, the retriever employs the same embedding model ($\mathcal{E}$) to process the user prompt, q , generating embedding $e_q = \mathcal{E}(q)$. Using the vector database, we calculate exact cosine similarity scores between the prompt’s embedding e_s and the embedding of each adapter in the matrix V . The similarity vector, $s_q = \frac{e_q^T V}{|e_q||V|}$, scores the adapter descriptions by similarity. The retriever simply returns indices of the top-k adapters $\mathcal{A}_k = \text{top-k}(s_q)$.

²We note that a large set of author descriptions are inaccurate, misleading, or absent. The *refiner* helped correct for human errors by using generated images as the ground truth, significantly improving our system.



Figure 4.4: **Qualitative comparison between Stylus over realistic (left) and cartoon (right) style Stable Diffusion checkpoints.** Stylus produces highly detailed images that correctly depicts keywords in the context of the prompt. For the prompt “A graffiti of a corgi on the wall”, our method correctly depicts a spray-painted corgi, whereas the checkpoint generates a realistic dog.

In our experiments, we find $k = 150$ is effective for StylusDocs. We denote the set of k descriptions of the adapters, $\mathcal{A}_k$ as D_k^* .

Composer

The *composer* serves a dual purpose: segmenting the prompt into tasks from a prompt’s keywords and assigning retrieved adapters to tasks. This implicitly filters out adapters that are not semantically aligned with the prompt and detects those likely to introduce foreign bias to the prompt through keyword grounding. For example, if the prompt is “pandas eating bamboo”, the composer may discard an irrelevant “grizzly bears” adapter and a biased “panda mascots” adapter.

The composer ($\mathcal{C}$) is a function of the prompt (q), the top K adapters ($\mathcal{A}_K$) from the retriever. Formally, denote the tasks identified by the composer as $\mathcal{T}(q) = \{t_1, t_2, \dots, t_n\}$. The composer produces a mapping from task to adapters:

$$\mathcal{C}(s, \mathcal{A}_K) = \{(t_i, \mathcal{A}_{k_i}) \mid t_i \in \mathcal{T}(q), \mathcal{A}_{k_i} \subseteq \mathcal{A}_K, \forall j \in \mathcal{A}_{k_i}, \text{Align}(\mathcal{A}_j, t_i)\} \quad (4.1)$$

where $\mathcal{A}_{k_i}$ is the subset of adapters per task t_i , $\text{Align}(\mathcal{A}_j, t_i)$ is a predicate that holds if the adapter, $\mathcal{A}_j$ is aligned with the task, t_i .

While the composer can be further improved by fine-tuning with human-labeled data (Ouyang et al., 2022b), we find that prompting a long-context Large Language Model (LLM) suffices. The LLM accepts the adapter descriptions and the prompt as part of its context and returns a mapping of tasks to a curated set of adapters. In practice, the alignment function is determined in the LLM’s chain-of-thought procedure before it outputs the final mapping of adapters to tasks. In our implementation, we choose Gemini 1.5, with a 128K context window, as the composer’s LLM (see Sec. 4.3 for the full prompt).

Stylus’s composer is similar to *reranking*. Rerankers employ cross encoders ($\mathcal{F}$) that compare the retriever’s individual adapter descriptions, generated from the refiner, against the

user prompt to determine better similarity scores: $\mathcal{F}(p, D^*)$. This prunes for adapters based on semantic relevance, thereby improving search quality, but not over keyword alignment. Our experimental ablations (§ 4.9) show that our composer outperforms existing rerankers (Cohere, `rerank-english-v2.0`) (Reimers, 2023).

Masking

The composer maps tasks to corresponding sets of highly relevant adapters. To further mitigate sensitivity to low-quality adapters, Stylus reduces the number of selected adapters with a straightforward masking scheme. Specifically, for each task, candidate masks are generated, and one is randomly selected to be applied over the set of adapters. Formally, for a given task, $m_i \in \{0, 1\}^{|\mathcal{A}_k|}$, is either a one hot encoding, $\vec{1}$, or $\vec{0}$, forming a set of possible masks, M_i . Across all tasks, masks are combined by taking the cross-product, $G = M_1 \times M_2 \times \dots \times M_n$. The combinatorial sets of masking schemes enable diverse linear combinations of adapters for a single prompt, leading to highly-diverse images (§ 4.8). This approach also curtails the number of final adapters merged into the base model, minimizing the risk of composing low-quality adapters that may introduce undesirable effects to the image (M. Zhong et al., 2024).

Merging

Stylus employs two key insights for effectively merging adapter weights. First, when applied to a single task, large adapter weights can introduce notable visual artifacts, such as over-saturation (Fig. 4.14a). Second, across multiple tasks, adapters tend to be orthogonal in the weight space, as they are designed to modify distinct, orthogonal concepts (Gandikota et al., 2023). Hence, Stylus computes the final adapter weights by *averaging* weights per task and *summing* weights across tasks. This approach ensures that the adapter weights per task remain appropriately scaled.

We mathematically illustrate our merging scheme below. Recall, the refiner outputs α_i , the recommended weight/coefficient, for each adapter. (§ 4.3). As shown in recent work (M. Zhong et al., 2024), multiple LoRAs can be merged with the base model weights (W_{base}). We arrive at our final merged model weights by a summing the adapter weights normalized by task. For a mapping, $\{(t_1, \mathcal{A}_{k_1}), (t_2, \mathcal{A}_{k_2}), \dots (t_n, \mathcal{A}_{k_n})\}$, and $g = (m_1, m_2, \dots, m_n) \in G$, the final model weight is:

$$W' = W_{base} + \beta \cdot \sum_{i \leq n} \sum_{j \in x_i} \alpha_j \Delta_j / |x_i| \quad (4.2)$$

where $x_i = Mask(m_i, \mathcal{A}_i)$ and Δ_j is the LoRA’s weight. We set $\beta = 0.8$ to mitigate image saturation, where assigning high adapter weights to an individual task (or concept) leads to sharp decreases in image quality (see Sec. 4.11). For batch inference, Stylus returns images sorted by CLIP score.

Details of the Refiner VLM

We provide a complete example input to the refiner’s VLM in Tab. 4.1. The prompt utilizes Chain-of-Thought (CoT) prompting, which decomposes the VLM’s goal of producing better adapter descriptions into two steps (Wei et al., 2022; Yao et al., 2024). Initially, the VLM categorizes the adapter’s task into one of several topics, such as concepts, styles, characters, or poses. Subsequently, the VLM is prompted to elaborate on why the adapter is associated with a particular topic and how it modifies images within that context. We found that this two-step logical process significantly improved the structure and quality of model responses.

Details of the Composer LLM

We provide a full example prompt of the composer’s LLM component in Tab. 4.2, which is plugged through the Gemini 1.5 endpoint (G. Team, 2023). Our experiments feed in descriptions of the top 150 adapters into the LLM’s context. Using a Chain-of-Thought (CoT) approach, the prompt is structured to first identify keywords or tasks, then allocate appropriate adapters to these tasks. If necessary, it merges keywords for adapters that span multiple tasks (Wei et al., 2022; Yao et al., 2024).

StylusDocs Characterization

This section describes **StylusDocs**, which comprises 76K Low Rank Adapters (LoRAs) from public repositories, including Civit AI and Hugging Face (Maier, 2022; Wolf et al., 2020). We excluded NSFW-labeled adapters from the Civit AI dataset, which originally contained over 100K LoRAs. Figure 4.5 illustrates the distribution of adapters across various semantic categories and their popularity, measured by download counts. A significant majority, 70%, of adapters belong to the character and celebrity category, primarily consisting of anime or game characters. Another 13% of adapters modify image style, 8% adjust clothing, and 4% represent various concepts (Fig. 4.5a). These statistics indicate that our experiments consider a minor proportion of adapters, as the COCO dataset does not feature characters or celebrities (T.-Y. Lin et al., 2015). Despite this, Stylus outperforms base Stable Diffusion. Furthermore, the popularity of adapters follows a Pareto distribution, where the top adapters receive exponentially more downloads than the others (Fig. 4.5a). However, the top adapter accounts for only 0.5% of total downloads, which suggests that the distribution is long-tailed.

4.4 Evaluation Setup

Adapter Testbed. Adapter selection requires a large database of adapters to properly evaluate its performance. However, existing methods (C. Huang et al., 2024; Z. Zhao et al., 2024) only evaluate against 50-350 adapters for language-based tasks, which is insufficient for our use case, since image generation relies on highly fine grained tasks that span across many concepts, poses, styles, and characters. To bridge this gap, we introduce **StylusDocs**,



Image Prompts

Prompt 1: Photo of Dwayne Johnson, wearing military clothes and cap, dramatic lighting, `<lora:TheRockV3:0.9>`.

Prompt 2: Photo of Dwayne Johnson, wearing a Superman suit, high quality, `<lora:TheRockV3:1>`.

Prompt 3: Photo of Dwayne Johnson, wearing an Armani tuxedo, `<lora:TheRockV3:0.9>`

Model Card Description

- Title: Dwayne "The Rock" Johnson (LoRA)
 - Tags: Celebrity, Photorealistic, Hollywood, Celeb
 - Trigger Words: Th3R0ck
 - Description: Had to make this one, due to Kevin Hart Lora. Recommended lora strength: 0.9. *% Author descriptions may be misleading or incomplete.*
-

Your goal is to improve the description of a model adapter's task for Stable Diffusion, with images, prompts, and descriptions pulled from popular model repositories. Above, we have provided the following information and the associated constraints:

1. Examples of generated images (from left to right) from the adapter and the corresponding user-provided prompts.

- Some prompts may specify the adapter weight (i.e. `<lora:NAME:WEIGHT>`). If provided, infer the adapter's name and weight. Prioritize this weight over the author's recommended weight.
- 2. The adapter's model card from the original author. This includes the title, tags, trigger words, and description.
- The model card description may be incorrect, misleading, or incomplete.
- The model card may specify the weight of the model adapter, or the recommended range. Find the recommended weight of the adapter (default is 0.8).

% Chain-of-Thought Prompting

Again, your mission is to provide a clear description of the model's adapter purpose and its impact on the image. To do so, you should implicitly categorize the model adapter into only one of the following topics: [Concept, Style, Pose, Action, Celebrity/Character, Clothing, Background, Building, Vehicle, Animal, Action]. Do not associate an adapter with a topic that is vague or uninteresting.

First, describe the topic associated with the adapter and explain how this adapter alters the images, based on the common elements observed in the example images. Your requirements are:

- Do not describe any training or dataset-related details.
- Provide additional context from your prior knowledge if there is insufficient information.
- Do not hallucinate and repeat text. Output only english words and sentences.

Second, recommend an optimal weight for the adapter as a float. Do not specify a range, only give one value.

Please format your output as follows:

Example 1: *[Description of adapter and its weight]*

Example 2: *[Description of adapter and its weight]*

Table 4.1: Full prompt for the refiner VLM to generate better adapter descriptions.

Retrieved Adapter Descriptions
42: This LoRA is for the concept of dragon, a mythical creature. It generates images of dragons with a variety of different appearances, including both Western and Eastern styles... 120: This LoRA steers the image generation towards a fantasy castle, with a focus on the building and its surroundings. The castle is depicted as a grand structure, often with towering spires, intricate architecture, and a sense of grandeur... 3478: This LoRA is designed to generate images of a Chinese dragon breathing fire. It generates images of a dragon with a long, serpentine body, covered in scales, with a large head and sharp teeth. The dragon is breathing fire, with flames coming out of its mouth... 1337: This LoRA is designed to generate images of animals breathing fire. It generates images of animals, such as rabbits, dragons, and frogs, breathing fire. The fire is shown as a bright, orange-yellow flame that is coming out of the animal’s mouth... ...
Provided above are the IDs and descriptions for different model adapters (e.g. LoRA) for Stable Diffusion that may be related to the prompt. Your goal is to fetch adapters that can improve image fidelity. The prompt is: <i>Dragon breathing fire on a castle.</i> <i>% Chain-of-Thought Prompting</i> First, segment the prompt into different tasks: concepts, styles, poses, celebrities, backgrounds, objects, actions, or adjectives from the prompt’s keywords. Here are the requirements for tasks: • Tasks should never introduce new information to the prompt. The topic must be selected from the prompt’s keywords. • Different tasks must be orthogonal from each other. • All tasks combined must span the entirety of the prompt. • Prioritize choosing narrower tasks. You may merge tasks if a relevant adapter spans several tasks. Second, for each task, provide 0-5 of the most relevant model adapters to the task. For each adapter, infer an adapter’s main function from its description. This function must directly match at least one task and the context of the prompt. If the adapter is indirectly relevant, do not include it. Here are the requirements for adapters: • Adapters should only be used at most once across all tasks. If an adapter is used in one task, it should not be used in another task. • Adapters should not introduce novel concepts or biases to the topic or the prompt. Do not include such adapters. • Adapters cannot encompass a broader scope relative to its assigned task. For example, if the task is about a “dog”, the adapter cannot be about general “animals”. • Adapters cannot be too narrow in scope relative to its assigned task. For example, if a task is about pandas, do not choose highly specific pandas such as the character “Po” from Kung Fu Panda. However, it is acceptable to choose adapters that modify the style of the task, such as “Red Pandas”. • If an adapter spans multiple tasks, merge these tasks together. For example, if there is an adapter that is about “fluffy cats”, merge the topics “fluffy” and “cats” together. • Avoid choosing NSFW and anthropomorphic adapters. Finally, for each selected adapter, provide a strong reason for why this adapter is relevant to the prompt, directly matches the keyword, and improves image quality. Give me the answer only. Please format your output as follows: Example 1: [<i>Dictionary of tasks to the associated adapter ids and reasons for their selection.</i>] Example 2: [<i>Dictionary of tasks to the associated adapter ids and reasons for their selection.</i>]

Table 4.2: Full prompt for the composer LLM.

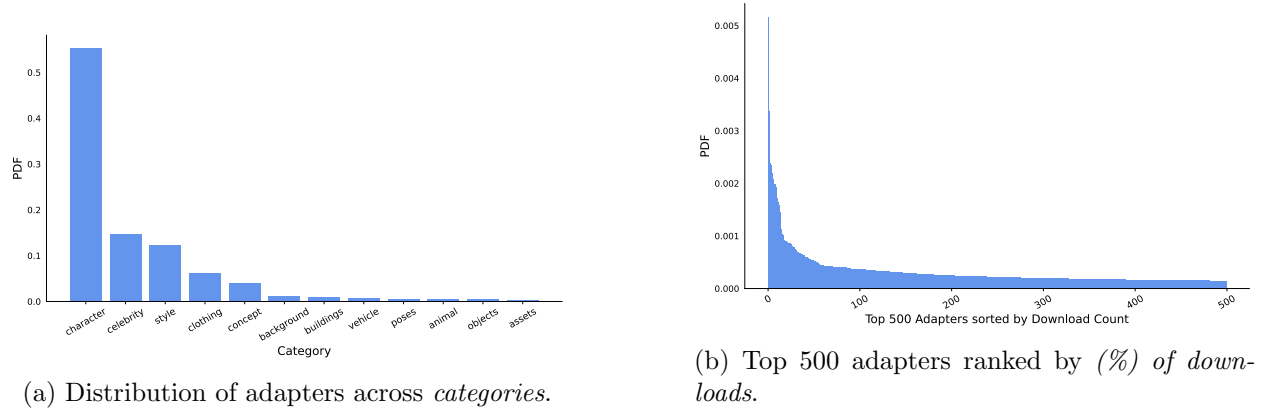


Figure 4.5: **Characterization of Civit Adapter in StylusDocs.** (a) Most adapters are categorized as characters or celebrities. (b) Adapter popularity exhibits a power-law distribution, with the top adapters receiving exponentially more downloads than the others.

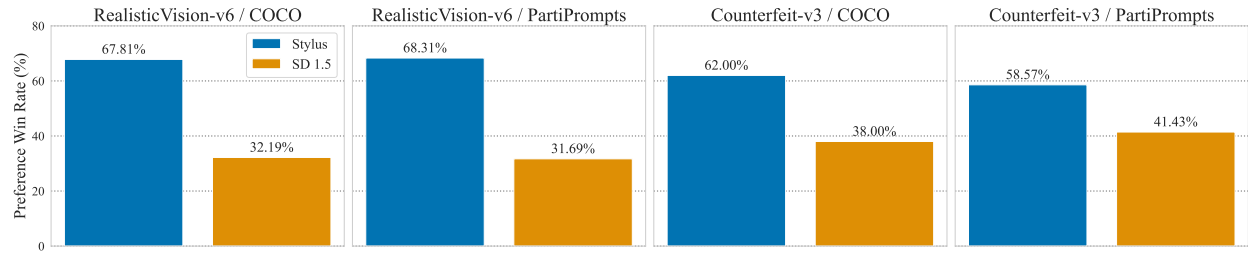
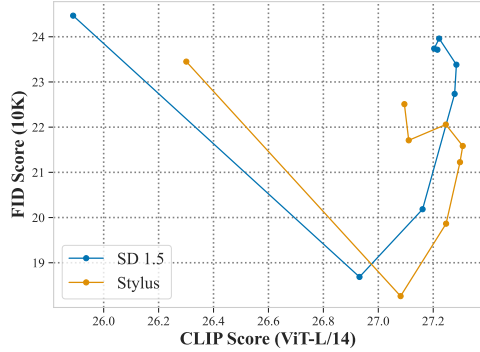


Figure 4.6: **Human Evaluation.** Stylus achieves higher preference scores (2:1) over different datasets and Stable Diffusion checkpoints.

a comprehensive dataset that pulls 75K LoRAs from popular model repositories, Civit AI and HuggingFace (Maier, 2022; Wolf et al., 2020). This dataset contains precomputed OpenAI embeddings (J. Lin et al., 2023) and improved adapter descriptions from Gemini Ultra-Vision (G. Team, 2023), the output of Stylus’s refiner component (§ 4.3). We further characterize the distribution of adapters in Sec. 4.3.

Generation Details. We assess Stylus against Stable-Diffusion-v1.5 (Rombach et al., 2022) as the baseline model. Across experiments, we employ two well-known checkpoints: **Realistic-Vision-v6**, which excels in producing realistic images, and **Counterfeit-v3**, which generates cartoon and anime-style images. Our image generation process integrates directly with Stable-Diffusion WebUI (AUTOMATIC1111, 2022) and defaults to 35 denoising steps using the default DPM Solver++ scheduler (Lu et al., 2023). To replicate high-quality images from existing users, we enable high-resolution upscaling to generate 1024x1024 from 512x512 images, with the default latent upscaler (Karras et al., 2022) and denoising strength set to 0.7. For images generated by Stylus, we discovered adapters could shift the image style away from the checkpoint’s original style. To counteract this, we introduce a *debias*



	CLIP (Δ)	FID (Δ)
Stylus	27.25 (+0.03)	22.05 (-1.91)
Reranker	25.48 (-1.74)	22.81 (-1.15)
Retriever-only	24.93 (-2.29)	24.68 (+0.72)
Random	26.34 (-0.88)	24.39 (+0.43)
SD v1.5	27.22	23.96

(a) Clip/FID Pareto Curve for COCO. (b) CLIP/FID scores and deltas over different retrieval methods (with CFG=6).

Figure 4.7: Automatic Evaluation Metrics. Figure (a) plots the CLIP/FID pareto curve. We observe Stylus shifts the curve down (improved visual fidelity, FID) and to the right (improved textual alignment, CLIP score) over a range of guidance values (CFG): [1, 1.5, 2, 3, 4, 6, 9, 12]. Table (b) evaluates Stylus against different retrieval methods. Stylus outperforms existing retrieval-based methods, attains the best FID score, and achieves similar CLIP score to Stable Diffusion.

