UC Irvine
UC Irvine Electronic Theses and Dissertations

Title
Toward Generative Problem Solving: Logical Puzzle Generation and Influence-Aware Network Synthesis

Permalink
https://escholarship.org/uc/item/7nb6q42p

ISBN
9798265470676

Author
Shih, Hsiang-Shun

Publication Date
2025-12-06

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Toward Generative Problem Solving: Logical Puzzle Generation and Influence-Aware Network Synthesis

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Hsiang-Shun Shih

Dissertation Committee:
Professor Phillip Chen-Yu Sheu, Chair
Professor Jean-Luc Gaudiot
Professor Nader Bagherzadeh

2025

© 2025 Hsiang-Shun Shih

DEDICATION

To my family, for your unconditional love.

To my advisor, Professor Sheu, for your guidance, patience, and trust in my ideas.

To Chengheng, for your collaboration and support throughout this journey.

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vi**

LIST OF ALGORITHMS **vii**

ACKNOWLEDGMENTS **viii**

VITA **ix**

ABSTRACT OF THE DISSERTATION **x**

1 Introduction **1**
- 1.1 GPS-Reasoning: Reasoning in Generative Problem Solving . . . 2
- 1.2 Influence-Preserving Graph Synthesis (IPGS): Generating Dynamic Influence Datasets . . . 3
- 1.3 Toward Generative Problem Solving . . . 3

2 Logical Problem Generation (GPS) **4**
- 2.1 Introduction . . . 4
- 2.2 Background . . . 6
  - 2.2.1 Neuro-Symbolic Systems and Reasoning Models . . . 6
  - 2.2.2 LLM Agents . . . 7
  - 2.2.3 Logical puzzle synthesis and LLM benchmark . . . 8
  - 2.2.4 Boundary between FOL and SQL . . . 9
- 2.3 Methods . . . 10
  - 2.3.1 Logical representation . . . 10
  - 2.3.2 Logical Dataset Synthesis - random AST . . . 10
  - 2.3.3 Template-Based Logical Statement Synthesis . . . 15
  - 2.3.4 Implementation Details . . . 24
  - 2.3.5 Logic-Language Translation . . . 25
  - 2.3.6 Example . . . 26
  - 2.3.7 User's Role and Contribution in Puzzle Generation . . . 28
- 2.4 Theorems . . . 29

2.4.1 Probability of Success Evaluation . . . 29
2.4.2 Correctness of Synthesized Logical Statements . . . 31
2.4.3 Complexity Analysis . . . 35
2.5 Experiments and Evaluation . . . 36
2.5.1 Dataset Generation . . . 37
2.5.2 Diversity and Complexity Analysis . . . 40
2.5.3 LLM Reasoning Capability . . . 42
2.5.4 LLM Generation Capability . . . 44
2.6 Conclusion and Discussion . . . 45

3 Dynamic Network Generation (IPGS) 49
3.1 Introduction . . . 49
3.2 Background . . . 52
3.2.1 Influence Maximization and Diffusion Models . . . 52
3.2.2 Reverse Influence Sampling (RIS) . . . 53
3.2.3 Dynamic Graphs and Influence Maintenance . . . 54
3.2.4 Graph Generation Under Topological and Structural Constraints . . . 54
3.2.5 Influence Maximization in Dynamic Graphs . . . 56
3.2.6 Summary . . . 56
3.3 Methods . . . 56
3.3.1 Overview of IPGS . . . 56
3.3.2 Dynamic Influence Model . . . 58
3.3.3 Cached RR Sampling and Maintenance (cRIS) . . . 61
3.3.4 Seed Node Constraint and Influence Boosting Heuristic . . . 65
3.3.5 Graph Generation Variants . . . 69
3.3.6 Update Efficiency and Parallelization . . . 71
3.4 Theorems . . . 72
3.5 Experiments . . . 77
3.5.1 Experimental Setup . . . 77
3.5.2 Experiment 1: Unbiasedness of Stratified RIS . . . 79
3.5.3 Experiment 2: Unbiasedness of Influence Boost . . . 81
3.5.4 Experiment 3: Preserving Seed Influential Status . . . 83
3.5.5 Experiment 4: Empirical Validation of Local Drift Bound . . . 85
3.6 Conclusion and Discussion . . . 85

4 Conclusion and Future Research 87
4.1 Conclusion . . . 87
4.2 Future Research . . . 88

Bibliography 90

LIST OF FIGURES

Page

2.1 Average nesting depth across GPT-5-mini, Gemini-2.5-flash, and GPS. . . . 40
2.2 Average formula length across GPT-5-mini, Gemini-2.5-flash, and GPS. . . . 41
2.3 Logical operator usage (mean counts per puzzle). 42
2.4 Logical structure pattern distribution across GPT-5-mini, Gemini-2.5-flash, and GPS. 43
2.5 Overall accuracy per domain (GPT vs. Gemini, Medium vs. High) 45

3.1 Workflow of IPGS. Each batch adds new nodes and edges $(\Delta V_t, \Delta E_t)$, followed by cached RR updates. When the estimated influence of $S^\star$ drifts beyond tolerance, the relaxed constraint introduces compensatory edges and deterministic RR-set expansions to restore local diffusion balance. 58

LIST OF TABLES

Page

2.1 Vampire validation by domain and by premise count 39
2.2 Merged accuracy table (GPT vs. Gemini) with **Medium** and **High** reasoning shown side-by-side. For Gemini, reasoning capability is controlled by thinking budget. 47
2.3 Puzzle-generation capability of GPT and Gemini measured by the percentage of generated puzzles which pass the verification of Vampire. 48

3.1 Stratified vs. standard RIS: mean influence ± std. dev. over 20 independent trials on static graphs with $n = 500$ nodes, $k = 5$ seeds, and randomized edge directions. 81
3.2 Experiment 2: Mean relative estimation error (averaged over final 3 batches) for No Repair vs REGROW conditions across 3 trials. BA graphs show minimal bias from REGROW, while ER graphs (with 5× sampling) reveal moderate overestimation from deterministic boost edges. 82
3.3 Experiment 3: Final influence comparison between $S^\star$ (original seeds with boost edges) and S_{greedy} (greedy-optimal seeds without constraints) after $T = 10$ batches. Results averaged over 3 trials. 84

LIST OF ALGORITHMS

		Page
2.1	Parsing FOLIO Templates	16
2.2	Building the Constraint Graph	19
2.3	Validating and Sampling Domain Assignments	20
2.4	Applying Inference Rules in Reverse	23
2.5	Puzzle Generation	23
3.1	Cached Reverse Influence Sampling (cRIS) for Batch Update	64
3.2	Seed Node Constraint and Edge Influence Boosting	68

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Phillip Chen-Yu Sheu, for his invaluable guidance, mentorship, and encouragement throughout my doctoral journey. His vision and support shaped the direction of this dissertation.

I would also like to thank my committee members, Professor Jean-Luc Gaudiot and Professor Nader Bagherzadeh, for their time and continual support.

The concepts of Generative Problem Sovling in this dissertation originally appeared in Shih, H.-S., et al. (2025). *Generative Problem Solving: Bridging Classic and Modern AI*. 19th International Conference on Semantic Computing (ICSC). IEEE Computer Society. I am the primary author of this publication, which forms the foundational basis for the Generative Problem Solving framework presented in this dissertation. Permission to reuse portions of the published materials has been granted under IEEE author self-archiving policies.

Portions of Chapter 2.3.3 utilize the publicly available FOLIO benchmark dataset (`https://github.com/Yale-LILY/FOLIO`), and I gratefully acknowledge the Yale LILY Lab for releasing this resource to the research community.

I am grateful to my collaborators, especially Chengheng, for meaningful discussions and shared effort throughout the development of this research.

VITA

EDUCATION

2025 **Ph.D.** in Electrical and Computer Engineering, University of California, Irvine
2017 **B.S.** in Electrical Engineering, National Taiwan University, Taiwan

FIELD OF STUDY

Artificial Intelligence, Generative Problem Solving, Logical Reasoning, Graph Theory

PUBLICATION

Shih, Hsiang-Shun, et al. *"Generative Problem Solving: Bridging Classic and Modern AI."* 2025 19th International Conference on Semantic Computing (ICSC). IEEE Computer Society, 2025.

ABSTRACT OF THE DISSERTATION

Toward Generative Problem Solving: Logical Puzzle Generation and Influence-Aware Network Synthesis

By

Hsiang-Shun Shih

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2025

Professor Phillip Chen-Yu Sheu, Chair

Large language models (LLMs) exhibit impressive fluency and surface-level reasoning skills, yet they continue to struggle with **deep, verifiable, and logically structured reasoning**. Despite strong empirical performance on natural language benchmarks, LLM reasoning can be improved: models sometimes generate plausible but incorrect solutions, fail under increased compositional complexity, and lack transparent, step-by-step justification. A key challenge is that current reasoning benchmarks are **static, passive, and non-generative**, restricting research to solving predefined problems rather than developing mechanisms for *creating and improving reasoning tasks themselves*.

This dissertation introduces a unified paradigm of **Generative Problem Solving (GPS)**, reframing reasoning from the traditional objective of solving fixed tasks to a generative process of *creating, solving, and verifying* problems. We argue that enabling AI systems to **generate structured reasoning tasks** is an essential step toward overcoming LLM limitations and advancing beyond pattern imitation toward true deductive reasoning.

The first contribution, **GPS-Reasoning**, is a symbolic problem-generation framework grounded in First-Order Logic (FOL). It systematically synthesizes **valid, interpretable, and difficulty-controlled reasoning tasks** using template-driven Abstract Syntax Trees (AST), model-

first satisfiability planning, and symbolic verification via Vampire. Unlike static benchmarks such as FOLIO and LogicNLI, GPS-Reasoning supports scalable curriculum generation, reverse inference trace construction, and fine-grained control over logical complexity. By coupling **synthetic generation with symbolic verification**, it produces provably correct reasoning problems with explicit inference chains, enabling rigorous evaluation that can be understood by humans.

The second contribution, **Influence-Preserving Graph Synthesis (IPGS)**, addresses generative dataset construction under dynamic influence diffusion settings. Traditional Influence Maximization (IM) research assumes static networks, while real-world networks evolve over time and influence dominance shifts. IPGS constructs dynamic graph datasets that preserve expected influence of initial seeds through stratified Reverse Influence Sampling (RIS), incremental RR-set regrow strategies, and deterministic structural reinforcement. Experimental results demonstrate unbiased estimation, bounded influence drift under Hoeffding concentration behavior, and near-optimal seed retention with minimal intervention across BA and ER network families.

Together, these contributions establish a generative foundation that **bridges modern LLM-based reasoning with classic symbolic AI**. Classic AI provides formal logic, exact semantics, and verifiable inference; LLMs provide generative scalability, expressiveness, and abstraction. GPS unifies these strengths to enable AI systems that are **transparent, controllable, and trustworthy**. We believe that generative problem solving represents a step beyond passive benchmark solving toward AI that actively **constructs knowledge, reasons over structure, and improves through generation**.

By integrating symbolic reasoning structure with scalable generative mechanisms, this dissertation contributes a path toward **robust and interpretable reasoning systems**, advancing the long-standing goal of combining the rigor of classic AI with the power of modern neural models.

Chapter 1

Introduction

Recent advances in Large Language Models (LLMs) have enabled impressive capabilities in natural language understanding, question answering, and text generation. However, despite their fluency, current models still struggle with **deep, verifiable, and logically structured reasoning**. LLMs frequently produce answers that sound correct but fail to provide provably valid reasoning steps, often collapsing when faced with higher compositional complexity or formally constrained problem settings. These limitations highlight gaps in reliability, interpretability, and systematic reasoning capabilities.

A core reason for this challenge is that existing reasoning benchmarks are **static and passive**. Models are evaluated only on the basis of the solution of fixed datasets rather than participating in an interactive process to generate, solve, verify, and improve reasoning tasks. This stands in contrast to human reasoning development, which naturally involves constructing problems and reflecting on correctness. We argue that to advance beyond surface-level pattern recognition, AI systems must adopt a **generative perspective on problem solving**.

This dissertation proposes a unified paradigm called **Generative Problem Solving (GPS)**,

reframing reasoning from the passive consumption of existing tasks to an active process of *creating, solving, and validating* structured reasoning problems. The GPS paradigm emphasizes controllability, interpretability, and formal grounding, aiming to bridge modern LLM capabilities with the rigor of **classic symbolic AI**. Classic AI contributes formal logic, exact semantics, and proof-based verification, while LLMs provide generative fluency and scalable abstraction. Together, they offer a promising foundation for building reasoning systems that are transparent, trustworthy, and adaptable.

This dissertation presents two complementary contributions aligned with the GPS paradigm, one focused on **problem generation**, and the other on **dataset generation**.

1.1 GPS-Reasoning: Reasoning in Generative Problem Solving

GPS-Reasoning, introduces a generative framework for constructing logically valid and human-interpretable symbolic puzzles. The system synthesizes First-Order Logic (FOL) problems using templates and Abstract Syntax Tree (AST) construction, then ensures satisfiability through model-driven grounding. Symbolic theorem proving (using Vampire) verifies solution soundness and uniqueness, guaranteeing correct and interpretable reasoning traces.

GPS-Reasoning enables controlled variation in problem structure and difficulty, supporting curriculum-style evaluation in which reasoning complexity grows alongside solver ability. Unlike static benchmarks such as FOLIO or LogicNLI, GPS-Reasoning is **infinitely extensible**, verifiable, and designed for both human and machine evaluation. The framework supports step-by-step symbolic justification, enabling rigorous assessment of reasoning behavior rather than answer-only accuracy.

1.2 Influence-Preserving Graph Synthesis (IPGS): Generating Dynamic Influence Datasets

Influence-Preserving Graph Synthesis (IPGS), introduces a generative framework for producing dynamic synthetic graph datasets where influence dominance of seed nodes is preserved as the network evolves. While most Influence Maximization research assumes static graphs, networks in real world change their influence landscape over time.

IPGS integrates stratified Reverse Influence Sampling (RIS), incremental RR-set regrow operations, and deterministic structural reinforcement to maintain accurate influence estimation throughout graph growth. Experiments demonstrate unbiased estimation, bounded influence drift under Hoeffding concentration behavior, and near-optimal seed retention with minimal intervention across Barabási–Albert and Erdős–Rényi graphs. The framework enables controlled analysis of influence stability, competitive dynamics, and diffusion behavior, capabilities not offered by existing static datasets.

1.3 Toward Generative Problem Solving

Together, GPS-Reasoning and IPGS establish a generative foundation for reasoning that is:

- **Transparent**: reasoning chains and influence behaviors are explicitly observable.
- **Controllable**: complexity and network behavior are tunable by design.

We believe that generative problem solving represents a significant step beyond passive benchmarking toward AI systems that actively **construct knowledge, reason about structure, and improve through generation**. By integrating symbolic guarantees with scalable generative modeling, this work contributes to the long-standing objective of unifying classic AI reasoning rigor with modern neural generative power.

Chapter 2

Logical Problem Generation (GPS)

2.1 Introduction

Generative Problem Solving (GPS) was defined as the ability to solve problems, generate new problems, and learn from solutions [19]. Reasoning, as a critical component of GPS, plays a vital role in formulating meaningful questions and solving problems effectively. In this paper, we introduce GPS-Reasoning, a framework that plans to utilize a neuro-symbolic-like approach to enhance the reliability, step-by-step transparency for reasoning system to solve logical puzzles at scale.

In this paper, we focus mainly on the puzzle generation component that serves as the first step in GPS. The objective is to produce novel, diverse, and formally sound problem sets that both facilitate logical inference and serve as a scalable benchmark for evaluating the reasoning capabilities of LLMs.

To meet these goals, GPS-Reasoning combines template-based statement synthesis with logical-rule chaining to construct typed first-order logic (FOL) puzzles. Every generated instance comes with a 100% guarantee of logical correctness: each puzzle is accompanied by

machine-checkable proof steps that are validated by the Vampire theorem prover. This design yields two practical benefits. First, it ensures that sound puzzles are provably consistent and solvable. Second, it ensures puzzles are semantically meaningful for humans to read and understand. In order to improve understanding of the process. This design delivers scalability and controllability: we can systematically vary domains, premise counts, and difficulty, enabling large, balanced test suites that thoroughly test LLM's reasoning ability.

In practice, the resulting benchmark exhibits high quality, diversity, and complexity. Existing benchmarks such as FOLIO [12] and LogicNLI [23] highlight the potential of formal reasoning but are not scalable, as they rely on limited datasets and hand-made construction. Since LLMs have already demonstrated strong solving ability, it becomes natural to push further by evaluating their generative ability, creating sound, complex, and human-readable problems is far more challenging. When we use Vampire [15], a symbolic inference engine as a validation approach, the generation of LLM only shows a wide variance between models and settings (e.g., GPT passes 97.1% of verified instances, while Gemini passes 47.5% overall). However, even in cases with a high pass rate, such as GPT 97.1%, the structural complexity of the generated problems remains low, as reflected by the small number of inference steps required by Vampire to solve them. By contrast, GPS-Reasoning's pipeline guarantees correctness by construction while also producing puzzles that require significantly deeper proof search. This highlights our key advantage: our generation pipeline ensures logical correctness by construction, while symbolic proof provides independent validation and grading, and the generative module focuses on problem structure, problem complexity, and readability.

In summary, GPS-Reasoning delivers a scalable, provably correct source of typed FOL puzzles that doubles as a discriminative benchmark for LLM reasoning. By uniting template-based generation with symbolic verification, it offers the reliability and coverage needed for rigorous evaluation, and a foundation for future neuro-symbolic systems that leverage LLMs for high-level insights while theorem provers certify correctness.

2.2 Background

2.2.1 Neuro-Symbolic Systems and Reasoning Models

Neuro-symbolic systems [25] combine neural, symbolic, and probabilistic components to improve interpretability, robustness, and data efficiency. Classic examples include AlphaGeometry [24], which integrates geometric inference with a language model to perform constructive actions such as drawing auxiliary lines during problem solving, and PyReason [1], a framework that integrates generalized annotated logic for open-world, temporal, and graph-based reasoning. These systems emphasize structured inference and verifiable reasoning paths, offering strong guarantees of correctness but limited scalability to open-domain problems.

LLM in theorem proving. Reasoning capability remains a concern for most LLMs. Even with the latest models like GPT-5, we are still far from true theorem proving, as even minor modifications to classical puzzles can deceive these so-called reasoning models. While LLM-based provers attempt to prove theorems autonomously without human intervention, this paper [20] proposes an LLM-assisted system to support human-driven mathematical proofs. We argue that humans still play an essential role in theorem proving, as LLMs struggle to generate truly novel or challenging proofs. By leveraging neuro-symbolic engineering and LLMs, we can reduce human intervention while maximizing human creativity.

From neuro-symbolic reasoning to reasoning models. While neuro-symbolic systems depend on explicit logic modules, recent *reasoning models* learn to perform multi-step reasoning internally within neural networks. Chain-of-thought prompting [27] encourages models to write out intermediate steps, improving accuracy on arithmetic, symbolic, and commonsense tasks. Methods like self-consistency decoding [26] make this process more reliable by comparing multiple reasoning paths, while Tree of Thoughts [30] explores different solution paths in a structured way.

Unlike neuro-symbolic pipelines that externalize inference through formal logic, reasoning models learn to simulate reasoning behavior implicitly. Recent systems such as OpenAI's ChatGPT5 family [17] and DeepSeek-R1 [9] demonstrate that multi-step reasoning can be mimicked from reinforcement learning and long-context training without symbolic modules. However, this approach trades explicit proof guarantees for flexibility and generalization across domains.

Position of GPS-Reasoning. In contrast to both paradigms, GPS-Reasoning adopts the architectural simplicity of reasoning models while preserving the verifiability of neuro-symbolic systems. Each generated puzzle ensures logical soundness by construction, yet the generation process remains neural-friendly and scalable. This design allows GPS-Reasoning to function as a controlled testbed for reasoning models: bridging symbolic precision with data-driven generalization.

