GRAIN: Bridging Name and Narrative Shifts in Real-World Graph Reasoning through Invariance-Rewarded Agentic RL

Zike Yuan[1,2], Han Zhang[2], Jianzhi Yan[1,2], Le Liu[1,2], Cai Ke[1,2], Huozhi Zhou, Jian Xie, Jiran Yin, Yukun Cao[3], Yue Yu[2], Hui Wang[2], Ming Liu[1,2,*], Bing Qin[1,2,*]

[1]Harbin Institute of Technology, Shenzhen, China
[2]Pengcheng Laboratory, Shenzhen, China
[3]Xidian University, Xi'an, China
{yuanzk,wangh06}@pcl.ac.cn
{mliu,qinb}@ir.hit.edu.cn
caoyukun@xidian.edu.cn

Abstract

Despite their potential in standardized graph tasks, Large Language Models (LLMs) remain brittle to real-world shifts in node identifiers and task formulation. While deterministic graph tools are invariant to such shifts, extracting topological structures from noisy text is highly fragile for LLMs, which often overfit to surface patterns. Moreover, mitigating these parsing failures via multi-agent systems incurs prohibitive latency. To address this, we propose GRAIN, a single-agent framework optimized via reinforcement learning. GRAIN models reasoning as a semantic parsing and tool-execution pipeline, guided by a Structure Invariance Reward. By validating extracted intermediate graphs against ground-truth topologies, this reward forces the LLM to learn robust text-to-structure mappings rather than memorizing linguistic artifacts. We also introduce GRIT, a benchmark evaluating sensitivity to such linguistic shifts. GRAIN outperforms multi-agent baselines by 16.45% in accuracy with approximately 24% lower latency. Furthermore, it demonstrates superior structural generalization, halving the out-of-distribution (OOD) gap of SFT models (from 15.77% to 7.80%) and maintaining robustness on large-scale graphs beyond the training distribution.

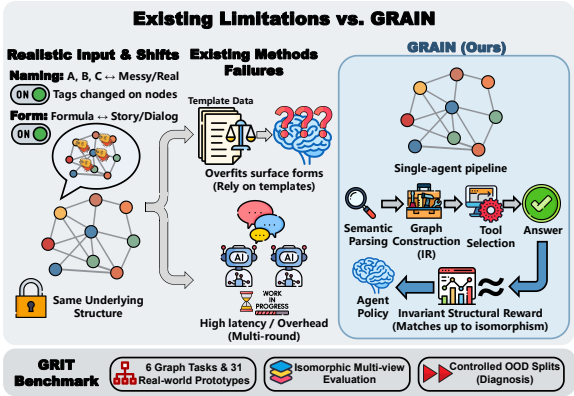

Figure 1: GRAIN mitigates sensitivity to **naming and formulation shifts** in real-world graph reasoning via a single-agent tool-use pipeline trained with **isomorphism-aware structural rewards**.

1 Introduction

Large Language Models (LLMs) exhibit great versatility in language, code, and tool utilization. However, critical domains including social network analysis, knowledge graph reasoning, and software dependency management necessitate robust reasoning over inherent graph structures. While recent approaches address classical tasks using text-serialized graphs (Wang et al., 2023), they often depend on benchmarks with sanitized identifiers. However, extracting accurate graph structures from text heavily laden with naming irregularities, alias collisions, and semantic noise is a fundamental challenge in Information Extraction (IE) and Semantic Parsing. Such idealized settings overlook these real-world linguistic complexities, which ultimately limits reliability in practical applications. We examine the robustness gap between clean benchmarks and practical graph reasoning tasks, revealing that LLMs are surprisingly brittle to superficial variations. As shown on the left side of Figure 1, this instability stems from two sources: ❶ **Node-label sensitivity**, where altering naming schemes (e.g., Random vs. Semantic) shifts performance despite structural equivalence. ❷ **Task-form sensitivity**, where performance degrades when moving from synthetic formulations to real-world narratives. Our analysis (see Figure 2 and Table 10) highlights a distinct disparity: while closed-source models generally maintain accuracy but suffer from unpredictable variance across naming schemes, open-source models exhibit severe degradation under non-canonical settings, especially in realistic scenarios. These observations underscore the critical need to explicitly model and mitigate robustness failures caused by naming and

*Corresponding authors: Ming Liu, and Bing Qin.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 32584–32604
October 24-29, 2026 ©2026 Association for Computational Linguistics

formulation shifts.

Prior research investigates LLM graph representations, yet critical limitations persist. **First**, while textualization studies (Fatemi et al., 2024) reveal sensitivity to serialization (e.g., labeling), they typically overlook distribution shifts between standardized templates and noisy real-world formulations. **Second**, optimization paradigms like CoT and Supervised Fine-Tuning (SFT) struggle with structural generalization; SFT specifically overfits surface patterns, causing performance degradation under isomorphic variations, scale shifts (*i.e.*, generalizing to graphs with significantly more nodes than the training set), or computational intensity (e.g., NP-hard problems) (Chen et al., 2024; Guo et al., 2025; Zhang et al., 2025). **Finally**, although multi-agent frameworks (Yuan et al., 2025b; Zhang et al., 2024a) improve accuracy via external tools, they suffer from inherent inefficiency: reliance on multi-round interactions incurs prohibitive token costs and latency, rendering them impractical for real-time deployment.

To address these limitations, we identify three critical design requirements: (i) **Explicit Pipeline Modelling**: Instead of relying on standard, free-form Chain-of-Thought (CoT) generation where LLMs mix logical reasoning with mental arithmetic, we decompose reasoning into explicit *semantic parsing* and *algorithmic execution*. This imposes verifiable constraints on intermediate representations, ensuring structural validity before computation. (ii) **Structural Invariance**: To decouple reasoning from specific identifiers, we vary naming and narratives while preserving the underlying structure, compelling the model to learn invariant rules rather than memorizing surface artifacts. (iii) **Unified Agentic RL Optimization**: As static supervision overfits surface forms, we employ RL with structural rewards to enforce generalization. This agentic RL formulation distills the high-accuracy reasoning capabilities—typically requiring complex multi-agent collaboration—into a single model's weights, achieving state-of-the-art performance without incurring their prohibitive latency.

Guided by these principles, we propose **GRAIN** (**G**raph **R**easoning **A**gent with **IN**variance), a single-agent RL framework for robust graph reasoning. GRAIN acts as an agentic pipeline that autonomously parses noisy queries, selects optimal external algorithms, and constructs necessary structural arguments. This ensures computational correctness while focusing the LLM purely on semantic parsing and information extraction. To mitigate sensitivity to surface shifts, we train GRAIN on diverse isomorphic narratives. Crucially, we optimize a unified RL objective with an *invariance-oriented structural reward*. By validating generated intermediate graphs against ground-truth topologies, this signal provides direct feedback on structure recovery, forcing the policy to learn invariant structural rules rather than memorizing superficial identifiers.