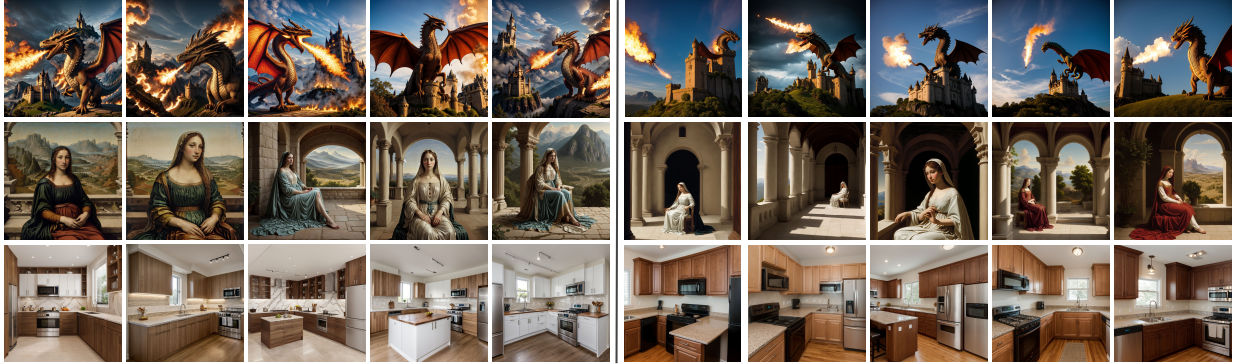


Figure 4.8: Image Diversity. Given the same prompt, our method (left) generates more diverse and comprehensive sets of images than that of existing Stable Diffusion checkpoints (right). Stylus’s diversity comes from its masking scheme and the composer LLM’s temperature parameter.

prompt injected at the end of a user prompt to steer images back to the checkpoint’s style³. We launched 16 replicas of Stylus and Stable Diffusion on 8 A100-80GB GPUs for 4 weeks to generate images for evaluation.

³The debias prompts are “realistic, high quality” for *Realistic-Vision-v6* and “anime style, high quality” for *Counterfeit-v3*, respectively.

4.5 Human Evaluation

To demonstrate our method’s general applicability, we evaluate Stylus over a cross product of two datasets, Microsoft COCO (T.-Y. Lin et al., 2015) and PartiPrompts (J. Yu et al., 2022), and two checkpoints, which generate realistic and anime-style images respectively. Examples of images generated in these styles are displayed in Figure 4.4; Stylus generates highly detailed images that better focus on specific elements in the prompt.

To conduct human evaluation, we enlisted four users to assess 150 images from both Stylus and Stable Diffusion v1.5 for each dataset-checkpoint combination. These raters were asked to indicate their preference for Stylus or Stable-Diffusion-v1.5. In Fig. 4.6, users generally showed a preference for Stylus over existing model checkpoints. Although preference rates were consistent across datasets, they varied significantly between different checkpoints. Adapters generally improve details to their corresponding tasks (e.g. generate detailed elephants); however, for anime-style checkpoints, detail is less important, lowering preference scores.

4.6 Automatic Benchmarks

We assess Stylus using two automatic benchmarks: CLIP (Hessel et al., 2022), which measures the correlation between a generated images’ caption and users’ prompts, and FID (Heusel et al., 2018), which evaluates the diversity and aesthetic quality of image sets. We evaluate COCO 2014 validation dataset, with 10K sampled prompts, and the Realistic-Vision-v6 checkpoint. Fig. 4.7a shows that Stylus shifts the Pareto curve towards greater efficiency, achieving better visual fidelity and textual alignment. This improvement aligns with our human evaluations, which suggest a correlation between human preferences and the FID scores.

4.7 VLM as a Judge

We use *VLM as a Judge* to assess two key metrics: textual alignment and visual fidelity, simulating subjective assessments (Dubois et al., 2023). For visual fidelity, the VLM scores based on disfigured limbs and unrealistic composition of objects. When asked to make subjective judgements, autoregressive models tend to exhibit bias towards the first option presented. To combat this, we evaluate Stylus under both orderings and only consider judgements that are consistent across reorderings; otherwise, we label it a tie. In Fig. 4.9a, we assess evaluate 100 randomly sampled prompts from the PartiPrompts dataset (J. Yu et al., 2022). Barring ties, we find visual fidelity achieves 60% win rate between Stylus and the Stable Diffusion realistic checkpoint, which is conclusively consistent with the 68% win rate from our human evaluation. For textual alignment, we find negligible differences between Stylus and the Stable Diffusion checkpoint. As most prompts lead to a tie, this indicates Stylus does not introduce additional artifacts. We provide the full prompt in Sec. 4.11.

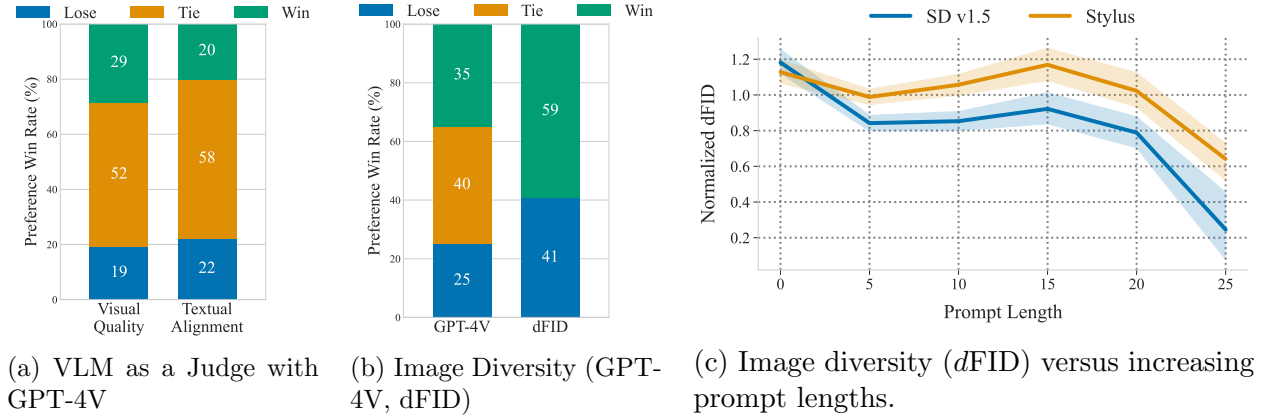


Figure 4.9: Figure (a) and (b) evaluate the preference win rate using GPT-4V as a judge. Stylus achieves higher preference scores as judged by GPT-4V for visual quality and image diversity. Figure (c) shows that Stylus achieves higher diversity scores than Stable Diffusion when prompt length increases.

4.8 Diversity per Prompt

Given identical prompts, Stylus generates highly diverse images due to different composer outputs and masking schemes. Qualitatively, Fig. 4.8 shows that Stylus generates dragons, maidens, and kitchens in diverse positions, concepts, and styles. To quantitatively assess this diversity, we use two metrics:

$dFID$: Previous evaluations with FID (Heusel et al., 2018) show that Stylus improves image quality and diversity *across prompts*⁴. We define $dFID$ specifically to evaluate diversity per prompt, calculated as the variance of latent embeddings from InceptionV3 (Szegedy et al., 2016). Mathematically, $dFID$ involves fitting a Normal distribution $\mathcal{N}(\mu, \Sigma)$ to the latent features of InceptionV3, with the metric given by the trace of the covariance matrix, $dFID = \text{Tr } \Sigma$.

GPT-4V: We use *VLM as a Judge* to assess image diversity between images generated using Stylus and the Stable Diffusion checkpoint over PartiPrompts. Five images are sampled per group, Stylus and SD v1.5, with group positions randomly swapped across runs to avoid GPT-4V’s positional bias (M. Zhong et al., 2024). Similar to VisDiff, we ask GPT-4V to rate on a scale from 0-2, where 0 indicates no diversity and 2 indicates high diversity (Dunlap et al., 2024). Full prompt and additional details are provided in Sec. 4.11.

Fig. 4.9b displays preference rates and defines a win when Stylus achieves higher $dFID$ or receives a higher score from GPT-4V for a given prompt. Across 200 prompts, Stylus prevails in approximately 60% and 58% cases for $dFID$ and GPT-4V respectively, excluding ties. Figure 4.9c compares Stylus with base Stable Diffusion 1.5 across prompt lengths, revealing that Stylus consistently produces more diverse images. Additional results measuring diversity

⁴FID fails to disentangle image fidelity from diversity (Sajjadi et al., 2018; Naeem et al., 2020).



Figure 4.10: **Different Retrieval Methods.** Stylus outperforms all other retrieval methods, which choose adapters that either introduce foreign concepts to the image or override other concepts in the prompt, reducing textual alignment.

	CLIP (Δ)	FID (Δ)
No-Refiner	24.91 (-2.31)	24.26 (+0.30)
Gemini-Ultra Refiner	27.25 (+0.03)	22.05 (-1.91)
GPT-4o Refiner	28.04 (+0.82)	21.96 (-2.00)
SD v1.5	27.22	23.96

Table 4.3: **Refiner’s impact on End2End performance.** Without a refiner, Stylus performs worse than SD v1.5 due to the poor quality of author-provided descriptions. Annotating adapters with GPT-4o significantly improves adapter descriptions and achieves higher CLIP/FID scores than Stylus’s default refiner VLM, Gemini-Ultra.

per keyword are presented in Sec. 4.11.

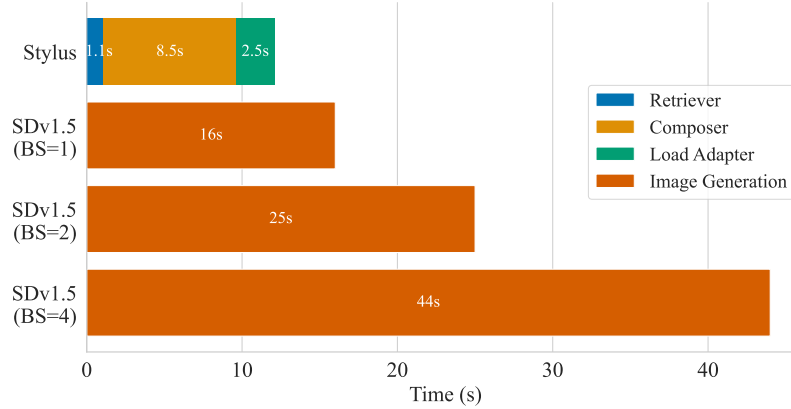


Figure 4.11: **Comparison of Stylus’s inference overheads with Stable Diffusion’s inference time by batch size (BS).** At BS=1, Stylus accounts for 75% of the image generation time, primarily due to the composer processing long context prompts from adapter descriptions. However, Stylus’s overhead decreases when batch size increases.

4.9 Ablations

Impact of Refiner

Table 4.3 evaluates the impact of different refiner pipelines on Stylus’s end-to-end performance. Below, we describe each refiner baseline:

No-Refiner: Stylus uses baseline adapter descriptions sourced from popular repositories such as HuggingFace (Wolf et al., 2020; Maier, 2022). These descriptions are often low-quality and underspecified. Hence, Stylus chooses the wrong adapters and attains lower CLIP and FID scores relative to SDv1.5.

Gemini-Ultra Refiner: This refiner, used throughout all our experiments, employs Gemini-Ultra to auto-generate enhanced adapter descriptions, improving both relevance and specificity. Consequently, Stylus attains better CLIP and FID scores than SDv1.5.

GPT-4o Refiner: The GPT-4o refiner, OpenAI’s most advanced model, outputs the best adapter descriptions, yielding the highest performance gains across CLIP and FID scores. This baseline demonstrates that Stylus’s end-to-end performance is highly dependent on the quality and specificity of adapter descriptions.

Alternative Retrieval-based Methods

We benchmark Stylus’s performance relative to different retrieval methods. For all baselines below, we select the top three adapters and merge them into the base model.

Random: Adapters are randomly sampled without replacement from StylusDocs.

Retriever: The retriever emulates standard RAG pipelines (Z. Zhao et al., 2024; Lewis et al., 2021), functionally equivalent to Stylus without the composer stage. Top adapters are

fetches via cosine similarity over adapter embeddings.

Reranker: An alternative to Stylus’s composer, the reranker fetches the retriever’s adapters and plugs a cross-encoder that outputs the semantic similarity between adapters’ descriptions and the prompt. We evaluate with Cohere’s reranker endpoint (Reimers, 2023).

As shown in Tab. 4.7b, Stylus achieves the highest CLIP and FID scores, outperforming all other baselines which fall behind the base Stable Diffusion model. First, both the retriever and reranker significantly underperform compared to Stable Diffusion. Each method selects adapters that are *similar* to the prompt but potentially introduce unrelated biases. In Fig. 4.10, both methods choose adapters related to elephant movie characters, which biases the concept of elephants and results in depictions of unrealistic elephants. Furthermore, both methods incorrectly assign weights to adapters, causing adapters’ tasks to overshadow other tasks within the same prompt. In Fig. 4.10, both the reranker and retriever generate images solely focused on singular items—beds, chairs, suitcases, or trains—while ignoring other elements specified in the prompt. We provide an analysis of failure modes in 4.11.

Conversely, the random policy exhibits performance comparable, but slightly worse, to Stable Diffusion. The random baseline chooses adapters that are orthogonal to the user prompt. Thus, these adapters alter unrelated concepts, which does not affect image generation. In fact, we observed that the distribution of random policy’s images in Fig. 4.10 were nearly identical to Stable Diffusion.

4.10 Failure Modes and Limitations

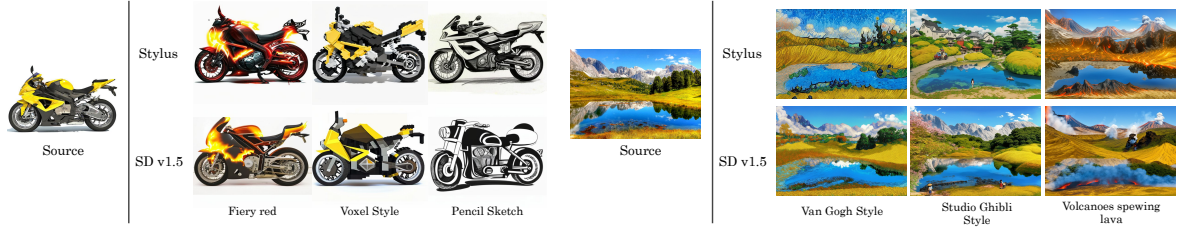
Stylus improves fidelity and diversity, but it does not eliminate the core risks of composing adapters. In practice, we observe several recurring failure modes: image saturation from overly aggressive adapter weights, task blocking where one adapter suppresses other requested concepts, reduced diversity among repeated subjects, low-quality adapters that corrupt generation, and retrieval errors that introduce the wrong concept altogether. We summarize these behaviors in the appendix failure mode analysis (Appendix 4.11), where we also provide qualitative examples and discuss which issues Stylus mitigates versus which remain open problems.

4.11 Efficiency and Extensions

Breakdown of Stylus’s Inference Time

This section breaks down the latency introduced by various components of Stylus. We note that image generation time is independent of Stylus, as adapter weights are merged into the base model (Hu et al., 2021).

Figure 4.11 demonstrates the additional time Stylus contributes to the image generation process across different batch sizes (BS), averaged over 100 randomly selected prompts.



(a) **Image Translation.** Stylus chooses relevant adapters that better adapt new styles and elements into existing images.



(b) **Inpainting.** Stylus chooses adapters that can better introduce new characters or concepts into the inpainted mask.

Figure 4.12: Stylus over different image-to-image tasks.

Specifically, Stylus adds 12.1 seconds to the image generation time, with the composer accounting for 8.5 seconds. The composer’s large overhead is due to long-context prompts, which include adapter descriptions for the top 150 adapters and can reach up to 20K+ tokens. Finally, when the BS is 1, Stylus presents a 75% increase in overhead to the image generation process. However, Stylus’s latency remains consistent across all batch sizes, as the composer and retriever run only once. Hence, for batch inference workloads, Stylus incurs smaller overheads as batch size increases.

Image-Domain Tasks

Beyond text-to-image, Stylus applies across various image-to-image tasks. Fig. 4.12 demonstrates Stylus applied to two different image-to-image tasks: image translation and inpainting.

Image translation: Image translation involves transforming a source image into a variant image where the content remains unchanged, but the style is adapted to match the prompt’s definition. Stylus effectively converts images into their target domains by selecting the appropriate LoRA, which provides a higher fidelity description of the style. We present examples in Fig 4.12a. For a yellow motorcycle, Stylus identifies a voxel LoRA that more effectively decomposes the motorcycle into discrete 3D bits. For a natural landscape, Stylus

successfully incorporates more volcanic elements, covering the landscape in magma.

Inpainting: Inpainting involves filling in missing data within a designated region of an image, typically outlined by a binary mask. Stylus excels in accurately filling the masked regions with specific characters and themes, enhancing visual fidelity. We provide further examples in Fig. 4.12b, demonstrating how Stylus can precisely inpaint various celebrities and characters (left), as well as effectively introduce new styles to a rabbit (right).

VLM as a Judge

The full prompts to GPT-4V as a judge for textual alignment, visual fidelity, and image diversity are specified in Tables 4.4 and 4.5.

To distinguish the two images (or groups of images), the VLM exploits multi-turn prompting: we provide each image (or group of images) labeled with **IMAGE/GROUP A** or **IMAGE/GROUP B**. Note that the **ACK** messages are not generated by the VLM; instead, it is part of the VLM’s context window. We provide the rubric, detailed instructions, reminders, and example model outputs in our prompt. For scoring, the VLM employs Chain-of-Thought (CoT) prompting to output scores 0–2, similar to VisDiff (Dunlap et al., 2024; Wei et al., 2022; Yao et al., 2024). We observe that larger ranges (5–10) leads the model towards abstaining from making decisions, as it avoids outputting extreme scores. However, the score range 0–2 provides the VLM sufficient granularity to express preferences and prompt the model to summarize the key differences.

Textual Alignment. The VLM scores how well a generated image follows the prompt’s specifications. We note that prompts with negations (e.g. “concert with no fans” or “harbor with no boats”) fail for both Stylus and the Stable Diffusion checkpoint. Hence, we prompted the VLM to assign better scores for images that produced fewer fans or boats. Furthermore, as adapters can potentially block existing concepts in the image (see Fig. 4.14b), the VLM allocates partial credit in scenarios where images partially capture the set of keywords in the prompt.

Visual Quality. Our evaluation assesses visual quality through three metrics: clarity, disfigurements, and detail. First, the VLM assigns low clarity scores if an image is blurry, poorly lighted, or exhibits poor compositional quality. We note that LoRAs are trained over specific tasks or concepts; the model determines how to compose different concepts. For instance, a rhinoceros LoRA combined with a motorcycle LoRA led to images of motorcycles draped with rhinoceros hide. As such, the VLM assigns partial credit when the model fails to combine concepts in a meaningful way. Second, the VLM assigns lower scores by judging if an image has disfigured parts. For instance, diffusion models have trouble accurately depicting a human hand, oftentimes generating extra fingers. Finally, the VLM’s final score depends on the detail of image. We find that adapters are able to bring greater detail to certain concepts. For example, an elephant adapter generates elephants with much greater detail than that of the base model. However, we note that the VLM is not good at detecting subtleties in detail.