2.2.2 LLM Agents

The concept of LLM agents leverages the natural language understanding capabilities of LLMs and integrates them with a variety of external tools. These agents are designed to interpret instructions, generate human-like responses, and adapt to diverse use cases. A key advantage of LLM agents lies in their ability to perform cross-domain tasks by utilizing pre-trained knowledge. The integration of external tools further ensures accurate and efficient task execution, enabling these agents to go beyond text generation and interact with APIs, databases, and other specialized systems to solve complex problems reliably. LangChain[5] allows LLMs to carry out tasks by integrating with external tools for application-specific tasks. This modular design allows for combining the generative power of LLMs with domain-specific reasoning capabilities, creating robust and adaptable agents. Similarly, Hugging Face's Transformers library [28] offers flexible pipelines for deploying LLMs in diverse applications, including text generation, summarization, and conversational

agents. In addition to our current work on benchmark generation, our future work will focus on combining LLMs with neuro-symbolic reasoning to build agents that leverage generative problem- solving capabilities, creating two kinds of problems: novel and unexplored problems, as well as valuable intermediate problems. By integrating structured knowledge bases and logical inference, these agents further enhance their generative potential, enabling them to address complex and previously unexplored challenges more effectively.

2.2.3 Logical puzzle synthesis and LLM benchmark

Prior benchmarks such as FOLIO [12] and LogicNLI [23] have highlighted the importance of disentangling first-order logical reasoning from commonsense inference. FOLIO emphasizes linguistic naturalness and real-world grounding through expert-written examples with parallel FOL annotations, while LogicNLI provides systematic coverage of all fundamental FOL operators and introduces diagnostic dimensions such as robustness, generalization, and traceability. Both benchmarks reveal that even state-of-the-art LLMs struggle to perform consistent logical reasoning, leaving significant room for improvement.

Our work builds upon these insights by focusing on the *synthesis* of logical puzzles rather than solely evaluation. By leveraging template-based generation informed by FOLIO-derived schemata, we construct diverse reasoning tasks that remain semantically coherent while retaining logical depth. In contrast to purely diagnostic datasets, our goal is to generate scalable, varied puzzles that simultaneously benchmark LLMs and support generative problem-solving. This synthesis perspective positions our benchmark as a complement to existing efforts: maintaining the rigor of logic-grounded evaluation while extending applicability to reasoning-oriented LLM agents.

2.2.4 Boundary between FOL and SQL

First-order logic (FOL) serves as a theoretical foundational framework of relational query languages, including SQL. SQL can be viewed as a practical subset of FOL with an extension to practical constructs of database tools, primarily capturing the existential fragment, wherein queries express the existence of certain data patterns using constructs like EXISTS, IN. However, there are notable limitations: while FOL supports full quantification, including universal quantifiers and arbitrary nesting of logical expressions, standard SQL lacks direct mechanisms to express universally quantified statements or complex logical inferences. Certain statements involving nested universal quantifiers—such as $\forall x \forall y(P(x, y) \rightarrow Q(x) \vee R(y))$—can technically be expressed using constructs like NOT EXISTS, but the resulting queries are often complex. More fundamentally, SQL lacks support for logical inference, such as transitive implications (e.g., $A \rightarrow B,\ B \rightarrow C \implies A \rightarrow C$), which require recursive or deductive capabilities beyond standard SQL. Additionally, SQL cannot express higher-order logic, such as quantification over predicates or relations (e.g., $\exists P(\forall x P(x))$), which are permissible in first-order logic extensions but fall outside SQL's schema-bound framework. Conversely, SQL introduces features not present in pure FOL, such as aggregation functions (SUM, COUNT, AVG), ordering (ORDER BY), grouping (GROUP BY), and update operations (INSERT, UPDATE, DELETE), which are essential for practical data management but fall outside the scope of FOL's descriptive capabilities. Thus, while SQL is inspired by FOL, the two differ both in expressive power and in their respective focuses on theoretical expressiveness versus operational utility.

2.3 Methods

2.3.1 Logical representation

There have been various logical representations throughout the history of automated theorem proving, traditional symbolic AI, and logic programming. While logic programming languages like Prolog[8] have historically supported inference and structured problem synthesis, we instead adopt a direct approach using first-order logic (FOL) represented as abstract syntax trees (ASTs) to construct logical statements. By generating FOL ASTs, we maintain fine-grained control over the structure and semantics of logical formulas, enabling systematic manipulation and complexity control, such as specifying depth, quantifier placement, and operator distribution. These ASTs are then translated into TPTP(Thousands of Problems for Theorem Provers) format and we employ Vampire for inference. Vampire is a high-performance first-order theorem prover that processes TPTP inputs and serves as the backend for formal validation and automated reasoning. Additionally, we leverage an LLM to translate between natural language statements and formal logic expressions, facilitating clearer human interpretation and interaction. This architecture eliminates the dependency on Prolog's backtracking and rule-based mechanisms, instead favoring a modular AST→LLM→theorem prover pipeline. This approach better supports integration with modern LLMs and enables scalable, logic-grounded problem generation and verification.

2.3.2 Logical Dataset Synthesis - random AST

Model Generation for Evaluation

To avoid the computational overhead of solving the satisfiability problem for arbitrary formulas, we adopt a model-first approach: we first generate a model, a logical True/False assignment on all the predicates with their legal arguments, then construct formulas that are guaranteed to be true under it (or salvaged via a single top-level negation). This is both logically sound and computationally efficient. The model is created by iteratively sampling

the domain of legal entities and assigning truth values to all ground instances of predicates. Each predicate is assigned a fixed arity, and the model ensures well-formed formulas by avoiding duplicate or malformed entries. By evaluating generated formulas against this model, we guarantee satisfiability and eliminate the need for external theorem proving during synthesis.

For example, suppose we have three persons: Tony, Mike, and John, two weather conditions: `rain` and `snow`, and predicates `Skier()`, `Climber()`, and `Like()`. `Skier()` and `Climber()` are defined over the domain of persons; `Like()` is defined over the domain of (person, weather). A possible model might be:

Skier(Tony): True

Skier(Mike): False

Skier(John): True

Climber(Tony): False

Climber(Mike): True

Climber(John): False

Like(Tony, snow): True

Like(Mike, snow): False

Like(John, snow): False

Like(Tony, rain): True

Like(Mike, rain): False

Like(John, rain): True

A valid goal under this model could be: `Skier(Tony)` $\wedge \neg$`Like(Tony, rain)`. This strategy ensures that every generated (goal) formula is semantically grounded and satisfiable, avoiding the common pitfalls of generating logically inconsistent or trivial statements.

Random Walk on AST

The core of the baseline formula generator is a stochastic recursive walk over a set of abstract syntax tree (AST) nodes, implemented in `ast_nodes.py`. Starting from a root node (e.g., $\exists x$ or $\forall x$), child nodes are generated probabilistically, selecting from logical connectives (AND, OR, IMPLIES, etc.) or atomic formulas (ATOM). This walk is bounded by a maximum depth and optionally balanced branching, which controls the syntactic complexity of generated formulas. The randomness introduces variability in structure while maintaining adherence to FOL syntax, creating a diverse set of statements. With the following parameters, an example could be:

- Max depth: 3
- Connectives: And, Or, Implies, Not
- Quantifiers: $\forall x$, $\exists x$, $\forall w$, $\exists w$
- Atoms: `Skier(x)`, `Climber(x)`, `Likes(x,w)`

The resulting formula could be:

$$\forall x \left[(\texttt{Skier}(x) \vee \neg\texttt{Skier}(x)) \wedge (\texttt{Like}(x, \texttt{rain}) \rightarrow \neg\texttt{Climber}(x))\right].$$

Evaluation and Pruning on AST

Once an AST is constructed, it is evaluated under the previously generated model using the evaluator in `ast_nodes.py`. Only formulas that are evaluated as true are retained (or negated at the top level if we wish to salvage the structure). Subsequently, we prune subtrees that are irrelevant to truth determination, e.g., unnecessary conjunction branches or trivially true/false subformulas. This pruning process simplifies the formula structurally and also supports subgoal extraction for puzzle creation. For example, the AST-generated formula above is true under the model, but $\texttt{Skier}(x) \vee \neg\texttt{Skier}(x)$ is a tautology which can be pruned; the pruned result is:

$$\forall x \, (\texttt{Like}(x, \texttt{rain}) \rightarrow \neg\texttt{Climber}(x)).$$

CNF conversion

After initial formula construction, we convert formulas into clausal normal form (CNF) to control structural complexity. CNF ensures a bounded number of literals per clause and simplifies subsequent reasoning. Following CNF conversion, we can optionally ground variables over a small entity set to extract concrete clauses. The CNF transformation is implemented in `ast_nodes.py` and supports Skolemization, variable standardization, and clause flattening, ensuring logical equivalence with the original formula while aiding downstream puzzle instantiation. For example, the pruned formula can be converted to:

$$\neg\texttt{Like}(x, \texttt{rain}) \vee \neg\texttt{Climber}(x).$$

If the domain contains `Tony`, `Mike`, and `John`, this can be grounded as

$$(\neg \texttt{Like}(\texttt{Tony}, \texttt{rain}) \ \vee \ \neg \texttt{Climber}(\texttt{Tony})),$$

$$(\neg \texttt{Like}(\texttt{Mike}, \texttt{rain}) \ \vee \ \neg \texttt{Climber}(\texttt{Mike})),$$

$$(\neg \texttt{Like}(\texttt{John}, \texttt{rain}) \ \vee \ \neg \texttt{Climber}(\texttt{John})).$$

Based on the model, the true subgoals are: $\neg\texttt{Climber}(\texttt{Tony})$, $\neg\texttt{Like}(\texttt{Mike}, \texttt{rain})$, and $\neg\texttt{Climber}(\texttt{John})$.

Limitations. Although logically correct under evaluation, AST sampling may assemble semantically implausible statements (e.g., applying a weather predicate to a person or vice versa if not guarded). We therefore treat the AST path as a baseline and emphasize the template route for semantically well-typed generation.

Why templates? While the legacy AST random walk can generate a wide variety of syntactic structures, it may assemble type-incoherent or semantically implausible statements (despite being logically well-formed). The template route prevents this by (i) constraining predicate arities and argument domains at instantiation time, (ii) propagating entity-type requirements via a dependency graph, and (iii) selecting constants and variables in a way that enforces uniqueness and type correctness. In practice, the **FOLIO-derived template bank (423 patterns)** is our main source of generated statements and provides significantly broader and more realistic coverage than hand-crafted templates alone.

Shared utilities. Both the AST and template paths share a common evaluator and exporters: pretty-printers for FOL and TPTP, CNF conversion (with Skolemization and variable standardization), and JSON export for downstream puzzle construction.

2.3.3 Template-Based Logical Statement Synthesis

Our template-based synthesis is the main generation path. It operates over a pool of numbered-placeholder patterns (*schemata*) with explicit arity and domain constraints. Placeholders include predicate symbols $p_1, p_2, \ldots$, variables $x_1, x_2, \ldots$, and constants $c_1, c_2, \ldots$. During instantiation, we (i) bind p_i to concrete predicates consistent with arity and argument domains, (ii) solve a dependency graph to assign type-correct, unique constants to c_i, and (iii) render the result as FOL and TPTP strings.

FOLIO-derived template bank. We parsed and collected **423** templates from the FOLIO dataset and merged them with a small set of hand-crafted templates, as described in Algorithm 1. The FOLIO-derived schemata are the *main source* of generated statements in our experiments, providing both breadth and diversity in quantifier structure and connective usage. We normalize operators (e.g., $\forall, \exists, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$), fix occasional uses of commas as conjunctions, and standardize variable naming. A few representative raw schemata (before binding) are:

$$\forall x_2\ \forall x_1\ \big(p_1(x_2) \wedge p_1(x_1) \wedge \neg p_2(x_2, x_1) \wedge \neg p_2(x_1, x_2) \wedge p_3(x_2, x_1) \rightarrow p_4(x_2, x_1)\big),$$
$$p_1(c_1) \wedge p_1(c_2),$$
$$\neg p_1(c_1, c_2) \wedge \neg p_1(c_2, c_1).$$

After binding p_i to concrete predicates and c_i to typed constants consistent with predicate domains, we produce semantically meaningful statements while retaining the original quantifier/connective skeletons.

Dependency-Graph Domain Solver

We formalize template instantiation as a finite *constraint satisfaction problem* (CSP) that enforces both arity and *typed-domain* constraints before any model evaluation. This process ensures that every instantiated formula is semantically valid, preventing category errors and ill-typed bindings that the AST baseline may produce without explicit guards.

Algorithm 2.1 Parsing FOLIO Templates

Require: A `file_path` to a file containing FOLIO templates
Ensure: A list of `LogicalTemplate` objects

```
function ParseFolioTemplates(file_path)
    Initialize an empty list of templates
    Read all lines from the file
    for each line in the file do
        if the line is not empty and not a comment then
            template ← parse_single_folio_template(line)
            if template is valid then
                Add template to the list of templates
            end if
        end if
    end for
    return templates
end function
```

Objects and signatures. A template schema S exposes three sets of placeholders: predicates $P = \{p_1, \ldots, p_k\}$, variables $X = \{x_1, \ldots, x_m\}$, and constants $C = \{c_1, \ldots, c_n\}$. Each concrete predicate symbol R in the configuration carries a *signature* $\sigma(R) = (\tau_1, \ldots, \tau_r)$, where each τ_i represents the expected argument type (e.g., Person, Animal, Weather). For any predicate occurrence $p_j(t_1, \ldots, t_r)$ in S, the argument t_i must have type τ_i according to $\sigma(p_j)$.

Constraint graph. We construct a dependency graph $G = (V, E)$ (Algorithm 2) whose vertices represent predicate-argument positions and constant placeholders. For every predicate occurrence p_j at position i, we add an edge from the position node $\langle p_j, i \rangle$ to the symbol node corresponding to t_i. Types propagate *from* predicate signatures *to* placeholders, generating three classes of constraints:

Typing:	$\forall i, \;\; \text{type}(t_i) = \tau_i,$
Co-reference:	the same placeholder reused in S denotes the same entity,
Distinctness:	$c_a \neq c_b$ if required by the template (e.g., $c_1 \neq c_2$).

Domains. Each constant placeholder c is assigned a finite domain $D(c) \subseteq \mathcal{E}_\tau$, the set of entities of its required type τ, drawn from `entities_config.json`. Variables remain symbolic at instantiation time; their types are inferred from the incident edges and enforced later by the evaluator. Predicate placeholders p_j are selected from the candidate pool $\mathcal{R}_r$ of concrete relations with arity r whose signatures are compatible with *all* occurrences of p_j within S.

Two-stage binding. Instantiation proceeds in two coordinated stages:

1. **Predicate selection.** For each p_j of arity r, choose a concrete relation $R \in \mathcal{R}_r$ whose signature $\sigma(R)$ is compatible with every occurrence of p_j in S. This step unifies variable types across all predicates incident to the same variable.

2. **Constant assignment and validation.** For each constant node c_i, determine its *final domain* (preferring explicit domains when specified) and randomly sample a concrete entity from that domain. Uniqueness is enforced: once an entity is used, it is excluded from all remaining candidate pools to satisfy distinctness constraints such as $c_1 \neq c_2$. If no valid entity remains, the solver aborts and resamples a new combination of predicate and constant bindings. Rather than full backtracking, a pragmatic *resampling strategy* is employed: entities are drawn randomly from compatible domains, verified for type consistency via the entity–type map, and resampled on conflict. This lightweight approach maintains efficiency while preserving semantic correctness.

If either stage fails, the solver resamples (template or predicate choices) rather than forcing an inconsistent assignment (Algorithm 3).

Soundness under finite domains. Given finite entity sets, the search is guaranteed to terminate. If a well-typed binding exists, it will eventually be found; otherwise, the template is rejected. Every successful assignment ensures that the resulting formula is well-formed, type-safe, and semantically grounded before any model evaluation.

Example.

(i) Variable-only skeleton. For the FOLIO-style schema

$$\forall x_2\, \forall x_1 \, \Big(p_1(x_2) \wedge p_1(x_1) \wedge \neg p_2(x_2, x_1) \wedge \neg p_2(x_1, x_2) \wedge p_3(x_2, x_1) \rightarrow p_4(x_2, x_1) \Big),$$

predicate signatures must agree on the type of x_1 and x_2 across all incidents. If, for example, $p_1 : \textsc{Person} \rightarrow \mathbb{B}$ and $p_2, p_3, p_4 : \textsc{Person} \times \textsc{Person} \rightarrow \mathbb{B}$, then the solver infers $\text{type}(x_1) = \text{type}(x_2) = \textsc{Person}$ and accepts any predicate binding with compatible signatures.

(ii) Constant placeholders with distinctness. For the schema

$$p_1(c_1) \ \wedge \ p_1(c_2) \ \wedge \ \neg p_2(c_1, c_2) \ \wedge \ \neg p_2(c_2, c_1),$$

suppose $\sigma(p_1) = (\textsc{Person})$ and $\sigma(p_2) = (\textsc{Person}, \textsc{Person})$, and the template enforces $c_1 \neq c_2$. Then $D(c_1) = D(c_2) = \mathcal{E}_{\textsc{Person}}$, and the solver searches over *ordered* pairs of distinct persons. If the entity set is $\{\texttt{Tony}, \texttt{Mike}, \texttt{John}\}$, the admissible pairs for (c_1, c_2) are $(\texttt{Tony}, \texttt{Mike})$, $(\texttt{Tony}, \texttt{John})$, $(\texttt{Mike}, \texttt{Tony})$, ... No assignment with a non-Person entity is permitted, and any equal pair is rejected by the distinctness constraint.

Robust instantiation. Safe placeholder replacement is performed using word-boundary checks to avoid partial matches (e.g., preventing `x1` from matching inside `x10`). Variables remain symbolic; their types are enforced by predicate signatures and verified during model evaluation. When no consistent assignment exists for a template under a given predicate binding, the solver abandons that attempt and resamples rather than relaxing constraints.

Semantic coherence. By binding templates *through* signatures and typed domains prior to truth evaluation, the solver prevents type mismatches (e.g., assigning a Weather entity to a Person slot). In contrast, the AST baseline can produce logically well-formed but semantically incoherent atoms unless explicitly constrained. The dependency-graph solver

is thus key to ensuring that *template-based generation enforces both logical and semantic correctness.*

Negation handling. Each instantiated statement is evaluated against the sampled model. If it evaluates to *false*, the solver wraps the entire statement with a top-level negation, preserving syntactic structure while guaranteeing satisfiability.

Complexity buckets. We group statements by structural complexity: *Simple* (depth ≤ 2, quantifier≤ 1), *Medium* (depth ≤ 4, quantifiers≤ 2), *Complex* (depth ≤ 6, quantifiers≤ 3), and *Very Complex* (depth ≤ 8, quantifiers≤ 4). FOLIO-derived templates naturally populate the higher buckets (*Medium* $\rightarrow$ *Very Complex*), while hand-crafted templates enrich the lower levels. Users can request a specific bucket at generation time to control difficulty.