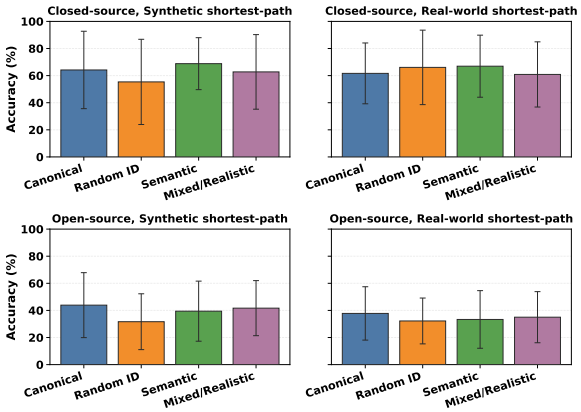

Figure 2: **Node-label Sensitivity.** Large error bars reveal high instability: open-source models degrade on Random IDs, while closed-source models struggle with canonical forms.

To facilitate systematic diagnosis, we introduce **GRIT**, a multi-task corpus covering six graph problems across 31 real-world prototypes. By pairing standard formulations with diverse narratives and namings, GRIT enables controlled **OOD** evaluation on fixed structures. Experiments demonstrate that GRAIN leverages this structural feedback to optimize the "language → structure" chain, achieving exceptional robustness against surface shifts while maintaining low computational complexity. Our primary contributions are:

- We quantify LLM sensitivity to node labeling and task formulation under a controlled isomorphic setup. We demonstrate that surface-level variations induce significant performance volatility, even when the underlying structure and semantics remain fixed.
- We propose **GRAIN**, a single-agent RL framework that leverages an invariance-oriented structural reward. By directly optimizing intermediate structure recovery and tool invocation, GRAIN achieves superior robustness across diverse naming schemes and task forms.
- We release **GRIT**, a multi-task benchmark featuring isomorphic multi-view coverage with explicit OOD splits. This resource enables systematic,

reproducible evaluation of graph reasoning robustness against naming and formulation shifts.

2 Related Work

LLMs for Graph Reasoning. Graph reasoning is essential to many real-world applications, including social network analysis, knowledge graph reasoning (Chen, 2024; Li et al., 2026), and software dependency management. However, existing benchmarks show that LLM performance declines substantially as graph size and task complexity increase, particularly when moving from basic connectivity problems to NP-hard tasks (Wang et al., 2023; Tang et al., 2025; Yuan et al., 2025a; Luo et al., 2024; Zhang et al., 2024c). To better understand these limitations, recent research has begun to shift from evaluating only final answers toward diagnosing intermediate reasoning processes (Taylor et al., 2024). Meanwhile, instruction-tuning methods improve graph reasoning accuracy through specialized data and curricula (Cao et al., 2025; Chen et al., 2024; Guo et al., 2025; Wang et al., 2025; Yuan et al., 2026). Nevertheless, these methods are typically optimized for standardized inputs and therefore remain brittle to changes in graph serialization formats and prompting schemes (Xu et al., 2026). This gap highlights the need for LLMs that can reason robustly over inherent graph structures rather than relying on specific surface representations.

Serialization Sensitivity. Graph serialization is a critical determinant of LLM performance. Prior work reveals that different encoding schemes (Fatemi et al., 2024) and even the descriptive order of edges (Li et al., 2024a) can drastically alter reasoning accuracy, highlighting a severe sensitivity to surface-form variations. To mitigate this reliance on raw textualization, approaches like GraphToken (Perozzi et al., 2024) propose injecting structured signals via parameter-efficient encodings, though robustness against diverse naming conventions remains an open challenge.

Agents and Tool Learning. Tool-augmented paradigms enhance reliability via external APIs or code execution (Yao et al., 2022; Schick et al., 2023; Qin et al., 2024; Jin et al., 2025; Zhang et al., 2024b; Zeng et al., 2025; Xiong et al., 2026). In the graph domain, multi-agent frameworks improve performance by decomposing tasks into coordinated roles (Li et al., 2024b; Yuan et al., 2025b; Han et al., 2026; Cao et al., 2024; Qian et al., 2026). Yet, these systems incur high latency and token costs due to extensive inter-agent communication, highlighting the necessity for efficient single-agent solutions.

Split	Node Range	# Tasks	# Scen.	# Graphs	Total Qs.
Train	4–40	6	31	2160	17,280
Test	10–40	6	31	360	2,760
Large Test	40–60	6	31	120	960
OOD Test	10–40	6	24*	180	1,440

*Denotes distinct scenarios unseen during training.

Table 1: Statistics of the GRIT benchmark splits.

3 GRIT: A Benchmark for Robust Graph Reasoning

We introduce GRIT to evaluate LLM robustness against two real-world distribution shifts: *identifier shift* (node naming variations) and *task-form shift* (standard vs. narrative formulations). Unlike benchmarks with fixed identifiers, GRIT provides controlled multi-view instances varying surface realizations while preserving underlying structures, serving as a rigorous diagnostic tool for structural information extraction and semantic parsing.

Task coverage and scenarios. GRIT covers six fundamental tasks spanning polynomial to NP-hard complexities (West et al., 2001). We instantiate them into 31 scenarios (e.g., logistics routing, social networks), yielding naturalistic narratives laden with complex syntactic embeddings and semantic distractors. This offers significantly richer linguistic variation than canonical definitions (details in Appendix A).

Multi-view question construction. Each base instance generates **8** distinct queries by crossing **two forms** (*Standard* vs. *Realistic* narrative) with **four naming schemes**: *Canonical*, *Random IDs* (testing tokenization robustness), *Semantic*, and *Noisy/Mixed* (testing coreference resolution and alias mapping). This factorial design isolates specific axes of variation, enabling controlled evaluation of both identifier and task-form shifts.