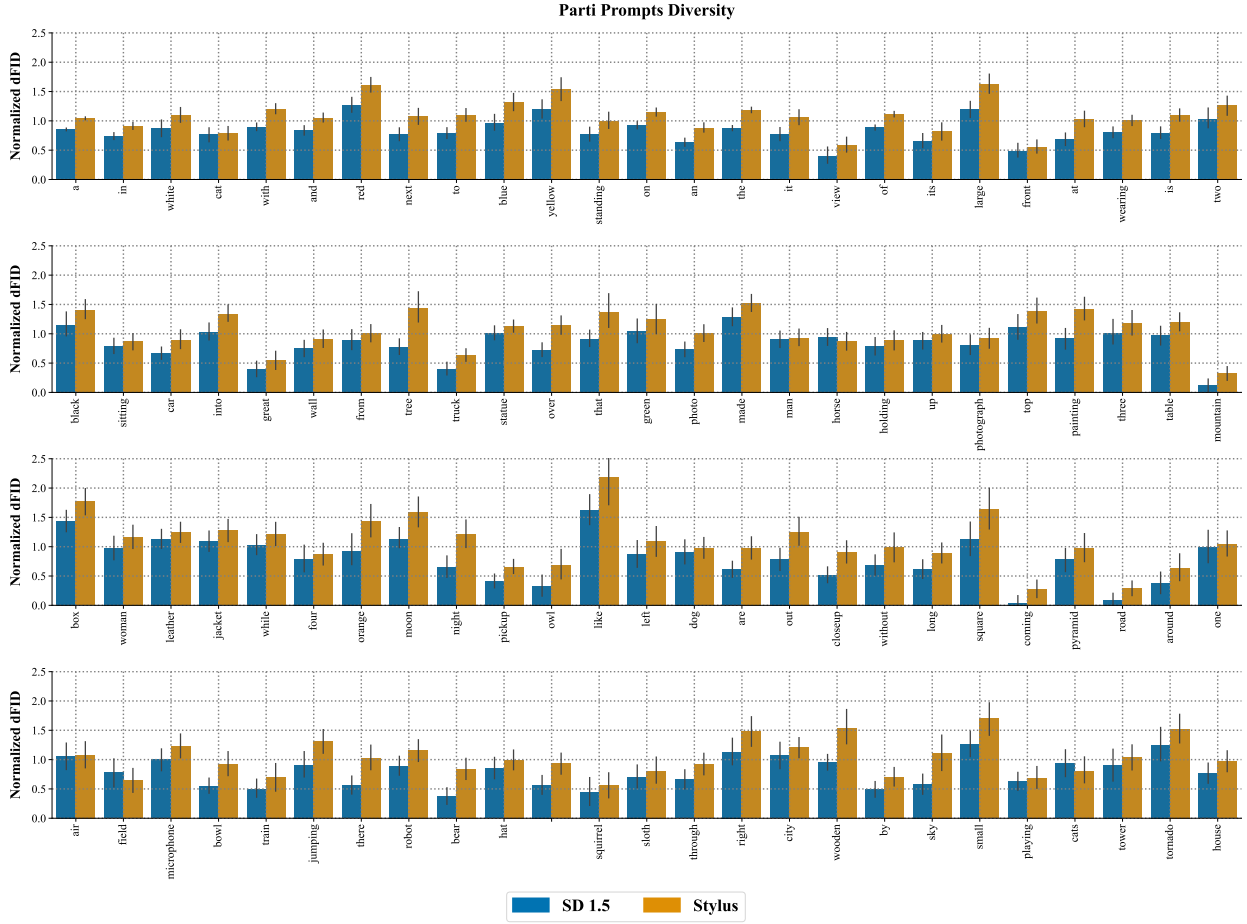


Figure 4.13: $dFID$ for top 100 keywords in PartiPrompts dataset. Stylus leads to consistently higher diversity when compared to Stable Diffusion checkpoints, especially for words describing concepts and attributes.

Diversity. For each prompt, we generate five images each for Stylus and the Stable Diffusion checkpoint. These images are then assessed with a VLM judge, GPT-4V, which rates and ranks them based on diversity. In Tab. 4.5, we measure diversity through two metrics. The first metric, theme interpretation, measures diversity based on the interpretation of the prompt, which is often underspecified. We find that different thematic interpretations improve model response due to non-ambiguity. The second metric measures diversity by the variance of focus across different subjects. We find that many prompts often under-specify which subject is the focus of the image.

System Prompt: You are a photoshop expert judging which image has better composition quality. Scoring: Compositional quality scores can be 2 (very high quality), 1 (visually aesthetic but has elements with distortion, missing features, or extra features), or 0 (low visual quality, with texture, blur, or visual-artifact issues). Composition aspects: • Clarity: If the image is blurry, poorly lit, or has poor composition (objects obstructing each other), it gets a 0. • Disfigured Parts: This applies to body parts of humans and animals as well as objects like motorcycles. If a hand has 6 fingers but is otherwise normal, it gets a 1. If the fingers themselves are disfigured, it gets a 0. • Detail: If a sail shows dynamic ripples and ornate patterns, give a 2. If it is monochrome and flat, give a 1. If it looks cartoonish and inconsistent with the environment, give a 0. • If the prompt is ‘shoes without laces’ but the shoes have laces, this is somewhat aligned and gets a 1. • If the prompt is ‘a concert without fans’ but there are fans in the image, prefer the images that show fewer fans.	Scoring: Alignment scores can be 2 (fully aligned), 1 (incorporates part of the theme but not all), or 0 (not aligned). Examples: • If the prompt is ‘shoes’ and an image is a sock, this is not aligned and gets a 0.
User: This is IMAGE A. Reply ‘ACK’. <i>% Generated Image from Group A</i>	
Assistant: ACK	
User: This is IMAGE B. Reply ‘ACK’. <i>% Generated Images from Group B</i>	
Assistant: ACK	
User: Rate the quality of the images in GROUP A and GROUP B. For each image, provide a score and explanation. Image A Quality: <SCORE> (<EXPLANATION>) Image B Quality: <SCORE> (<EXPLANATION>) Preference: Group <CHOICE> (<EXPLANATION>) <i>% Prevent VLM from returning neutral results.</i> I’ll make my own judgement using your results; your response is just an opinion as part of a rigorous process. Additional requirements: • You must pick a group for ‘Better Quality’ / ‘Better Alignment’; ‘neither’ is not an option. • If it’s a close call, make a choice first and then explain why in parentheses.	

Table 4.4: Full prompt judging compositional quality (left) or textual alignment (right) using VLM.

System Prompt: You are a photoshop expert judging which set of images is more diverse. Scoring: Diversity scores can be 2 (very diverse), 1 (somewhat diverse), or 0 (not diverse). Diversity can be decomposed into 1) interpretation of the theme and 2) the main subject. • Theme Interpretation: The theme can vary based on interpretation. “It’s raining cats and dogs” may be literal (cats and dogs falling from the sky) or figurative (heavy rain). A set showing both is more diverse; a set showing only weather or only animals gets a 1. • Main Subject: A set mixing apples and children dressed as apples is more diverse than one with only children dressed as apples, because the main subject changes across images.
User: This is GROUP A. Reply ‘ACK’. <i>% Set of 5 Generated Images from Group A</i>
Assistant: ACK
User: This is GROUP B. Reply ‘ACK’. <i>% Set of 5 Generated Images from Group B</i>
Assistant: ACK
User: Rate the diversity of the images in GROUP A and GROUP B. For each group, provide a score and explanation. Group A Diversity: <SCORE> (<EXPLANATION>) Group B Diversity: <SCORE> (<EXPLANATION>) Preference: Group <CHOICE> (<EXPLANATION>) <i>% Prevent VLM from returning neutral results.</i> I’ll make my own judgement using your results; your response is just an opinion as part of a rigorous process. Additional requirements: • Do not forget to reward different main subjects in the diversity score. • You must pick a group for ‘More Diversity’; ‘neither’ is not an option. • If it’s a close call, make a choice first and then explain why in parentheses.

Table 4.5: Full prompt judging diversity using VLM.

Additional Diversity Scores

Fig. 4.13 decomposes $dFID$ scores over the top 100 keywords in the PartiPrompts dataset. We highlight that the largest differences stem from concepts, appearances, attributes, or styles. For example, Stylus excels over concepts ranging from animals (“bears”, “sloth”, and “squirrel”) to objects (“microphone”, “box”, and “jacket”). Selected attributes can include but are not limited to “white”, “blue”, and “photographic”. Regardless of keyword, Stylus attains higher diversity scores across the board.

Failure Modes

We detail different failure modes that were discovered while developing Stylus.

Image saturation. The quality of image generation is highly dependent on adapters’ weights. If the assigned weight is above the recommended value, the adapter negatively impacts image generation, leading to a growing number of visual inconsistencies and artifacts.



(a) **Image Saturation.** Assigning too high of a weight to a “James Bond” adapter leads to significant degradation in visual fidelity.



(b) **Task Blocking.** Adapters can block a prompt’s or other adapter’s tasks (i.e. toy trains, person in orange blanket, or man eating cake).

(c) **Task Diversity.** Adding an adapter reduces diversity of instances within a single task (i.e. teddy bears, woman, and apples).



(d) **Low quality adapters.** Low quality adapters can significantly impact visual fidelity. We blacklist such adapters.

(e) **Retrieval Errors.** Retrieval errors can lead to foreign biases in image generation and deliberate misinterpretations of the prompt.

Figure 4.14: **Categorization of Different Failure Modes.**

In Fig. 4.14a, assigning a high weight to a “James Bond” LoRA increases images exposure and introducing significant visual tearing. Stylus mitigates over-saturation with its refiner component, which extracts the right adapter weights from the adapter’s model card. Lastly,

Stylus uniformly weights adapters based on their associated tasks, ensuring that similar adapters do not significantly impact their corresponding tasks.

Task Blocking. Composing adapters presents the risk of overwriting existing concepts or tasks specified in the prompt and other selected adapters. We illustrate several examples in Figure 2: a train LoRA overrides the toy train concept (left), a park bench LoRA masks a person in an orange blanket (middle), and a fancy cake LoRA erases the image of a man eating the cake (right). Task blocking typically arises from two main issues: the adapter weight set too high or too many adapters merged into the base model. Stylus addresses this by reducing an adapter’s weight with uniform weighting per task, while the masking scheme reduces the number of selected adapters. Although Stylus does not completely solve task blocking, it offers simple heuristics to mitigate the issue.

Task Diversity. Merging adapters into the base model overwrites the base model’s prior distribution over an adapter’s corresponding tasks. If an adapter is not finetuned on a diverse set of images, diversity is significantly reduced among different instances of the same task. We present three examples in Fig. 4.14c, over different prompts that specify multiple instances of the same task (teddy bears, women, and apples). We observe that all instances of each task are highly identical with one another. Stylus offers no solution to address or mitigate this problem.

Low quality adapters. Low quality adapters can significantly degrade the quality of image generation, as shown by corrupted images in Fig. 4.14d. This issue typically arises from poor training data or from fine-tuning the adapter for too many epochs. Stylus attempts to blacklist such adapters. However, our blacklist is not comprehensive, and as a result, Stylus may still occasionally select low-quality adapters.

Retrieval Errors. Stylus’s retrieval process involves three stages, each introducing potential errors that can compound in later stages. For instance, the refiner may return incorrect descriptions of an adapter’s task, while the composer may classify the adapter into an incorrect task. We detail three examples in Figure 4. Stylus selects an “okapi” (forest giraffe) LoRA, known for its distinctive zebra-like appearance, causing the generated giraffes to adopt the okapi’s skin texture. In the middle, Stylus selects a flowery vase LoRA, a misinterpretation of the prompt “orange flowers placed in a vase.” On the right, the composer incorrectly chooses a human baby adapter for the prompt “a baby daikon radish in a tutu.”, resulting in images of babies instead of daikons. Stylus includes an option to self-repair faulty composer outputs with multi-turn conversations, which can improve adapter selection.

4.12 Discussion

The strategic composition and routing of adapters in Stylus introduce a new dimension of model performance, broadening the scope of potential applications. One such application is the automatic creation of agentic workflows (J. Zhang et al., 2024; Zhuge et al., 2024). For instance, Stylus’s composer can decompose a complex task into a graph of subtasks and assign them to specialized agents to improve end-to-end performance. Additionally, routing can extend beyond adapters to encompass different models, allowing Stylus to optimize the cost-performance tradeoff by dynamically selecting between high-performing, resource-intensive models and more efficient, lower-cost models (Ong et al., 2024; Mohammadshahi et al., 2024). Finally, for fact verification, adapters have shown significant potential in reducing hallucinations (Tian et al., 2023; Gekhman et al., 2024). Stylus can selectively use domain-specific, fine-tuned models to enhance factual accuracy and better verify claims.

As demonstrated in Sec. 4.4, Stylus demonstrates significant potential for improvement, as adapter composition introduces future research challenges beyond the scope of this work. A summary of Stylus’s failure cases are provided in Fig. 4.14. Specifically, adapters can *restrict certain concepts* from appearing in an image and *limit diversity* among multiple subjects within a scene. While Stylus does not fundamentally solve these challenges, Stylus reduces the likelihood of these problems occurring by reducing the number of adapters through its masking algorithm. Lastly, Stylus introduces noticeable overheads to the inference pipeline, primarily stemming from the composer’s long context prompts, which can be accelerated with various sequence parallel techniques (H. Liu et al., 2023; B. Lin et al., 2024).

4.13 Conclusion

We propose Stylus, a flexible algorithm that automatically selects and composes adapters to generate better images. Our method leverages a three-stage framework that precomputes adapters as lookup embeddings and retrieves most relevant adapters based on prompts’ keywords. To evaluate Stylus, we develop **StylusDocs**, a processed dataset featuring 75K adapters and pre-computed adapter embeddings. Our evaluation of Stylus, across automatic metrics, humans, and vision-language models, demonstrates that Stylus achieves better visual fidelity, textual alignment, and image diversity than existing Stable Diffusion checkpoints.

Part III

Conclusion

Chapter 5

Diversity Is Necessary but Not Sufficient for Creativity

5.1 Summary of Learnings

This thesis studies resampling diversity of large-scale models across language and image generation. We analyzed the reinforcement learning objective and highlighted how the objective is inherently mode-seeking. We then proceed to show how mode-collapsed models can be post-hoc fine-tuned on a small balanced dataset to regain diversity. This simple exercise exposes the importance of midtraining and SFT as a heavy-weight solution for preserving and correcting known biases and imbalances. However, we focus our main efforts on training-free approaches as others are better positioned to address improving production pre-training data for diversity and innovative algorithms on that data to reduce collapse.

We highlight two approaches to improving diversity. In SimpleStrat, we propose stratified sampling to introduce external randomness to the sampling with partitions elicited with a game of 20 questions. We find that this leads to better resampling diversity without impacting generation quality and improvements stack orthogonally with temperature. We demonstrate 4X improvement in multi-answer question answering and creative writing. In Stylus, we show random subsampling of candidate LoRAs can provide more diversity in the weight space. In the Stylus setting, we use FID to show the diversity matches the diversity of the natural image distribution and also diversity as judged by VLM.

Together these contributions highlight methods of ameliorating lost diversity without modifying the training algorithms or training data. Akin to how ϵ -greedy policies in classic reinforcement learning (Rummery et al., 1994; Watkins et al., 1992) induce exploration by adding more randomness, both methods find effective ways to induce more randomness to the sampling. However, unlike prior settings, random actions in token generation can push the language model off the rails. By their design, language models narrow the space of $|Vocab|^{context.length}$ responses to small manifold of fluent continuations. As such, both methods are careful to introduce randomness while remaining in the model’s training distribution.

5.2 Future Work

Trained Delegation

The SimpleStrat auto-stratification can be viewed as a delegation agent. The overall goal is to generate many diverse solutions and auto-stratification’s job is to generate prompts for each spawned agent to get a solution while orchestrating to ensure each subagent is searching a different subspace. The application of proposing tasks can be a good place to demonstrate the benefits of training the delegator. A natural follow-up study is to co-train the SimpleStrat model with the policy model completing tasks on the websites. A shared challenge of co-training agents is how to assign credit to the different agents. SimpleStrat has a nice answer to this. SimpleStrat draws parallels to classic decision trees as there is a space of solutions and we want the most informative cuts of the solution space. Since we operate under the assumption that we have full annotation of the space of acceptable solutions, we can train SimpleStrat auto-stratification on maximizing information gain. Concretely, we can fine-tune an auto-stratification model by having it generate splits, then on a fixed set of solutions measure the information gain. Other choices include entropy, balance ratio, and Gini impurity. This allows us to decouple the strength of the proposer from the strength of the model under training. Three key questions can be tested. First, does training on splitting solution spaces in one domain translate to another domain. Training on CoverageQA may lead to good reasoning patterns for assessing information gain that generalizes to task proposing. Secondly, does training on information gain lead to better webagent policies even without co-training. Third, when co-training the agents, is there a benefit to training auto-stratification with both the goldilocks objective (do the tasks lead to 50% success rate on the current policy) and the information gain (do the splits provide good coverage on a hidden set of tasks). Training the LLM to auto-stratify may lead to better delegation in other domains as well.

SimpleStrat: Sampling without Replacement

A second extension to SimpleStrat is “sampling-without-replacement,” or settings where we allow ourselves to observe what solutions we have previously generated when generating a new one. The simplest setting to consider is adjusting the probabilities from heuristic estimation as we see solutions. Consider a simplified setting where we simply split on 50/50 boy and girl names as the stratification. If, due to random chance, we have already sampled 8 boy names and only 2 girl names, then the empirical distribution is heavily skewed toward boys. We would therefore like the probability of sampling a girl name to increase on subsequent draws, so that the running sample distribution is pulled back toward the desired stratification.

More generally, rather than sampling each partition from a fixed heuristic probability, we maintain the observed count n_i for each partition i and compare its empirical frequency to the target stratification probability q_i . A Bayesian way to do this is to place a Dirichlet

prior α over the partitions and use the posterior mean

$$\hat{p}_i = \frac{\alpha_i + n_i}{\sum_j \alpha_j + \sum_j n_j}$$

as a smoothed estimate of how represented partition i currently is. We then reweight the probability of sampling from partition i inversely to its current overrepresentation:

$$w_i \propto \frac{q_i}{\hat{p}_i}, \quad P(\text{next partition} = i) = \frac{w_i}{\sum_j w_j}.$$

For example, suppose our target stratification is

$$q_{\text{boy}} = q_{\text{girl}} = 0.5,$$

and we use a symmetric prior

$$\alpha_{\text{boy}} = \alpha_{\text{girl}} = 1.$$

After observing 8 boy names and 2 girl names, the posterior means are

$$\hat{p}_{\text{boy}} = \frac{1 + 8}{1 + 1 + 8 + 2} = \frac{9}{12} = 0.75,$$

and

$$\hat{p}_{\text{girl}} = \frac{1 + 2}{1 + 1 + 8 + 2} = \frac{3}{12} = 0.25.$$

The unnormalized sampling weights are therefore

$$w_{\text{boy}} \propto \frac{0.5}{0.75} = \frac{2}{3}, \quad w_{\text{girl}} \propto \frac{0.5}{0.25} = 2.$$

After normalizing, the next sampling probabilities become

$$P(\text{boy}) = \frac{2/3}{2/3 + 2} = \frac{1}{4}, \quad P(\text{girl}) = \frac{2}{2/3 + 2} = \frac{3}{4}.$$

Thus, after seeing 8 boys and 2 girls, the next sample would be drawn using a 25/75 boy–girl sampling ratio. This does not require knowing in advance how many total samples will be generated; it simply uses the samples seen so far to adaptively correct future sampling probabilities toward the target stratification. Additional adjustments can be made to account for the size of conditional probabilities with multiple partitions and settings where we know the total number of samples we are looking for.