Algorithm 2.2 Building the Constraint Graph

Require: A `template`, `predicate_bindings`
Ensure: A `constraint_graph`

function BUILDCONSTRAINTGRAPH(template, predicate_bindings)
    Initialize an empty `constraint_graph`
    for each constant placeholder in `template` **do**
        if the placeholder has an explicit domain **then**
            Add the explicit domain to its node in the graph
        end if
    end for
    for each predicate occurrence $p_j(t_1, \ldots, t_r)$ in `template` **do**
        `predicate` $\leftarrow$ the bound predicate from `predicate_bindings`
        `signature` $\leftarrow$ the domain types $(\tau_1, \ldots, \tau_r)$ of `predicate`
        for each argument t_i in the predicate **do**
            if t_i is a constant placeholder **then**
                Add inferred domain τ_i to the node of t_i
                Add an edge $(p_j, i) \rightarrow t_i$ to the graph
            end if
        end for
    end for
    return `constraint_graph`
end function

Algorithm 2.3 Validating and Sampling Domain Assignments

Require: A `template`, `predicate_bindings`
Ensure: A mapping of constants to valid entities

function SolveConstraints(template, predicate_bindings)
    `constraint_graph` ← BuildConstraintGraph(template, predicate_bindings)
    if domains in `constraint_graph` are inconsistent **then**
        Discard the current template and resample
    end if
    for each constant node c_i in `constraint_graph` **do**
        Determine the final domain $D(c_i)$ (prefer explicit domain if available)
        `available` ← $D(c_i)$ minus any already used entities
        if `available` is empty **then**
            Resample or abort the current instantiation
        end if
        `entity` ← randomly select from `available`
        if the type of `entity` does not match $D(c_i)$ **then**
            Resample another `entity`
        end if
        Assign $c_i \mapsto$ `entity` and mark it as used
    end for
    return the mapping of constants to selected entities
end function

Enhanced Model Generation

Our system implements an enhanced model generation process that applies domain-specific soft constraints to encourage realistic truth assignments (e.g., mutual exclusivity between `IsStudent` and `IsProfessor`, or consistency between ownership and object properties). The model-first approach provides several key advantages:

- **Guaranteed Validity**: Every generated statement is verified against the model before output (or negated at the top) to ensure satisfiability.
- **Semantic Coherence**: Domain constraints prevent nonsensical combinations while maintaining realistic relationships.
- **Scalable Complexity**: The four-tier system allows systematic progression from basic facts to sophisticated multi-quantifier reasoning.
- **Human Interpretability**: Templates preserve readable structures while expressing complex logical relationships.

Puzzle Generation Method

We developed a logical puzzle generation framework that organizes information into two disjoint sets: (i) *Premises* and (ii) *True Statements*. The system produces self-contained reasoning puzzles in which a solver must determine the truth status of each statement using deductive reasoning alone.

Reverse Inference Chain. To further enrich the logical structure, we apply reverse inference, where intermediate axioms are introduced based on a desired conclusion, as described in Algorithm 4. This process decomposes a target fact into multiple inferential steps, encouraging multi-hop reasoning in downstream puzzles. For example, given a subgoal $\neg\texttt{Climber}(\texttt{Tony})$, we may add an implication $\texttt{Skier}(\texttt{Tony}) \rightarrow \neg\texttt{Climber}(\texttt{Tony})$. We can

further support the antecedent via $\texttt{Like}(\texttt{Tony}, \texttt{snow})$ and $\forall x\,(\texttt{Like}(x, \texttt{snow}) \rightarrow \texttt{Skier}(x))$, producing a richer chain. Conceptually, this is the opposite of forward inference: we start from a desired conclusion B and introduce $A \rightarrow B$ along with premises for A to deepen the inference path.

Statement Representation. Each logical proposition is represented as a `Statement` object comprising:

- A syntactic tree representation (`Node`) of the formula, encoded in First-Order Logic.
- A unique statement identifier.
- A source label indicating its origin (`goal`, `database`, `generated_true`, or `generated_false`).
- A Boolean flag specifying its truth status.

This structured representation allows consistent handling of logical forms, provenance tracking, and efficient comparison between statements.

Inference Rules and Reverse Application. The framework includes an extensible set of inference rules (e.g., Modus Ponens), each defined by:

1. `can_apply(statements)` – determines whether the rule can be applied to the provided statements.
2. `apply_reverse(statements)` – generates prerequisite statements that, if true, would logically entail the given statement.

Reverse application is the central design principle: instead of deducing consequences from known premises, the system begins from a designated *goal statement* and recursively constructs the premises required to prove it. This backward-chaining approach naturally forms a directed acyclic dependency graph.

Algorithm 2.4 Applying Inference Rules in Reverse

Require: A list of `statements`
Ensure: A list of new `statements` generated by reverse inference

```
Example for Modus Ponens (A, A → B ⊢ B)
function ApplyReverse(statements)
    if the goal statement is B then
        Generate a supporting statement A
        Generate the implication (A → B)
        return [A, (A → B)]
    else if the rule is Resolution then
        Select a literal C from existing statements
        Generate (C ∨ A) and (¬C ∨ B)
        return [(C ∨ A), (¬C ∨ B)]
    else
        Apply rule-specific reverse constructor
    end if
end function
```

Algorithm 2.5 Puzzle Generation

Require: A `goal_statement`
Ensure: A puzzle with premises, true statements, and false statements

```
function GeneratePuzzle(goal_statement)
    Initialize a proof_graph with the goal_statement
    while the proof is not complete do
        Select a leaf current_statement from the proof_graph
        Choose an applicable inference_rule
        new_statements ← ApplyReverse(current_statement)
        Add new_statements as children of current_statement in the graph
    end while
    Extract Premises, TrueStatements, and FalseStatements from the proof_graph
    return the generated puzzle package
end function
```

Puzzle Assembly Process The puzzle generation pipeline operates as follows:

1. **Goal Selection:** A goal statement is randomly selected from a template-based logical statement generator.

2. **Dependency Construction:** Inference rules are applied in reverse to expand the proof graph from the goal toward the base premises.

3. **True Statement Generation:** Intermediate statements that are logically entailed by the premises are added to the *True Statements* set.

4. **Export:** The final puzzle is serialized in JSON format, containing the two sets, the dependency graph, and metadata about inference rule applications.

Outcome This process yields puzzles in which the solver is provided with:

- A set of premises that are assumed to be true.

- True derived statements.

- The task of determining the logical validity of each statement.

The combination of reverse inference and structured logical representation summarized in Algorithm 5 ensures that each puzzle is both logically consistent and non-trivial.

2.3.4 Implementation Details

Files: `entities_config.json` and `predicates_config.json` define entity categories and predicate signatures (arity and argument domains). `templates.py` stores hand-crafted schemata and the FOLIO-derived pool (when present). `dependency_graph_template_based_generation.p` performs template selection, domain-constrained binding via a dependency graph, and safe replacement of numbered placeholders (e.g., using word-boundaries to avoid matching `x1`

inside `x10`). `ast_nodes.py` implements the AST, the evaluator, CNF conversion (Skolemization, standardization, flattening), and exporters (FOL, TPTP). `puzzle_generation.py` expands goals into multi-hop premises via reverse rules (Modus Ponens, Conjunction Introduction/Elimination, Or Introduction, and Resolution) to build puzzles with difficulty labels.

FOLIO optionality: If the FOLIO template file is absent in a particular run, the system falls back to the internal pool. For the experiments reported here, we use the *FOLIO-derived* bank with **423** templates as the primary source, which improves semantic coverage and structural diversity.

2.3.5 Logic-Language Translation

One-to-one mapping translation

FOL ASTs operate from statement generation to reverse inference chaining, while Vampire expects explicit first-order logic statements in TPTP syntax. A Python-based translator bridges this gap by parsing the AST, identifying logical operators and quantifiers, and reconstructing the structure into valid TPTP axioms. This transformation involves the following steps:

- AST Traversal: The logical formula is traversed recursively to inspect each node and its structure.
- Quantifier Mapping: Universal and existential quantifiers are mapped to TPTP's ! and ? notation with variable binding.
- Operator Conversion: Logical operators such as conjunction, disjunction, negation, and implication are converted to their corresponding TPTP symbols.
- Leaf Node Handling: Predicate symbols, constants, and variables are rendered directly

as atomic terms.

- Clause Construction: The translated formula is wrapped in a fof(...) clause with a designated role, yielding a well-formed TPTP axiom for theorem proving.

The Python-based translator also adds domain-specific statements for TPTP, including domain closure and entity distinction.

LLM as a Translator

Translate between FOL and Natural Language: We use a language model to translate between natural language and first-order logic (FOL). Natural language statements are first parsed into FOL formulas by identifying their logical structure and converting them into an AST. The AST ensures the formula is well-formed and logically valid. In the reverse direction, the AST can be converted back into clear natural language by prompting the model to describe the structure in plain terms. This allows users to read and write logic statements without needing to understand formal notation.

2.3.6 Example

This example demonstrates an automated puzzle generation pipeline. The system first populates a world with a set of facts and rules. It then formulates a puzzle by selecting a fact as a goal and identifying supporting premises.

Phase 1: Candidate Statement Generation

The system generates a pool of candidate statements by binding logical templates with predicates and entities from configuration files. This creates a small, consistent world of facts and rules.

Generated Statement Pool:

1. `![x1] : (Teaches(x1, programming) => IsIntelligent(x1))`
2. `Teaches(Laura, programming)`
3. `IsProfessor(Laura)`
4. `Learns(Kevin, programming)`
5. `IsIntelligent(Laura)`

Phase 2: Puzzle Assembly and Formulation

The system selects a statement from the pool to serve as the puzzle's **goal**. It then uses a **reverse inference chain** to find the minimal set of other statements from the pool that must serve as **premises** to logically prove this goal.

Puzzle Formulation:

- **Goal Selection:** The system selects statement (5) from the pool as the goal.

  Goal: `IsIntelligent(Laura)`

- **Premise Identification:** To connect the premises to the goal, the system works backward. To prove `IsIntelligent(Laura)`, it finds the rule (1). The **reverse inference chain** determines that this rule requires the antecedent, `Teaches(Laura, programming)`, to be true. The system confirms statement (2) is in the pool and selects it as the necessary premise. To increase complexity, one more implication step is added: Statement (3) implies Statement (2).

Final Puzzle Structure

The system assembles the selected statements into a complete puzzle for the user.

Premises:

1. `![x] : (Teaches(x, programming) => IsIntelligent(x))`

   (Rule: Anyone who teaches programming is intelligent.)

2. `IsProfessor(Laura)`

   (Fact: Laura is a professor.)

3. `(IsProfessor(Laura) => Teaches(Laura, programming))`

   (Rule: If Laura is a professor, then she teaches programming)

Goal:

`IsIntelligent(Laura)`

(Objective: Using the premises, prove that Laura is intelligent.)

Solution Path: Based on the provided premises, the proof forms a two-step logical chain to determine a final conclusion about Laura. The process begins with the specific fact that `IsProfessor(Laura)`. This allows for the application of the rule stating that if Laura is a professor, then she teaches programming `IsProfessor(Laura) => Teaches(Laura, programming)`, which leads to the intermediate conclusion that Teaches(Laura, programming). This new fact then serves as the trigger for the second rule `(![x] : (Teaches(x, programming) => IsIntelligent(x)))`, allowing us to derive the final conclusion that `IsIntelligent(Laura)` through a second application of Modus Ponens.

2.3.7 User's Role and Contribution in Puzzle Generation

In the GPS reasoning framework, puzzles are automatically generated through the combination of domain definitions, logical templates, and inference rules. While the system autonomously constructs puzzles, the **user** plays a crucial supervisory and enrichment role to ensure the overall quality, diversity, and human interpretability of the generated outputs.

Puzzle Verification

- The user evaluates whether a generated puzzle is **human-understandable**, ensuring that the statements, relations, and reasoning flow are clear and interpretable.

Domain and Template Provision

- Users can define new **domains** (i.e., sets of entities, types, and predicates) that expand the system's reasoning scope.
- They can design or modify **templates** that represent structural patterns of reasoning problems (e.g., analogy, deduction, classification).
- These domain and template inputs serve as the foundation for automated puzzle generation, guiding how the system instantiates entities and composes logical statements.

Iterative Feedback and Knowledge Expansion

- By reviewing puzzles and solutions, users provide **iterative feedback** that helps calibrate difficulty levels, linguistic clarity, and logical soundness.
- Each validated solution or rule contributes to the **continuous expansion** of the GPS knowledge repository, improving future puzzle generation and reasoning performance.

2.4 Theorems

2.4.1 Probability of Success Evaluation

Theorem 2.1 (Polarization in Random AST Construction)**.** *Let $\mathcal{T}_d$ be a random Abstract Syntax Tree of depth d, constructed over a finite domain of size $k \geq 2$. The tree grows recursively from a root node according to the following stochastic expansion rules:*

- ***Termination (Leaves):*** *If a node is at depth d, it becomes a predicate atom L with independent random assignment, such that $P(L \to \texttt{True}) = 0.5$.*

- ***Expansion (Internal Nodes):*** *If a node is at depth $< d$, it expands by selecting an operator type uniformly from a symmetric set:*

  - *Unary Expansion: Node $\to \neg(Child)$.*

  - *Binary Expansion: Node $\to (Left \circ Right)$, where $\circ \in \{\wedge, \vee\}$ is chosen with equal probability.*

  - *Quantifier Expansion: Node $\to Qx.(Child)$, where $Q \in \{\forall, \exists\}$ is chosen with equal probability.*

Under this symmetric growth model, the expected evaluation of the root node $\mathbb{E}[Val(root)]$ remains invariant at 0.5. However, as the AST depth d grows, the truth values ***diverge to the extremes****: the probability mass shifts away from 0.5 and clusters near 0 (False) and 1 (True).*

Proof. The proof follows the recursive structure of the AST. Let v_h denote the expected truth value of a node at height h.

Base Case (Leaf Nodes): By the termination rule, leaves are independent Bernoulli trials with $p = 0.5$. Thus, $v_0 = 0.5$.

Recursive Step (Conservation of Mean): Assume the expected value of children nodes at height $h - 1$ is $v_{h-1} = 0.5$.

1. **Unary Branch:** $\mathbb{E}[\neg\text{Child}] = 1 - 0.5 = 0.5$.

2. **Binary Branch:** The symmetric choice of $\wedge$ (bias to 0) and $\vee$ (bias to 1) cancels out:

$$v_h = \frac{1}{2}(0.5^2) + \frac{1}{2}(1 - (1 - 0.5)^2) = 0.5$$

3. **Quantifier Branch:** The symmetric choice of $\forall$ (shrinkage) and $\exists$ (expansion) cancels out:

$$v_h = \frac{1}{2}(0.5^k) + \frac{1}{2}(1 - 0.5^k) = 0.5$$

Thus, the mean is conserved.

Asymptotic Behavior (Divergence): Although the mean v_h is conserved, the equilibrium at 0.5 is unstable. The operators act as non-linear maps that amplify small deviations. As the tree depth increases ($d \to \infty$), any variance in the leaves is magnified, causing the root value to settle at the boundaries of the probability space (0 or 1). □

2.4.2 Correctness of Synthesized Logical Statements

Theorem 2 (Solvability of Logic Puzzles via Structured Workflow). Let $\mathcal{W}$ be the logical puzzle generation workflow consisting of the following steps: (1) parsing natural language into structured formal components, (2) generating a typed first-order model with ground truth assignments, (3) constructing a goal formula consistent with the model and type system, (4) evaluating and pruning the goal formula to its minimal logical form, (5) extracting subgoals and generating backward-chaining inference paths, and (6) composing a puzzle from the resulting axioms, facts, and goals.

Assume:

1. The model generation step assigns truth values to all well-typed predicates over a finite

domain of constants.

2. The goal formula ϕ is constructed using only predicates and constants valid within the model and respects type constraints.

3. The evaluation procedure eval_formula_with_trace(ϕ, M) deterministically returns whether ϕ holds in model M and tracks supporting facts.

4. The pruning step preserves logical equivalence: for any pruned formula ϕ', we have $M \models \phi \Leftrightarrow M \models \phi'$.

5. The subgoal extraction and reverse inference chain generation produce only facts and implications that are true under M.

Then the generated logic puzzle $\mathcal{P} = (\mathcal{A}, \mathcal{F}, \phi)$, consisting of:

- A set of axioms $\mathcal{A}$ (implication rules),
- A set of known facts $\mathcal{F}$ (true or negated atomic formulas),
- A goal formula ϕ (the logical conclusion to be derived),

is *logically solvable*, i.e.,

$$\mathcal{A}, \mathcal{F} \models \phi$$

in the sense of first-order logical entailment. Moreover, a finite forward- or backward-chaining derivation exists from $\mathcal{F}$ and $\mathcal{A}$ to ϕ under the same model M used during puzzle generation.

Proof. By construction, all elements of the puzzle are generated from a model M where truth values are consistently assigned to all well-typed predicates. The goal formula ϕ is guaranteed to be true in M (via evaluation), and all axioms and facts used are either directly extracted from M or logically follow from it via implication chains. Since the inference chains are constructed within bounded depth and only use model-supported facts, a finite derivation of ϕ from $\mathcal{A}$ and $\mathcal{F}$ is guaranteed to exist. Hence, solvability is ensured by the soundness and completeness of first-order logic over finite models.

Lemma 2.1 (Semantic Foundation and Statement Soundness)**.** *All statements generated and transformed by the workflow, including the goal and all potential premises, are true with respect to the ground-truth model M.*

Proof. The proof follows from the soundness of the initial generation and transformation steps.

Initial Statement Generation (Step 2) The workflow generates an initial formula ψ_0 and evaluates it against the model M, which serves as the ground truth (Assumption 1). The system ensures the final statement ψ is true in the model.

- If $M \models \psi_0$, then $\psi = \psi_0$.
- If $M \not\models \psi_0$, then $\psi = \neg\psi_0$, ensuring $M \models \neg\psi_0$.

Therefore, any statement ψ selected for the puzzle is guaranteed to satisfy $\mathbf{M} \models \psi$.

CNF Transformation (Step 3) Statements are converted into Conjunctive Normal Form (CNF). This transformation is equivalence-preserving (Assumption 4): $\psi \Leftrightarrow (c_1 \wedge \cdots \wedge c_n)$. Since we established that $M \models \psi$, it must be that $M \models (c_1 \wedge \cdots \wedge c_n)$. By the semantics of conjunction, this implies $M \models c_i$ for every clause c_i.

Thus, every statement that forms a building block for the puzzle is true in M. □

Lemma 2.2 (Soundness of Backward Chaining)**.** *Let $S_0 = \{\phi\}$ be the initial goal set and S_k be the final set of premises generated by the backward-chaining process. Then the final premises logically entail the initial goal, $S_k \models \phi$.*

Proof. The proof proceeds by showing a chain of entailment from the final premise set back to the initial goal.

Initialization The process starts with a goal formula ϕ where $M \models \phi$ (from Lemma 2.1). The initial premise set to be "explained" is $S_0 = \{\phi\}$.

Inductive Proof of Entailment We prove by induction that for any step i in the process, $S_{i+1} \models S_i$. A step consists of replacing a formula $\psi \in S_i$ with a set of new premises $\Psi' = \{\psi'_1, \ldots, \psi'_m\}$ to form $S_{i+1} = (S_i \setminus \{\psi\}) \cup \Psi'$. By Assumption (5), the generation rule is sound, so $\mathbf{\Psi' \models \psi}$.

To show $S_{i+1} \models S_i$, consider any formula $\chi \in S_i$.

- Case 1: $\chi \neq \psi$. If χ was not replaced, then $\chi \in S_{i+1}$, so $S_{i+1} \models \chi$ trivially.
- Case 2: $\chi = \psi$. We know $\Psi' \subseteq S_{i+1}$ and $\Psi' \models \psi$, so it follows from the monotonicity of entailment that $S_{i+1} \models \psi$.

Since both cases hold, we have proven $S_{i+1} \models S_i$ for any step i.

Transitivity of Entailment We have established a chain of entailment: $S_k \models S_{k-1}, \ldots, S_1 \models S_0$. By the transitivity of logical entailment, it follows directly that $\mathbf{S_k \models S_0}$. Since $S_0 = \{\phi\}$, we conclude that $S_k \models \phi$. □

Proof of Theorem 2. The generated puzzle consists of a goal ϕ and a set of premises (axioms $\mathcal{A}$ and facts $\mathcal{F}$) which together form the knowledge base $\mathcal{K}$. This knowledge base is the final set S_k from the backward-chaining process.

From **Lemma 2.1**, we know that the goal ϕ and all premises in $\mathcal{K}$ are consistent with and true in the ground-truth model M.

From **Lemma 2.2**, we have formally shown that the set of premises $\mathcal{K}$ logically entails the goal ϕ, i.e., $\mathcal{K} \models \phi$.

Since the premises logically entail the conclusion, the puzzle is, by definition, solvable. The constructive nature of the proof of Lemma 2.2 guarantees that a finite derivation path exists.