Graph generation and deterministic injection. GRIT features topologies spanning in-domain ($N \in [10, 40]$) and length-generalization ($N \in [40, 60]$) scales, with ground-truth answers derived via symbolic solvers. Crucially, to prevent hallucinations, narratives are *not* LLM-generated. Instead, we employ a **strictly deterministic, template-**

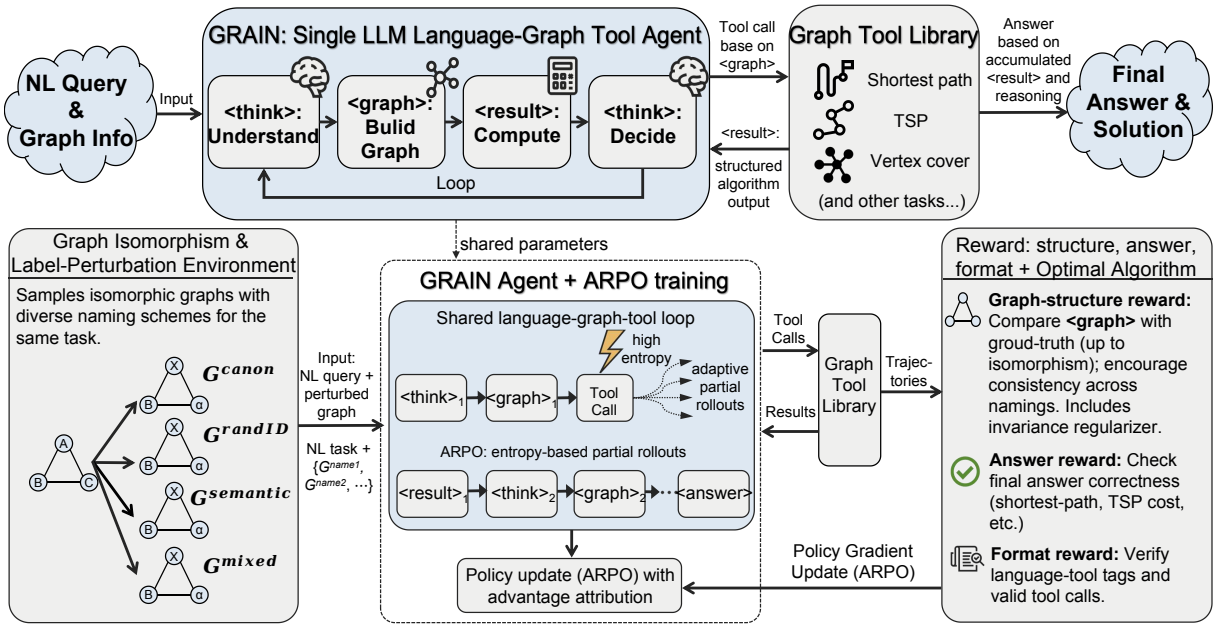

Figure 3: **Overview of the GRAIN framework. (Top) Inference:** The LLM builds a graph IR in `<graph>`, calls solvers, and outputs the answer. **(Bottom) Training: ARPO** trains under isomorphism-preserving name perturbations, with partial rollouts and a reward for structural invariance, correctness, and valid format.

based slot-filling engine to inject topologies into linguistic scenarios, guaranteeing 100% reliable isomorphism between the text and the underlying graph.

Benchmark splits. Table 1 summarizes the dataset splits. **Train** and **Test** share the same scenario pool, while **Large Test** evaluates length generalization on larger graphs ($N \in [40, 60]$). To isolate robustness against surface-level variations, the **OOD Test** set introduces *unseen* held-out scenarios and identifier realizations not observed during training.

4 GRAIN

To address the brittleness caused by identifier and task shifts, we propose **GRAIN**, a single-agent reinforcement learning framework for robust graph reasoning. GRAIN models the LLM as a language–graph–tool agent trained in an isomorphism-based environment with diverse naming and formulation views. Crucially, we enforce a *gated* reward structure that strictly penalizes format violations while incentivizing recovery of the underlying topology, ensuring the policy captures structural rules invariant to surface-level labels and wording.

4.1 Problem Formulation and Overall Objective

To formalize the task (Table 13), let $G = (V, E, w)$ be a weighted graph and $\mathcal{T}$ a task (e.g., TSP). Real-world queries incorporate variations in node naming $\nu : V \to \Sigma^*$ and formulation style ϕ (e.g., narrative complexity), defined as the natural language query $x = f(G, \nu, \phi, \mathcal{T})$. We denote the training graph distribution by $\mathcal{G}$ and the conditional distribution over surface forms (ν, ϕ) given G by $\mathcal{P}(G)$.

Given a policy π_θ, the full generation for input x is a trajectory $\tau(G, \nu, \phi; \pi_\theta)$, containing all tokens from `<think>` to `</answer>`. At a high level, we aim to train a policy that maximizes task performance while simultaneously recovering the underlying graph structure, robust to variations in naming and phrasing. We formulate this as maximizing the expected joint return:

$$\max_{\theta} \mathbb{E}_{\substack{G \sim \mathcal{G} \\ (\nu,\phi) \sim \mathcal{P}(G)}} \Big[R\big(\tau(G, \nu, \phi; \pi_\theta)\big) \Big], \qquad (1)$$

where the total return $R(\tau)$ composites the task-level reward and a structural invariance score, subject to strict formatting constraints as detailed in Section 4.3.

4.2 GRAIN: A Single Language–Graph–Tool Agent

GRAIN treats the LLM as a single agent interacting with a graph-tool library via structured text. For each input x, the agent outputs a sequence with four tagged segments (a complete step-by-step execution example is provided in Appendix C):

$$y = \langle \texttt{<think>}\ldots, \texttt{<graph>}\ldots, \texttt{<result>}\ldots, \texttt{<answer>}\ldots \rangle. \tag{2}$$

As illustrated in Figure 3, the overall reasoning pipeline consists of four stages:

Task understanding and planning (`<think>`). The agent reads x, identifies the task type $\mathcal{T}$, extracts entity mentions corresponding to nodes and edges, and sketches a plan of which graph algorithms to call and with which parameters.

Graph construction (`<graph>`). In the `<graph>` segment, the agent emits a structured representation $\hat{G}$ that specifies the node set, edges and weights, and task-specific parameters (e.g., source/target nodes or the set of cities for TSP). The runtime system parses `<graph>` into the input format of the graph-tool library.

Tool calls and result injection (`<result>`). The system invokes the corresponding graph algorithm, obtains a deterministic result $\hat{z}$, and injects it back into the context as a `<result>` segment.

Continued reasoning and final answer (`<answer>`). Upon seeing `<result>`, the agent may produce additional `<think>` and `<graph>` segments and trigger more tool calls, or directly synthesize the final answer in `<answer>`. The episode terminates at `</answer>`.

With the tool library ensuring computational correctness, learning focuses on the language-to-graph decision chain: entity resolution, reconstruction, and tool/parameter selection under noisy naming and diverse formulations.