This sampling with replacement setting is more aligned with using SimpleStrat in evolutionary approaches such as AlphaEvolve (Novikov et al., 2025) and GEPA (Agrawal et al., 2025). These are settings where the model is allowed to condition on previous solutions to decide what to try next. For evolutionary methods in particular, SimpleStrat can be used to make the LLM calls for candidate solution generation more diverse (often the prompt is

just given this implementation of bubble sort modify it to make the sorting more memory efficient). Another application is to provide a secondary diversity objective. OpenEvolve (O. Team, 2025) implements a code-diversity metric by comparing a candidate program c against a reference set $\mathcal{R}$ of previously generated programs. For a reference program $r \in \mathcal{R}$, the naive diversity heuristic is zero when the code strings are identical, and otherwise combines the absolute difference in code length, the difference in line count, and the symmetric difference between the sets of characters appearing in each program. Concretely, the pairwise score is computed as

$$d(c, r) = 0.1 |\text{len}(c) - \text{len}(r)| + 10 |\text{lines}(c) - \text{lines}(r)| + 0.5 |\text{chars}(c) \Delta \text{chars}(r)|.$$

The candidate’s diversity score is then the average pairwise distance to the reference set,

$$D(c; \mathcal{R}) = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} d(c, r).$$

This diversity signal can improve search efficiency, but it ignores program semantics and is coarse at best. In contrast, each property produced by auto-stratification can be used to define a more semantic diversity score by having an LLM estimate the probability that a solution satisfies the property. This can help ensure that the working set of candidates remains semantically diverse.

5.3 Open Questions

Language model response diversity as described in this thesis naturally raises two key open questions: diversity-preserving RL algorithms and ultimately, artificial creativity.

Diversity-preserving RL algorithms.

Recent efforts try to address this collapse either by optimizing multi-sample objectives such as Pass@K (Tang et al., 2025; Walder et al., 2025) or by explicitly rewarding lower-probability solutions (He et al., 2025; Song et al., 2025). These approaches are promising, but they often expose the same underlying tension: gains in sample-set diversity or Pass@K can come at the cost of single-sample correctness, and their benefits can vary sharply across settings. We have focused our efforts on approaches that do not require modifying the model’s training. However, DivPO (Lanchantin et al., 2025) and DAPO (Q. Yu et al., 2025) provide initial approaches to improving diversity more directly in RL training, with DAPO’s clip-higher modification explicitly aimed at improving sequence entropy – a diversity proxy. However, these solutions do not lead to substantial improvement by themselves. The question remains open whether algorithmic adjustments can be made to RLVR training to improve diversity.

It’s also possible that RLVR is not easily adjusted to satisfy diversity-hungry users. Potentially interesting approaches to replace RLVR include Energy-based Fine-Tuning (Jelassi

et al., 2026) and Discrete Diffusion (Austin et al., 2021; Lou et al., 2024) lend themselves to directly reward diversity. Perhaps we’ll look back in a few years and conclude substantial rethinking was needed to unlock diversity just as diffusion and flow matching has taken robotics by storm (McAllister et al., 2025; Chi et al., 2023; Chi et al., 2024)

Creativity.

Originally, this research set out to philosophize as to the quintessential human question of creativity. Taking inspiration from our user-study interviews with Hollywood and New York’s most decorated writers in 2023 and echoed in Chakrabarty et al., 2024 and Ippolito et al., 2022, one truth was very clear and has resoundingly held true as chatbots have matured (Behrooz et al., 2024). Story-telling is not about putting words on the page; story-telling is about being the kind of person who has a story worth telling. In one interview, a decorated screenwright¹ described their process as years of storyboarding and iteration after iteration of character development; committing the story to text happens in a mere week. As F. R. Leavis would say, the true work of a writer is not just to capture the “essential human experience” (Leavis, 1948).

In an effort to prototype co-creative companions for writers’ block, an interesting quirk appeared: models have an uncanny preference for certain character names. This same observation is articulated in user-studies including Chakrabarty et al., 2024. where writers expressed frustration that models have “one narrative that it knows how to write, and writes exactly [that] same story over and over again.” This led me down the path of SimpleStrat hoping to measure and improve semantic repetition. However, the broader question of how to advance machine creativity ultimately requires a more comprehensive answer to a more fundamental question: What is human creativity?

What do the experts have to say when judging human artistic output? As Terry Eagleton in *Literary Theory* would argue *Literature* is no more than “valued writing” subjective to the reader and writers of the work (Eagleton, 2011). Intellectual property law would say *patentable* requires a non-obvious, novel, and useful contribution (United States Congress, 1952a; United States Congress, 1952b; United States Congress, 1952c). For Clement Greenberg, modernist painting’s *newness* comes not simply from originality, but from its self-critical turn inward: its break from inherited conventions in order to reveal what is unique to painting as a medium (Greenberg, 1993). While we are far from consensus, creativity is most obviously missing to human judges when responses are predictable or cliché Chakrabarty et al., 2024 and most evident when operating at the edge of expectation.

Some progress with smaller models by Padmakumar et al., 2026 tries to directly measure novelty as unique n-gram not found in the training corpus while maintaining quality via LLM-as-a-Judge. In practice, the best measure we have is open-ended human judgement of the same ilk as Gomez et al.’s and Yang et al.’s, in which humans or LLMs, respectively, evaluate

¹*Screenwright* is used here by analogy with playwright and emphasizes the craft of “making” work for the screen, echoing the older -wright formation meaning maker or builder.

AI creativity and find that human-generated stories outperform AI-generated ones (Gómez-Rodríguez et al., 2023; K. Yang et al., 2023). These works leave the interpretation of "creative" or "interestingness" entirely up to the individual subjective judgement of the human judges. It remains, perhaps, one of the most captivating open questions: what can be done to improve LLM creativity? And perhaps, ironically, that will tell us more about us as readers than about the author.

Regrettably, my AIs and I did not have the time to complete a thesis on creativity just as we also did not have the time to make it shorter.²

²The quote original appears in Blaise Pascal's Provincial Letters from 1657 (Pascal, 1859): "I have made this longer than usual because I have not had time to make it shorter." It's often misattributed to Mark Twain because he would totally have written a quip like this.

Bibliography

- Agrawal, Lakshya A., Shangyin Tan, Dilara Soylu, Noah Ziem, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab (2025). *GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning*. arXiv: 2507.19457 [cs.CL]. URL: <https://arxiv.org/abs/2507.19457>.
- Aliannejadi, Mohammad, Julia Kiseleva, Aleksandr Chuklin, Jeff Dalton, and Mikhail Burtsev (2020). *ConvAI3: Generating Clarifying Questions for Open-Domain Dialogue Systems (ClariQ)*. arXiv: 2009.11352 [cs.CL]. URL: <https://arxiv.org/abs/2009.11352>.
- Austin, Jacob, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg (2021). “Structured Denoising Diffusion Models in Discrete State-Spaces”. In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 17981–17993.
- AUTOMATIC1111 (Aug. 2022). *Stable Diffusion Web UI*. URL: <https://github.com/AUTOMATIC1111/stable-diffusion-webui>.
- Balcan, Maria-Florina, Travis Dick, Tuomas Sandholm, and Ellen Vitercik (2018). “Learning to Branch”. In: *CoRR* abs/1803.10150.
- Barrett, Clark, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli (2009). “Satisfiability Modulo Theories”. In: *Handbook of Satisfiability*. Ed. by Armin Biere, Hans van Maaren, and Toby Walsh. IOS Press. Chap. 26, pp. 825–885.
- Beauchemin, Maxime (2014). *Apache Airflow*. URL: <http://airflow.apache.org/>.
- Behrooz, Morteza, Yuandong Tian, William Ngan, Yael Yungster, Justin Wong, and David Zax (2024). “Holding the Line: A Study of Writers’ Attitudes on Co-creativity with AI”. In: *arXiv preprint arXiv:2404.13165*. DOI: 10.48550/arXiv.2404.13165. URL: <https://arxiv.org/abs/2404.13165>.
- Bello, Irwan, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio (2016). “Neural combinatorial optimization with reinforcement learning”. In: *arXiv preprint arXiv:1611.09940*.
- Bender, Emily M., Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell (2021). “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 610–623. DOI: 10.1145/3442188.3445922.

- Bertolissi, Clara, Daniel R Dos Santos, and Silvio Ranise (2018). “Solving multi-objective workflow satisfiability problems with optimization modulo theories techniques”. In: *Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies*, pp. 117–128.
- Bjørner, Nikolaj, Anh-Dung Phan, and Lars Fleckenstein (2015). “ ν Z - An Optimizing SMT Solver”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Christel Baier and Cesare Tinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 194–199. ISBN: 978-3-662-46681-0.
- Bradley, Herbie, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman (2023). “Quality-diversity through ai feedback”. In: *arXiv preprint arXiv:2310.13032*.
- Caballero, Ethan, . OpenAI, and Ilya Sutskever (Aug. 2016). *Description2Code Dataset*. DOI: 10.5281/zenodo.5665051. URL: <https://github.com/ethancaballero/description2code>.
- Cao, Yong, Haijiang Liu, Arnav Arora, Isabelle Augenstein, Paul Röttger, and Daniel Herscovich (2025). “Specializing Large Language Models to Simulate Survey Response Distributions for Global Populations”. In: *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Albuquerque, New Mexico: Association for Computational Linguistics, pp. 3141–3154. DOI: 10.18653/v1/2025.naacl-long.162. URL: <https://aclanthology.org/2025.naacl-long.162/>.
- Cassano, Federico, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha (2023). “Knowledge transfer from high-resource to low-resource programming languages for code llms”. In: *arXiv preprint arXiv:2308.09895*.
- Chakrabarty, Tuhin, Vishakh Padmakumar, Faeze Brahman, and Smaranda Muresan (2024). “Creativity Support in the Age of Large Language Models: An Empirical Study Involving Professional Writers”. In: *Proceedings of the 16th Conference on Creativity & Cognition*. Association for Computing Machinery, pp. 132–155. DOI: 10.1145/3635636.3656201.
- Chase, Harrison (Oct. 2022). *LangChain*. URL: <https://github.com/langchain-ai/langchain>.
- Chen, Hao, Abdul Waheed, Xiang Li, Yidong Wang, Jindong Wang, Bhiksha Raj, and Marah I. Abdin (2024). “On the Diversity of Synthetic Data and its Impact on Training Large Language Models”. In: *arXiv preprint arXiv:2410.15226*. DOI: 10.48550/arXiv.2410.15226.
- Chen, Nuo, Zinan Zheng, Ning Wu, Linjun Shou, Ming Gong, Yangqiu Song, Dongmei Zhang, and Jia Li (2023). “Breaking language barriers in multilingual mathematical reasoning: Insights and observations”. In: *arXiv preprint arXiv:2310.20246*.
- Chi, Cheng, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song (2023). “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion”. In: *Proceedings of Robotics: Science and Systems (RSS)*.

- Chi, Cheng, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song (2024). “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion”. In: *The International Journal of Robotics Research*.
- Chronopoulou, Alexandra, Matthew E. Peters, Alexander Fraser, and Jesse Dodge (2023). *AdapterSoup: Weight Averaging to Improve Generalization of Pretrained Language Models*. arXiv: 2302.07027 [cs.CL].
- Chung, John Joon Young, Ece Kamar, and Saleema Amershi (2023). “Increasing diversity while maintaining accuracy: Text data generation with large language models and human interventions”. In: *arXiv preprint arXiv:2306.04140*.
- Cook, Stephen A. (1971). “The Complexity of Theorem-Proving Procedures”. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158. DOI: 10.1145/800157.805047.
- Dai, Haixing, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao, Shaochen Xu, Wei Liu, Ninghao Liu, et al. (2023). “Auggpt: Leveraging chatgpt for text data augmentation”. In: *arXiv preprint arXiv:2302.13007*.
- De Moura, Leonardo and Nikolaj Bjørner (2008). “Z3: An efficient SMT solver”. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 337–340.
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].
- Dubey, Abhimanyu, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. (2024). “The llama 3 herd of models”. In: *arXiv preprint arXiv:2407.21783*.
- Dubois, Yann, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto (2023). “AlpacaFarm: A Simulation Framework for Methods that Learn from Human Feedback”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., pp. 30039–30069. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/5fc47800ee5b30b8777fdd30abcaaf3b-Paper-Conference.pdf.
- Dunlap, Lisa, Alyssa Umino, Han Zhang, Jiezhi Yang, Joseph E Gonzalez, and Trevor Darrell (2023). “Diversify your vision datasets with automatic diffusion-based augmentation”. In: *Advances in neural information processing systems* 36, pp. 79024–79034.
- Dunlap, Lisa, Yuhui Zhang, Xiaohan Wang, Ruiqi Zhong, Trevor Darrell, Jacob Steinhardt, Joseph E. Gonzalez, and Serena Yeung-Levy (2024). “Describing Differences in Image Sets with Natural Language”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Dutertre, Bruno and Leonardo De Moura (2006). “The yices smt solver”. In: *Tool paper at http://yices.csl.sri.com/tool-paper.pdf* 2.2, pp. 1–2.

- Eagleton, Terry (2011). *Literary Theory: An Introduction*. 25th Anniversary Edition. Originally published in 1983; revised edition published in 1996. Malden, MA: John Wiley & Sons. ISBN: 9781118306291.
- Euler, Leonhard (1743). *De integratione aequationum differentialium altiorum graduum*.
- Fan, Angela, Mike Lewis, and Yann Dauphin (2018). “Hierarchical neural story generation”. In: *arXiv preprint arXiv:1805.04833*.
- Fan, Wenqi, Yao Ma, Qing Li, Jianping Wang, Guoyong Cai, Jiliang Tang, and Dawei Yin (2022). “A Graph Neural Network Framework for Social Recommendations”. In: *IEEE Transactions on Knowledge and Data Engineering* 34.5, pp. 2033–2047.
- Fremont, Daniel J., Alexandre Donzé, and Sanjit A. Seshia (2017). “Control Improvisation”. In: *arXiv preprint arXiv:1704.06319*. DOI: 10.48550/arXiv.1704.06319. arXiv: 1704.06319 [cs.FL].
- Fremont, Daniel J., Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia (2023). “Scenic: A Language for Scenario Specification and Data Generation”. In: *Machine Learning* 112.10, pp. 3805–3849. DOI: 10.1007/s10994-021-06120-5.
- Gal, Rinon, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or (2022). *An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion*. arXiv: 2208.01618 [cs.CV].
- Game Freak (Feb. 27, 1996). *Pocket Monsters Red and Pocket Monsters Green*. Game Boy video games. Japan.
- Gandikota, Rohit, Joanna Materzyńska, Tingrui Zhou, Antonio Torralba, and David Bau (2023). “Concept Sliders: LoRA Adaptors for Precise Control in Diffusion Models”. In: *arXiv preprint arXiv:2311.12092*.
- Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang (2024). *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv: 2312.10997 [cs.CL].
- Gasse, Maxime, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi (2019). “Exact combinatorial optimization with graph convolutional neural networks”. In: *arXiv preprint arXiv:1906.01629*.
- Gekhman, Zorik, Gal Yona, Roei Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig (2024). *Does Fine-Tuning LLMs on New Knowledge Encourage Hallucinations?* arXiv: 2405.05904 [cs.CL]. URL: <https://arxiv.org/abs/2405.05904>.
- Ghosh, Dibya (2018). *KL Divergence for Machine Learning*. URL: <https://dibyaghosh.com/blog/probability/kldivergence/> (visited on 04/11/2026).
- Gilmer, Justin, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl (2017). “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR, pp. 1263–1272.
- Gog, Ionel, Sukrit Kalra, Peter Schafhalter, Joseph E. Gonzalez, and Ion Stoica (2022). “D3: A Dynamic Deadline-Driven Approach for Building Autonomous Vehicles”. In: *Proceedings of the Seventeenth European Conference on Computer Systems*. Rennes, France: Association for Computing Machinery, pp. 453–471.

- Gómez-Rodríguez, Carlos and Paul Williams (Dec. 2023). “A Confederacy of Models: a Comprehensive Evaluation of LLMs on Creative Writing”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, pp. 14504–14528. DOI: 10.18653/v1/2023.findings-emnlp.966. URL: <https://aclanthology.org/2023.findings-emnlp.966/>.
- Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). “Generative Adversarial Nets”. In: *arXiv preprint arXiv:1406.2661*. URL: <https://arxiv.org/abs/1406.2661>.
- Greenberg, Clement (1993). “Modernist Painting”. In: *Art and Literature*. Ed. by John O’Brian. Vol. 4. The Collected Essays and Criticism. Essay originally delivered as a radio lecture in 1960 and published in revised form in 1961. Chicago: University of Chicago Press, pp. 85–93.
- Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger (June 2017). “On Calibration of Modern Neural Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1321–1330. URL: <https://proceedings.mlr.press/v70/guo17a.html>.
- Gurobi Optimization, LLC (2021). *Gurobi Optimizer Reference Manual*. URL: <https://www.gurobi.com>.
- Ha, David, Andrew Dai, and Quoc V. Le (2016). *HyperNetworks*. arXiv: 1609.09106 [cs.LG].
- Halawi, Danny, Fred Zhang, Chen Yueh-Han, and Jacob Steinhardt (2024). “Approaching Human-Level Forecasting with Language Models”. In: *arXiv preprint arXiv:2402.18563*.
- Havrilla, Alex, Andrew Dai, Laura O’Mahony, Koen Oostermeijer, Vera Zisler, Alon Albalak, Fabrizio Milo, Sharath Chandra Raparthy, Kanishk Gandhi, Baber Abbasi, et al. (2024). “Surveying the Effects of Quality, Diversity, and Complexity in Synthetic Data From Large Language Models”. In: *arXiv preprint arXiv:2412.02980*. DOI: 10.48550/arXiv.2412.02980.
- He, Haoran, Shangbin Feng, Zhaoxuan Lin, Zhiye Zuo, Yu Hao, Fei Xia, Lingpeng Kong, Zichang Huang, Xiang Yue, Yifei Zhou, and Yizhong Wang (2025). *Rewarding the Unlikely: Lifting GRPO Beyond Local Mode-Seeking*. arXiv: 2506.02355 [cs.CL]. URL: <https://arxiv.org/abs/2506.02355>.
- Hendrycks, Dan, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt (2021). *Measuring Coding Challenge Competence With APPS*. arXiv: 2105.09938 [cs.SE]. URL: <https://arxiv.org/abs/2105.09938>.
- Henry, Julien, Mihail Asavoaie, David Monniaux, and Claire Mar˘za (2014). “How to Compute Worst-Case Execution Time by Optimization modulo Theory and a Clever Encoding of Program Semantics”. In: LCTES ’14. Association for Computing Machinery, pp. 43–52.