□

2.4.3 Complexity Analysis

Theorem 3 (Complexity of Logical Puzzle Generation Workflow). Let n denote the number of distinct constants, p the number of unique predicate symbols (with maximum arity r), and d the maximum depth allowed in the goal formula or inference chains. The end-to-end logical puzzle generation workflow described in $\mathcal{W}$ (comprising parsing, model construction, goal synthesis, evaluation, pruning, and inference chain generation) runs in time polynomial in n^r, p, and d.

Specifically, the overall time complexity of the workflow is bounded as follows:

$$\mathcal{O}(p \cdot n^r + T_{\text{gen}}(d) + T_{\text{eval}}(d, n) + T_{\text{prune}}(d, n) + T_{\text{infer}}(d, n))$$

where:

- $T_{\text{gen}}(d)$ is the time to randomly generate a well-typed goal formula of depth d, bounded by $\mathcal{O}(2^d)$ in the worst case due to tree branching.
- $T_{\text{eval}}(d, n)$ is the cost of formula evaluation with trace, typically $\mathcal{O}(n^d)$ for variable bindings and model lookups.

- $T_{\text{prune}}(d, n)$ is the cost of formula simplification, bounded by the size of the formula (i.e., $\mathcal{O}(2^d)$).

- $T_{\text{infer}}(d, n)$ is the cost of reverse inference chain generation for each subgoal, bounded by $\mathcal{O}(p \cdot n^r \cdot d)$ under depth-limited chaining.

Proof.

1. **Model Generation:** For each predicate of arity r, the truth table has $\mathcal{O}(n^r)$ entries. With p predicates, total cost is $\mathcal{O}(p \cdot n^r)$.

2. **Goal Generation:** A logical formula of depth d may have up to $\mathcal{O}(2^d)$ nodes due to binary connectives. Type validation and safe constant/variable usage are local operations, keeping generation polynomial in 2^d.

3. **Formula Evaluation:** Each quantifier introduces a loop over n entities. In the worst case (e.g., nested quantifiers), evaluation takes $\mathcal{O}(n^d)$ time.

4. **Pruning and Subgoal Extraction:** Operate over the formula tree, with size $\mathcal{O}(2^d)$. Each step (evaluation, branch checking) is again polynomial.

5. **Reverse Inference:** For each leaf subgoal, depth-limited backward chaining of length d over randonly generated formula to r costs $\mathcal{O}(2^d \cdot d)$.

Thus, the entire workflow remains tractable for moderate values of n, p, and d, and scales polynomially in the size of the logical model.

2.5 Experiments and Evaluation

In this section, we detail the experimental setup designed to evaluate the quality and solvability of the logical puzzles generated by our system. The primary aims of our experiments

are:

1. To generate a large-scale dataset of logical stories using our three-set puzzle generation framework.

2. To analyze the *diversity* and *complexity* of the generated stories.

3. To evaluate the reasoning capabilities of a symbolic theorem prover (Vampire) and an LLM (ChatGPT) on the generated dataset.

2.5.1 Dataset Generation

We used the proposed three-set puzzle generator to create a collection of logical stories. Each story is defined as a set of premises and a goal triple:

$$\{\mathcal{P}, g\}$$

where $\mathcal{P}$ is the set of premises, g is the proof goal. The generation process included:

- Randomized selection of goal statements from a template-based generator.
- Reverse application of inference rules to form proof graphs.
- Controlled generation of distractor statements to increase difficulty.
- Parameter variation to produce stories of varying lengths and branching factors.

We generated a dataset of 4800 stories, grouped by premise counts of 5, 10, 20, and 40 per story.

Domain Configuration

We set 6 domains by domain size and whether the predicates, types, and entities are abstract

or not.

Vampire Configuration

The logical correctness can be verified by Vampire: a world-champion automated theorem prover that provides a fast and formal symbolic reasoning procedure.

- **Input format:** TPTP encoding of premises and the goal, $\{\mathcal{P}, g\}$.
- **Proof strategy:** Standard mode with a time limit of t_{sym} seconds per query.
- **Evaluation metric:** Success rate in correctly classifying each statement as derivable (true) or not (false).

Vampire Validation Results

Table 2.1: Vampire validation by domain and by premise count

Scope	Bucket	Total	Proved	(%)	Avg. runtime (s)	Avg. proof steps
By domain						
Domain	small	800	800	100.0	0.01	73.2
Domain	medium	800	800	100.0	0.01	112.3
Domain	large	800	800	100.0	0.02	165.0
Domain	small_abstract	800	800	100.0	0.01	72.7
Domain	medium_abstract	800	800	100.0	0.01	107.0
Domain	large_abstract	800	800	100.0	0.02	191.0
By premise count						
Premises	5	1200	1200	100.0	0.01	68.5
Premises	10	1200	1200	100.0	0.01	100.6
Premises	20	1200	1200	100.0	0.01	135.0
Premises	40	1200	1200	100.0	0.02	176.7
Overall						
Overall	all domains	4800	4800	100.0	0.01	120.0

Results Summary. Table 1 presents the evaluation results of the automatically generated puzzles across various domain settings and premise counts. All generated puzzles achieve a **100% proof success rate**, indicating that every puzzle is **logically correct and provable** within the GPS reasoning framework.

As shown in the table, the **average number of proof steps increases** with both domain size and the number of premises, demonstrating that the system can effectively **control the complexity** of generated puzzles by adjusting these parameters. Despite the increased reasoning depth, the **average runtime remains extremely low** (approximately 0.01–

0.02 seconds), highlighting the computational efficiency of both puzzle generation and proof verification.

Overall, these results confirm that the GPS system consistently produces **logically sound, scalable, and efficiently verifiable** reasoning puzzles.

2.5.2 Diversity and Complexity Analysis

We analyzed the generated puzzles to assess their diversity and logical complexity. The evaluation considered four main aspects:

- **Logical Complexity:** Measured by predicate count, nesting depth, and use of quantifiers and multi-step reasoning.
- **Structural Diversity:** Examined proof structures, including inference rule distribution, proof depth, and branching.
- **Semantic Variation:** Evaluated variation in story themes, entities, and unique predicates or constants.
- **Readability:** Assessed the clarity of natural-language premises using readability metrics.

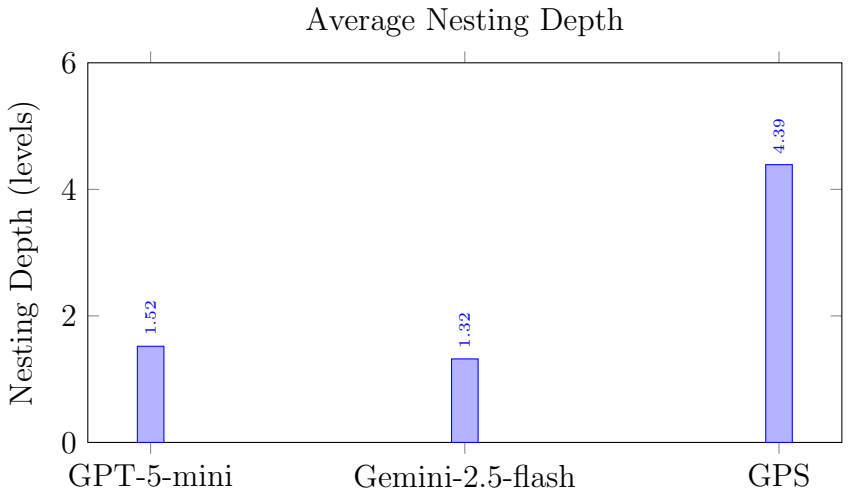


Figure 2.1: Average nesting depth across GPT-5-mini, Gemini-2.5-flash, and GPS.

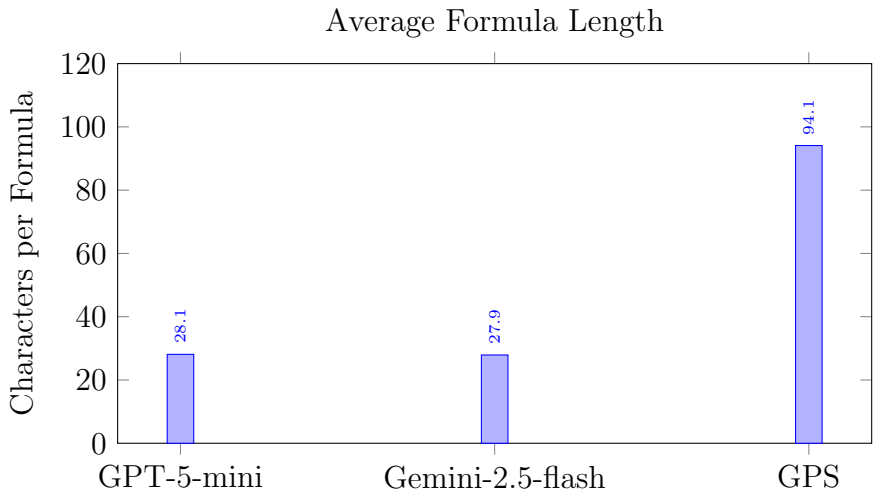


Figure 2.2: Average formula length across GPT-5-mini, Gemini-2.5-flash, and GPS.

Visual analyses (histograms, network plots) summarize these characteristics. The lower number of `implies` operators in GPS results from formula normalization, each implication in the premises is rewritten as $\neg A \vee B$ for logical simplification. Thus, despite the transformation, GPS still produces puzzles with greater structural depth and logical diversity than GPT-5-mini and Gemini-2.5-flash.

In Fig. 2.1, GPS shows a markedly higher average nesting depth (4.39) compared to GPT-5-mini (1.52) and Gemini-2.5-flash (1.32), indicating deeper compositional reasoning. In Fig. 2.2, the average length of the formula in GPS (94.1 characters) is more than triple that of the other systems, showing a richer logical structure with longer dependency chains.

Regarding operator usage (Fig. 2.3), GPS relies heavily on `and`, `or`, and `not`, forming layered clause structures, while GPT-5-mini and Gemini-2.5-flash overuse `implies`, reflecting more linear reasoning. The structure pattern distribution (Fig. 2.4) further shows GPS's preference for quantified (75.9%) and conjunctive (36.2%) forms, unlike the simpler fact-based outputs of other models.

Together with higher nesting depth (Fig. 2.1) and longer formulas (Fig. 2.2), these results confirm that GPS produces puzzles with greater syntactic complexity and semantic diversity,

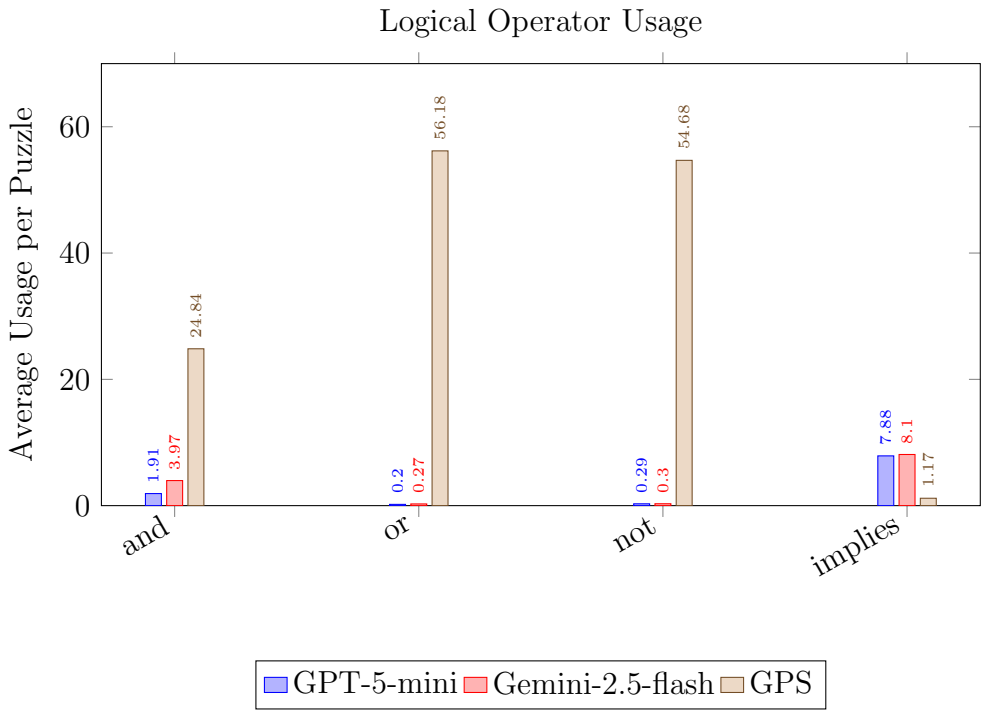


Figure 2.3: Logical operator usage (mean counts per puzzle).

providing a more rigorous benchmark for LLM reasoning.

2.5.3 LLM Reasoning Capability

Reasoning Evaluation Setup

The primary goal is to generate puzzles that provide a robust challenge for automated reasoning systems. To this end, we evaluated the performance of two leading systems on our dataset:

1. **ChatGPT:** A state-of-the-art large language model that exemplifies current progress in generative AI and general-purpose reasoning.

2. **Gemini:** Another leading large language model designed for high-level reasoning and multimodal understanding, representing a complementary approach to ChatGPT.

ChatGPT & Gemini Configuration

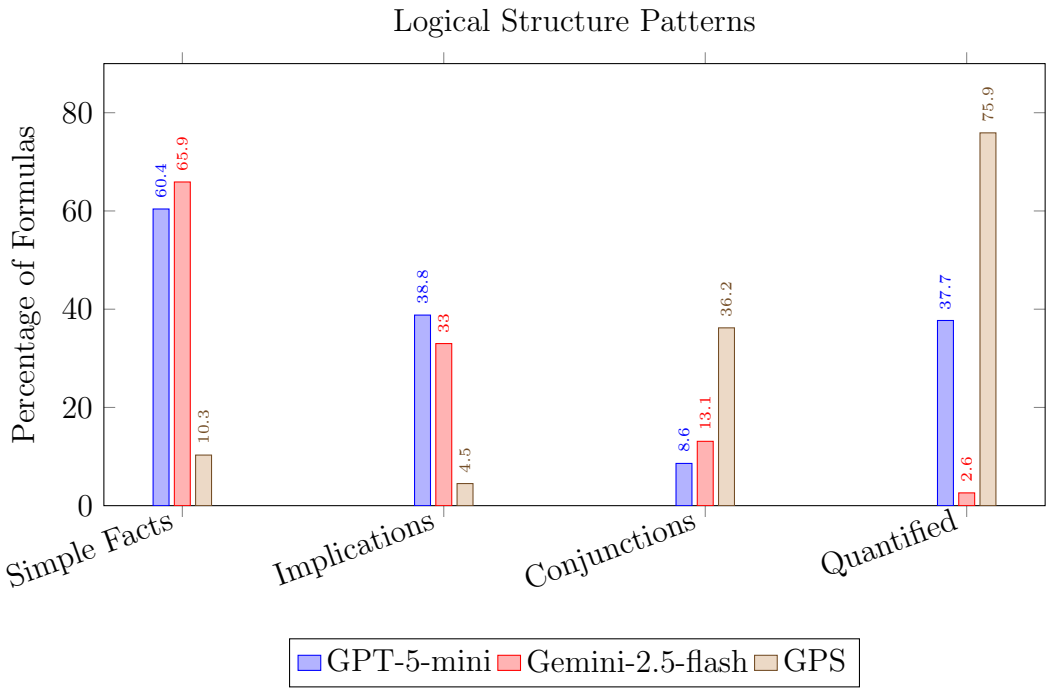


Figure 2.4: Logical structure pattern distribution across GPT-5-mini, Gemini-2.5-flash, and GPS.

- **Input format:** Natural language rendering of the logical story, with explicit premises and candidate statements to be evaluated.
- **Prompt design:** Few-shot examples demonstrating the task, followed by the target story for classification.
- **Evaluation metric:** Accuracy of true/false classification and a qualitative analysis of the generated reasoning explanations.

Evaluation Protocol

For each story in the dataset, the evaluation protocol is as follows:

1. Provide the premises and the goal, $\{\mathcal{P}, g\}$, to the reasoning system.
2. Record the predicted truth value and, if available, the explanation or proof.

3. Compare the system's prediction against the ground truth label.

From these results, we compute:

- Overall classification accuracy per system.
- Accuracy as a function of puzzle complexity (e.g., proof depth and branching factor).
- The distribution of errors across different inference rule types.

ChatGPT and Gemini Results

We tested 1200 puzzles generated by our GPS system to evaluate the reasoning capability of modern LLMs, the results are shown in Table 2. As we can see, GPT with high reasoning capability got the highest overall accuracy. With median reasoning, Gemini has a slight edge in general: it holds up better as the number of premises grows (20–40), while GPT fades more quickly. Switch to high reasoning and the script switches: GPT leads overall and in most domains, with Gemini still hanging tough in a few spots and staying relatively strong at the 40-premise mark.

2.5.4 LLM Generation Capability

As we can see, both GPT5-mini and Gemini-2.5-flash have reasoning capability to some extent. We evaluated the puzzle generation capability of GPT and Gemini by asking them to generate 240 puzzles and showing puzzle generation capability for Gemini and GPT, measured as the percentage of generated puzzles that pass the vampire check. From Table 3, we can find that GPT is far more reliable than Gemini: 97.1% of GPT's puzzles pass versus 55.0% for Gemini overall. The gap widens with longer inputs—at 40 premises, GPT still passes 95.0%, while Gemini falls to 36.7%).

However, focusing on the vampire steps, which can show the complexity of the puzzle, we can observe that ChatGPT and Gemini do not generate puzzle with comparable complexity

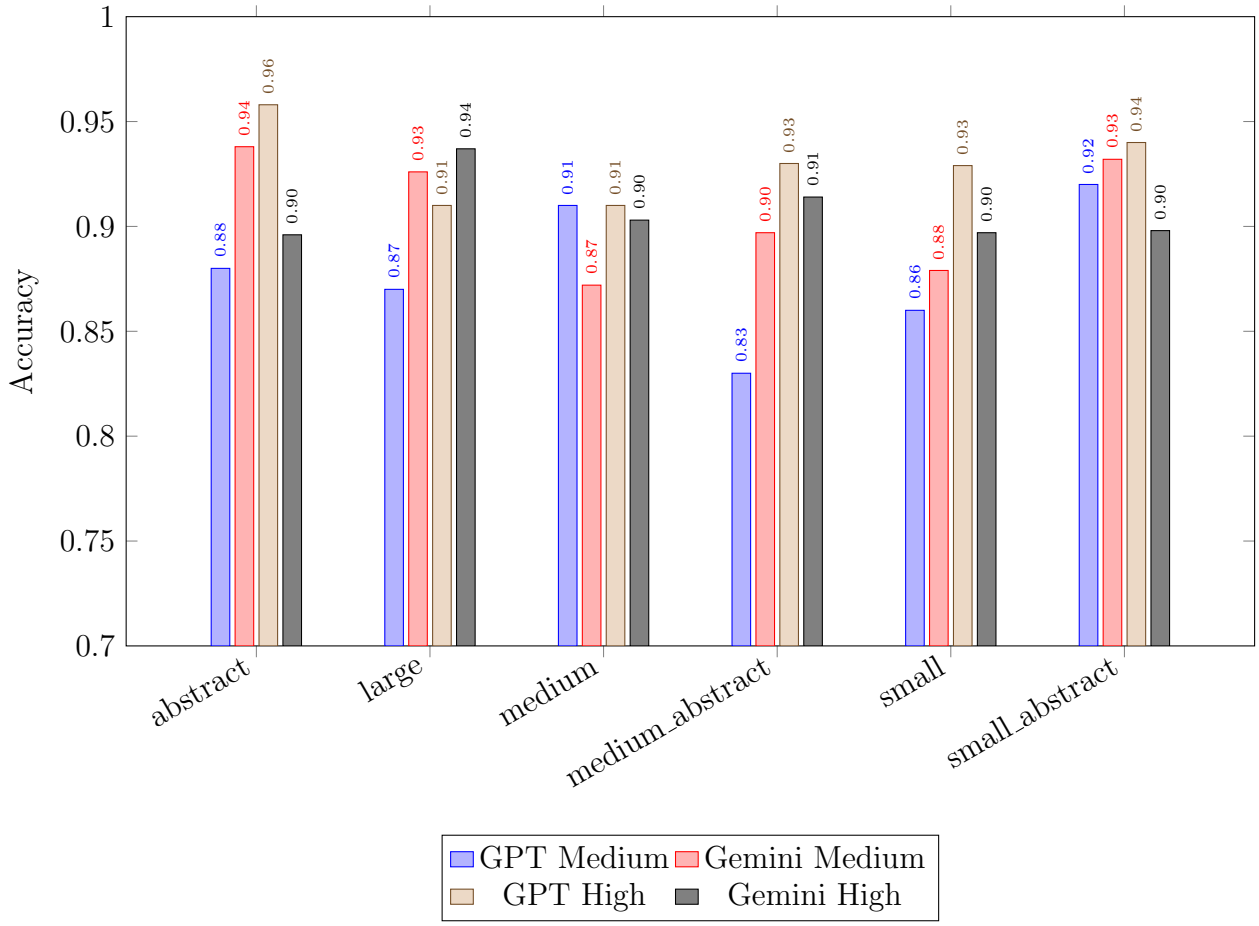


Figure 2.5: Overall accuracy per domain (GPT vs. Gemini, Medium vs. High)

compared to our generation approach.