4.3 Isomorphic Environment and Gated Rewards

To simulate real-world variability, we construct an isomorphism-based perturbation environment. For each training graph G and task $\mathcal{T}$, we generate diverse naming schemes ν (Canonical, Random, Semantic, Mixed) and query styles ϕ, yielding an instance family $x = f(G, \nu, \phi, \mathcal{T})$.

The RL state s_t comprises the input x, generated history, and injected `<result>` segments; episodes terminate at `</answer>`. To enforce strict protocol adherence while optimizing structural recovery, we employ a **gated reward formulation** consisting of three components:

Answer Reward $r_{\text{ans}}(\tau)$. This term checks whether the `<answer>` segment provides the correct task-specific value (shortest-path length, TSP tour cost, etc.) and assigns a binary or shaped reward accordingly.

Structure Similarity Score $s_{\text{inv}}(\tau)$. We parse the extracted graph $\hat{G}$ from the `<graph>` segment and compute the **Jaccard Index** over the canonical edge sets of $\hat{G}$ and the ground-truth G. Specifically, we align entity names via the environment's mapping and represent edges as sets of canonical tuples `(source, target, weight)`. This set-theoretic implementation mathematically prevents out-of-bounds dimension errors that would occur in matrix-based similarity (e.g., Cosine) if the LLM hallucinates or misses nodes. Furthermore, unlike computationally NP-hard metrics such as Graph Edit Distance (GED), this signal is highly efficient to compute while granting dense, continuous partial credit ($s_{\text{inv}} \in [0, 1]$) for topological recovery.

Gated Total Reward $R(\tau)$. Instead of treating formatting as a soft regularization term, we impose it as a hard constraint. The final trajectory return is defined as:

$$R(\tau) = \begin{cases} \rho_{\text{err}}, & \text{if invalid,} \\ r_{\text{ans}}(\tau) + \lambda_{\text{inv}}\, s_{\text{inv}}(\tau), & \text{if valid.} \end{cases} \tag{3}$$

where ρ_{err} penalizes malformed trajectories (e.g., missing tags). For valid outputs, the reward sums task correctness and structure similarity (equally weighted with $\lambda_{\text{inv}} = 1.0$).

Training utilizes the underlying G to compute $s_{\text{inv}}(\tau)$, whereas inference operates without labels. By scoring diverse views (ν, ϕ) of the same G with this signal, the policy learns invariant topological rules rather than memorizing surface phrasing.

4.4 Reinforcement Learning with ARPO

To address cold-start challenges and ensure format compliance, we perform a brief **SFT warm start**. Expert trajectories are automatically generated on small synthetic graphs ($|V| \leq 14$) using the graph-tool library. Minimizing standard autoregressive

Model Family	Method / Setting	S. Path	Coloring	TSP	V. Cover	BFS	Centr.	Avg.
Proprietary Models & Multi-Agent Systems								
GPT-5-nano	Zero-shot CoT	72.18	43.54	3.33	29.17	16.25	15.21	29.95
	Tool-use CoT	73.39	68.96	41.88	28.61	31.46	40.42	47.45
G1-3B	Zero-shot CoT	11.46	8.96	0.00	5.28	1.04	8.33	5.85
MA-GTS(GPT-4o-mini)	Multi-Agent	85.63	87.50	91.25	68.33	55.62	87.50	79.31
MA-GTS(Qwen3-4B-Ins.)†	Multi-Agent	13.75	12.50	1.88	4.17	0.62	7.50	6.74
Llama-3.2 Series (3B)								
Llama-3.2-Ins	Zero-shot CoT	1.88	13.96	0.21	3.61	4.38	4.58	4.77
	Tool-use CoT	2.92	11.88	0.21	4.44	0.83	5.42	4.28
GRAIN (Ours)	SFT + ARPO	45.93	51.46	34.79	15.83	71.88	64.58	47.41
Qwen-3 Series (4B)								
Qwen-3-Ins	Zero-shot CoT	53.13	28.75	1.88	5.28	70.42	12.71	28.69
	Tool-use CoT	48.13	36.88	71.25	38.61	60.42	42.29	49.60
Qwen-3-Base	Zero-shot CoT	49.38	22.29	2.08	4.44	48.96	11.88	23.17
	Tool-use CoT	35.42	45.28	40.00	37.50	70.42	42.29	45.15
	SFT (Text-only)	22.13	46.88	3.54	33.89	71.25	45.00	37.11
	SFT (Tool-use CoT)	**99.58**	**93.13**	96.88	88.33	82.50	74.17	89.10
GRAIN (Ours)	SFT + ARPO	99.38	92.92	**97.92**	**93.30**	**98.33**	**92.70**	**95.76**

Table 2: **Main Results on GRIT Tasks.** We report the accuracy (%) across six graph reasoning tasks. The best results are **bolded**, and the second-best results are underlined. "Ins" denotes Instruct models. For Qwen-3-Base, we compare CoT and SFT under both text-only and tool-use settings. † denotes zero-shot transfer of the six-agent MA-GTS pipeline to Qwen3-4B-Instruct on the 920-example Test set; its micro-average is 6.85%. GRAIN consistently achieves SOTA performance.

cross-entropy loss enables the model to acquire correct tag usage, valid tool syntax, and basic construction skills. This model serves as both the RL initialization θ_0 and the reference policy π_{ref}.

We then further optimize the policy using Reinforcement Learning with Verifiable Rewards (RLVR). Writing $x = f(G, \nu, \phi, \mathcal{T})$ and $\tau \sim \pi_\theta(\cdot \mid x)$, our training objective is:

$$J(\theta) = \mathbb{E}_{x,\,\tau\sim\pi_\theta(\cdot|x)}\Big[R(\tau) - \beta\,\mathrm{KL}\big(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\big)\Big], \tag{4}$$

where $R(\tau)$ follows the gated formulation in Eq. (3). We optimize $J(\theta)$ using **ARPO** (Agentic Reinforced Policy Optimization) (Dong et al., 2026), which is uniquely suited for our problem setting:

Entropy-based adaptive partial rollouts. In our pipeline, naming and query perturbations primarily impact entity resolution and graph reconstruction, empirically manifesting as token entropy spikes. ARPO detects these high-uncertainty moments to trigger adaptive partial rollouts (branching), enabling the agent to explore diverse graph construction hypotheses when confronting ambiguous identifiers.

Branch-aware advantage estimation. ARPO standardizes returns across trajectories sharing a prefix, assigning shared advantages to prefix tokens (task understanding) and individual ones to branch suffixes (graph construction). This ensures the structure reward s_{inv} targets high-entropy grounding decisions, facilitating efficient learning.