- Hessel, Jack, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi (2022). *CLIP-Score: A Reference-free Evaluation Metric for Image Captioning*. arXiv: 2104.08718 [cs.CV].
- Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (2017). “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *Advances in Neural Information Processing Systems*. Vol. 30. URL: <https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html>.
- (2018). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. arXiv: 1706.08500 [cs.LG].
- Hinton, Geoffrey (2015). “Distilling the Knowledge in a Neural Network”. In: *arXiv preprint arXiv:1503.02531*.
- Hopfield, John J. and David W. Tank (2004). ““Neural” computation of decisions in optimization problems”. In: *Biological Cybernetics* 52, pp. 141–152.
- Howell, David C (1992). *Statistical methods for psychology*. PWS-Kent Publishing Co.
- Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv: 2106.09685 [cs.CL].
- Huang, Chengsong, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin (2024). *LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition*. arXiv: 2307.13269 [cs.CL].
- Huang, Sheng-Wei, Che-Tsung Lin, Shu-Ping Chen, Yen-Yi Wu, Po-Hao Hsu, and Shang-Hong Lai (2018). “Auggan: Cross domain adaptation with gan-based data augmentation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 718–731.
- Hubert, Thomas, Raj Mehta, Laurent Sartran, et al. (2026). “Olympiad-level Formal Mathematical Reasoning with Reinforcement Learning”. In: *Nature* 651, pp. 607–613. DOI: 10.1038/s41586-025-09833-y.
- Ippolito, Daphne, Ann Yuan, Andy Coenen, and Sehmon Burnam (2022). *Creative Writing with an AI-Powered Writing Assistant: Perspectives from Professional Writers*. DOI: 10.48550/arXiv.2211.05030. arXiv: 2211.05030 [cs.HC]. URL: <https://arxiv.org/abs/2211.05030>.
- Jain, Naman, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica (2024a). “LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code”. In: *arXiv preprint arXiv:2403.07974*. DOI: 10.48550/arXiv.2403.07974.
- Jain, Naman, Manish Shetty, King Han, Koushik Sen, and Ion Stoica (2024b). “R2E: Turning any Github Repository into a Programming Agent Environment”. In: *Proceedings of the 41st International Conference on Machine Learning*. URL: <https://proceedings.mlr.press/v235/jain24b.html>.
- Jang, Joel, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu (2023). *Personalized*

- Soups: Personalized Large Language Model Alignment via Post-hoc Parameter Merging*. arXiv: 2310.11564 [cs.CL].
- Jelassi, Samy, Mujin Kwun, Rosie Zhao, Yuanzhi Li, Nicolo Fusi, Yilun Du, Sham M. Kakade, and Carles Domingo-Enrich (2026). “Matching Features, Not Tokens: Energy-Based Fine-Tuning of Language Models”. In: *arXiv preprint arXiv:2603.12248*.
- Joshi, Mandar, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer (2017). “TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1601–1611. DOI: 10.18653/v1/P17-1147. URL: <https://aclanthology.org/P17-1147/>.
- Karras, Tero, Miika Aittala, Timo Aila, and Samuli Laine (2022). *Elucidating the Design Space of Diffusion-Based Generative Models*. arXiv: 2206.00364 [cs.CV].
- Katz, Guy, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer (2017). “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *International conference on computer aided verification*. Springer, pp. 97–117.
- Kipf, Thomas N and Max Welling (2016). “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907*.
- Kovácsnai, Gergely, Csaba Biró, and Balázs Erdélyi (2017). “Generating Optimal Scheduling for Wireless Sensor Networks by Using Optimization Modulo Theories Solvers.” In: *SMT*, pp. 15–27.
- Kraft, Amy (Mar. 25, 2016). *Microsoft shuts down AI chatbot after it turned into a Nazi*. CBS News. URL: <https://www.cbsnews.com/news/microsoft-shuts-down-ai-chatbot-after-it-turned-into-racist-nazi/> (visited on 04/10/2026).
- Kuhn, Lorenz, Yarin Gal, and Sebastian Farquhar (2022). “Clam: Selective clarification for ambiguous questions with large language models”. In: *arXiv preprint arXiv:2212.07769*.
- Lanchantin, Jack, Angelica Chen, Shehzaad Dhuliawala, Ping Yu, Jason Weston, Sainbayar Sukhbaatar, and Ilia Kulikov (2025). “Diverse preference optimization”. In: *arXiv preprint arXiv:2501.18101*.
- Leavis, F. R. (1948). *The Great Tradition: George Eliot, Henry James, Joseph Conrad*. London: Chatto & Windus.
- Leino, K Rustan M (2010). “Dafny: An automatic program verifier for functional correctness”. In: *International conference on logic for programming artificial intelligence and reasoning*. Springer, pp. 348–370.
- Lemieux, Caroline, Rohan Padhye, and Koushik Sen (2023). “CodaMOSA: Escaping Coverage Plateaus in Test Generation with Pre-Trained Large Language Models”. In: *Proceedings of the 45th International Conference on Software Engineering*, pp. 919–931. DOI: 10.1109/ICSE48619.2023.00085. URL: <https://doi.org/10.1109/ICSE48619.2023.00085>.
- Leofante, Francesco, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella (2017). “On the synthesis of guaranteed-quality plans for robot fleets in logistics

- scenarios via optimization modulo theories”. In: *2017 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, pp. 403–410.
- Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv: 2005.11401 [cs.CL].
- Lewkowycz, Aitor, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra (2022). “Solving Quantitative Reasoning Problems with Language Models”. In: *arXiv preprint arXiv:2206.14858*. DOI: 10.48550/arXiv.2206.14858.
- Li, Daiqing, Aleks Kamko, Ehsan Akhgari, Ali Sabet, Linmiao Xu, and Suhail Doshi (2024). *Playground v2.5: Three Insights towards Enhancing Aesthetic Quality in Text-to-Image Generation*. arXiv: 2402.17245 [cs.CV].
- Li, Hongyu, Zhijian Liu, Yixuan Lai, Jing Huang, Shiyu Wei, Yushi Su, Hoifung Poon, and Tao Yu (2024). “Revisiting Catastrophic Forgetting in Large Language Model Tuning”. In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19753–19771. URL: <https://aclanthology.org/2024.emnlp-main.1094/>.
- Li, Yujia, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals (Dec. 2022). “Competition-level code generation with AlphaCode”. In: *Science* 378.6624, pp. 1092–1097. ISSN: 1095-9203. DOI: 10.1126/science.abq1158. URL: <http://dx.doi.org/10.1126/science.abq1158>.
- Lin, Bin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, Shen Li, Zhigang Ji, Tao Xie, Yong Li, and Wei Lin (2024). *Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache*. arXiv: 2401.02669 [cs.DC]. URL: <https://arxiv.org/abs/2401.02669>.
- Lin, Jimmy, Ronak Pradeep, Tommaso Teofili, and Jasper Xian (2023). *Vector Search with OpenAI Embeddings: Lucene Is All You Need*. arXiv: 2308.14963 [cs.IR].
- Lin, Stephanie, Jacob Hilton, and Owain Evans (2021). “TruthfulQA: Measuring How Models Mimic Human Falsehoods”. In: *arXiv preprint arXiv:2109.07958*.
- Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár (2015). *Microsoft COCO: Common Objects in Context*. arXiv: 1405.0312 [cs.CV].
- Liu, Hao, Matei Zaharia, and Pieter Abbeel (2023). *Ring Attention with Blockwise Transformers for Near-Infinite Context*. arXiv: 2310.01889 [cs.CL]. URL: <https://arxiv.org/abs/2310.01889>.

- Liu, Jerry (Nov. 2022). *LlamaIndex*. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index.
- Liu, Jiawei, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang (2023). “Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”. In: *Advances in Neural Information Processing Systems*. Vol. 36. URL: <https://arxiv.org/abs/2305.01210>.
- Liu, Nan, Yilun Du, Shuang Li, Joshua B. Tenenbaum, and Antonio Torralba (2023). *Unsupervised Compositional Concepts Discovery with Text-to-Image Generative Models*. arXiv: 2306.05357 [cs.CV].
- Lou, Aaron, Chenlin Meng, and Stefano Ermon (2024). “Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution”. In: *Proceedings of the 41st International Conference on Machine Learning*.
- Lowerre, Bruce P and B Raj Reddy (1976). “Harpy, a connected speech recognition system”. In: *The Journal of the Acoustical Society of America* 59.S1, S97–S97.
- Lu, Cheng, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu (2023). *DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models*. URL: <https://openreview.net/forum?id=4vGwQqviud5>.
- Luo, Michael, Justin Wong, Brandon Trabucco, Yanping Huang, Joseph E. Gonzalez, Zhifeng Chen, Ruslan Salakhutdinov, and Ion Stoica (2024). *Stylus: Automatic Adapter Selection for Diffusion Models*. Manuscript.
- Ma, Tengyu (2023). *Vectorize your data to gear up your AI stack*. URL: <https://docs.voyageai.com/docs/embeddings>.
- Madaan, Aman, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark (2023). “SELF-REFINE: Iterative Refinement with Self-Feedback”. In: *arXiv preprint arXiv:2303.17651*. DOI: 10.48550/arXiv.2303.17651.
- Mahrooghi, Ilia, Aryo Lotfi, and Emmanuel Abbe (2026). “Goldilocks RL: Tuning Task Difficulty to Escape Sparse Rewards for Reasoning”. In: *arXiv preprint arXiv:2602.14868*. DOI: 10.48550/arXiv.2602.14868.
- Maier, Justin (2022). *The home of open-source Generative AI*. URL: <https://civitai.com/>.
- Malthus, Thomas Robert (1798). *An Essay on the Principle of Population*. J. Johnson, London.
- Mangrulkar, Sourab, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan (2022). *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*. <https://github.com/huggingface/peft>.
- Manolios, Panagiotis, Jorge Pais, and Vasilis Papavasileiou (2015). “The Inez Mathematical Programming Modulo Theories Framework”. In: *Computer Aided Verification*. Ed. by Daniel Kroening and Corina S. Păsăreanu. Springer International Publishing, pp. 53–69.
- McAllister, David, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa (2025). *Flow Matching Policy Gradients*. arXiv: 2507.21053 [cs.LG]. URL: <https://arxiv.org/abs/2507.21053>.

- Meng, Rui, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz (2024). *SFR-Embedding-Mistral: Enhance Text Retrieval with Transfer Learning*. Salesforce AI Research Blog. URL: <https://blog.salesforceairesearch.com/sfr-embedded-mistral/>.
- Merrill, William and Ashish Sabharwal (2023). “The Expressive Power of Transformers with Chain of Thought”. In: *arXiv preprint arXiv:2310.07923*. DOI: 10.48550/arXiv.2310.07923.
- Meyerson, Elliot, Mark J Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K Hoover, and Joel Lehman (2023). “Language model crossover: Variation through few-shot prompting”. In: *arXiv preprint arXiv:2302.12170*.
- Min, Sewon, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer (2020). “AmbigQA: Answering ambiguous open-domain questions”. In: *arXiv preprint arXiv:2004.10645*.
- Mirhoseini, Azalia, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. (2020). “Chip placement with deep reinforcement learning”. In: *arXiv preprint arXiv:2004.10746*.
- Mohammadshahi, Alireza, Arshad Rafiq Shaikh, and Majid Yazdani (2024). *Routoo: Learning to Route to Large Language Models Effectively*. arXiv: 2401.13979 [cs.CL]. URL: <https://arxiv.org/abs/2401.13979>.
- Moon, Suhong, Marwa Abdulhai, Minwoo Kang, Joseph Suh, Widyadewi Soedarmadji, Eran Kohen Behar, and David M. Chan (2024). “Virtual Personas for Language Models via an Anthology of Backstories”. In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, pp. 19864–19897. DOI: 10.18653/v1/2024.emnlp-main.1110. URL: <https://aclanthology.org/2024.emnlp-main.1110/>.
- Moritz, Philipp, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica (2018). “Ray: A Distributed Framework for Emerging AI Applications”. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 561–577.
- Morris, G Elliott (2022). *Strength in Numbers: How Polls Work and why We Need Them*. WW Norton & Company.
- Moura, Leonardo de and Nikolaj Bjørner (2008). “Z3: An efficient SMT solver”. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 337–340.
- Mouret, Jean-Baptiste and Jeff Clune (2015). “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909*.
- Naeem, Muhammad Ferjad, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo (2020). “Reliable Fidelity and Diversity Metrics for Generative Models”. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 7176–7185. URL: <http://proceedings.mlr.press/v119/naeem20a.html>.
- Nair, Vinod, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O’Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et

- al. (2020). “Solving Mixed Integer Programs Using Neural Networks”. In: *arXiv preprint arXiv:2012.13349*.
- Ng, Nathan, Kyunghyun Cho, and Marzyeh Ghassemi (2020). “SSMBA: Self-supervised manifold based data augmentation for improving out-of-domain robustness”. In: *arXiv preprint arXiv:2009.10195*.
- Novikov, Alexander, Ngan Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehra-bian, et al. (2025). “AlphaEvolve: A coding agent for scientific and algorithmic discovery”. In: *arXiv preprint arXiv:2506.13131*.
- Ong, Isaac, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica (2024). *RouteLLM: Learning to Route LLMs with Preference Data*. arXiv: 2406.18665 [cs.LG]. URL: <https://arxiv.org/abs/2406.18665>.
- OpenAI et al. (2024). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL]. URL: <https://arxiv.org/abs/2303.08774>.
- Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe (2022a). “Training language models to follow instructions with human feedback”. In: *arXiv preprint arXiv:2203.02155*. DOI: 10.48550/arXiv.2203.02155.
- (2022b). *Training language models to follow instructions with human feedback*. arXiv: 2203.02155 [cs.CL].
- Padmakumar, Vishakh, Yueh-Han Chen, Jane Pan, Valerie Chen, and He He (2026). “Measuring LLM Novelty As The Frontier Of Original And High-Quality Output”. In: *International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=i7QNKZioN6>.
- Pahuja, Vardaan, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Awadallah (2025). “Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents”. In: *arXiv preprint arXiv:2502.11357*.
- Pascal, Blaise (1859). *The Provincial Letters*. Trans. by Thomas M’Crie. Letter XVI; original French letter dated December 4, 1656. Edinburgh: Johnstone and Hunter.
- Petryk, Suzanne, David M. Chan, Anish Kachinthaya, Haodi Zou, John Canny, Joseph E. Gonzalez, and Trevor Darrell (2024). “ALOHa: A New Measure for Hallucination in Captioning Models”. In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*. Mexico City, Mexico: Association for Computational Linguistics, pp. 342–357. DOI: 10.18653/v1/2024.naacl-short.30. URL: <https://aclanthology.org/2024.naacl-short.30/>.
- Pietsch, Malte, Timo Möller, Bogdan Kostic, Julian Risch, Massimiliano Pippi, Mayank Jobanputra, Sara Zanzottera, Silvano Cerza, Vladimir Blagojevic, Thomas Stadelmann,

- Tanay Soni, and Sebastian Lee (Nov. 2019). *Haystack: the end-to-end NLP framework for pragmatic builders*. URL: <https://github.com/deepset-ai/haystack>.
- Pillutla, Krishna, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui (2021). “MAUVE: Measuring the Gap Between Neural Text and Human Text using Divergence Frontiers”. In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 4816–4828. URL: <https://arxiv.org/abs/2102.01454>.
- Platt, John (June 2000). “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods”. In: *Adv. Large Margin Classif.* 10.
- Podell, Dustin, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach (2023). *SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis*. arXiv: 2307.01952 [cs.CV].
- Putta, Pranav, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov (2024). “Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents”. In: *arXiv preprint arXiv:2408.07199*.
- Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever (2021). *Learning Transferable Visual Models From Natural Language Supervision*. arXiv: 2103.00020 [cs.CV].
- Raventós, Allan, Mansheej Paul, Feng Chen, and Surya Ganguli (2024). “Pretraining task diversity and the emergence of non-bayesian in-context learning for regression”. In: *Advances in Neural Information Processing Systems* 36.
- Reimers, Nils (2023). *Say Goodbye to Irrelevant Search Results: Cohere Rerank Is Here*. URL: <https://txt.cohere.com/rerank/>.
- Reimers, Nils and Iryna Gurevych (Nov. 2019). “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. URL: <https://arxiv.org/abs/1908.10084>.
- Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer (2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. arXiv: 2112.10752 [cs.CV].
- Rummery, Gavin A. and Mahesan Niranjan (1994). *On-line Q-learning using connectionist systems*. Technical Report CUED/F-INFENG/TR 166. University of Cambridge, Department of Engineering. URL: https://mi.eng.cam.ac.uk/reports/abstracts/robotics/rummery_tr166.html.
- Sajjadi, Mehdi S. M., Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly (2018). “Assessing Generative Models via Precision and Recall”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, pp. 5234–5243. URL: <https://proceedings.neurips.cc/paper/2018/hash/f7696a9b362ac5a51c3dc8f098b73923-Abstract.html>.

- Samvelyan, Mikayel, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram H Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, et al. (2024). “Rainbow teaming: Open-ended generation of diverse adversarial prompts”. In: *arXiv preprint arXiv:2402.16822*.
- Sebastiani, Roberto and Patrick Trentin (2015). “OptiMathSAT: A Tool for Optimization Modulo Theories”. In: *Computer Aided Verification*. Ed. by Daniel Kroening and Corina S. Păsăreanu. Springer International Publishing, pp. 447–454.
- Selsam, Daniel, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill (2019). “Learning a SAT Solver from Single-Bit Supervision”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.
- Shannon, Claude E. (1948). “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3, pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- Sharmanska, Viktoriia, Lisa Anne Hendricks, Trevor Darrell, and Novi Quadrianto (2020). “Contrastive examples for addressing the tyranny of the majority”. In: *arXiv preprint arXiv:2004.06524*.
- Shinn, Noah, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao (2023). *Reflexion: Language Agents with Verbal Reinforcement Learning*. arXiv: 2303.11366 [cs.AI].
- Shumailov, Ilia, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson (2023). “The Curse of Recursion: Training on Generated Data Makes Models Forget”. In: *arXiv preprint arXiv:2305.17493*. DOI: 10.48550/arXiv.2305.17493.
- Simpson, Edward H (1951). “The interpretation of interaction in contingency tables”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 13.2, pp. 238–241.
- Solar-Lezama, Armando, Liviu Tancau, Rastislav Bodik, Sanjit A. Seshia, and Vijay A. Saraswat (2006). “Combinatorial Sketching for Finite Programs”. In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 404–415. DOI: 10.1145/1168857.1168907. URL: <https://doi.org/10.1145/1168857.1168907>.
- Song, Xuefeng, Zihan Li, Yuxiang Zhang, Qi Zhang, and Xiang Chen (2025). *Outcome-Refining Process Supervision for Code Generation*. arXiv: 2501.13918 [cs.CL]. URL: <https://arxiv.org/abs/2501.13918>.
- Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna (2016). “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308. URL: <https://doi.org/10.1109/CVPR.2016.308>.
- Tang, Yunhao, Pranav Srinivasan, Yuxuan Song, Yiyang Zhou, Chunting Zhou, Zhiheng Huang, Jiacheng Huang, Wen-tau Yih, Qiying Yu, Jingjing Liu, and Hao Zhou (2025). *Inference-Time Scaling for Generalist Reward Modeling*. arXiv: 2504.02495 [cs.CL]. URL: <https://arxiv.org/abs/2504.02495>.

- Team, Gemini (2023). *Gemini: A Family of Highly Capable Multimodal Models*. arXiv: 2312.11805 [cs.CL].
- Team, OpenEvolve (2025). *OpenEvolve: Open-Source Language Model Evolution for Code and Search*. Project reference. URL: <https://github.com/codelion/openevolve>.
- Tian, Katherine, Eric Mitchell, Huaxiu Yao, Christopher D. Manning, and Chelsea Finn (2023). *Fine-tuning Language Models for Factuality*. arXiv: 2311.08401 [cs.CL]. URL: <https://arxiv.org/abs/2311.08401>.
- Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample (2023a). *LLaMA: Open and Efficient Foundation Language Models*. arXiv: 2302.13971 [cs.CL].
- Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom (2023b). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. arXiv: 2307.09288 [cs.CL].
- Towse, John N, Tobias Loetscher, and Peter Brugger (2014). “Not all numbers are equal: preferences and biases among children and adults when generating random sequences”. In: *Frontiers in psychology* 5, p. 19.
- Trabucco, Brandon, Kyle Doherty, Max Gurinas, and Ruslan Salakhutdinov (2023a). *Effective Data Augmentation With Diffusion Models*. arXiv: 2302.07944 [cs.CV].
- (2023b). “Effective data augmentation with diffusion models”. In: *arXiv preprint arXiv:2302.07944*.
- Trabucco, Brandon, Gunnar Sigurdsson, Robinson Piramuthu, and Ruslan Salakhutdinov (2025). “InSTA: Towards Internet-Scale Training For Agents”. In: *arXiv preprint arXiv:2502.06776*.
- United States Congress (1952a). *35 U.S.C. §101: Inventions Patentable*. United States Code. Current version accessed via Office of the Law Revision Counsel, U.S. House of Representatives. URL: <https://uscode.house.gov/view.xhtml?path=/prelim@title35/part2/chapter10&edition=prelim>.
- (1952b). *35 U.S.C. §102: Conditions for Patentability; Novelty*. United States Code. As amended, including changes under the Leahy-Smith America Invents Act. URL: <https://uscode.house.gov/view.xhtml?path=/prelim@title35/part2/chapter10&edition=prelim>.