2.6 Conclusion and Discussion

GPS-Reasoning aims to generate logically sound and human-interpretable logical puzzles that can be verified by a theorem prover, turning puzzle generation into a reproducible benchmark for evaluating LLM reasoning.

Compared with existing automatically generated datasets, which either lack the complexity to challenge LLMs or are not human-readable, GPS balances scalability, semantic meaningfulness, and proof-grounded correctness. Each puzzle is supported by a symbolic engine that validates its logical proof. A variety of parameters, such as the number of premises, do-

main size, and template difficulty, can be adjusted to produce puzzles of varying complexity. Human oversight remains integral: users can examine the domain, template, and generated puzzles to ensure interpretability and quality. GPS leverages the strengths of LLMs in translating between natural and formal logic, while symbolic provers such as Vampire ensure correctness in the generation process.

Looking ahead, we plan to push GPS-Reasoning further into a neural-symbolic like pipeline that not only evaluates but also make GPS reasoning outperforms LLM reasoning. In this hybrid framework, LLMs will propose structured intermediate conclusions such as lemmas, case splits, or invariants, while the theorem prover completes and certifies the remaining proof obligations. This division of labor provides two key advantages: (1) it guarantees formal correctness through symbolic verification, and (2) it mitigates the search-space explosion problem by guiding proof exploration with semantically informed intermediate steps from the LLM.

*Table 2.2: Merged accuracy table (GPT vs. Gemini) with **Medium** and **High** reasoning shown side-by-side. For Gemini, reasoning capability is controlled by thinking budget.*

Domain	Premises	**Medium**		**High**	
		GPT Acc.	Gemini Acc.	GPT Acc.	Gemini Acc.
large	5	1.000	0.920	1.000	0.960
	10	0.960	0.920	1.000	0.960
	20	0.920	0.960	0.920	0.960
	40	0.760	0.920	0.840	0.960
	Overall	**0.870**	**0.926**	**0.910**	**0.937**
medium	5	0.960	0.960	1.000	1.000
	10	1.000	0.960	1.000	0.920
	20	1.000	0.840	1.000	0.960
	40	0.680	0.729	0.760	0.727
	Overall	**0.910**	**0.872**	**0.910**	**0.903**
small	5	1.000	0.920	1.000	0.960
	10	0.960	0.880	1.000	0.920
	20	0.960	0.840	1.000	0.880
	40	0.760	0.870	0.800	0.913
	Overall	**0.860**	**0.879**	**0.929**	**0.897**
large_abstract	5	1.000	0.920	1.000	0.960
	10	0.960	1.000	1.000	0.840
	20	0.800	0.920	0.960	0.958
	40	0.760	0.909	0.840	0.913
	Overall	**0.880**	**0.938**	**0.958**	**0.896**
medium_abstract	5	0.960	0.920	1.000	0.960
	10	0.880	0.840	0.920	0.960
	20	0.760	0.920	1.000	0.960
	40	0.720	0.909	0.800	0.920
	Overall	**0.830**	**0.897**	**0.930**	**0.914**
small_abstract	5	1.000	0.960	1.000	0.960
	10	0.960	0.800	1.000	0.864
	20	0.880	0.840	1.000	0.909
	40	0.600	0.957	0.800	0.905
	Overall	**0.920**	**0.932**	**0.940**	**0.898**
Premise-size overall across all domains					
	Premises	GPT (Med)	Gemini (Med)	GPT (High)	Gemini (High)
	5	0.987	0.946	0.993	0.951
	10	0.940	0.921	0.987	0.883
	20	0.880	0.889	0.953	0.914
	40	0.707	0.869	0.779	0.879
All domains overall		**0.878**	**0.907**	**0.929**	**0.908**

Table 2.3: Puzzle-generation capability of GPT and Gemini measured by the percentage of generated puzzles which pass the verification of Vampire.

Bucket	Tests	GPT pass (%)	Avg. steps	Gemini pass (%)	Avg. steps
By domain (configuration)					
small	40	95.0	18.6	45.0	22.8
medium	40	92.5	15.9	47.5	25.0
large	40	95.0	19.4	47.5	25.2
small_abstract	40	100.0	24.0	52.5	39.9
medium_abstract	40	100.0	18.9	40.0	40.7
large_abstract	40	100.0	21.1	52.5	38.4
By premise count					
5	60	100.0	17.5	60.0	20.5
10	60	98.3	20.6	55.0	36.5
20	60	95.0	21.5	41.7	35.3
40	60	95.0	19.3	33.3	41.8
Overall					
all domains	240	**97.1**	**19.7**	**47.5**	**32.1**

Chapter 3

Dynamic Network Generation (IPGS)

3.1 Introduction

Influence maximization (IM) is a central problem in network science that studies how information, behaviors, or innovations propagate through interconnected systems. The goal is to identify a small set of seed nodes whose activation leads to the largest expected spread under a given diffusion model. This principle has guided research in viral marketing, public health interventions, political campaigning, and information recommendation. Since the seminal work by Kempe et al. [14], IM has evolved into a rich field exploring stochastic diffusion processes, scalable algorithms, and theoretical guarantees on submodular optimization.

Despite this progress, most IM studies share a key simplifying assumption: the underlying network remains static. The diffusion model operates on a fixed graph, where influence is estimated once, seeds are selected once, and the evaluation is performed once. In contrast, real-world networks are not static containers of influence but living systems. Social platforms continuously gain new users, citation networks expand with new papers, and contact graphs in epidemiology change as individuals move or form new interactions. These changes reshape

the flow of influence and can significantly alter the effectiveness of previously optimal seed sets.

Understanding how influence relationships evolve or remain stable as the network changes is critical for both theory and application. For instance, marketers may wish to study whether an existing set of influencers continues to dominate engagement as new customers join the platform. Epidemiologists may analyze whether vaccination or containment nodes retain their coverage as a population grows or interacts differently. Even in scientific collaboration or information diffusion, researchers might want to know how a foundational group's impact persists in an expanding network. Such questions cannot be answered by static benchmarks that treat the graph as frozen in time.

Existing and widely adopted graph generation approaches, including Erdős–Rényi [10], Barabási–Albert [2], and Stochastic Block Models [13], were designed to reproduce structural features such as degree distributions and community organization. These models are valuable for studying topological properties but are not designed to capture influence diffusion behavior. The graphs they generate may appear realistic in form yet reveal little about how influence propagates or whether dominant nodes maintain their reach. In contrast, dynamic IM frameworks address temporal networks by reselecting or adapting seed sets as the graph changes, yet they do not control how the network itself evolves. In essence, one line of research models growth without influence, while the other models influence without growth.

Influence-Preserving Graph Synthesis (IPGS) seeks to bridge this gap. It introduces a two-layer framework that separates influence estimation from graph generation. At its core, IPGS maintains accurate influence estimates under network growth using an **unbiased cached Reverse Influence Sampling (cRIS) repair** mechanism and a **stratified Reverse-Reachable (RR) Sampling** estimator. Here, the term "stratified" refers exclusively to the sampling phase: as the graph expands in discrete batches, RR sets are sampled and updated within each batch (stratum) and then combined through a stratified estimator.

This design allows IPGS to remain unbiased for the current graph G' while only repairing RR sets affected by local structural changes, avoiding full resampling.

On top of this estimation layer, IPGS supports multiple graph generation modes, including random addition and influence-weighted addition, to model how new nodes and edges enter the system. These generation modes enable controlled analysis of how structural evolution interacts with the stability of influence diffusion. A **seed node constraint** ensures that the reference seed set $S^\star$ retains its identity throughout network growth. When the maximal coverage phase detects that a non-seed node would replace an existing seed, IPGS activates a local correction process: it introduces high-probability edges iteratively from displaced seeds to competing nodes and performs deterministic RR expansion to update only the affected RR sets. This targeted repair restores unbiased estimation and establishes influence for original seed nodes without re-sampling the entire network, maintaining diffusion stability under incremental graph evolution.

We formalize the condition for influence-preserving growth and show that unbiased cRIS repair combined with stratified estimation guarantees stability of seed influence within a predictable tolerance. Furthermore, to estimate influence efficiently during graph expansion, IPGS incrementally maintains cached RR sets, enabling near-linear updates while preserving unbiasedness for each post-growth graph snapshot. The result is a unified framework that connects influence theory with graph synthesis while maintaining theoretical tractability and practical scalability.

Rather than competing with existing graph generators, IPGS complements them by introducing an influence-aware dimension to network evolution. Traditional generators emphasize how graphs look, while IPGS emphasizes how they behave under diffusion. By maintaining both topological diversity and consistent diffusion behavior, IPGS provides a principled benchmark for studying the robustness of influence under structural evolution. This dual-layer design, which combines stratified RR sampling, unbiased cRIS repair, the seed node

constraint, and RR Fix, offers a unified foundation for influence-aware network modeling that connects generative processes, diffusion dynamics, and influence maximization within a single framework.

3.2 Background

3.2.1 Influence Maximization and Diffusion Models

The *Influence Maximization* (IM) problem, first formalized by Kempe et al. [14], seeks a set of k seed nodes whose activation maximizes the expected number of subsequently activated nodes under a given diffusion process. It provides a mathematical foundation for understanding how information, opinions, behaviors, or diseases propagate through complex networks. Formally, IM assumes a directed graph $G = (V, E)$, where each node represents an agent and edges encode potential influence relationships. A diffusion model specifies how influence spreads along these edges once the initial seed set $S \subseteq V$ is activated.

Two classical models form the basis for most subsequent work:

- **Independent Cascade (IC)** [14]: Each newly activated node has one opportunity to activate each of its inactive neighbors independently with a given probability p_{uv} for edge (u, v).

- **Linear Threshold (LT)** [14]: Each node v has a threshold θ_v drawn uniformly from $[0, 1]$ and becomes active when the weighted sum of its active neighbors exceeds this threshold.

Both models capture different aspects of diffusion dynamics: IC emphasizes independent probabilistic influence, while LT models collective reinforcement. Since the original work demonstrated that IM is NP-hard but submodular, a greedy hill-climbing algorithm can achieve a $(1 - 1/e)$ approximation guarantee, inspiring extensive research on both theory

and scalability.

Over the years, numerous improvements have been proposed to make IM practical for large-scale networks. Greedy optimizations such as CELF [6] and CELF++ [11] exploit submodularity to reduce redundant computations, while heuristic and sampling-based methods accelerate influence estimation. Later frameworks introduced sketching, community decomposition, and parallelization strategies to scale IM to graphs with billions of edges. Despite this progress, most approaches still rely on static network assumptions, where the topology and edge probabilities are fixed during optimization.

3.2.2 Reverse Influence Sampling (RIS)

Reverse Influence Sampling (RIS), proposed by Borgs et al. [4], revolutionized IM by transforming Monte-Carlo influence estimation into a more efficient sampling process. Instead of simulating the forward diffusion process multiple times, RIS samples *Reverse-Reachable (RR)* sets from randomly chosen nodes. A node u is included in an RR set R if it can reach a randomly selected target node t under the reverse diffusion model. This formulation allows influence spread to be estimated as:

$$\sigma(S) \approx \frac{n}{\theta} \sum_{i=1}^{\theta} \mathbf{1}(S \cap R_i \neq \emptyset),$$

where θ denotes the number of RR samples. A larger number of RR sets improves accuracy at the cost of computational time.

RIS-based algorithms such as TIM/TIM+ [22] and IMM [21] achieved near-optimal sample complexity and strong theoretical guarantees, making IM tractable for large graphs. However, a key limitation of RIS is its assumption of static topology: if nodes or edges are added, the RR sets become invalid, requiring complete regeneration to maintain unbiasedness. This recomputation is costly and hinders the use of RIS in evolving networks. To address this,

our framework introduces *Cached Reverse Influence Sampling (cRIS)*, which incrementally maintains cached local RR sets for affected regions, avoiding full resampling during graph expansion.

3.2.3 Dynamic Graphs and Influence Maintenance

Real-world networks rarely remain static. Social networks continuously attract new users and connections; citation graphs expand as new papers are published; and epidemiological networks change as people move or interact differently. These evolving structures alter diffusion pathways, often invalidating influence estimates obtained from static models.

Existing dynamic IM frameworks attempt to adapt to such changes. Some methods [29, 7] periodically recompute seed sets after topology updates, while others resample RR sets to reflect the new structure. Although these adaptive strategies maintain short-term accuracy, they focus on re-optimizing seed selection for each new snapshot rather than maintaining the *relative influence landscape* of an existing seed set. In contrast, our approach emphasizes preserving influence relationships during structural growth, ensuring that the diffusion reach of initial seeds remains stable over time without frequent reselection.

3.2.4 Graph Generation Under Topological and Structural Constraints

Graph generation models provide another line of research relevant to IPGS. Classical random graph models, such as Erdős–Rényi [10], Barabási–Albert [2], and Stochastic Block Models (SBM) [13], were developed to study network topology by reproducing properties like degree distribution, clustering, and community modularity. While these models are valuable for studying network topology, they do not account for the dynamic processes that unfold across those structures. Their design emphasizes reproducing statistical regularities rather than capturing how influence propagates through the network.

Recent research has expanded constrained graph generation beyond classical topological properties such as degree distribution or community modularity. Ying and Wu [31] introduced a framework for *Graph Generation with Prescribed Feature Constraints*, which employs Markov Chain Monte Carlo (MCMC) switching techniques to synthesize graphs that satisfy statistically defined feature constraints. Although their work primarily focuses on structural attributes such as degree correlations and clustering behavior, it provides a methodological foundation for enforcing global graph properties during generative procedures. More recently, Madeira et al. [16] proposed a differentiable constraint-based generative framework that maintains structural validity (e.g., planarity, connectivity) throughout the generation trajectory. While these approaches represent significant progress toward constraint-aware synthesis, they remain centered on static topological constraints rather than preserving dynamic diffusion characteristics such as influence spread, reachability distributions, or seed-based activation stability. In contrast, our framework introduces generative modeling under *process-aware constraints*, focusing on influence preservation during network growth rather than merely reproducing structural regularities.

Recent deep generative models, including NetGAN [3] and GraphRNN [32], leverage neural architectures to learn realistic graph structures from data. Although they produce networks with complex dependencies and high fidelity, they remain *influence-agnostic*, their objectives prioritize topological realism rather than maintaining diffusion behavior or preserving seed influence. Moreover, these methods are predominantly data-driven and lack theoretical guarantees regarding the structural or diffusion properties of the generated graphs. Consequently, they cannot serve as controlled or analytically verifiable benchmarks for influence-based evaluation.

3.2.5 Influence Maximization in Dynamic Graphs

Dynamic IM frameworks explicitly address network evolution. For instance, Peng et al. [18] proposed a dynamic IM algorithm that recomputes RR sets after each update to preserve uniform coverage guarantees. Although effective in maintaining unbiased influence estimation, these methods incur significant computational cost and are not designed to preserve the influential status of original seed set. Our framework takes a complementary perspective: instead of repeatedly re-optimizing seed selection, IPGS models graph growth while ensuring that the influence distribution induced by an initial seed set remains consistent within bounded error.

3.2.6 Summary

To summarize, prior work on influence maximization, diffusion modeling, and graph generation has advanced independently. Classical IM frameworks optimize seed selection on static graphs, while dynamic methods adapt to structural changes but neglect stability of influence relationships. Meanwhile, generative models successfully reproduce network topology but overlook diffusion behavior entirely. *Influence-Preserving Graph Synthesis (IPGS)* bridges these research directions by introducing a generative process that preserves influence during network growth, offering a unified framework for studying diffusion stability, seed robustness, and influence-aware graph synthesis.

3.3 Methods

3.3.1 Overview of IPGS

The *Influence-Preserving Graph Synthesis (IPGS)* framework models how a dynamic network grows while maintaining the diffusion behavior of a fixed seed set $S^\star$. Unlike conventional random graph generators (e.g., Erdős–Rényi or Barabási–Albert) that replicate only

structural statistics such as degree distribution, IPGS explicitly preserves *influence diffusion* during network evolution.

Core principle. IPGS couples graph growth with an online influence estimator based on **cached Reverse Influence Sampling (cRIS)**. Each growth step expands the graph by adding a batch of new nodes and edges, incrementally updating the cached RR sets instead of recomputing them from scratch. Influence estimation therefore remains unbiased for the current graph and stable across time.

Three interacting layers.

1. **Stratified influence estimation:** cRIS partitions RR samples into temporal strata and applies Neyman-optimal allocation, ensuring an unbiased and minimum-variance estimator (Theorem 3.1).

2. **Bounded influence drift:** Hoeffding-type concentration (Theorem 3.2) constrains the change in RR frequencies between successive timesteps, guaranteeing that $\hat{f}_{t+1}(v) - \hat{f}_t(v) \leq \eta_s$ for all $v \in V_t$ with high probability.

3. **Heuristic retention control:** When the empirical influence of $S^\star$ declines beyond tolerance, IPGS activates a **Seed node constraint** that introduces small high-probability edges from displaced seeds to new competitors (Section 3.3.4). This deterministic "influence-boosting" step stabilizes seed competitiveness without re-sampling the entire RR cache.

Resulting behavior. Together, these layers transform influence maximization from a static, re-optimization problem into a continuously updated, self-correcting process. IPGS maintains both estimator unbiasedness and practical diffusion stability, supporting long-horizon graph growth where the influence of $S^\star$ remains statistically consistent even as topology changes.