5 Evaluation

5.1 Experimental Setup

Benchmarks and Data. We evaluate primarily on **GRIT**, spanning six tasks (SHORTEST PATH, COLORING, TSP, VERTEX COVER, BFS, CENTRALITY). Each instance features **8 views** derived from **31** scenarios: **2** forms (*Standard*/*Realistic*) × **4** naming schemes (*Canonical*, *Random IDs*, *Semantic*, *Noisy/Mixed*). Training uses graphs with $N \leq 40$; evaluation covers **ID** (seen), **OOD-Form** (unseen naming–narrative pairs, $N \leq 40$), and **OOD-Large-Size** ($N \in [40, 60]$). To ensure unbiased evaluation, the node sizes in our test sets

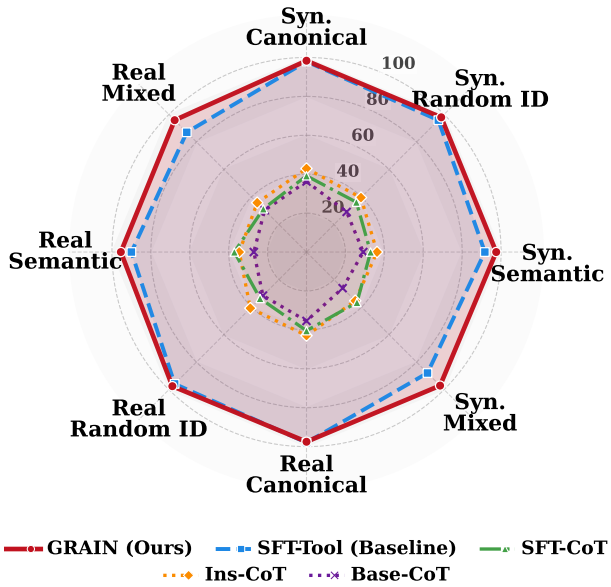

Figure 4: **Robustness Evaluation.** GRAIN (Red) achieves consistent robustness across shifts, unlike SFT-Tool (Blue) which degrades significantly in noisy, real-world scenarios.

follow a strictly balanced, uniform distribution (detailed in Appendix H). We also report results on external benchmarks **GraphInstruct** (Luo et al., 2024) and **GraphArena** (Tang et al., 2025).

Models and Baselines. We compare **Zero-shot CoT** (Wei et al., 2022), **Tool-use CoT**, **SFT**, and **GRAIN** (SFT+**ARPO**). Baselines include **GPT-5-nano** (Achiam et al., 2023), the graph-specific RL model **G1-3B** (Guo et al., 2025), and the multi-agent **MA-GTS** (Yuan et al., 2025b). For open models, we evaluate **Llama-3.2-Ins** (Touvron et al., 2023) (3B) and **Qwen-3-Ins** (Bai et al., 2023) (4B) ("Ins" denotes Instruct). Additionally, we train **GRAIN** on **Llama-3.2** (3B) to test transferability. On **Qwen-3-Base** (4B), we conduct a comprehensive comparison across Zero-shot, Tool-use, SFT (Text/Tool), and GRAIN to isolate the specific effects of tool access and training.

Tool Library and Inference Protocol. Tool-use methods share a unified interface: models generate graph representations, invoke verifiable solvers, and derive answers. We enforce identical decoding budgets and tool-call limits (text-only baselines disable tools).

Training Details. Training relies exclusively on **GRIT Train**. **SFT** is performed on the full split to provide initializations. **GRAIN** warm-starts from the tool-use SFT checkpoint, employing a two-stage curriculum: first stabilizing structured generation on small graphs (**4–14** nodes), then running **ARPO** on the **10–40** node range by sampling **two** underlying graphs per task and size. Implementation details are provided in Appendix B.

Evaluation Metrics. We report **final-answer accuracy** (%), counting a prediction as correct iff it strictly matches the ground truth. Tables present per-task performance and the macro-average (**Avg.**), computed using a unified scoring script across all settings.

5.2 Performance

5.2.1 Overall Performance on GRIT

Table 2 compares GRAIN against representative baselines across six graph reasoning tasks. GRAIN achieves consistent state-of-the-art performance across varying model scales, demonstrating the effectiveness of our proposed framework. The analysis of the case study is presented in the Appendix C.

Surpassing Supervised Baselines. GRAIN significantly outperforms SFT under identical configurations. On Qwen-3-Base, GRAIN boosts the strong SFT baseline (89.10%) to **95.76%**. Substantial gains on complex tasks like Vertex Cover (+4.97%) and BFS (+15.83%) validate that our *invariance-oriented structural reward* effectively optimizes non-differentiable decision chains, surpassing the performance ceiling of standard imitation learning.

Multi-Agent Comparison. MA-GTS with GPT-4o-mini achieves 79.31%, whereas GRAIN reaches 95.76% with a compact Qwen-3 backbone. As a compact-model transfer diagnostic, the same six-agent MA-GTS pipeline with Qwen3-4B-Instruct obtains a 6.74% macro-average (6.85% micro-average) on the 920-example Test set, with consistently low accuracy across tasks. Because training, backbone initialization, and agent organization differ from GRAIN, this result should not be interpreted as an isolated single- versus multi-agent effect; rather, it indicates that inference-only multi-agent decomposition does not automatically resolve graph-grounding and executable tool-call failures in this setting. Appendix F reports the complete pipeline diagnostic.

Empowering Compact Models. GRAIN also unlocks the reasoning potential of smaller architectures. While Llama-3.2-Instruct fails in zero-shot and standard tool-use settings (<5%), GRAIN boosts its average accuracy to **47.41%**, matching the performance of the proprietary GPT-5-nano

Method	GRIT Distribution			Unseen Tasks	
	Test	OOD	Gap↓	G-Ins	Arena
Baseline: Qwen3-4B-SFT					
Zero-shot CoT	37.11	27.29	9.82	32.83	13.0
Tool-use CoT	89.10	73.33	15.77	92.06	88.0
GRAIN-4B (Ours)					
SFT + ARPO	**95.76**	**87.96**	**7.80**	**97.05**	**91.0**

Table 3: **OOD Robustness & Generalization.** We report the accuracy drop (Δ) on GRIT and zero-shot performance on unseen benchmarks (GraphInstruct, GraphArena).

Method	Rounds	Tokens	Latency (s)		Acc.
	(Avg)	(Avg)	Avg	p90	(%)
GPT-5-nano (Single-Agent)					
Zero-shot CoT	1.00	17.1k	117.3	200.2	29.95
Tool-use CoT	2.00	22.1k	126.1	212.8	47.45
Multi-Agent Framework					
MA-GTS (4o-mini)	6.00	13.5k	58.4	96.1	79.31
GRAIN-4B (Ours)					
SFT + ARPO	3.29†	**3.3k**	**44.4**	**82.5**	**95.76**

† Denotes interaction rounds with the local Python environment.