- `//uscode.house.gov/view.xhtml?path=/prelim@title35/part2/chapter10&edition=prelim`.
- United States Congress (1952c). *35 U.S.C. §103: Conditions for Patentability; Non-obvious Subject Matter*. United States Code. As amended, including changes under the Leahy-Smith America Invents Act. URL: <https://uscode.house.gov/view.xhtml?path=/prelim@title35/part2/chapter10&edition=prelim>.
- Verhulst, PF (1838). “Notice on the law that the population follows in its growth”. In: *Corresp Math Phys* 10, pp. 113–26.
- Vijayakumar, Ashwin K, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra (2016). “Diverse beam search: Decoding diverse solutions from neural sequence models”. In: *arXiv preprint arXiv:1610.02424*.
- Walder, Christian and Darius Karkhanis (2025). *Optimizing Language Models for Pass@k*. arXiv: 2502.06257 [cs.LG]. URL: <https://arxiv.org/abs/2502.06257>.
- Wallace, Eric, Kai Yuanqing Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel (2024). “The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions”. In: *arXiv preprint arXiv:2404.13208*. DOI: 10.48550/arXiv.2404.13208.
- Wang, Hanqing, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun (2024). *LoRA-Flow: Dynamic LoRA Fusion for Large Language Models in Generative Tasks*. arXiv: 2402.11455 [cs.CL].
- Wang, Xinyi, Yulia Tsvetkov, Sebastian Ruder, and Graham Neubig (Nov. 2021). “Efficient Test Time Adapter Ensembling for Low-resource Language Varieties”. In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Ed. by Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih. Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 730–737. DOI: 10.18653/v1/2021.findings-emnlp.63. URL: <https://aclanthology.org/2021.findings-emnlp.63>.
- Wang, Xuezhi, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou (2022). “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *arXiv preprint arXiv:2203.11171*. DOI: 10.48550/arXiv.2203.11171.
- Watkins, Christopher J. C. H. and Peter Dayan (1992). “Technical Note: Q-Learning”. In: *Machine Learning* 8.3–4, pp. 279–292. DOI: 10.1023/A:1022676722315. URL: <https://link.springer.com/article/10.1023/A:1022676722315>.
- Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou (2022). “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 24824–24837. URL: <https://arxiv.org/abs/2201.11903>.
- Wei, Jason and Kai Zou (2019). “Eda: Easy data augmentation techniques for boosting performance on text classification tasks”. In: *arXiv preprint arXiv:1901.11196*.
- Wilson, Chris (May 2016). “Baby Names: Why There Are More for Girls Than Boys”. In: *Time*. URL: <https://time.com/4322881/baby-names-girls-boys/> (visited on 09/28/2024).

- Wimmer, Lisa, Yusuf Sale, Paul Hofman, Bernd Bischl, and Eyke Hüllermeier (2023). “Quantifying Aleatoric and Epistemic Uncertainty in Machine Learning: Are Conditional Entropy and Mutual Information Appropriate Measures?” In: *arXiv preprint arXiv:2209.03302*. DOI: 10.48550/arXiv.2209.03302. URL: <https://arxiv.org/abs/2209.03302>.
- Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush (2020). *HuggingFace’s Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs.CL].
- Wong, Justin, Yury Orlovskiy, Alexander Shypula, Michael Luo, Sanjit A. Seshia, and Joseph E. Gonzalez (2025). “SimpleStrat: Diversifying Language Model Generation with Stratification”. In: *Advances in Neural Information Processing Systems*. Vol. 38. URL: https://proceedings.neurips.cc/paper_files/paper/2025/hash/a99cbe29888dc87029faa80eb10d2b9-Abstract-Conference.html.
- Wu, Yaoxin, Wen Song, Zhiguang Cao, and Jie Zhang (2021). “Learning Large Neighborhood Search Policy for Integer Programming”. In: *Advances in Neural Information Processing Systems*.
- Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu (2021). “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1, pp. 4–24.
- Xu, Jingjing, Yuechen Wang, Duyu Tang, Nan Duan, Pengcheng Yang, Qi Zeng, Ming Zhou, and Xu Sun (Nov. 2019). “Asking Clarification Questions in Knowledge-Based Question Answering”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Hong Kong, China: Association for Computational Linguistics, pp. 1618–1629. DOI: 10.18653/v1/D19-1172. URL: <https://aclanthology.org/D19-1172>.
- Yan, Qi, Raihan Seraj, Jiawei He, Lili Meng, and Tristan Sylvain (2023). “Autocast++: Enhancing world event prediction with zero-shot ranking-based context retrieval”. In: *arXiv preprint arXiv:2310.01880*.
- Yang, Kevin, Dan Klein, Nanyun Peng, and Yuandong Tian (July 2023). “DOC: Improving Long Story Coherence With Detailed Outline Control”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, pp. 3378–3465. DOI: 10.18653/v1/2023.acl-long.190. URL: <https://aclanthology.org/2023.acl-long.190/>.
- Yang, Zongheng, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica (2022). “Balsa: Learning a query optimizer without expert demonstrations”. In: *arXiv preprint arXiv:2201.01441*.

- Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan (2024). “Tree of thoughts: Deliberate problem solving with large language models”. In: *Advances in Neural Information Processing Systems* 36.
- Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao (2022). “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *arXiv preprint arXiv:2210.03629*. DOI: 10.48550/arXiv.2210.03629.
- Yu, Jiahui, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu (2022). *Scaling Autoregressive Models for Content-Rich Text-to-Image Generation*. arXiv: 2206.10789 [cs.CV].
- Yu, Qiyang, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang (2025). “DAPO: An Open-Source LLM Reinforcement Learning System at Scale”. In: *arXiv preprint arXiv:2503.14476*. DOI: 10.48550/arXiv.2503.14476. URL: <https://arxiv.org/abs/2503.14476>.
- Yu, Tao, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev (2019). *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. arXiv: 1809.08887 [cs.CL]. URL: <https://arxiv.org/abs/1809.08887>.
- Yu, Yue, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang (2023a). “Large language model as attributed training data generator: A tale of diversity and bias”. In: *Advances in Neural Information Processing Systems* 36, pp. 55734–55784.
- (2023b). “Large Language Model as Attributed Training Data Generator: A Tale of Diversity and Bias”. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 55734–55784. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/2e10b2c2e1aa4f8083c37dfe269873f8-Abstract-Conference.html.
- Zhang, Edwin, Vincent Zhu, Naomi Saphra, Anat Kleiman, Benjamin L Edelman, Milind Tambe, Sham M Kakade, and Eran Malach (2024). “Transcendence: Generative Models Can Outperform The Experts That Train Them”. In: *arXiv preprint arXiv:2406.11741*.
- Zhang, Jiayi, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu (2024). *AFlow: Automating Agentic Workflow Generation*. arXiv: 2410.10762 [cs.AI]. URL: <https://arxiv.org/abs/2410.10762>.
- Zhao, Andrew, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang (2025). “Absolute Zero: Reinforced Self-play

- Reasoning with Zero Data”. In: *arXiv preprint arXiv:2505.03335*. DOI: 10.48550/arXiv.2505.03335.
- Zhao, Ziyu, Leilei Gan, Guoyin Wang, Wangchunshu Zhou, Hongxia Yang, Kun Kuang, and Fei Wu (2024). *LoraRetriever: Input-Aware LoRA Retrieval and Composition for Mixed Tasks in the Wild*. arXiv: 2402.09997 [cs.AI].
- Zhong, Ming, Yelong Shen, Shuohang Wang, Yadong Lu, Yizhu Jiao, Siru Ouyang, Donghan Yu, Jiawei Han, and Weizhu Chen (2024). *Multi-LoRA Composition for Image Generation*. arXiv: 2402.16843 [cs.CV].
- Zhong, Victor, Caiming Xiong, and Richard Socher (2017). “Seq2sql: Generating structured queries from natural language using reinforcement learning”. In: *arXiv preprint arXiv:1709.00103*.
- Zhou, Andy, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang (2023). “Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models”. In: *arXiv preprint arXiv:2310.04406*. DOI: 10.48550/arXiv.2310.04406.
- Zhu, Hang, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin (2021). “Network Planning with Deep Reinforcement Learning”. In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM ’21. Association for Computing Machinery, pp. 258–271.
- Zhu, Yaoming, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu (2018). “Texygen: A Benchmarking Platform for Text Generation Models”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 1097–1100. DOI: 10.1145/3209978.3210080.
- Zhuge, Mingchen, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber (2024). *Language Agents as Optimizable Graphs*. arXiv: 2402.16823 [cs.AI].
- Zou, Andy, Tristan Xiao, Ryan Jia, Joe Kwon, Mantas Mazeika, Richard Li, Dawn Song, Jacob Steinhardt, Owain Evans, and Dan Hendrycks (2022). “Forecasting future world events with neural networks”. In: *Advances in Neural Information Processing Systems* 35, pp. 27293–27305.

Appendix A

Ashera: Neural Optimization Modulo Theory

A.1 Introduction

In the work described below we study the challenge of solving optimization problems that have many redundant solutions by construction. This occurs due to symmetry. This is classically demonstrated by noting the slowness of the naive encoding for verifying the pigeonhole principle. The challenge is the solver cannot be confident there exists no assignment of pigeon to pigeonhole without trying each one. In practice, one of the key values of an expert is the art of encoding to avoid symmetries (symmetry-breaking). For the pigeonhole principle the canonical fix is requiring the pigeons of larger index to occupy larger index holes. Anecdotally, a similar trick in the torch compiler improved solve times from several days to hours.

In the following sections, we detail work on symmetric problems at the intersection of Satisfiability Modulo Theory (SMT) and Integer Linear Programming (ILP). We introduce a learning-based solution for warmstarting and combining the strengths of SMT and ILP for this problem.

Analysis of worst-case execution time (WCET) of software and hardware requires not only identifying valid executions of a program but also finding a provably slowest execution. Existing solvers for Satisfiability Modulo Theories (SMT) have found remarkable practical success in providing guarantees on feasibility problems in domains such as software and hardware verification (Dutertre et al., 2006; Moura et al., 2008; Leino, 2010; Katz et al., 2017). However, there has been limited progress on Optimization Modulo Theories (OMT) problems like WCET (Henry et al., 2014) that require provably cost-optimal solutions not just feasible solutions. In these scenarios, ranging from software security (Bertolissi et al., 2018; Henry et al., 2014) to scheduling and planning (Leofante et al., 2017; Kovásznai et al., 2017), require OMT to support an objective function.

While OMT is a more general optimization framework, solving it becomes more difficult.

There are two major challenges. First, existing approaches to OMT either use the Big M approximation to reduce the problem to Integer Linear Programming ILP (Gurobi Optimization, LLC, 2021) or do iterative calls to SMT (Sebastiani et al., 2015; Bjørner et al., 2015), which scales poorly. Second, the solvers are designed to be general purpose and problem agnostic. To address poor scalability in a per application basis, experts manually tune hyperparameters and problem encodings which often improves the solving time from days to a matter of minutes. Such solutions, while feasible, are often expensive and time-consuming, and ignore the past experience of solved problems.

In this paper, we present a codesigned neural OMT solver, called *Ashera* to address these two challenges by introducing two components: *an OMT engine* and *a neural diver*.

First, Ashera’s OMT engine iterates between an SMT solver and an optimality-aware theory solver, TS*. Because SMT decomposes disjunctions into guided search over conjunctive cases, we can avoid using Big M. In SMT, theory solvers (TS) are domain specific subsolvers for a conjunction of constraints. For instance, for LIA theory specific properties to prune and speed up the enumeration such as $x > 0 \wedge x < 1$ indicates no integer x is feasible. We introduce *optimality-aware theory solver*, TS*, which additionally has awareness of the optimization objective. More concretely for Linear Integer Arithmetic (LIA), we use Z3 for quantifier-free LIA SMT and Gurobi for ILP.

Second, our neural diver learns to directly predict promising partial assignments that lead to a better cost when extended to full assignments. We do this by training a Graph Neural Network (GNN) on existing solutions of related problems, extracting the knowledge from past experience. Then, the same SMT solver used in our OMT engine is used to check the validity of these candidate assignments to guarantee soundness. Our approach is inspired by the recent success of (Nair et al., 2020) that applies similar ideas to Mixed Integer Linear Programming (MILP). The efficiency of existing solvers can be substantially improved by transferring knowledge of solving one problem instance to another similar one. We observe that in practice, optimizations are done repeatedly on similar instances drawn from an underlying distribution, for example on a monthly basis for network planning (H. Zhu et al., 2021). As such, the problems to be solved regularly differ only slightly, but the optimization procedure needs to be redone, as modified constraints change the solution space. As a result, learning from previous solved problems can speed up the next optimization without any manual human input. Our approach differs from existing work that learn end-to-end a solver to replace existing systems (e.g., learned query optimization (Z. Yang et al., 2022) and chip placement (Mirhoseini et al., 2020)). In contrast, we co-design our OMT approach to contribute a new generalized solver while exploiting opportunities for learning-based components.

As existing benchmarks provide dozens of diverse OMT problems, we build a benchmark for evaluating learning-based OMT solvers. Our benchmark provides over 58,000 OMT problems from two families of problems: task scheduling and multi-agent Traveling Salesman Problem (TSP). Using this benchmark we demonstrate that Ashera can learn on problems with less variables and constraints and generalize to larger problems within the same problem class.

Our work makes the following contributions:

- A benchmark for evaluating learning-based OMT solving with problem families in scheduling and multi-agent traveling salesman problems.
- To our knowledge, we present the first learning-based OMT solver in Ashera. The solver focuses on support for Linear Integer Arithmetic (LIA), but the framework generalizes to other theories.
- For scheduling, Ashera in contrast is **3x faster** and solves **three more problems** than the widely-used commercial solver, Gurobi, on problems with 10 and 11 tasks, respectively. OptiMathSAT and Z3 are unable to solve any problems of 11 tasks within 1 hour timeout.
- Ashera is **18% faster** than OptiMathSAT on multiagent TSP with 15 waypoints, where Gurobi and Z3 timeout.

A.2 Related works

Solvers for OMT

The last decade has produced two prominent Optimization Modulo Theory (OMT) solvers: 1) OptiMathSAT (Sebastiani et al., 2015), which applies a binary search approach to discovering the optimal solution, and 2) *vZ* (Bjørner et al., 2015), which iteratively uses SMT to find a strictly better feasible solution and locally improve it by coordinate-wise search for strictly improving boundary solutions. OptiMathSAT further uses sorting networks to decouple the Boolean reasoning from the arithmetic solving. *vZ* exploits carefully engineered MaxSMT and pseudo Boolean solvers to provide competitive performance when the OMT problem lies within these domains. Neither of these works perform well at scale and applications are largely dominated by ILP approximations.

Similar to OMT, Inez (Manolios et al., 2015), a solver for Mathematical Programming Modulo Theory, integrates ILP solvers with solvers for first order theories. However, Inez relies on the Big M encoding or built in constraint handlers in ILP solvers to reason about disjunctions and instead focus on extending ILP solvers to support uninterpreted functions and support for user-provided axioms. Ashera explicitly reasons about disjunctions arising from the logic structure of OMT problems.

Neural Guidance for Combinatorial Optimization

The Integer Linear Program (ILP) is a well-studied class of optimization problems in large part due to its prominence in operations research, computer vision, scheduling, and other domains. Branch and bound, one particular tree search algorithm, is the best known approach for solving ILPs. Branch and bound is able to incrementally establish a lower and

upper bound via a feasible point and a solution to a linear relaxation of the ILP respectively until tightness is achieved.

In (Balcan et al., 2018), the authors propose to replace empirical heuristics for selecting variables to branch with a data-driven methodology that achieves provable complexity bounds. The authors show that they can learn a convex combination of scoring rules (which each determine the ordering of node branching) that is nearly optimal in expectation over a distribution of original ILPs. Furthermore, the optimization process over scoring rules can be seen as performing empirical risk minimization (ERM) of the original optimization objective subject to the input variable constraints.

Another approach taken by Wu et al. (Y. Wu et al., 2021) is to train a model to reconstruct locally optimal solutions. This model is trained by taking feasible solutions and resolving the optimization with a random subset of variables masked out. The model learns how to improve solutions locally and recognize strategies that generalize to adjacent regions in the feasible set. However, this approach fails to recognize that due to combinatorial explosion subproblem optimization is often negligible and the real challenge is identifying and proving optimal the global optima not local optimas.

The approach taken by Nair et al. (Nair et al., 2020) extends Balcan et al. (Balcan et al., 2018)’s approach with a neural diver, which takes the bipartite graph representation of variables and constraints to predict plausible partial assignments. These partial assignments can then be explored in parallel by instantiating Mixed Integer Linear Programming (MILP) instances over a smaller variable space. Our neural diver extends Nair et al. to support OMT problems.

Neural Guidance for Combinatorial Applications

Using neural networks to solve TSP (Bello et al., 2016) has been extensively studied dating back to the development of Hopfield neural networks in 1985 (Hopfield et al., 2004). More recent efforts such as Selsam et al. (Selsam et al., 2019) have attempted to learn end-to-end models for SAT solvers.

Alternatively, a presolve phase is employed before solving to explore promising regions. This approach is exemplified by NeuroPlan (H. Zhu et al., 2021), which applies neural guidance to large-scale network planning problem. These problems often takes days or weeks for integer linear programming (ILP) to find even a feasible solution. For this, NeuroPlan learns an RL agent to predict a good initial solution to large-scale network planning problem modeled as ILP. The RL agent constructs the solution progressively by picking which network connection to be used to increase the capacity between two nodes to satisfy the communication requirements, and verify the feasibility via efficient checking techniques, reusing previous computational results. Once the initial solution is found, a follow-up ILP solving becomes much faster.

A.3 Preliminaries

In this section, we provide some background on OMT and graph neural network architectures used in Ashera.

Optimization Modulo Theories (OMT)

Optimization Modulo Theories (OMT) extends SMT, guaranteeing that an optimal feasible solution is returned. In addition to the satisfaction formula, an OMT problem includes an objective function C which maps assignments to a total ordered set. When the domain of C is real or floating point, a tolerance δ must be specified.

A Satisfiability Modulo Theory (SMT) problem decides the satisfiability of a first-order formula within a theory (Barrett et al., 2009). We focus on the theory of Linear Integer Arithmetic (LIA), but note that SMT and OMT extend to other theories, for instance, arrays and strings. For LIA, consider *atomic formulas* as linear inequalities of the form: $atom_i \triangleq \vec{a}_i \cdot \vec{x} \bowtie b_i$ where $\bowtie \triangleq \{<, \leq, >, \geq, =\}$. These atoms may differ when considering different theories, and we denote a theory *literal* as an atomic formula or the negation of one.

We build up *clauses* and subsequently *formulas* in Conjunctive Normal Form (CNF) from these atoms as

$$clause_j \triangleq \bigvee_i literal_i \quad formula \triangleq \bigwedge_j clause_j.$$

With the standard interpretation of atoms, a model or assignment that satisfies a formula is one where the evaluation of the formula is True. We denote partial assignments as α and full assignments as $\mathcal{A}$.

We define a *Boolean backbone* $\mathcal{B}$ as the set of literals where the polarity of each literal depends on the truth value of the corresponding atom when applying a full assignment $\mathcal{A}$, i.e. a literal is in positive polarity if the corresponding atom evaluates to true. A Boolean (backbone) assignment is the truth value assignment to all literals in the formula. Note that an assignment uniquely specifies a Boolean backbone but multiple assignments can share a common backbone.

Lazy SMT solves with a two stage iterative process. First, a SAT solver identifies a candidate Boolean assignment, viewing clauses as simple Boolean functions. Then, once a Boolean assignment is chosen, the formula can be expressed as a conjunct of atomic theory constraints. As such, the constraints can then be passed along to a specialized theory solver that identifies a feasible solution that satisfies the conjunct of constraints. If this is not feasible, the solver picks another Boolean assignment factoring in the learned conflict. In some sense, this can be viewed as a two level search problem where Boolean backbone identifies a *logical neighborhood* for the theory solver to search in.