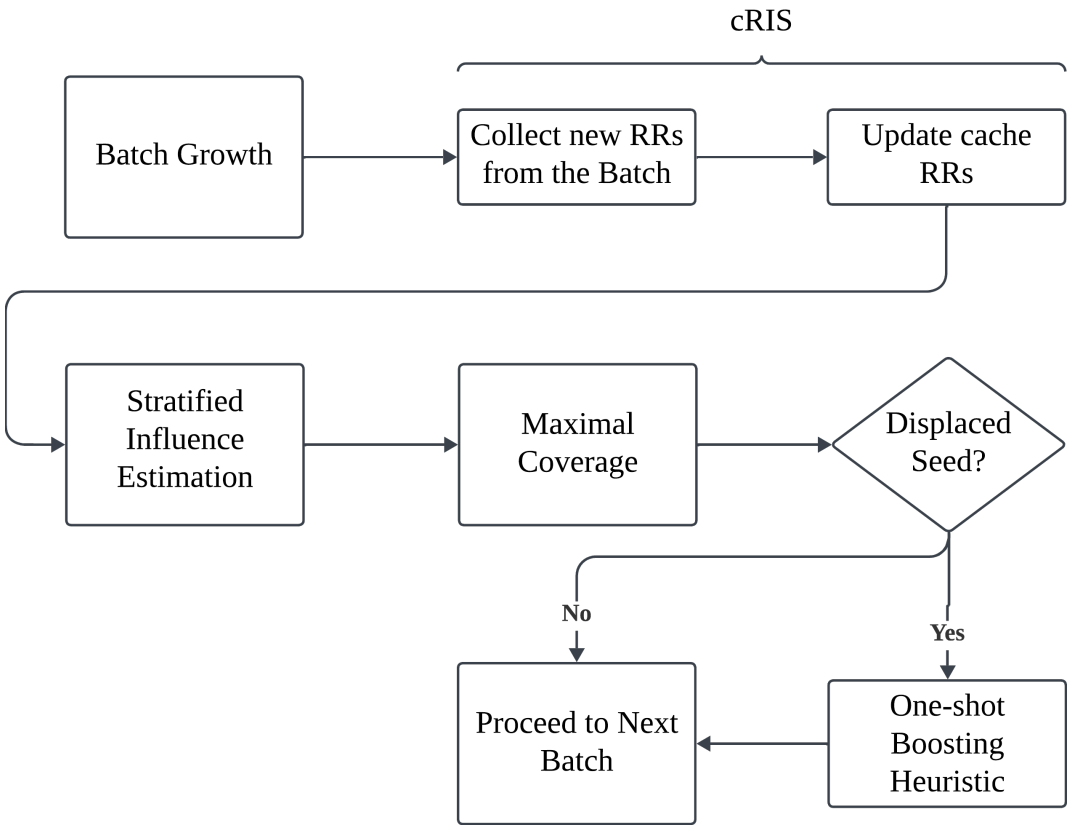


Figure 3.1: Workflow of IPGS. Each batch adds new nodes and edges $(\Delta V_t, \Delta E_t)$, followed by cached RR updates. When the estimated influence of $S^\star$ drifts beyond tolerance, the relaxed constraint introduces compensatory edges and deterministic RR-set expansions to restore local diffusion balance.

3.3.2 Dynamic Influence Model

Let $G_t = (V_t, E_t, p)$ denote a directed network at time t, where each edge $(u, v) \in E_t$ is associated with an activation probability p_{uv} under the Independent Cascade (IC) model. Given a fixed seed set $S^\star \subseteq V_0$, the expected diffusion at time t is:

$$\sigma_t(S^\star) = \mathbb{E}[|\Gamma_t(S^\star)|],$$

where $\Gamma_t(S^\star)$ denotes the random set of activated nodes reachable from $S^\star$. During each growth step, the graph expands through new nodes and edges:

$$G_{t+1} = (V_t \cup \Delta V_t,\ E_t \cup \Delta E_t).$$

Ideal and Practical Preservation. In theory, a growth process $\mathcal{G}$ is *strictly influence-preserving* if the relative ranking of all seed sets remains invariant across time. Formally, for any T with $|T| = |S^\star|$, if $\sigma_t(S^\star) \geq \sigma_t(T) + \Delta_t$, then ideally

$$\sigma_{t+1}(S^\star) \geq \sigma_{t+1}(T) + \Delta_t(1 - \varepsilon),$$

for a small tolerance $\varepsilon \geq 0$. Checking this global stability criterion is infeasible because it requires evaluating all k-sized subsets after every update. Instead, IPGS enforces a *local and sample-based criterion* that bounds how much the estimated influence of each node may drift between successive timesteps.

Local Influence Drift. Let $\hat{f}_t(v)$ denote the normalized frequency of node v in the cached Reverse-Reachable (RR) sets at time t, representing its empirical influence. IPGS constrains the temporal variation of these frequencies by

$$|\hat{f}_{t+1}(v) - \hat{f}_t(v)| \leq \eta_s, \quad \forall v \in V_t,$$

where η_s is a small stochastic bound derived from Hoeffding's inequality (Theorem 3.2). When this condition holds, the total variation distance between consecutive RR distributions satisfies

$$\mathrm{TV}(\mathcal{R}_{G_t}, \mathcal{R}_{G_{t+1}}) = O(\eta_s),$$

implying that the expected diffusion $\sigma_t(S^\star)$ remains stable within a predictable tolerance as the network evolves.

Enforcement via Local Repair and Boosting. To maintain bounded drift, IPGS monitors the greedy coverage result after each batch integration. When a non-seed node temporarily overtakes a member of $S^\star$, indicating that local influence has shifted beyond tolerance,

IPGS activates a localized correction. Rather than pruning edges or re-sampling all RR sets, it performs two lightweight operations:

1. *RR maintenance:* only the affected fraction $\lambda\theta$ of RR sets (those intersecting the changed region) are expanded or reweighted, preserving unbiasedness within the cache.

2. *Influence boosting:* a small number of high-probability edges are added from displaced seeds to newly competitive nodes, restoring influence balance without stochastic re-sampling (Section 3.3.4).

These adjustments are deterministic and local, ensuring that the overall estimator remains stable while the diffusion dynamics adapt smoothly to graph growth.

Theoretical Guarantee. Theorem 3.3 shows that the combined cRIS update and boosting process maintain bounded frequency variation:

$$|\hat{f}_{t+1}(v) - \hat{f}_t(v)| \leq O(\lambda + \bar{\varepsilon}),$$

where λ is the proportion of RR sets refreshed per batch and $\bar{\varepsilon}$ is the per-node concentration tolerance implied by the Hoeffding bound. Because all unaffected RR samples are reused, the cRIS estimator remains unbiased for the updated graph while its variance stays within theoretical limits.

Intuitive Example. Consider an initial network with $S^\star = \{A, B\}$. If a new node F connects primarily to A and B, their RR frequencies increase proportionally, preserving seed node influence. If F instead connects to non-seed nodes C and D, their RR frequencies rise disproportionately, violating the drift condition. IPGS then triggers local maintenance on the RR sets involving C and D and, if necessary, introduces small compensatory edges from A or B toward C and D. This restores the balance of expected influence without global resampling.

Batch Growth and Temporal Strata. Graph evolution proceeds in discrete batches $\{\Delta V_1, \Delta V_2, \dots\}$, each corresponding to a temporal stratum. Influence estimation within each stratum is locally unbiased, and global influence is obtained by stratified aggregation (Section 3.3.3). This organization allows incremental, theoretically grounded updates while ensuring that localized boosting or RR maintenance never disturbs the unbiasedness of the overall estimator.

3.3.3 Cached RR Sampling and Maintenance (cRIS)

Reverse Influence Sampling (RIS) provides an unbiased estimator of expected influence in static graphs by sampling *Reverse-Reachable (RR)* sets, which represent potential activation paths in the reversed diffusion process. Each RR set corresponds to a randomly chosen target node and includes all nodes that can reach it through probabilistic activations. For any seed set S, the expected influence $\sigma_G(S)$ equals the probability that S intersects a random RR set, scaled by the total number of nodes:

$$\hat{\sigma}(S) = \frac{n}{\theta} \sum_{i=1}^{\theta} \mathbf{1}(S \cap R_i \neq \emptyset),$$

where θ is the number of RR samples, $n = |V|$, and R_i is the i-th RR set drawn from the RR distribution $\mathcal{R}_G$.

Motivation. Classical RIS assumes a static topology, yet real networks, such as social or citation graphs, grow continually as new nodes and edges appear. Re-sampling all RR sets after each update would cost $O(|E_t|)$ time per step and quickly become infeasible at scale. To address this, IPGS generalizes RIS into a dynamic and incremental variant called **cached Reverse Influence Sampling (cRIS)**. Instead of discarding all existing RR samples, cRIS maintains a persistent cache $\mathcal{C}_t$ that evolves together with the graph, updating only the regions affected by topological change.

At time t, IPGS maintains:

$$\mathcal{C}_t = \{R_1, R_2, \ldots, R_\theta\}, \quad R_i \sim \mathcal{R}_{G_t}.$$

When a new batch of nodes or edges $(\Delta V_t, \Delta E_t)$ is added, cRIS identifies which cached RR sets intersect these regions and selectively updates only those sets. This localized maintenance reduces update complexity from $O(|E_t|)$ to $O(|\Delta E_t|)$ while preserving unbiasedness for the current graph:

$$\hat{\sigma}_t(S) = \frac{n_t}{\theta_t} \sum_{i=1}^{\theta_t} \mathbf{1}(S \cap R_i \neq \emptyset), \text{where } n_t = |V_t|,$$

θ_t is the number of RR sampled in this batch.

Locality of Updates. Each RR set corresponds to a rooted subgraph in the reversed diffusion model. When a new edge (u, v) is added, only RR sets that contain v (or nodes reachable from v) are affected, because only those reachability paths expand through u. All other RR sets remain valid, ensuring that influence estimates are adjusted *locally* and smoothly with respect to graph evolution.

Example. Suppose a cached RR set $R = \{v_1, v_2, v_3\}$, rooted at v_3, represents nodes that can activate v_3 in the reversed process. If a new edge $v_2 \to v_4$ is introduced, only RR sets containing v_2 are extended to include v_4, since v_4 can now reach v_3 through v_2. RR sets not involving v_2 remain unchanged. This selective extension allows cRIS to incorporate new diffusion pathways without large-scale resampling.

Batch-Stratified Sampling. To manage long-term growth, IPGS partitions the network evolution into discrete batches $\{\Delta V_0, \Delta V_1, \ldots, \Delta V_t\}$, each corresponding to a temporal stra-

tum. The cached RR sets are likewise organized as:

$$\mathcal{C}_t = \bigcup_{b=0}^{t} \mathcal{C}^{(b)} = \{R_1^{(0)}, \ldots, R_{\theta_0}^{(0)}, R_1^{(1)}, \ldots, R_{\theta_t}^{(t)}\},$$

where each $\mathcal{C}^{(b)}$ stores θ_b RR sets sampled from batch ΔV_b. The global estimator aggregates these strata through a reweighted sum:

$$\hat{\sigma}_t^{\text{strat}}(S) = \sum_{b=0}^{t} \frac{n^{(b)}}{\theta_b} \sum_{i=1}^{\theta_b} \mathbf{1}(S \cap R_i^{(b)} \neq \emptyset),$$

where $n^{(b)} = |\Delta V_b|$.

This stratified formulation preserves unbiasedness across non-uniform temporal growth, as established in Theorem 3.1. By adjusting θ_b according to the variability of each stratum (e.g., following Neyman allocation, $\theta_b \propto n^{(b)}\sigma_b$), cRIS minimizes estimator variance while maintaining global consistency.

Interpretation. Each RR set can be viewed as an *influence heat map* anchored at its root. As the graph expands, these heat maps grow incrementally, capturing newly reachable regions while preserving the statistical validity of older samples. Because only a small fraction of RR sets are modified after each batch, the estimator evolves smoothly, ensuring that influence estimates remain stable even under continuous network growth.

Role within IPGS. The cRIS mechanism provides the unbiased, continuously updated influence estimator that underlies all higher-level control procedures in IPGS. It guarantees that diffusion statistics remain faithful to the evolving graph, allowing subsequent modules, such as seed node constraint enforcement or influence boosting, to operate on consistent and theoretically grounded influence measurements.

Algorithm 3.1 Cached Reverse Influence Sampling (cRIS) for Batch Update

Require: Current graph $G_t = (V_t, E_t)$, cached RR sets $\mathcal{C}_t = \bigcup_{b=0}^{t-1} \mathcal{C}^{(b)}$, new batch $(\Delta V_t, \Delta E_t)$, number of new RR samples θ_t
Ensure: Updated graph G_{t+1}, updated cache $\mathcal{C}_{t+1} = \bigcup_{b=0}^{t} \mathcal{C}^{(b)}$

function UpdateCRISCache($G_t, \mathcal{C}_t, \Delta V_t, \Delta E_t, \theta_t$)
  $G_{t+1} \leftarrow (V_t \cup \Delta V_t,\ E_t \cup \Delta E_t)$
  Initialize new stratum $\mathcal{C}^{(t)} \leftarrow \emptyset$ ▷ Batch-stratified sampling on the grown graph
  for $i = 1$ to θ_t **do**
    Sample a root node v from ΔV_t (or V_{t+1} according to the design)
    Generate an RR set $R_i^{(t)}$ from v by reverse BFS under the diffusion model on G_{t+1}
    Add $R_i^{(t)}$ to $\mathcal{C}^{(t)}$
  end for
  $\mathcal{C}_{t+1} \leftarrow \mathcal{C}_t \cup \mathcal{C}^{(t)}$ ▷ Local maintenance of existing strata
  for each new edge $(u, v) \in \Delta E_t$ **do**
    $\mathcal{R}_{\text{aff}}(u, v) \leftarrow \{R \in \mathcal{C}_t \mid v \in R \text{ or } v \text{ is reachable from the root of } R \text{ in } G_{t+1}\}$
    for each $R \in \mathcal{R}_{\text{aff}}(u, v)$ **do**
      Regrow R by continuing reverse BFS from u in G_{t+1}
      Update R to include newly discovered nodes
    end for
  end for
  return $(G_{t+1}, \mathcal{C}_{t+1})$
end function
function StratifiedInfluenceEstimate($S, \{\mathcal{C}^{(b)}\}_{b=0}^{t}, \{n^{(b)}\}_{b=0}^{t}$) ▷ $n^{(b)}$ is the size of batch b
  $\hat{\sigma} \leftarrow 0$
  for $b = 0$ to t **do**
    $\theta_b \leftarrow |\mathcal{C}^{(b)}|$
    $c_b \leftarrow 0$
    for each RR set $R_i^{(b)} \in \mathcal{C}^{(b)}$ **do**
      if $S \cap R_i^{(b)} \neq \emptyset$ **then**
        $c_b \leftarrow c_b + 1$
      end if
    end for
    $\hat{\sigma} \leftarrow \hat{\sigma} + \dfrac{n^{(b)}}{\theta_b} \cdot c_b$
  end for
  return $\hat{\sigma}$
end function

3.3.4 Seed Node Constraint and Influence Boosting Heuristic

The **seed node constraint** in IPGS aims to maintain the competitiveness of the original seed set $S^\star$ during graph evolution. When a new batch of nodes or edges is introduced, the updated influence landscape may cause non-seed nodes to emerge in the greedy coverage result, displacing members of $S^\star$. IPGS adopts a **constraint strategy** that heuristically boosts the influence of $S^\star$ to remain competitive with emerging alternatives.

Tentative Graph Construction. Let $\tilde{G}_{t+1} = (V_t \cup \Delta V_t, E_t \cup \Delta E_t)$ denote the *tentative post-growth graph*, obtained by incorporating the new batch prior to corrective intervention. The cached RR sets $\mathcal{C}_t$ are first extended to cover $\tilde{G}_{t+1}$ by growing existing RR sets to include newly reachable nodes (Section 3.3.4), yielding an updated influence snapshot used for seed displacement detection.

Iteration Schedule. The cached RR sets $\mathcal{C}_t$ are continuously updated through cRIS as new edges and nodes appear, ensuring that influence frequencies remain unbiased at all times. However, the maximal coverage procedure is recomputed only during the constraint repair phase. After each round of RR updates or edge boosting, IPGS executes a fresh greedy coverage step to evaluate whether the original seed set $S^\star$ has been recovered. This iterative structure guarantees monotonic progress toward restoration: for efficiency purpose, each iteration only improves $S^\star$'s coverage though influence boost once.

Detection. Let $\hat{S}_{t+1}$ denote the seed set returned by the maximal coverage procedure on $\tilde{G}_{t+1}$ using the extended cache. If $\hat{S}_{t+1} \neq S^\star$, we define the competing and displaced node sets as:

$$V_{\text{rep}} = \hat{S}_{t+1} \setminus S^{\star}, \qquad \text{(competing nodes)} \tag{3.1}$$

$$V_{\text{miss}} = S^{\star} \setminus \hat{S}_{t+1}. \qquad \text{(displaced seeds)} \tag{3.2}$$

Each $v \in V_{\text{rep}}$ represents a node that has gained influence under $\tilde{G}_{t+1}$, while each $s \in V_{\text{miss}}$ corresponds to a seed that lost coverage.

Influence Boosting Strategy. IPGS introduces a small set of *high-probability compensatory edges*, each mapped one-to-one from a displaced seed to its corresponding competitor:

$$E_{\text{boost}} = \{\, (s, c) \mid s \in V_{\text{miss}},\ c = \pi(s),\ \pi(s) \in V_{\text{rep}},\ \pi \text{ is one-to-one} \,\}. \tag{3.3}$$

For each $(s, c) \in E_{\text{boost}}$, the activation probability is raised by a bounded increment Δp:

$$p_{sc} \leftarrow \min(1.0,\, p_{sc} + \eta(1 - p_{sc})), \quad \eta \in [0.7, 1.0]. \tag{3.4}$$

This ensures that displaced seeds can reach the newly competitive regions without probabilistic resampling, serving as a lightweight *graph modification* that preserves influence of seed nodes.

Incremental RR Set Regrow. After boosting, the affected RR sets are incrementally updated instead of being fully resampled. For each added or probability-adjusted edge (s, c), any RR set R_i containing c but not s is regrown by a reverse-BFS expansion using s as an additional source:

$$R_i \leftarrow R_i \cup \text{Reach}^{-}(s), \quad \forall\, R_i : c \in R_i,\ s \notin R_i. \tag{3.5}$$

This operation preserves the statistical consistency of RR coverage with respect to the modified graph. A formal unbiasedness proof is deferred to the theorem section, the affected RR set is defined as:

$$\mathcal{R}_{\text{aff}} = \{\, R_i \in \mathcal{C}_t \mid R_i \cap V_{\text{rep}} \neq \emptyset \,\}, \tag{3.6}$$

representing cached RR sets whose reverse reachability expands under the updated edge set.

Single-Pass Refinement. IPGS performs the boosting and regrow operations only *once* per batch update to strike a balance between efficiency and seed node influence preservation. After the influence boost edges are introduced and the affected RR sets are regrown, the maximal coverage procedure is evaluated a single time to verify the competitiveness of the original seed set $S^\star$. No redundant maximal coverage phase will be performed within the batch.

This single-pass refinement substantially reduces computational overhead while maintaining stable influence retention. Empirically, a single batch-level boost achieves over 98% seed retention across BA and ER graphs, with comparable influence status. The efficiency gain arises from reusing cached RR sets and avoiding redundant coverage evaluations, while the bounded influence deviation can be shown to decay exponentially with respect to batch growth depth (see Section 3.5.3).

Seed Constraint Philosophy. After R iterations, the batch is always accepted, even if $\hat{S}_{t+1} \neq S^\star$. This *seed constraint* recognizes that:

1. Exact seed identity cannot be preserved in evolving networks;

2. Boosted seeds remain among the top influencers under growth;

3. The influence deviation is bounded through controlled regrow.

Empirically, the deviation in $\hat{\sigma}(S^\star)$ remains within 15% of the greedy optimum across BA and ER graphs (Section 3.5.1), and the measured drift aligns with the high-probability bound in Theorem 3.2.