Table 4: GRAIN achieves peak accuracy (95.76%) with the lowest overhead (3.3k tokens, 44.4s), significantly outperforming proprietary and multi-agent baselines.

(47.45%). This indicates that our framework effectively instills structural reasoning capabilities even within compact parameter spaces.

5.2.2 Robustness against Naming and Formulation Shifts

Figure 4 illustrates stability across eight variations. GRAIN demonstrates exceptional **structural invariance**, maintaining a nearly uniform performance envelope across all axes. Conversely, SFT-Tool exhibits marked **node-label sensitivity**, with performance visibly collapsing under "Random" and "Mixed" identifiers. This confirms GRAIN effectively decouples reasoning from surface forms, mitigating the overfitting to canonical patterns inherent in standard SFT. Further analysis of the two sensitivities are presented in the Appendix D.

5.2.3 OOD Generalization and Scalability

We evaluate GRAIN's capability to generalize beyond its training distribution across two dimensions: domain shifts (Table 3) and graph scale (Figure 5).

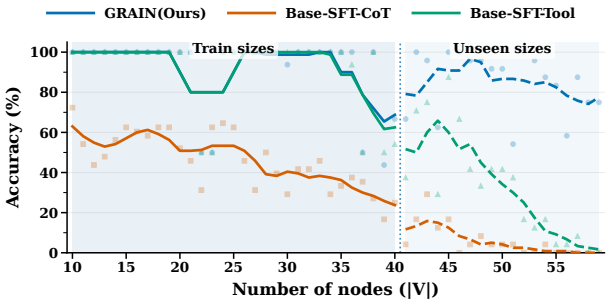

Figure 5: Performance as node count ($|V|$) exceeds the training range ($|V| \leq 40$). GRAIN demonstrates robust length generalization, sustaining accuracy on larger graphs where baselines collapse.

Method / Variant	Metrics		
	Gap↓	Var. (10^{-5})↓	Acc.↑
GRAIN (Full)	**1.14%**	**4.64**	**96.76%**
Component Analysis			
w/o Struct. Reward	11.14%	14.65	84.73%
w/o Diverse Naming	4.11%	341.83	79.21%
Optimization Method			
replace w/ GRPO	1.17%	7.05	96.51%

Table 5: **Ablation Study.** **Gap** denotes the performance drop ($|Standard - Real|$), and **Var.** (10^{-5}) measures stability across naming permutations.

Distribution Shift and Transferability. Table 3 highlights GRAIN's resilience: while SFT-Tool drops 15.77% on OOD splits due to overfitting, GRAIN halves this gap to **7.80%**. Furthermore, GRAIN demonstrates exceptional **zero-shot transferability** on GraphInstruct (97.05%) and GraphArena (91.0%). To verify baseline fairness, GRAIN achieves **86.60%** zero-shot accuracy on the unseen **G-REAL** dataset (MA-GTS's original benchmark; Appendix E). This confirms that our invariance-oriented rewards foster universally applicable structural extraction skills.

Length Generalization. Figure 5 illustrates performance as graph size scales beyond the training range ($|V| > 40$). While baselines degrade rapidly on larger graphs due to context overload, GRAIN maintains robust accuracy. This **length generalization** is a direct benefit of our explicit pipeline design: by offloading computational complexity to external algorithms, GRAIN limits the LLM's role to semantic parsing and information extraction, effectively decoupling performance from the exponential complexity of graph reasoning.

Method / Variant	Node-F1	Edge-F1	Exact
w/o Struct. Reward	96.03	94.95	63.10
SFT-Tool	98.45	97.96	90.21
replace w/ GRPO	97.23	97.20	95.22
GRAIN (Full)	**98.76**	**98.67**	**97.28**

Table 6: **Intermediate Structure Recovery.** Results on 2,760 GRIT examples. Edge-F1 denotes weighted edge recovery, while Exact requires the entire predicted weighted graph to match the ground-truth graph.

5.2.4 Ablation Study

Table 5 validates component contributions. Removing the **Structure Invariance Reward** widens the generalization gap ($1.14\% \rightarrow 11.14\%$) and drops accuracy to 84.73%, confirming its role in preventing overfitting. **Diverse Naming** is crucial for stability; omitting it spikes variance ($4.64 \rightarrow 341.83$) and degrades accuracy to 79.21%. Finally, **ARPO** outperforms GRPO by lowering variance ($7.05 \rightarrow 4.64$) and boosting accuracy ($96.51\% \rightarrow 96.76\%$), demonstrating the efficacy of branch-aware optimization.

Intermediate Structure Recovery. As shown in Table 6, GRAIN achieves **98.67%** weighted Edge-F1 and **97.28%** exact graph recovery. Removing the Structure Invariance Reward reduces exact recovery to **63.10%** despite retaining high node- and edge-level overlap, showing that partial overlap can mask whole-graph inconsistencies. Replacing ARPO with GRPO also reduces exact recovery from 97.28% to 95.22%. Appendix G provides a detailed decomposition of the remaining non-exact predictions.

5.2.5 Efficiency Analysis

Table 4 highlights GRAIN's optimal accuracy-efficiency trade-off.

Deployment Efficiency. Compared to GPT-5-nano's verbose reasoning (>17k tokens), GRAIN's concise structured generation cuts token usage by $\sim$85% (to **3.3k**) and achieves the lowest latency (**44.4s**).

Advantage over Multi-Agent Systems. While MA-GTS requires 6 interaction rounds, GRAIN's single-agent design outperforms it by **16.45%** in accuracy while reducing latency by **24%**. Our failure analysis (Appendix F) reveals that multi-agent systems frequently suffer from fatal information loss during handovers. Thus, for structural semantic parsing, GRAIN delivers SOTA performance without the prohibitive communication penalties of multi-agent architectures.

6 Conclusion

We address LLM fragility in graph reasoning under identifier and task shifts. We propose **GRAIN**, a single-agent RL framework utilizing a **Structure Invariance Reward** to decouple topology from surface variations. On our **GRIT** benchmark, GRAIN achieves state-of-the-art accuracy and efficiency, outperforming complex multi-agent systems. Furthermore, it generalizes robustly to OOD scenarios and larger graphs, validating the efficacy of invariant structural grounding.