Even though SMT is designed to efficiently explore disjunctive logic structures exploiting structure and symmetry in the encoding, the satisfiability task only requires reasoning about

feasibility. In contrast, an OMT solver must search for other solutions once after identifying a feasible solution potentially with differing Boolean backbone.

In the notation as introduced for SMT, we solve problems of minimizing cost, $C(\vec{x})$ such that the CNF formula holds, where in the theory of LIA variables are constrained to be integral and $C(\vec{x})$ a linear function with integral coefficients.

OMT raises the specific challenges of both explicitly reasoning about disjunctions and optimizing a cost function. Particularly, disjunctions lead to local optimas which may not be connected since there are no convexity guarantees.

Integer Linear Programming (ILP)

A Integer Linear Programming (ILP) problem is parameterized by the tuple $(\mathbf{A}, \vec{b}, \vec{c})$. The objective is to solve for an optimal choice of $\vec{x}$ such that $\vec{c} \cdot \vec{x}$ is minimized. However, $\vec{x}$ is constrained to satisfy $\mathbf{A}\vec{x} \leq \vec{b}$, where $\leq$ represents element wise less than or equal to. Further, these vectors can be over a mixture of real numbers and integers. We note that strict inequalities $\vec{a}_i \cdot \vec{x} < b_i$ can be encoded as $\vec{a}_i \cdot \vec{x} \leq b_i - 1$.

Although MILP does not explicitly support disjunctions, Big M encoding can allow practitioners to implicitly approximate disjunctions by adding an additional decision variable. Disjunctions of inequalities that appear in the original encoding, $(\text{LHS}_1 \leq \text{RHS}_1) \vee (\text{LHS}_2 \leq \text{RHS}_2)$ can be encoded by adding the decision binary variable α . Since MILP must be expressed as a list of constraints that always hold, a large constant M is then added to the inequality to trivialize the constraints when α deselects the literal. For our example, the disjunction becomes encoded as the following two constraints:

$$\begin{aligned} \text{LHS}_1 &\leq \text{RHS}_1 + \alpha M \\ \text{LHS}_2 &\leq \text{RHS}_2 + (1 - \alpha)M \end{aligned}$$

By using this encoding strategy, the assignment of α results in the selection of which clause must hold.

Graph Neural Network (GNN)

Recent progress in machine learning for optimization problems have been enabled by graph neural networks. In this section, we provide a brief introduction and key intuition behind these models. Interested readers can refer to (Zonghan Wu et al., 2021) for more details.

Graph Neural Networks (GNNs) are deep neural networks that take graph-structured inputs and make predictions on both individual nodes or edges and the entire graph. As formalized by (Gilmer et al., 2017), these models output a graph or node representation (i.e., a high-dimensional vector), via message passing over the graph structure. The GNN takes node features x_v and applies i rounds of message passing where the hidden state h_v^i of each node is updated based a learned function parameterized by θ on its neighbors' hidden

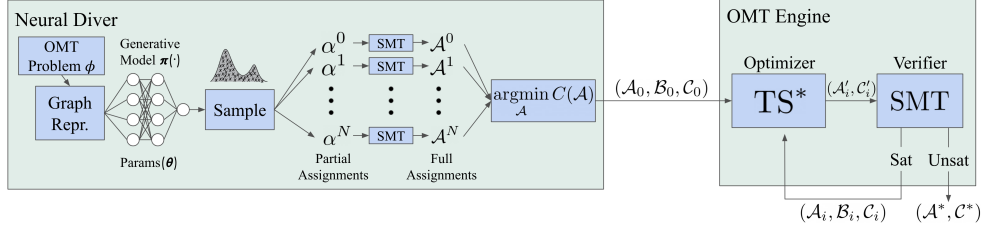


Figure A.1: Ashera workflow. The neural diver serves as a warm-starter providing an initial low cost feasible solution. After neural diving, the OMT engine in Ashera alternates between an *Optimization-aware Theory Solver* (TS*) for optimization and an SMT solver for verification. Each invocation of TS* returns a tighter blocking clause, restricting the SMT solver to search for strictly lower cost solutions than the current model.

states and the edge features: $m_v^{t+1} f_\theta(h_w^i, e_{vw})$ where w is a neighbor in the neighborhood of v , $N(v)$. The new hidden state is then $h_v^{i1} = \text{Agg}(h_v^i, \sum_{w \in N(v)} m_v^{t+1})$.

Inspired by Convolutional Neural Networks (CNNs), which exploit the inductive bias that neighboring pixels are often related, Graph Convolutional Networks, or GCN (Kipf et al., 2016), employ the same learned function across the same layer of the neural network irrespective of nodes. Analogous to computer vision, the shared function encourages the network to learn to recognize the same pattern occurring in connected subgraphs. This approach has seen wide success from analyzing social media graphs (W. Fan et al., 2022) to predicting molecule properties (Gilmer et al., 2017) and within combinatorial optimization has been used for network planning (H. Zhu et al., 2021) and chip placement (Mirhoseini et al., 2020).

Graph encoding for constrained programming. As done in (Gasse et al., 2019) and Nair et al. (Nair et al., 2020), one common graph representation for constrained programming is a graph, in which nodes represent constraints and variables. Edge between constraint nodes and variable nodes encode the coefficient of variables that appear in the constraint. Then, the message passing from variable to constraint in the graph neural network can be interpreted as how the variable embedding influence the constraint embedding (and vice versa). In practice, 2-3 rounds of message passing suffice to model higher-order constraints, e.g., two variables in the same constraint, etc.

A.4 Method Overview

In this section, we provide an overview of our method, illustrated in Fig.A.1. Ashera consists of two main components: an OMT engine (Section A.5) and a neural diver (Section A.6).

The neural diver generates an initial feasible assignments $\mathcal{A}_0$ by first using a neural heuristic trained on solutions of similar OMT problems. Given a $\mathcal{A}_0$, the assignment uniquely specifies a cost $\mathcal{C}_0$ and a boolean backbone $\mathcal{B}_0$. The initial feasible solution provides a warm

start to the OMT engine. We elaborate on how the neural diver is trained and how it selects initial feasible assignment in Section A.6.

Given the tuple $(\mathcal{A}_0, \mathcal{B}_0, \mathcal{C}_0)$, the OMT engine generates a blocking clause: $C(x) < \mathcal{C}_0$. This restricts the engine to only search for lower cost assignments. Using the boolean backbone, our OMT engine iterates between an optimizer that searches for a lower cost solution than the current assignment $\mathcal{A}_i$, and a verifier that checks if the optimized assignment $\mathcal{A}'_i$ is optimal. In a counterexample-guided fashion, the optimizer utilizes feasible solutions returned by the verifier to refine the search. We elaborate on the optimizer and verifier in Section A.5. When the verifier returns unsat, Ashera returns the best assignment, $\mathcal{A}^*$, and best cost, $\mathcal{C}^*$.

Ashera can be run in a *cold-start* settings when solutions to similar OMT problems are not available, or as a way to solve related OMT problems that are then used for training. In this setting, the neural diver can be replaced by a SMT solver. Although Ashera will not be able to learn from past examples, the SMT solver will still provide a valid albeit likely high cost assignment to the OMT engine.

A.5 OMT Engine

For exposition purposes, we first detail the cold-start setting of our OMT algorithm in this section, in which the neural diver is substituted with an SMT solver. In Section A.6, we introduce the full Ashera algorithm, including the neural diver and introduce Logical Neighborhood Search.

Logical Neighborhood Search

Efforts like Wu et al. (Y. Wu et al., 2021) approach optimization problems as a large neighborhood search problem where the optimizer first discovers feasible solutions. Then, it explores assignments which differ from the feasible assignment by a ϵ -ball. Our approach identifies a more natural notion of logical locality for OMT.

After obtaining the tuple $(\mathcal{A}_i, \mathcal{B}_i, \mathcal{C}_i)$ from the SMT solver, we perform optimization with an optimality-aware theory solver TS* which takes in a Boolean backbone $\mathcal{B}$ as input. Unlike existing approaches to neighborhood search, we aim to search for solutions that have similar logic structure (i.e. preserves the same Boolean literal backbone). This definition of locality allows us to use an off-the-shelf ILP solver as the optimality-aware theory solver to conduct the neighborhood search.

For each literal in the Boolean backbone $\mathcal{B}_i$, we add the negation of the false literal as an ILP constraint and the literal itself for true literals in the backbone. In this way, we do not need to encode disjunctions into the ILP encoding using Big M or convex hull. In A.10, we discuss some sound optimizations that can be done with this constraint generation process. By restricting to the convex region around the feasible solution, we can express the optimization purely as the conjunction of literals in $\mathcal{B}_i$. As we have identified a feasible

Figure A.2: Ashera: OMT Solver

Input: ϕ , an OMT formula.

Returns: $\mathcal{A}^*, \mathcal{C}^*$, the best assignment and cost with respect to the objective.

1. Initialize $\mathcal{A}^*, \mathcal{C}^* := \emptyset, \infty$.
2. Sample $partialAssignSamples := neuralDiver(\phi)$.
3. Construct $SMT_problem := createSMT(\phi)$.
4. For each α in $partialAssignSamples$, solve $SMT_problem$ under α . If the result is satisfiable and improves cost, update $\mathcal{A}^*, \mathcal{C}^*$.
5. Repeat:
6. Set $blocker := (cost \leq \mathcal{C}^* - \delta)$.
7. Solve $SMT_problem$ with the blocker to obtain $isSat, \mathcal{A}, \mathcal{B}, \mathcal{C}$.
8. If the problem is unsatisfiable, terminate the loop.
9. Construct $ILP_problem := createILP(\mathcal{B})$.
10. Update $\mathcal{A}^*, \mathcal{C}^* := solve(ILP_problem)$.
11. Return $\mathcal{A}^*, \mathcal{C}^*$.

solution $\mathcal{A}_i$, we effectively use ILP to improve on the found solution within the neighborhood which maintains the same logic assignment. Note that the search is localized to a connected region specified by the constraints, in which at least one feasible solution can be found (i.e., the current solution $\mathcal{A}_i$). However, unlike a LIA theory solver, ILP is cost-aware and able to optimize with respect to our objective function. The ILP solver produces a tuple $(\mathcal{A}'_i, \mathcal{C}'_i)$ as output, where $\mathcal{A}'_i$ is an assignment that achieves the optimized cost $\mathcal{C}'_i$ within the neighborhood.

To reach another disconnected region, we query SMT with the optimized cost $\mathcal{C}'_i$ for a feasible solution that's strictly better than the solution $\mathcal{A}'_i$ discovered by the last iteration of ILP. If this results in an unsatisfiable result, we terminate knowing there exists no better feasible solution to the OMT problem. Given ILP discovered the optimal solution for the particular logic backbone, the SMT solver will find a feasible solution with a different logic backbone.

We present our algorithm in full in Algorithm A.2 noting that a tolerance, δ , can be set

depending on the user’s domain expertise. For our integer example, $\delta = 1$ is natural.

A.6 Neural Diving

Inspired by work by Nair et al. (Nair et al., 2020), we adopt a neural diver which can be thought of as a warm-starter that identifies promising initial feasible solutions.

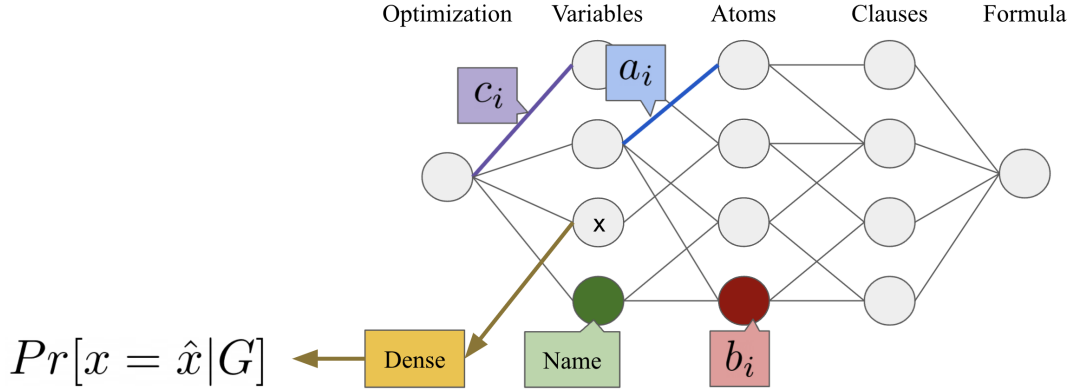


Figure A.3: Graph representation. We represent the OMT problem as a graph with edges connecting variable nodes and constraint/cost nodes. The logic structure is represented similarly to an abstract syntax tree (AST), with the AST leaf nodes coinciding with the atomic constraint nodes.

Graph Representation

We translate an OMT problem into a graph by encoding each variable as a node; we encode the variables and atomic formula as is done in (Gasse et al., 2019). We encode the elements of $\vec{a}_{ij}$ as weights on edges between variables and constraints. We add $\vec{b}_{ij}$ as node attributes for constraint nodes and variable names as node attributes on variable nodes. We also have a cost node connected to each of the variable nodes with weights corresponding to $\vec{c}$. Each atomic formula node serves as leaves in an adjoining tree representing the clauses and final formula.

Taking from the approach in (Nair et al., 2020), we implement a neural partial assignment generator based on a graph convolutional neural network (GCN). As shown in Fig. A.3, we build a graph connecting variable nodes and literal/constraint nodes with the edge weights indicating the linear coefficients in the literal. Finally, we encode the cost objective as a special literal node, with the coefficients as edge weights.

In contrast to the encoding for Nair et al. (Nair et al., 2020), OMT also includes disjunctions over the literals. As such, we encode the disjunction as a tree over the literal nodes.

Disjunction and conjunction nodes pass information between related clauses in rounds of message passing.

Formulating the Learning Problem

We consider the setting where an OMT solver is repeatedly solving similar problems. This means we can curate a training set of problems in this distribution based on historical queries or in simulation. Further, by design, our OMT engine can be run without the neural diver by replacing it with an initial SMT call. The cold-start OMT engine can be used to label training examples with optimal models. Using this labeled training set, we seek to reduce the required time to solve unseen problems from the same distribution.

With the graph encoding, call it G , our goal is to learn a function, f that estimates for each variable a probability distribution over potential values. We do this with a standard graph convolutional network (GCN) (Kipf et al., 2016). We learn this function f over examples G_i labeled with x_i^* , a cost optimal variable assignment. We treat integer variable values as independent classes train the model to classify each variable.

We use a GCN to learn an embedding for each variable, which we then pass through a linear layer to predict the class corresponding to the variable assignment. We find it sufficient to run two rounds of message passing for this application. With two rounds, the variables nodes can be aggregate information from two hop neighbors allowing the final learned embedding of the variable to be both influenced by variables that it shares an atomic constraint with and the clause that it belongs to.

Using the learned embedding, we optimize the following cross entropy loss:

$$\mathcal{L} = - \sum_{i=1}^m x_i^* \log p(x_i | G)$$

where x_i^* is the optimal assignment of the i -th variable and $\log p(x_i|G)$ is the probability of the assignment generated by the function f .

This loss intuitively maximizes the probability that the optimal assignment is selected. Note for each variable the GCN effectively approximate the probability distribution over variable assignments conditioned on G , $p(x|G)$. This is the desired f we sought to learn.

Partial Assignment Warm-Start

When the neural diver is used to solve a problem of interest, the diver makes a prediction based on the input problem G . This inference results in an estimated probability $P(x|G)$ returned as output logits. We sample variable assignments based on these logits and use the KL divergence to a uniform distribution to estimate confidence. If the KL divergence is larger than a confidence threshold, C , we abstain from assigning (see Appendix A.10 for implementation details). This results in partial assignments α^i as only variables that are easy to predict have assignments. To get full feasible solutions $\mathcal{A}^i$, we call an SMT solver on

each partial assignment to get a complete feasible assignment. We do this by adding to the existing OMT formula equality constraints $x = k$ where x is a variable and k is the sampled assignment from the partial assignment generator.

We then use Ray (Moritz et al., 2018) to search for a valid assignment in parallel for a user-specified time, T , with K parallel threads. If a thread discovers an unsatisfiable partial assignment, it continues searching with another sample from the generative model until the time expires. After running for T , the diver returns the best assignment discovered. In the case that it does not find any feasible solutions, the OMT engine runs SMT first to get a feasible assignment. By default Ashera uses $T = 5s$, $K = 5$, and $C = 1$, but these parameters can be tuned on a validation set in practice for each problem family.

A.7 Experiment Setup

For this work, we look at two families of real-world OMT problems: 1) DAG job scheduling, and 2) multi-agent traveling salesman problem (TSP). Disjunctions native to these two families are particularly challenging as they result in a disconnected feasible set and a redundancy of equally optimal solutions. For instance, in DAG scheduling, the each assignment of tasks to resources defines a nontrivial scheduling subproblem, that tend to be similar but not identical to each other. We describe the dataset generation, baseline solvers in this section, and include experimental details such as hardware setup in Appendix A.10.

Dataset Generation. In order to train and evaluate a learning-based OMT solver, we generate instances for each family of problems labeled with their optimal assignment.

Baseline Solvers. To compare with existing ILP solvers (e.g., Gurobi), we encode the OMT benchmarks as ILP problems using the standard Big M encoding to approximate the disjunctions. Big M encoding introduces an additional binary choice variable α to determine which clause in the disjunction is enforced as an ILP constraint. This increases the number of variables combinatorially in the number of disjunctions and requires the ILP solver to optimize over all disjunctive branch simultaneously.

For a fair comparison, the models were evaluated without GPU assistance, but we expect improved performance if accelerators are available at inference time. As Gurobi is highly parallelized, to make fair comparison of algorithmic cost, we compare results based on process time unless otherwise noted and impose a 1 hr limit for experiments. We do not include training time in our comparison as it required only 2 hour on a single GPU and depends heavily on the amount of data. Further, targeted applications of repeated OMT solves occur on a weekly basis allowing for training between invocations.

A.8 Results

In this section, we present our empirical results of existing OMT tools, *vZ*, OptiMathSAT, Gurobi (ILP with Big M), and Ashera. Our results show that Ashera scales to larger prob-

Table A.1: OMT Solver Performance on Scheduling. We report PAR-2 scores of process time to account for timeouts and provide number of solved problems in parenthesis.

Number of Tasks	Average PAR-2 Score in Seconds (Number Solved in 1 hr)				
	Gurobi	<i>vZ</i>	OptiMathSAT	Ashera	Ashera Cold-start
5	2.21 (20)	0.02 (20)	0.02 (20)	2.25 (20)	2.35 (20)
6	2.30 (20)	0.11 (20)	0.03 (20)	2.31 (20)	2.40 (20)
7	2.78 (20)	1.00 (20)	0.17 (20)	2.33 (20)	2.50 (20)
8	3.84 (20)	16.27 (20)	1.28 (20)	3.04 (20)	3.44 (20)
9	26.29 (20)	149.39 (20)	31.17 (20)	16.78 (20)	13.95 (20)
10	400.01 (20)	5246.23 (9)	841.21 (20)	135.97 (20)	181.34 (20)
11	3562.15 (15)	7200.00 (0)	7200.00 (0)	3204.00 (18)	2924.77 (18)

lems, outperforming all three baselines by as much as 5x compared to the next best solver on the scheduling and multi-agent TSP tasks. We compare performances with Par-2 score, which is the solving time for solved instances and two times timeout for unsolved ones.

Task Scheduling for Directed Acyclic Graphs

We generate a total of 43,596 similar DAG scheduling problems using the same problem encoding but varying the number of tasks and CPUs. We split up the evaluation based on the number of tasks in the scheduling problem. For a realistic setting, we consider two GPUs, and we have all the task with the same expected runtime of 15 seconds and release times of 2. This symmetry is notoriously difficult for traditional ILP solutions. To further make the instances comparable and always feasible, we scale up the deadline as the number of tasks increase, and only have one randomly placed dependency between two tasks in the taskset. All task within a problem instance have the same deadline as reported in Table A.4. In this section, we only consider Ashera trained and tested on the *same* number of tasks and defer analysis Ashera’s ability to adapt to tasks sizes not seen in training in Section A.8. We use $T = 1m$ for 10 tasks and default values otherwise.