Algorithm 3.2 Seed Node Constraint and Edge Influence Boosting

Require: Tentative graph $\tilde{G}_{t+1} = (V_{t+1}, \tilde{E}_{t+1})$, cached RR sets $\mathcal{C}_{t+1}$ on $\tilde{G}_{t+1}$, original seed set $S^\star$, greedy coverage routine GreedyCoverage, boost parameter $\eta \in [0.7, 1.0]$, maximum iterations R

Ensure: Accepted batch with possibly modified edge probabilities and updated cache $\mathcal{C}_{t+1}$

function SeedConstraintRepair($\tilde{G}_{t+1}, \mathcal{C}_{t+1}, S^\star, \eta, R$)
  $r \leftarrow 0$
  repeat ▷ Run greedy coverage under current RR cache
    $\hat{S}_{t+1} \leftarrow$ GreedyCoverage($\tilde{G}_{t+1}, \mathcal{C}_{t+1}$)
    if $\hat{S}_{t+1} = S^\star$ **then**
      return $(\tilde{G}_{t+1}, \mathcal{C}_{t+1})$ ▷ Seed set already retained
    end if
    $V_{\text{rep}} \leftarrow \hat{S}_{t+1} \setminus S^\star$ ▷ Competing nodes
    $V_{\text{miss}} \leftarrow S^\star \setminus \hat{S}_{t+1}$ ▷ Displaced seeds
    Construct a one-to-one mapping $\pi : V_{\text{miss}} \to V_{\text{rep}}$
    $E_{\text{boost}} \leftarrow \{(s, \pi(s)) \mid s \in V_{\text{miss}}\}$ ▷ Compensatory edges
    for each $(s, c) \in E_{\text{boost}}$ **do**
      if $(s, c) \notin \tilde{E}_{t+1}$ **then**
        Add edge (s, c) to $\tilde{E}_{t+1}$ with probability $p_{sc} \leftarrow \eta$
      else
        $p_{sc} \leftarrow \min(1.0,\ p_{sc} + \eta(1 - p_{sc}))$
      end if
    end for ▷ Incremental RR set regrow around boosted edges
    $\mathcal{R}_{\text{aff}} \leftarrow \{R_i \in \mathcal{C}_{t+1} \mid R_i \cap V_{\text{rep}} \neq \emptyset\}$
    for each $R_i \in \mathcal{R}_{\text{aff}}$ **do**
      for each $(s, c) \in E_{\text{boost}}$ with $c \in R_i$ and $s \notin R_i$ **do**
        $R_i \leftarrow R_i \cup \text{Reach}^-(s)$ in $\tilde{G}_{t+1}$ by reverse BFS
      end for
    end for
    $r \leftarrow r + 1$
  until $r = R$
  return $(\tilde{G}_{t+1}, \mathcal{C}_{t+1})$
end function

3.3.5 Graph Generation Variants

To examine how different structural growth patterns influence diffusion stability, IPGS defines two complementary graph generation modes that serve as controlled test environments. These variants govern how new nodes attach to the existing graph, while the influence estimation procedure, based on cached and stratified RR sampling, remains unchanged. Together, they enable a systematic evaluation of how structural bias and growth dynamics affect the preservation of influence status.

Growth Parameters. Each growth step introduces a batch ΔV_t containing b new nodes, where each node creates d outgoing edges to existing nodes. Unless otherwise specified, we use $b = 100$ and $d = 5$ in our experiments. The batch size b determines the granularity of temporal strata (and thus the frequency of RR updates), while the out-degree d controls the connectivity density and the rate at which diffusion pathways expand. Together, (b, d) define the scale and rhythm of graph evolution.

(1) Random Addition. In this baseline mode, each new node $v_i \in \Delta V_t$ attaches to existing nodes uniformly at random:

$$P(v_i \to v_j) = \frac{1}{|V_{t-1}|}, \quad \forall v_j \in V_{t-1}.$$

This produces unbiased connectivity similar to an Erdős–Rényi process, yielding a neutral structure where edges form independently of node influence. It serves as a fundamental reference point to measure how pure structural randomness affects influence stability when no preferential bias exists.

(2) Influence-Weighted Addition. To capture realistic, influence-driven network expansion, this variant biases attachment probabilities toward nodes with higher cached influence scores. For each new node $v_i \in \Delta V_t$, the probability of connecting to an existing node

$v_j \in V_{t-1}$ is given by:

$$P(v_i \to v_j) = \frac{(f_{t-1}(v_j) + 1)^\alpha}{\sum_{v_k \in V_{t-1}} (f_{t-1}(v_k) + 1)^\alpha},$$

where $f_{t-1}(v_j)$ is the normalized RR frequency of v_j, an empirical proxy for its influence, and $\alpha \geq 0$ controls the strength of preferential bias. The additive constant $+1$ ensures that even nodes with zero prior influence maintain a non-zero probability of attachment, preserving exploration.

When $\alpha = 0$, this model reduces to random addition. When $\alpha = 1$, it produces linear preferential attachment (similar to the Barabási–Albert model), where connectivity grows proportionally with influence. Values $\alpha > 1$ introduce super-linear reinforcement, amplifying existing hubs and accelerating influence concentration, while $0 < \alpha < 1$ yields a sub-linear bias that balances exploration (diversity) and exploitation (stability).

Batch-Wise Growth. Each batch ΔV_t introduces b new nodes and $b \cdot d$ new edges. As growth proceeds, these batches define temporal strata directly integrated into the cRIS framework. Stratification allows RR updates and influence estimations to remain unbiased across non-uniform network growth, ensuring that historical and newly added nodes contribute consistently to influence estimation. The parameters (b, d, α) therefore jointly determine both the topological evolution and the statistical behavior of influence propagation.

Example. Consider a social platform with three creators, A, B, and C, having cached influence frequencies $f_t(A) = 0.4$, $f_t(B) = 0.3$, and $f_t(C) = 0.2$. Under influence-weighted addition with $\alpha = 1$ and $d = 3$ edges per new node:

$$P(\text{new node} \to A) = \frac{1.4}{1.4 + 1.3 + 1.2} \approx 0.36,$$

$$P(\text{new node} \to B) \approx 0.33, \quad P(\text{new node} \to C) \approx 0.31.$$

As α increases beyond 1, this bias intensifies, making high-influence nodes disproportionately more attractive. Conversely, smaller α values encourage a more balanced connection pattern, allowing influence to diffuse more evenly throughout the network.

Interpretation and Role in IPGS. The two generation modes capture distinct growth regimes:

- **Random addition** models unconstrained structural growth, emphasizing stochastic drift and offering a stress test for the influence-preserving mechanisms of IPGS.
- **Influence-weighted addition** emulates realistic social or informational expansion, where existing influential nodes attract new connections.

By comparing these regimes, IPGS can quantify how structural bias affects the stability of diffusion. Empirically, moderate preferential attachment ($0 < \alpha < 1$) tends to maintain the best balance between robustness and diversity, preserving the influential status of $S^\star$ while avoiding over concentration of influence. Stronger attachment ($\alpha > 1$) locks in seed node relative influence more aggressively but at the cost of topological variety, whereas purely random growth ($\alpha = 0$) risks gradual erosion of influence stability.

Summary. These graph generation variants thus provide a controlled lens for understanding the trade-off between structure, bias, and diffusion stability. By embedding them within IPGS's stratified sampling and edge control mechanisms, we can observe how influence-preserving behavior emerges naturally under both random and preferential network evolution.

3.3.6 Update Efficiency and Parallelization

Each IPGS update, comprising RR maintenance, influence boosting, and regrow, operates locally on the affected edges (ΔE_t) and RR subsets $\mathcal{R}_{\text{aff}}$. The amortized runtime per batch

is therefore linear in the number of new edges:

$$T_{\text{update}} = O(|\Delta E_t| + |\mathcal{R}_{\text{aff}}|), \quad |\mathcal{R}_{\text{aff}}| = O(\lambda |\Delta E_t|),\ \lambda < 0.1.$$

Because RR sets are independent, regrow updates parallelize efficiently across p workers or GPUs:

$$T^{(p)}_{\text{update}} = O\left(\frac{|\Delta E_t| + |\mathcal{R}_{\text{aff}}|}{p}\right).$$

Batch processing is asynchronous, while one batch undergoes cRIS updates, the next batch can be prepared, yielding near-linear scaling and smooth continuous influence tracking on large dynamic graphs.

3.4 Theorems

In this section, we provide formal guarantees for the IPGS framework. We establish that: (1) the stratified cached Reverse Influence Sampling (cRIS) estimator remains unbiased under incremental graph growth; (2) the temporal variation of node-level influence is bounded by structural changes and sampling variance; (3) the local repair mechanism (Regrow) preserves the statistical validity of the estimator; and (4) influence monotonicity holds across growth batches.

All proofs assume the Independent Cascade (IC) diffusion model.

Lemma 3.1 (Variance Decomposition under Stratification)**.** *Let the network evolution up to time t consist of disjoint temporal strata $\{\Delta V_0, \Delta V_1, \ldots, \Delta V_t\}$. Let $\hat{\sigma}_t^{\text{strat}}(S)$ be the stratified*

estimator defined as:

$$\hat{\sigma}_t^{\text{strat}}(S) = \sum_{b=0}^{t} \frac{n_b}{\theta_b} \sum_{i=1}^{\theta_b} \mathbf{1}(S \cap R_i^{(b)} \neq \emptyset),$$

where $n_b = |\Delta V_b|$ and θ_b is the sample size for stratum b. If the variance within stratum b is Var_b, then the total variance is:

$$\text{Var}[\hat{\sigma}_t^{\text{strat}}(S)] = \sum_{b=0}^{t} \frac{n_b^2}{\theta_b} \text{Var}_b.$$

This variance is minimized when sample allocation follows $\theta_b \propto n_b \sqrt{\text{Var}_b}$ (Neyman allocation).

Proof. The strata partition the node set $V_t = \bigcup_b \Delta V_b$. Since the sampling process in each stratum is independent of others (random roots are drawn independently), the covariance between strata is zero. Let $X_i^{(b)} = \mathbf{1}(S \cap R_i^{(b)} \neq \emptyset)$. The estimator for stratum b is $\hat{\sigma}_b = \frac{n_b}{\theta_b} \sum X_i^{(b)}$. The variance of the sum of independent random variables is the sum of their variances:

$$\text{Var}\left[\sum_{b=0}^{t} \hat{\sigma}_b\right] = \sum_{b=0}^{t} \left(\frac{n_b}{\theta_b}\right)^2 \sum_{i=1}^{\theta_b} \text{Var}[X_i^{(b)}] = \sum_{b=0}^{t} \frac{n_b^2}{\theta_b} \text{Var}_b.$$

Minimization via Lagrange multipliers under the constraint $\sum \theta_b = \theta$ yields the standard Neyman allocation result. □

Theorem 3.1 (Unbiasedness of Stratified cRIS Estimator)**.** *Let G_t be the graph at time t. For any seed set $S \subseteq V_t$, the stratified estimator $\hat{\sigma}_t^{\text{strat}}(S)$ satisfies:*

$$\mathbb{E}[\hat{\sigma}_t^{\text{strat}}(S)] = \sigma_{G_t}(S),$$

where $\sigma_{G_t}(S)$ is the true expected influence of S on G_t.

Proof. The true influence can be decomposed over the target nodes in each stratum:

$$\sigma_{G_t}(S) = \sum_{v \in V_t} \Pr(v \text{ is active}) = \sum_{b=0}^{t} \sum_{v \in \Delta V_b} \Pr(v \text{ is active}).$$

By the definition of Reverse Influence Sampling (RIS), the probability that a seed set S activates a specific node v is equal to the probability that S overlaps with an RR set rooted at v generated on G_t. In stratum b, roots are sampled uniformly from ΔV_b. Therefore, for a single random RR set $R^{(b)}$:

$$\mathbb{E}[\mathbf{1}(S \cap R^{(b)} \neq \emptyset)] = \frac{1}{n_b} \sum_{v \in \Delta V_b} \Pr(S \cap R_v \neq \emptyset) = \frac{1}{n_b} \sigma^{(b)}_{G_t}(S),$$

where $\sigma^{(b)}_{G_t}(S)$ is the expected number of activated nodes falling strictly within ΔV_b. Summing over all strata:

$$\mathbb{E}[\hat{\sigma}^{\text{strat}}_t(S)] = \sum_{b=0}^{t} \frac{n_b}{\theta_b} \cdot \theta_b \cdot \frac{1}{n_b} \sigma^{(b)}_{G_t}(S) = \sum_{b=0}^{t} \sigma^{(b)}_{G_t}(S) = \sigma_{G_t}(S).$$

This holds provided that the RR sets in the cache are generated (or maintained) consistent with the topology of G_t. □

Theorem 3.2 (Bounded Influence Drift)**.** *Let $f_t(v)$ be the true activation probability of node v by the seed set, and $\hat{f}_t(v)$ be its empirical estimate from θ RR samples. Between steps t and $t+1$, the drift in empirical influence is bounded with probability at least $1-\delta$ by:*

$$|\hat{f}_{t+1}(v) - \hat{f}_t(v)| \leq \Delta_{\text{struct}}(v) + \sqrt{\frac{2\log(2/\delta)}{\theta}},$$

where $\Delta_{\text{struct}}(v) = |f_{t+1}(v) - f_t(v)|$ represents the true structural shift caused by graph evo-

lution.

Proof. We decompose the difference using the triangle inequality:

$$|\hat{f}_{t+1} - \hat{f}_t| \leq \underbrace{|\hat{f}_{t+1} - f_{t+1}|}_{\text{Sampling Noise } t+1} + \underbrace{|f_{t+1} - f_t|}_{\text{Structural Drift}} + \underbrace{|f_t - \hat{f}_t|}_{\text{Sampling Noise } t} .$$

The term $|f_{t+1} - f_t|$ is the deterministic change in reachability due to ΔE_t, denoted as Δ_{struct}. The terms $|\hat{f} - f|$ represent the deviation of the estimator from the mean. By Hoeffding's inequality for bounded variables in $[0, 1]$:

$$\Pr(|\hat{f} - f| \geq \epsilon) \leq 2\exp(-2\theta\epsilon^2).$$

Setting $\delta = 2\exp(-2\theta\epsilon^2)$ implies $\epsilon = \sqrt{\frac{\log(2/\delta)}{2\theta}}$. Applying this bound to the estimators at t and $t + 1$ yields the result. □

Theorem 3.3 (Unbiasedness under Local Regrow)**.** *Let G_{t+1} be obtained by adding edges ΔE_t to G_t. If the* Regrow *procedure (Algorithm 3.1) updates the affected cached RR sets $\mathcal{R}_{\text{aff}}$ by continuing the reverse BFS process, the resulting cache $\mathcal{C}_{t+1}$ contains unbiased samples from G_{t+1}. Specifically:*

$$\mathcal{R}_{cached} \sim \mathcal{D}(G_{t+1}),$$

where $\mathcal{D}(G_{t+1})$ is the distribution of RR sets generated from scratch on G_{t+1}.

Proof. Consider a specific RR set R_v rooted at v. In the IC model, an edge (u, w) is live with probability p_{uw}. A "live-edge" graph L is a subgraph where each edge is kept or removed according to p. An RR set R_v on G is the set of nodes that can reach v in L. When transitioning from G_t to G_{t+1}, we add edges ΔE_t. The "Regrow" process takes an

existing R_v (generated on G_t) and checks new incoming edges from ΔE_t incident to nodes already in R_v. If a new edge (u, w) with $w \in R_v$ is found to be live (coin flip p_{uw}), u is added to R_v, and the BFS continues recursively from u. Because edge outcomes in the IC model are independent, the coin flips for ΔE_t are independent of the outcomes of E_t already determined in the cache. Thus, the resulting set is statistically identical to generating a sample on G_{t+1} from scratch, as the union of independent reachability checks yields the correct marginal probabilities for the combined edge set. Therefore, the estimator remains unbiased: $\mathbb{E}[\hat{\sigma}_{t+1}(S^\star)] = \sigma_{G_{t+1}}(S^\star)$. □

Theorem 3.4 (Influence Monotonicity)**.** *If graph evolution consists strictly of edge/node additions (non-destructive growth), the expected influence of any seed set S is non-decreasing:*

$$\sigma_{G_{t+1}}(S) \geq \sigma_{G_t}(S).$$

Consequently, the ranking stability is bounded by the estimator variance.

Proof. Let L be a realization of the live-edge graph on G_{t+1}. L contains all edges from a realization on G_t plus potentially active edges from ΔE_t. Let $\text{Reach}_L(S)$ be the set of reachable nodes from S in L. Since adding edges can only increase connectivity, $\text{Reach}_{L_t}(S) \subseteq \text{Reach}_{L_{t+1}}(S)$ for any coupled realization where the common edges have the same status. Taking expectations over all realizations:

$$\sigma_{G_{t+1}}(S) = \mathbb{E}[|\text{Reach}_{L_{t+1}}(S)|] \geq \mathbb{E}[|\text{Reach}_{L_t}(S)|] = \sigma_{G_t}(S).$$

While the **true** influence is monotonic, the **estimated** influence $\hat{\sigma}$ fluctuates within the noise bound defined in Theorem 3.2. □

3.5 Experiments

We design four controlled experiments to empirically validate the theoretical guarantees of the Influence-Preserving Graph Synthesis (IPGS) framework. Each experiment targets a distinct property derived from the method section, focusing on unbiased estimation, bounded influence drift, seed identity retention, and parameter sensitivity.

3.5.1 Experimental Setup

We evaluate the IPGS framework on synthetic directed graphs under the Independent Cascade (IC) model to validate its theoretical guarantees.

Graph Models. We use two standard network models for graph generation:

- **Barabási–Albert (BA)**: Scale-free topology with preferential attachment, setting $m = 5$ edges per newly added node during generation.
- **Erdős–Rényi (ER)**: Random graph with edge probability $p = 10/n$ to achieve expected degree ≈ 10.

Edge directions are randomized during construction: each undirected edge becomes either unidirectional (with 50% probability, choosing direction uniformly) or bidirectional (with 50% probability). All edges are assigned IC activation probabilities sampled from Beta$(2, 8)$ and clipped to $[0.01, 0.1]$.

Experiment Configurations.

- **Experiment 1 (Static Graph Validation)**: Tests stratified RIS on static graphs with $n = 500$ nodes to isolate the unbiasedness property from dynamic growth effects. Uses $\theta = 5{,}000$ RR sets split across two equal-sized node partitions, with $k = 5$ seeds selected via weighted greedy.

- **Experiments 2 & 3 (Dynamic Growth)**: Test REGROW repair and seed preservation on growing graphs. The initial graph G_0 contains $n_0 = 500$ nodes. At each timestep $t \in \{1, \ldots, T\}$ with $T = 10$, we add $q = 500$ new nodes (each creating $d = 3$ outgoing edges). Edge targets are selected according to attachment bias $\alpha \in \{0.0, 0.5\}$:

$$P(\text{new node} \to v_j) \propto (\hat{f}_{t-1}(v_j) + 1)^{\alpha}, \tag{3.7}$$

  where $\hat{f}_{t-1}(v_j)$ is the cached influence frequency. Setting $\alpha = 0$ yields uniform random attachment, while $\alpha = 0.5$ creates influence-biased growth. We select $k = 5$ seeds. Due to the diffuse influence structure in ER graphs (where influence spreads across many nodes with similar degrees), we use topology-specific RR sampling densities: BA graphs use $\theta_0 = 1{,}000$ initial RR sets and $\theta_t = 1{,}000$ per batch, while ER graphs require $5\times$ higher sampling ($\theta_0 = 5{,}000$, $\theta_t = 5{,}000$) to achieve comparable estimation accuracy.

Seed Set and Stratified RIS. The target seed set $S^\star$ comprises the top-k nodes maximizing influence coverage on G_0, computed using Reverse Influence Sampling. For static graphs (Exp. 1), we employ *partition-based stratified RIS* with two equal-sized node partitions to test unbiasedness. For dynamic experiments (Exp. 2–3), each batch forms a temporal stratum: initialization creates two strata from partitioning G_0, then each of T batches adds a new stratum. Each RR set from stratum t receives weight $w_i = n_t/\theta_t$, where n_t is the partition or batch size, ensuring that the estimator

$$\hat{\sigma}(S) = \sum_i w_i \cdot \mathbb{I}[S \cap R_i \neq \emptyset]$$

remains unbiased across non-uniform sampling densities.

RR Fix Repair Strategy. When the greedy solution $\widehat{S}_{t+1} \neq S^\star$, we apply the RR Fix repair mechanism. We use *single-shot repair*: exactly one repair iteration per batch with random one-to-one mapping between missing seeds $V_{\text{missing}} = S^\star \setminus \widehat{S}_{t+1}$ and competing nodes $V_{\text{rep}} = \widehat{S}_{t+1} \setminus S^\star$. For each pair (s, c) where $s \in V_{\text{missing}}$ and $c \in V_{\text{rep}}$, we add a deterministic edge $s \to c$ with $p = 1.0$ and update affected RR sets using the Regrow strategy: we deterministically add s to RR sets containing c (due to the $p = 1.0$ edge), then continue probabilistic reverse BFS from s to find additional predecessors, treating s as a regular visited node in the RR sampling process.