Limitations

A limitation of our current framework lies in the management of tool contexts. Given the vast array of graph algorithms, embedding comprehensive tool definitions directly into the system prompt leads to excessive text length, which consumes valuable context window and may distract the model from core reasoning tasks. Future work will explore adopting standards like the Model Context Protocol (MCP) (Hou et al., 2025) to handle tool definitions dynamically, moving away from static inclusion to optimize context efficiency.

Acknowledgments

The research in this article is supported by the National Science Foundation of China (U22B2059, 62276083), Key Research and Development Program of Heilongjiang Province (2024ZX01A05) and the 5G Application Innovation Joint Research Institute's Project (A003).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yukun Cao, Zengyi Gao, Zhiyang Li, Xike Xie, S Kevin Zhou, and Jianliang Xu. 2024. Lego-graphrag: Modularizing graph-based retrieval-augmented generation for design space exploration. *arXiv preprint arXiv:2411.05844*.

Yukun Cao, Shuo Han, Zengyi Gao, Zezhong Ding, Xike Xie, and S Kevin Zhou. 2025. Graphinsight: Unlocking insights in large language models for graph structure understanding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12096–12134.

Huajun Chen. 2024. Large knowledge model: Perspectives and challenges. *Data Intell.*, 6(3):587–620.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. 2026. Agentic reinforced policy optimization. In *International Conference on Learning Representations*, volume 2026, pages 16981–17017.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. Talk like a graph: Encoding graphs for large language models. In *International conference on learning representations*, volume 2024, pages 43909–43934.

Xiaojun Guo, Ang Li, Yifei Wang, Stefanie Jegelka, and Yisen Wang. 2025. G1: Teaching llms to reason on graphs with reinforcement learning. *arXiv preprint arXiv:2505.18499*.

Shuo Han, Yukun Cao, Zezhong Ding, Zengyi Gao, S Kevin Zhou, and Xike Xie. 2026. See or say graphs: Agent-driven scalable graph understanding with vision-language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 41565–41589.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Transactions on Software Engineering and Methodology*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Xin Li, Weize Chen, Qizhi Chu, Haopeng Li, Zhaojun Sun, Ran Li, Chen Qian, Yiwei Wei, Zhiyuan Liu, Chuan Shi, et al. 2024a. Can large language models analyze graphs like professionals? a benchmark, datasets and models. *Advances in Neural Information Processing Systems*, 37:141045–141070.

Xin Li, Qizhi Chu, Yubin Chen, Yang Liu, Yaoqi Liu, Zekai Yu, Weize Chen, Chen Qian, Chuan Shi, and Cheng Yang. 2024b. Graphteam: Facilitating large language model-based graph analysis via multi-agent collaboration. *arXiv preprint arXiv:2410.18032*.

Zhiyang Li, Ao Ke, Yukun Cao, and Xike Xie. 2026. Kg-vip: Bridging knowledge grounding and visual perception in multi-modal llms for visual question answering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 35144–35157.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. 2024. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *arXiv preprint arXiv:2403.04483*.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. 2026. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. Grapharena: Evaluating and exploring large language models on graph computation. In *International Conference on Learning Representations*, volume 2025, pages 48118–48145.

Alexander K Taylor, Anthony Cuturrufo, Vishal Yathish, Mingyu Derek Ma, and Wei Wang. 2024. Are large-language models graph algorithmic reasoners? *arXiv preprint arXiv:2410.22597*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861.

Yuyao Wang, Bowen Liu, Jianheng Tang, Nuo Chen, Yuhan Li, Qifan Zhang, and Jia Li. 2025. Graph-r1: Unleashing llm reasoning with np-hard graph problems. *arXiv e-prints*, pages arXiv–2508.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Douglas Brent West et al. 2001. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River.

Siheng Xiong, Oguzhan Gungordu, James C Kerce, and Faramarz Fekri. 2026. Adaptive information control for search-augmented llm reasoning. *arXiv preprint arXiv:2602.01672*.

Hao Xu, Xiangru Jian, Xinjian Zhao, Wei Pang, Chao Zhang, Suyuchen Wang, Qixin Zhang, Zhengyuan Dong, Joao Monteiro, Bang Liu, et al. 2026. Graphomni: A comprehensive and extensible benchmark framework for large language models on graph-theoretic tasks. In *International Conference on Learning Representations*, volume 2026, pages 116120–116234.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Zike Yuan, Yukun Cao, Han Zhang, Jianzhi Yan, Le Liu, Yue Yu, Hui Wang, Ming Liu, Bing Qin, et al. 2026. Egl-sca: Structural credit assignment for co-evolving instructions and tools in graph reasoning agents. *arXiv preprint arXiv:2605.10366*.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025a. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7925–7948.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025b. Ma-gts: A multi-agent framework for solving complex graph problems in real-world applications. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 19297–19315.

Daojian Zeng, Lin Zhou, Zhiheng Zhang, and Lincheng Jiang. 2025. Autogen: Automated tool learning data generation with domain-specific structured data. *Data Intell.*, 7(4):1108–1128.

Qifan Zhang, Xiaobin Hong, Jianheng Tang, Nuo Chen, Yuhan Li, Wenzhong Li, Jing Tang, and Jia Li. 2024a. Gcoder: Improving large language model for generalized graph problem solving. *arXiv preprint arXiv:2410.19084*.

Xilin Zhang, Zhixin Mao, Ziwen Chen, and Shen Gao. 2024b. Effective tool augmented multi-agent framework for data analysis. *Data Intell.*, 6(4):923–945.

Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. 2024c. Can llm graph reasoning generalize beyond pattern memorization? In *Findings of the association for computational linguistics: EMNLP 2024*, pages 2289–2305.

Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xinyun Liu, and Yulia Tsvetkov. 2025. Generalizable llm learning of graph synthetic data with reinforcement learning. *arXiv e-prints*, pages arXiv–2506.

A GRIT Benchmark Implementation Details

This appendix complements the statistics in Table 1 and task descriptions in Table 8 by detailing the generation pipeline.

Graph Topology Generation. Underlying graphs are synthesized using two random models via NetworkX: **Erdős-Rényi** (edge probability $p \in [0.1, 0.3]$) . For weighted tasks, weights are integers uniformly sampled from $[1, 10]$. We perform rejection sampling to ensure connectivity (for undirected graphs) or strong connectivity (for TSP).

Ground Truth & Tie-Breaking. To ensure deterministic evaluation, we apply specific constraints to algorithmic solvers:

- **BFS Traversal:** Neighbors are visited in *lexicographical order* (e.g., node 2 before node 10) to guarantee a unique traversal sequence.
- **TSP:** Formulated as Metric TSP on complete graphs; we compute the exact minimum-cost Hamiltonian cycle using dynamic programming solvers.
- **Graph Coloring:** We compute the exact chromatic number using a greedy strategy with backtracking.