We report the performance of baseline solvers and Ashera in Table A.1 categorized by the number of tasks in the problem instance. The table indicates that existing baselines OptiMathSAT has the best performance until 8 tasks while *vZ* performs badly on 10 tasks. Gurobi however, continues to solve instances with 11 tasks where *vZ* and OptiMathSAT timeout. Ashera in contrast is 3x faster and solves three more problems than Gurobi on problems with 10 and 11 tasks, respectively. The results demonstrates the effectiveness of Ashera when the problem size scales. This suggests the benefit of applying Ashera to large-scale real-world applications, which typically have hundreds of variables.

Note. We generated scheduling benchmarks with 11 to 20 tasks, but all methods timeout within 1 hour at 11 tasks. Our benchmark also contains multi-agent TSP problems with 8 to 10 waypoints per cluster, but all methods timeout within 1 hour at 8 waypoints per cluster.

Table A.2: OMT Solver Performance on Multi-Agent TSP. We report PAR-2 scores of process time to account for timeouts and provide number of solved problems in parenthesis.

# Waypoints per Cluster	Average PAR-2 Score in Seconds (Number Solved in 1 hr)				
	Gurobi	<i>vZ</i>	OptiMathSAT	Ashera	Ashera Cold-start
3	0.91 (22)	0.16 (22)	0.12 (22)	2.48(22)	3.93 (22)
4	76.51 (22)	0.80 (22)	0.94 (22)	4.51(22)	7.89 (22)
5	5154.00 (10)	5.74 (22)	7.85 (22)	20.50(22)	24.29 (22)
6	7200.00 (0)	7200.00 (0)	90.54 (22)	104.00(22)	102.91 (22)
7	7200.00 (0)	7200.00 (0)	919.26 (22)	754.33 (22)	793.88 (22)

Multi-agent TSP

In our multi-agent TSP benchmark, we generate 2500 instances of TSP with two clusters of waypoints arranged in a polygon, and additionally 22 problems for testing. We vary the distance between the center of the cluster and the origin where the vehicles start and vary the radius of the polygon. For simplicity, we provide as many vehicles as there are clusters and ensure restrictions on weight are not constraining. Additionally, the first waypoint is always the starting point for the vehicles.

Table A.2 shows Ashera outperforms all three baselines on the largest problems, solving faster than OptiMathSAT while Gurobi and *vZ* solve none. In Table A.2, we report Ashera’s performance when trained on instances with the same number of waypoints as those of the test set. We use $C = 0.5$ and otherwise use default parameters.

Cold-Start Ashera

We ablate the learned component of Ashera and see that Ashera performs faster on scheduling problems with 5 to 8 tasks. We attribute this to a 2 second overhead incurred in order to perform neural network inference on CPU and initialize Ray (Moritz et al., 2018). We leave for future work improvements afforded by hardware accelerators such as GPUs. Cold-start Ashera outperforms neural Ashera on scheduling problems with 11 tasks. For multi-agent TSP, Table A.2 shows neural diving provides modest improvement compared to cold-start due to a larger number of variables and constraints. We note that Cold-start Ashera already outperforms or matches performances of the baselines.

Neural Diver Performance

One important aspect for neural-based solvers is its performance to transfer to similar but new problems. Specifically, we look at two settings: 1) the performance of Ashera when tested on larger problems than seen in training, and 2) the neural diver applied to Gurobi and OptiMathSAT.

Table A.3: Ablation on Neural Diving. We both test performance transfer to larger problems trained on smaller problems (Curriculum) and applying neural diving on existing solvers (Neural Diver + Gurobi/OptiMathSAT).

Number of Tasks	Average PAR-2 score in Seconds (Number Solved in 1 hr)			
	Neural Diver + Gurobi	Neural Diver + OptiMathSAT	Ashera	Ashera Curriculum
5	2.34 (20)	2.37 (20)	2.25 (20)	– [‡]
6	2.42 (20)	2.43 (20)	2.31 (20)	2.28 (20)
7	2.66 (20)	2.44 (20)	2.33 (20)	2.30 (20)
8	4.00 (20)	3.24 (20)	3.04 (20)	2.36 (20)
9	19.44 (20)	12.03 (20)	16.78 (20)	13.97 (20)
10	288.09 (20)	288.09 (20)	135.97 (20)	174.19 (20)
11	2649.02 (17)	5728.87 (9)	3204.00 (18)	3118.62(17)

Performance transfer to larger problems. In this setting, Ashera is trained only on problems that are smaller than the test-time number of task. On scheduling, Table A.3 shows Ashera performs comparably to when it is trained on the same sized problems. On TSP, the performance is also comparable (see Appendix). This ablation indicates that Ashera can be trained on smaller problems from the same family to scale to larger problems of interest.

Neural Gurobi and Neural OptiMathSAT. We further consider the performance of the learned neural diver on Gurobi and OptiMathSAT. For this setting, we replace the Ashera OMT engine with Gurobi or OptiMathSAT. As the diver provides a Z3 verified upper bound on the cost, all OMT solvers are compatible and can use the upper bound as a hint. On problems with 11 tasks, OptiMathSAT can solve 9 problems compared to none before and Gurobi can solve two additional problems. This demonstrates the use of neural guidance on OMT problems extends beyond our specific solver design.

Note. [‡]We do not evaluate on 5 tasks as there are no smaller problems.

A.9 Conclusions

Our work presents Ashera, a neural OMT solver, which performs up to 3x faster and solves three more problems than the widely-used commercial solver, Gurobi, on problems with 10 and 11 tasks, respectively. Traditional solvers, OptiMathSAT and Z3, are unable to solve any problems of 11 tasks within 1 hour timeout. Further, Ashera is 18% faster than OptiMathSAT on multiagent TSP with 15 waypoints, where Gurobi and Z3 timeout. As OMT problems solved in practice tend to be solved on a regular basis, we make the case for learned-based OMT solver that train on a set of similar problems encountered previously

in the application or in simulation. We contribute benchmark of problem families including DAG task scheduling and multi-agent TSP for evaluating learning-based OMT solvers.

A.10 Supplementary Details

DAG Scheduling with Deadlines

The scheduling problem belongs to a widely applicable family of scheduling problems. It requires discovering the optimal placement of tasks to resources as well as assigning start times to tasks. Further, assignments must satisfy constraints on both resources and dependencies between tasks. We desire to maximize slack – the buffer time before the deadline that a task is expected to complete. As such it’s often non-trivial to find any feasible schedule, much less an optimal one.

This family of problems appears in both the workflow management platform Apache Airflow (Beauchemin, 2014) and in the DAG scheduler used for the dynamic deadline-driven execution model for self driving (Gog et al., 2022).

We consider a set of N tasks, $T = \{t_i | i \in [1, N]\}$, and a dependency matrix M where $M_{ij} = 1$ if t_i must complete before t_j otherwise 0. We further consider a set of deadlines and expected runtimes denoted as d_i and e_i , respectively.

In addition to runtime, we also consider resource requirements and placements. We denote r_i to be 1 if t_i requires a GPU and 0 otherwise.

We seek to optimize with respect to two sets of variables s_i and p_i which denote the start time and placement of t_i . Let N_G and N_C be the number of GPUs and CPUs, respectively. We encode $p_i = k$ to be in $[1, N_G]$ if it’s placed on the k^{th} GPUs of N_G and $(k - N_G)^{th}$ CPU if it’s in $[N_G, N_G + N_C]$.

Our cost objective is

$$\sum_{i=0}^N d_i - (s_i + e_i)$$

with the following constraints:

- **Basic constraints.** For all i , $0 \leq s_i$ and $0 < p_i$.
- **Finish before deadline.** For all i , $s_i + e_i \leq d_i$.
- **Placement constraints.** For all i , if $r_i = 1$, $1 \leq p_i \leq N_G$. Otherwise, $r_i = 0$, $1 \leq p_i \leq N_G + N_C$.
- **Dependency Respecting.** For all i, j , if $M_{ij} = 1$, $s_i + e_i \leq s_j$.
- **Exclusion.** For all i, j , $p_i = p_j \implies (s_i + e_i \leq s_j \vee s_j + e_j \leq s_i)$.

Big M for Scheduling Unfortunately, the exclusion constraint requires a disjunction. In order to compare against ILP solvers, we use the Big M strategy as presented in A.2. We introduce an additional two variables per pair of tasks i, j to 1) choose if task i and task j utilize the same resources and 2) choose if task i completes execution before task j begins or vice versa. We provide the Big M version of the exclusion constrain in Appendix A.10.

Multi-Agent Traveling Salesman Problem

The multi-agent Traveling Salesman Problem (TSP) appears in practical route planning applications including package deliveries in warehouse operations. A multi-agent TSP is specified by the distances between the W waypoints and the number of vehicles V . In this problem, the optimizer must find an ordering o_i in which waypoint, i , is visited by vehicle, v_i . We denote the starting waypoint as s . Our objective is to minimize the sum of the times t_i when a waypoint is visited:

$$\sum_{i=0}^N t_i$$

Due to space constraints, we highlight the following constraints and present the full encoding for multi-agent TSP in Appendix A.10:

- **Visited.** All waypoints w must be visited by at least one vehicle.
- **Deterministic.** After visiting a waypoint, w , a vehicle visits at most one waypoint w' immediately afterwards.
- **Ordering.** The starting waypoint has $o_s = 0$. For all waypoints, w , visited in order, o_w , the waypoint's predecessor p_w must have order $o_{p_w} = o_w - 1$. This prevents tours that do not include the starting point.
- **Weight Constraint.** The sum of the weights of the vehicles is less than a given value M .
- **Visit Time.** For all waypoints w , the visit time t_w if vehicle v_w visits it is at least the $t_{p_w} + \tau_{v,p,w}$ where t_p is the time when the preceding waypoint was visited and τ is the travel time from p to w by vehicle v .
- **Exclusion.** If vehicle v is traveling from w to w' from t_w to $t_{w'}$, there cannot be a waypoint w'' visited by v while it's traveling.

Big M for Multi-agent Traveling Salesman Problem

We again use the Big M encoding to encode disjunctions. The most complex disjunction requiring the disjunction of conjunctions is the ordering condition for a waypoint w :

$$\bigvee_{w' \in W, v \in V} \mathbf{M}_{v,w',w} \wedge (o_{w',v} = o_{w,v} - 1)$$

In the disjunction of conjuncts, all the inequalities in the conjunct share the same choice variable, ensuring that all constraints in the disjunctive case hold simultaneously if chosen.

Big M Exclusion Constraint We use Big M to encode the exclusion constraint for MILP. The implication

$$p_i = p_j \implies (s_i + e_i \leq s_j \vee s_j + e_j \leq s_i)$$

can be encoded as

$$(p_i < p_j) \vee (p_j < p_i) \vee (s_i + e_i \leq s_j) \vee (s_j + e_j \leq s_i).$$

By creating two binary variables $\alpha_{i,j}$ and $\beta_{i,j}$ per constraint, we introduce the following constraints in replacement using Big M:

- **Case 1** ($\alpha_{i,j} = 0, \beta_{i,j} = 0$) :

$$p_i - p_j < M\alpha_{i,j} + M\beta_{i,j}$$

- **Case 2** ($\alpha_{i,j} = 0, \beta_{i,j} = 1$) :

$$p_j - p_i < M\alpha_{i,j} + M(1 - \beta_{i,j})$$

- **Case 3** ($\alpha_{i,j} = 1, \beta_{i,j} = 0$) :

$$s_i + e_i - s_j \leq M(1 - \alpha_{i,j}) + M\beta_{i,j}$$

- **Case 4** ($\alpha_{i,j} = 1, \beta_{i,j} = 1$) :

$$s_j + e_j - s_i \leq M(1 - \alpha_{i,j}) + M(1 - \beta_{i,j})$$

Multi-Agent Traveling Salesman Problem Full Encoding

The multi-agent traveling salesman problem can be defined as an optimization problem where the aggregation over the time when the waypoints are visited is minimized.

We index the vehicles and waypoints respectively from 0 to $|V| - 1$ and $|W| - 1$, where V and W are the set of vehicles and waypoints and $|\cdot|$ denotes the cardinality of a set. The constraints are defined over the following variables:

- u_v - whether or not a vehicle is being used
- $\mathbf{M}_{v,w,w'}$ - a Boolean 3D array indicating if vehicle v travels from waypoint w to waypoint w'
- p_w - the preceding waypoint from which the vehicle visits w
- x_w - the vehicle that visits waypoint w

- h - starting waypoint
- o_w - order that waypoint w is visited by a vehicle v . Note that this is an ordering per vehicle not globally for all vehicles.
- t_w - the time when waypoint w is visited
- m_{max} - total mass allowed

We get the following constants from an oracle.

- $\tau_{v,w,w'}$ - time for agent v to travel from w to w'
- $c_{v,w,w'}$ - energy consumption
- γ_v - vehicle weight.

Our optimization seeks to minimize the aggregated time t when the waypoints are visited under the following constraints:

1. Each waypoint except the harbor must be visited by a vehicle:

$$\forall w' \in W \setminus \{h\}. \sum_{v \in V, w \in W} \mathbf{M}_{v,w,w'} = 1$$

The harbor has to be visited by the vehicles that are used.

$$\forall v \in V. \sum_{w \in W} \mathbf{M}_{v,w,h} = u_v$$

2. From one waypoint only one other waypoint is visited next (determinism), and according to fixed order:

$$\forall w \in W \setminus \{h\}. \sum_{v \in V, w' \in W \setminus \{h\}} \mathbf{M}_{v,w,w'} = 1$$

Vehicles that are used should leave the harbor.

$$\forall v \in V. \sum_{w' \in W} \mathbf{M}_{v,h,w'} = u_v$$

3. No self loop allowed.

$$\forall v \in V, w \in W. \overline{\mathbf{M}_{v,w,w}}$$

4. If a point has order o then it must have been reach from another point with order $o-1$. For the harbor starting point we have,

$$\forall v \in V. o_{h,v} = 0$$

for $o_{w,v}$ where v is unused we similarly constrain it to be zero:

$$\forall w \in W \setminus \{h\}, v \in V. (o_{w,v} = 0) \\ \vee \bigvee_{w' \in W} \mathbf{M}_{v,w,w'}$$

and also

$$\forall w \in W \setminus \{h\}. \bigvee_{w' \in W, v \in V} (\mathbf{M}_{v,w',w} \wedge o_{w',v} = o_{w,v} - 1)$$

5. For each waypoint w , a vehicle must visit another waypoint w' from w if it travels from some waypoint w'' to w :

$$\forall v \in V, w \in W. \left(\sum_{w'' \in W} \mathbf{M}_{v,w'',w} = 1 \rightarrow \sum_{w' \in W} \mathbf{M}_{v,w,w'} = 1 \right)$$

6. Constraint for p_w :

$$\forall w \in W. p_w = \sum_{v \in V, w' \in W} w' \cdot \mathbf{M}_{v,w',w}$$

7. Constraint for x_w :

$$\forall w \in W. x_w = \sum_{v \in V, w' \in W} v \cdot \mathbf{M}_{v,w',w}$$

8. A vehicle is used when:

$$\forall v \in V. (u_v \leftrightarrow \bigvee_{w, w' \in W} \mathbf{M}_{v,w,w'})$$

9. The total weight is less than a given value:

$$\sum_{v \in V} u_v \cdot \gamma_v < m_{max}$$

10. The total time each agent takes is equal to t , which is in the minimization problem:

$$\sum_{v \in V, w \in W, w' \in W \setminus \{h\}} \mathbf{M}_{v,w,w'} \cdot \tau_{v,w,w'} = t$$

Literal Dropping

To further improve performance, we recognize that literals can be dropped safely if the formula is expressed as negation normal form (NNF), in which the negation operator is only applied to atoms and the only allowed Boolean operators are conjunction and disjunction. We formally state the observation as follows.

Observation 1. *Given a satisfying assignment $\mathcal{A}$ to an SMT formula ϕ in NNF over LIA, let L^+ and L^- be the sets of true and false (theory) literal that appear in ϕ by applying $\mathcal{A}$. Literals $l \in L^+$ define a valid solution space for variables in ϕ . That is, $l \in L^-$ can be dropped while maintaining soundness.*

Proof. First, observe that removing false literals $l \in L^-$ in an NNF ϕ does not affect the satisfiability of ϕ (for $l \in L^+$ alone satisfies ϕ). Thus, passing L^+ to an LIA theory solver, any solution returned by the theory solver must satisfy $l \in L^+$ and hence will satisfy ϕ . $\square$

Moreover, since $L^+ \cup L^-$ imposes more constraints than L^+ does, the optimal value found in L^+ can be greater (resp. smaller) than that found in $L^+ \cup L^-$ when maximizing (resp. minimizing) an objective. This allows us to push bounds even further in the ILP solving phase.

Summary of Benchmarks Table A.4 and Table A.5 provide a summary of the number of variables and constraints for each size of problem in DAG scheduling and multiagent TSP, respectively. *Note.* [†]We generated scheduling benchmarks with 11 to 20 tasks, but all methods timeout within 1 hour at 11 tasks.

Table A.4: Scheduling Benchmark Summary.

Number of Tasks [†]	Number of Training Cases	Number of Test Cases	Average Variable Count	Average Constraint Count	Deadline
5	811	20	16	48	50
6	1459	20	19	63	57
7	2383	20	22	80	65
8	3630	20	25	99	72
9	5250	20	28	120	80
10	7291	20	31	143	87
11	9801	20	-	-	-
12	12831	20	-	-	-

Hardware Setup and Software Versions

For evaluation, we used c5.xlarge AWS cloud instances, which have 3.0 GHz Intel Xeon Platinum processors with 4 vCPUs and 8 GiB of RAM. For training, we used two Quadro GV100 GPUs with 32GB GPU Memory. We implemented parallel assignment search using

Table A.5: Multi-Agent TSP Benchmark Summary.

# Waypoints per Cluster	Number of Training Cases	Number of Test Cases	Average Variable Count	Average Constraint Count
3	2500	22	137	428
4	2500	22	211	654
5	2500	22	301	928
6	2500	22	401	1250
7	2500	22	519	1620
8	2500	22	-	-

Ray 1.10 (Moritz et al., 2018), a distributed programming framework. Our evaluation uses the following solver versions as baselines: Gurobi v9.5.1 (Gurobi Optimization, LLC, 2021), Z3 4.8.16 (De Moura et al., 2008), and OptiMathSAT 1.7.3 (Sebastiani et al., 2015). The same version of Gurobi is used for Logical Neighborhood Search and likewise of Z3 as the verifier in completing partial assignments and verifying soundness in the OMT engine.

Implementation Details of Diver

KL divergence. We use the KL divergence to compare our learned distribution from a uniform distribution over the allowed variables values. We use static analysis to extract single variable inequalities ($k < x$) to obtain simple upper and lower bounds.

If the learned distribution is similar to this restricted uniform distribution it is likely the variable is symmetric in the data and an assignment to the variable is likely to precipitate simpler subproblems.

Parallel search. To augment the parallel search, we additionally ran a thread that simply ran the OMT engine in parallel with the diving exploration. This way if the problem is small and easy to solve, the solver will not be penalized to severely for diving.

The diver on the other hand explored up to 5 partial assignments at a time as sampled from the generative model. This pool of partial assignments always includes the highest probability partial assignment. This assignment is gotten by taking the highest probability class from generative model using an argmax. We found this practically useful as this assignment is an often promising partial assignment but as the number of predicted variables increases the odds of getting this particular sample decreases.

This page was intentionally left blank