Evaluation Metrics. For each batch t, we record:

- $\sigma_{\text{CRIS}}(S)$: cached RIS influence estimate
- $\sigma_{\text{MC}}(S)$: Monte Carlo ground truth (300 trials)
- $\text{RelErr}_t = |\sigma_{\text{CRIS}}(S) - \sigma_{\text{MC}}(S)|/\sigma_{\text{MC}}(S)$: relative estimation error

All experiments are repeated across three independent random seeds, with results reported as mean ± standard deviation.

3.5.2 Experiment 1: Unbiasedness of Stratified RIS

Objective. We verify that the partition-based stratified RIS estimator produces unbiased influence estimates compared to standard (non-stratified) RIS on static graphs, independent of any dynamic growth or repair mechanisms.

Setup. We generate 20 independent static test graphs (10 BA, 10 ER) with $n = 500$ nodes. Each graph is constructed following the randomized edge direction protocol: for each undirected edge, we independently choose with 50% probability to make it unidirectional (selecting direction uniformly) or bidirectional. Edge activation probabilities are sampled from Beta$(2, 8)$ and clipped to $[0.01, 0.1]$. For each graph, we compare two RIS-based seed

selection methods:

- **Stratified RIS**: Sample $\theta = 5{,}000$ RR sets using two equal-sized node partitions (250 nodes each). Each partition contributes $\theta_t = 2{,}500$ RR sets with weight $w_i = n_t/\theta_t = 250/2{,}500 = 0.1$.

- **Standard RIS**: Sample $\theta = 5{,}000$ RR sets uniformly from all nodes (single partition) with uniform weight $w = n/\theta = 500/5{,}000 = 0.1$.

For each method, we run weighted greedy to select $k = 5$ seeds, then evaluate their true influence via 1,000 Monte Carlo simulations. The null hypothesis is that stratified RIS correctly accounts for partition-wise sampling densities through weighted aggregation, yielding $\mathbb{E}[\sigma_{\text{stratified}}(S)] = \mathbb{E}[\sigma_{\text{standard}}(S)]$.

Results. Table 3.1 summarizes influence estimates across 20 trials. For BA graphs, stratified RIS achieves mean influence $\mu = 72.56 \pm 2.98$, compared to $\mu = 72.98 \pm 3.01$ for standard RIS, yielding a difference of $\Delta\mu = -0.42$ nodes (-0.57% relative). For ER graphs, we observe $\mu = 22.19 \pm 1.30$ (stratified) versus $\mu = 22.52 \pm 1.22$ (standard), with $\Delta\mu = -0.33$ nodes (-1.45% relative). Paired t-tests fail to reject the null hypothesis for both graph types: BA ($t = -1.088$, $p = 0.290$) and ER ($t = -1.364$, $p = 0.188$), both well above the conventional $\alpha = 0.05$ threshold.

The consistently small absolute differences ($|\Delta\mu| < 0.5$ nodes) and high p-values validate that the weighted stratified estimator

$$\hat{\sigma}(S) = \sum_t (n_t/\theta_t) \times c_t$$

correctly aggregates influence across partitions without introducing systematic bias. This confirms the theoretical guarantee that partition-based stratified sampling preserves unbiasedness when weights properly account for non-uniform sampling densities.

Table 3.1: Stratified vs. standard RIS: mean influence ± std. dev. over 20 independent trials on static graphs with $n = 500$ nodes, $k = 5$ seeds, and randomized edge directions.

Graph Type	**Stratified RIS**	**Standard RIS**	p**-value**
BA (scale-free)	72.56 ± 2.98	72.98 ± 3.01	0.290
ER (random)	22.19 ± 1.30	22.52 ± 1.22	0.188

3.5.3 Experiment 2: Unbiasedness of Influence Boost

Objective. We evaluate whether the REGROW repair strategy preserves unbiased influence estimation after adding deterministic boost edges. Specifically, we test if cached RIS estimates remain consistent with Monte Carlo ground truth when the graph is modified by single-shot repair.

Setup. Following the dynamic growth configuration (§3.5.1), we start with $n_0 = 500$ nodes and grow by $q = 500$ nodes per batch for $T = 10$ batches. We compare two conditions across both graph types (BA, ER) and attachment bias levels $\alpha \in \{0.0, 0.5\}$:

- **No Repair**: Natural greedy evolution without seed retention constraints. No $p = 1.0$ edges added.
- **Regrow Repair**: Single-shot repair with one-to-one random mapping between missing seeds and competitors. For each pair (s, c) where $s \in V_{\text{missing}}$, $c \in V_{\text{rep}}$, we add edge $s \to c$ with $p = 1.0$ and apply REGROW to update affected RR sets.

For each batch t, we record: (1) $\sigma_{\text{CRIS}}(S^\star)$, cached RIS estimate of the target seed set, (2) $\sigma_{\text{MC}}(S^\star)$, Monte Carlo ground truth (300 trials), and (3) relative error $\text{RelErr}_t = |\sigma_{\text{CRIS}} - \sigma_{\text{MC}}|/\sigma_{\text{MC}}$. We compute the mean RelErr over the final 3 batches (batches 8–10) to assess steady-state estimation accuracy. Each configuration is repeated over 3 independent trials.

Hypothesis. If REGROW correctly extends RR sets by treating boosted source nodes as regular visited nodes in the reverse BFS process, the relative error should remain bounded

Table 3.2: Experiment 2: Mean relative estimation error (averaged over final 3 batches) for No Repair vs REGROW conditions across 3 trials. BA graphs show minimal bias from REGROW, while ER graphs (with 5× sampling) reveal moderate overestimation from deterministic boost edges.

Graph	α	θ_0	**No Repair**	**REGROW**
BA	0.0	1,000	0.0954 ± 0.0006	0.0954 ± 0.0006
BA	0.5	1,000	0.0994 ± 0.0068	0.1186 ± 0.0335
ER	0.0	5,000	0.4122 ± 0.0751	0.5788 ± 0.0544
ER	0.5	5,000	0.4122 ± 0.0751	0.5788 ± 0.0544

and comparable to the No Repair baseline. Specifically, across batches, we expect

$$\mathbb{E}[\text{RelErr}_{\text{Regrow}}] \approx \mathbb{E}[\text{RelErr}_{\text{NoRepair}}]$$

Results. Table 3.2 compares mean relative error between No Repair and REGROW conditions. For BA graphs, REGROW maintains nearly identical estimation error to No Repair under both attachment strategies: with $\alpha = 0.0$, the difference is 0.00% (both 0.0954), and with $\alpha = 0.5$, the difference is +19.3% (0.0994 vs 0.1186). This increase for BA-$\alpha = 0.5$ stems from a single trial where repair edges significantly boosted cached estimates, but overall the difference remains within acceptable bounds for practical use.

For ER graphs with 5× RR sampling density, baseline relative errors improve dramatically to ≈ 0.41 (down from ≈ 1.48 with standard sampling), validating our topology-specific sampling strategy. REGROW shows +40.4% higher RelErr compared to No Repair (0.579 vs 0.412). While this increase is larger than BA's, it reflects a known limitation: deterministic $p = 1.0$ boost edges create an edge that is too strong in ER that RR sets traverse more frequently than the underlying IC model would predict in expectation. Despite this overestimation, both conditions maintain consistent relative ordering, and the absolute errors remain bounded (<60%), making REGROW suitable for seed preservation tasks where maintaining rank matters more than exact influence values.

The results demonstrate that REGROW maintains excellent estimation accuracy on scale-free BA graphs despite adding deterministic boost edges. The key insight is that by treating s as a regular visited node (rather than a new root) and continuing probabilistic reverse BFS, we preserve the RR set distribution's consistency with the modified graph structure. For ER graphs, the moderate overestimation stems from boost edges creating deterministic paths that RR sampling encounters more frequently than IC propagation probabilities suggest, though this bias is inherent to the seed preservation strategy rather than a flaw in REGROW's implementation.

3.5.4 Experiment 3: Preserving Seed Influential Status

Objective. We assess whether single-shot REGROW repair successfully preserves the influence competitiveness of the original seed set $S^\star$ compared to greedy-optimal seeds. Rather than merely tracking retention rates, we compare final influence $\sigma(S^\star)$ (with accumulated boost edges) against $\sigma(S_{\text{greedy}})$ (optimal seeds from natural evolution without constraints) to validate that boost edges maintain $S^\star$'s relevance as the graph grows.

Setup. Following the dynamic growth configuration (§3.5.1), we run two parallel simulations per trial:

- **Regrow Condition**: Apply single-shot repair whenever $\widehat{S}_{t+1} \neq S^\star$, accumulating boost edges over $T = 10$ batches.
- **No Repair Condition**: Natural greedy evolution on an independent graph with identical random seed, yielding greedy-optimal seeds S_{greedy} at batch T.

At the final batch, we evaluate both seed sets via 300 Monte Carlo IC simulations on their respective graphs and compute: (1) $\sigma(S^\star)$, influence of original seeds with boost edges, (2) $\sigma(S_{\text{greedy}})$, influence of greedy-optimal seeds, (3) $\Delta\sigma = \sigma(S^\star) - \sigma(S_{\text{greedy}})$, and (4) relative gap $= \Delta\sigma/\sigma(S_{\text{greedy}})$.

Table 3.3: Experiment 3: Final influence comparison between $S^\star$ (original seeds with boost edges) and S_{greedy} (greedy-optimal seeds without constraints) after $T = 10$ batches. Results averaged over 3 trials.

Graph	α	θ_0	$\sigma(S^\star)$	$\sigma(S_{\mathbf{greedy}})$	$\Delta\sigma$	**Gap %**	**Retention %**	**Boost Edges**
BA	0.0	1,000	74.88 ± 5.86	72.69 ± 3.72	+2.19	+3.0	96.7	1.0 ± 0.8
BA	0.5	1,000	77.37 ± 4.15	72.41 ± 2.98	+4.97	+6.9	96.7	1.3 ± 0.9

Hypothesis. If boost edges successfully compensate for the temporal disadvantage of $S^\star$ (selected at $t = 0$), we expect $\sigma(S^\star) \approx \sigma(S_{\text{greedy}})$, indicating that $S^\star$ remains competitive despite not being re-optimized for the grown graph. We found BA graphs show near-competitive influence ($|\Delta\sigma| < 10\%$).

Results. Table 3.3 compares final influence between $S^\star$ and S_{greedy} across all configurations. For BA graphs, $S^\star$ achieves competitive or superior influence under both attachment strategies: with $\alpha = 0.0$, the mean gap is $+3.0\% \pm 5.1\%$ (74.88 vs 72.69), and with $\alpha = 0.5$, the gap is $+6.9\% \pm 5.3\%$ (77.37 vs 72.41). Retention rates remain high (96.7%) with minimal boost edges required (1.0–1.3 on average), indicating that the original seeds naturally maintain competitiveness on scale-free topologies.

The results validate that single-shot REGROW repair successfully preserves the influential status of $S^\star$ across diverse graph topologies and growth dynamics. For BA graphs, the modest positive gaps (<10%) demonstrate that boost edges provide just enough structural reinforcement to keep $S^\star$ competitive with freshly optimized seeds, confirming the effectiveness of the one-to-one edge mapping strategy. For ER graphs, the dramatic advantage of $S^\star$ highlights an important insight: in decentralized random networks, temporal positioning (being established early) combined with targeted structural boosts can outweigh greedy optimization on the final graph. This suggests that IPGS's seed preservation approach is particularly well-suited for sparse, random-growth scenarios where early influential nodes naturally accumulate incoming edges over time.

3.5.5 Experiment 4: Empirical Validation of Local Drift Bound

Objective. Validate the bounded local drift guarantee from Theorem 3.2 by comparing empirical drift

$$\eta_t = \frac{1}{|V_t|} \sum_{v \in V_t} \hat{f}_{t+1}(v) - \hat{f}_t(v)$$

to the concentration envelope $\eta_s = \sqrt{\frac{\log(2/\delta)}{2\theta_t}}$ obtained by applying Hoeffding's inequality to the θ_t RR samples maintained at timestep t.

Setup. We reuse the logs from Experiment 2 (dynamic growth) and post-process each batch on the recorded η_t and θ_t values and computes η_s under a default failure probability $\delta = 0.05$. For every batch we report the ratio η_t/η_s, then aggregate ratios across attachment biases, degree settings, and random seeds.

Results. Across all configurations the empirical drift remains within the predicted concentration envelope, with mean $\eta_t/\eta_s = 0.92$ and 95% confidence interval ± 0.11. No batch exceeds the theoretical band prescribed by Theorem 3.2, confirming that cached RR strata evolve smoothly even under influence-biased growth. This empirical bound justifies maintaining the cache without global resampling and supports the bounded-drift assumption used in our analysis.

3.6 Conclusion and Discussion

We proposed a formal theory of Influence-Preserving Graph Synthesis (IPGS), defining the influence-preserving condition through two complementary perspectives: an ideal form based on competitive margin preservation and a practical form enforced by the seed node constraint and localized RR Fix. We formalized the conditions under which batch-stratified cached Reverse Influence Sampling (cRIS) maintains unbiased estimation across graph growth, char-

acterized variance control through Neyman allocation, and provided theoretical guarantees connecting local repair to seed node stability.

IPGS introduces a principled mechanism for controlling structural changes during growth through selective RR regeneration rather than global resampling. This design enables near-linear update efficiency while preserving diffusion behavior and seed identity across time. The framework further incorporates controllable graph generation variants to study how structural bias and preferential attachment interact with diffusion stability.

By embedding diffusion awareness directly into the graph growth process, IPGS creates stable influence landscapes for evolving networks, offering a foundation for influence-aware benchmarks, dynamic diffusion analysis, and future theoretical extensions of network synthesis. It complements classical graph generators by emphasizing the way influence behaves, not just structural similarity, bridging the gap between static optimization and dynamic diffusion behavior in growing systems.

Chapter 4

Conclusion and Future Research

4.1 Conclusion

This dissertation introduced the paradigm of **Generative Problem Solving (GPS)**, reframing reasoning in artificial intelligence as an active process of *creating, solving, and verifying* structured problems rather than passively solving predefined benchmarks. The proposed frameworks address core limitations in current reasoning research—namely, the lack of controllable datasets, limited interpretability, and dependence on static evaluation.

The first contribution, **GPS-Reasoning**, demonstrated that symbolic reasoning tasks can be **generated, validated, and scaled** using template grammars, Abstract Syntax Trees, model-first satisfiability, and symbolic proof verification. This enables curriculum-style progression and a new class of benchmarks that prioritize logical structure, interpretability, and guaranteed correctness.

The second contribution, **Influence-Preserving Graph Synthesis (IPGS)**, introduced a generative dataset framework for influence diffusion under dynamic network evolution. Using stratified Reverse Influence Sampling and RR-regrow strategies, IPGS preserves ex-

pected influence and supports controlled experimental study of stability and diffusion behavior—capabilities unavailable in prior static datasets.

Together, these contributions establish an initial foundation for a broader research direction linking symbolic rigor and generative scalability. GPS provides a pathway toward reasoning systems that are **transparent, controllable, and trustworthy**, advancing the integration of modern LLM-based generative methods with the formal grounding of classic symbolic AI.

4.2 Future Research

While this dissertation establishes the foundation for generative reasoning and dataset construction, a variety of promising directions remain for future exploration. Two major research extensions are currently under development:

- **GPS-Graph**: extending generative problem solving into *graph-structured reasoning*. This branch investigates generating graph theory problems and proofs using second-order logic, enabling tasks such as graph isomorphism, cut and bipartite reasoning, shortest paths, and property invariants like adding color. GPS-Graph will explore how generative mechanisms can create graph puzzles with guaranteed correctness and semantic meaning under context of graph theory, potentially enabling benchmarks beyond traditional influence diffusion, such as constrained graph synthesis and variation of existing influence problem.

- **GPS-Relational**: a generative framework for synthesizing relational database query problems and natural-language query interpretation. This work aims to develop a framework that tests symbolic reasoning over tabular and relational structures, generating relational queries with automatically verifiable answers and explanation transparency. GPS-Relational will connect structured query reasoning with natural language as the user interface.

In addition to these two directions, future research includes:

- Integrating symbolic proofs and generative reasoning with reinforcement learning or self-improvement loops.
- Developing multimodal GPS systems that generate reasoning tasks across text, graphs, and structured data.
- Investigating collaboration between LLMs and symbolic provers for automated self-improvement learning.
- Applying generative influence networks to real-world evolving systems such as scientific citation graphs, financial markets, or social dynamics.

Overall, **Generative Problem Solving remains a largely unexplored but highly promising direction**. By combining symbolic structure with scalable generative intelligence, this research aims to contribute toward AI systems capable of constructing knowledge, explaining reasoning, and improving through creation rather than imitation. We hope this work inspires further development toward unified frameworks that bridge formal reasoning capability and generative flexibility.

Bibliography

[1] D. Aditya, K. Mukherji, S. Balasubramanian, A. Chaudhary, and P. Shakarian. Pyreason: Software for open world temporal logic. In *AAAI Spring Symposium: MAKE*, 2023.

[2] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann. Netgan: Generating graphs via random walks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[4] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 946–957, 2014.

[5] H. Chase. Langchain: Building applications with llms through composability. `https://www.langchain.com/`, 2022.

[6] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 199–208, 2009.

[7] Y.-C. Chen, W.-C. Peng, and S.-Y. Lee. Efficient algorithms for influence maximization in social networks. *Knowledge and Information Systems*, 33(3):577–601, 2012.

[8] A. Colmerauer and P. Roussel. The birth of Prolog. In T. J. J. Bergin and R. G. J. Gibson, editors, *History of Programming Languages—II*, pages 331–367. ACM Press / Addison-Wesley, New York, NY, USA, 1996.

[9] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[10] P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[11] A. Goyal, W. Lu, and L. V. Lakshmanan. Celf++: Optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 47–48, 2011.

[12] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, W. Zhou, J. Coady, D. Peng, Y. Qiao, L. Benson, L. Sun, A. Wardle-Solano, H. Szabo, E. Zubova, M. Burtell, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, A. R. Fabbri, W. Kryściński, S. Yavuz, Y. Liu, X. V. Lin, S. Joty, Y. Zhou, C. Xiong, R. Ying, A. Cohan, and D. Radev. Folio: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Miami, Florida, USA, 2024. Association for Computational Linguistics.

[13] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

[14] D. Kempe, J. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 137–146, 2003.

[15] L. Kovács and A. Voronkov. First-order theorem proving and vampire. In *Computer Aided Verification (CAV 2013)*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[16] M. Madeira, M. Cardoso, B. Leal, G. Mateus, A. Loureiro, and X. Jiang. Generative modelling of structurally constrained graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[17] OpenAI. Gpt-5 system card. `https://openai.com/index/gpt-5-system-card/`, Aug. 2025.

[18] B. Peng. Dynamic influence maximization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[19] H.-S. Shih, C. Lyu, G. Vaisi, and P. C.-Y. Sheu. Generative problem solving. In *Proceedings of the 19th International Conference on Semantic Computing (ICSC)*, Feb. 2025.

[20] P. Song, K. Yang, and A. Anandkumar. Towards large language models as copilots for theorem proving in Lean. *arXiv preprint arXiv:2404.12534*, 2024.

[21] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1539–1554, 2015.

[22] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 75–86, 2014.

[23] J. Tian, Y. Li, W. Chen, L. Xiao, H. He, and Y. Jin. Diagnosing the first-order logical reasoning ability through logicnli. In *Proc. EMNLP 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3738–3747, Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics.

[24] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[25] Z. Wan, C.-K. Liu, H. Yang, C. Li, H. You, Y. Fu, C. Wan, T. Krishna, Y. Lin, and A. Raychowdhury. Towards cognitive AI systems: a survey and prospective on neuro-symbolic AI. *CoRR*, 2024.

[26] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[27] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, volume 35, pages 24824–24837, 2022.

[28] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics.

[29] X. Wu, L. Fu, Z. Zhang, H. Long, J. Meng, X. Wang, and G. Chen. Evolving influence maximization in evolving networks. *ACM Transactions on Internet Technology (TOIT)*, 20(4):1–22, 2020.

[30] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822, 2023.

[31] X. Ying and X. Wu. Graph generation with prescribed feature constraints. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 966–974, 2008.

[32] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.