Narrative Injection Pipeline. We implement a template-based slot-filling engine. For each of the 31 scenarios (Table 8), we define a domain context (e.g., "Logistics") and specific entity lists. The engine injects graph topology into these templates under four naming schemes. For **Random IDs**, we use high-entropy integers (e.g., 8392, 1024) to test tokenization robustness; for **Noisy/Mixed**, we introduce aliases (referring to the same node by multiple names).

OOD Split Construction. Unlike random splits, the **OOD Test** set is constructed by explicitly **holding out** 24 specific scenario templates and naming distributions during training. For instance, if "City Traffic" appears in Train, "Server Routing" is reserved for OOD. This ensures that high performance reflects true structural reasoning transfer rather than domain text memorization.

Please refer to Table 11 for illustrative examples of the node naming variations (Canonical, Random IDs, Semantic, and Noisy/Mixed) used in our robustness evaluation. The complete system prompt, detailing the tool-use schema and reasoning guidelines for our GRAIN framework, is listed in Table 12.

B Training Implementation Details

This appendix details the hyperparameters and infrastructure used for the two-stage training pipeline of GRAIN: (1) Supervised Fine-Tuning (SFT) and (2) Agentic Reinforcement Learning via ARPO.

B.1 Infrastructure

All experiments were conducted on a computational node equipped with **4× NVIDIA A100 (80GB) GPUs**. For memory-efficient full-parameter fine-tuning, we utilized `DeepSpeed` with ZeRO-3 Offload strategy. For the RL stage, we integrated the `vLLM` inference engine to enable high-throughput trajectory rollouts. The training was performed in **BF16** precision to maintain numerical stability.

B.2 Stage 1: Supervised Fine-Tuning (SFT)

The SFT stage initializes the model's ability to use tools and generate valid JSON graph representations. We perform full-parameter fine-tuning on the **Qwen-3-Base (4B)** and **Llama-3.2-Base (3B)** models.

As listed in Table 7 (refer to main text or appendix table), we utilize a global batch size of 8 and a conservative learning rate of 7×10^{-6} with a cosine decay scheduler. Notably, we set a context window of **15,000 tokens** to accommodate the verbose serialization of larger graphs (up to 40 nodes) and the corresponding CoT reasoning.

Parameter	**Value**
General Settings	
Base Model	Qwen-3-4B
Tuning Method	Full Fine-tuning
Precision	BF16
Context Window	15,000 tokens
DeepSpeed Stage	ZeRO-3 (Offload)
GPUs	2× NVIDIA A100 GPUs
Hyperparameters	
Global Batch Size	8
Gradient Accumulation	2
Epochs	3
Learning Rate	7×10^{-6}
LR Scheduler	Cosine
Warmup Ratio	0.1

Table 7: **Implementation Details.** Hyperparameters and settings used for full-parameter fine-tuning.

Graph Task	Canonical Graph-Theoretic Formulation	Representative Real-World Problem Formulations (Noun Phrases)
Centrality	Compute centrality scores of one or several nodes in a graph and identify structurally "key" nodes.	City traffic hubs; emergency response centers; social network key persons; information relay stations
Shortest Path	In a weighted directed or undirected graph, find the minimum-cost path from a source node to a target node.	Data-center latency routing; logistics cost routes; robot energy-efficient paths; network routing paths
TSP	In a weighted complete graph, find a minimum-cost tour that visits every node once and returns to the start (decision version NP-complete, optimization version NP-hard).	Parcel delivery tour; sales representative tour; food delivery tour; garbage collection tour; maintenance inspection tour; school bus route
Min Graph Coloring	Color all nodes of a graph using as few colors as possible so that adjacent nodes have different colors.	Wi-Fi channel assignment; cellular frequency planning; radio channel allocation; event loudspeaker placement; exam timetabling; map region coloring; interference-aware equipment layout
Graph Traversal (BFS)	Perform breadth-first search from a given source node and output the visit order or layer structure.	Post-disaster inspection sweep; tourist exploration route; UAV area scanning; building safety inspection
Vertex Cover	Select a minimum set of nodes such that every edge has at least one endpoint in the set.	Server monitoring placement; road intersection cameras; railway security posts; social liaison selection; pipeline sensor placement; campus patrol posts

Table 8: Detailed descriptions of the six graph reasoning tasks included in the GRIT benchmark. This table complements the main results by providing the canonical graph-theoretic formulation and representative real-world problem scenarios for each task.

B.3 Stage 2: Reinforcement Learning (ARPO)

We employ our proposed **ARPO** (Agentic Reasoning Policy Optimization) algorithm, utilizing the GRPO estimator. The model is warm-started from the tool-use SFT checkpoint. The detailed hyperparameters are provided in Table 9.

Curriculum Warm-up Strategy. As mentioned in the experimental setup, we adopt a two-phase curriculum to ensure training stability:

1. **Phase 1 (Stabilization):** We first train on a subset of smaller graphs ($N \in [4, 14]$). This phase focuses on stabilizing the model's adherence to the structured output format and tool invocation syntax without the distraction of long-context complexity.
2. **Phase 2 (Main Optimization):** We then scale to the full training distribution ($N \in [10, 40]$). To balance diversity, we sample two distinct underlying graphs per size per task during this phase.

Extended Context and Rollouts. Graph reasoning trajectories involving tool interactions significantly increase sequence length. Consequently, we extend the maximum context window to **36,000 tokens** (24k for prompts + 12k for generation). We set the KL coefficient to 0.0, allowing the policy to explore the solution space freely, constrained only by the group-relative reward signal. The training uses a batch size of 16 with 8 rollouts per prompt ($G = 8$) to stabilize the baseline estimation.

Parameter	Value
Optimization & Data	
Actor LR	1×10^{-6}
Train Batch Size	16
Mini-batch Size	4
Epochs	2
KL Coefficient	0.0
Generation & Environment	
Rollout Engine	vLLM
Max Prompt Length	24,000 tokens
Max Response Length	12,000 tokens
Rollouts per Prompt (G)	8
Reward Function	Graph Correctness
Tool Integration	Sync with Tool
Search & Exploration	
Beam Size	2
Branch Probability	0.5
Entropy Weight	0.2
Initial Rollouts	2

Table 9: **Hyperparameters for RL Training (ARPO).** We employ the GRPO estimator with vLLM-based rollouts. Note the extended context window (36k total) to handle graph reasoning trajectories.

Task	Model	Node Naming Schemes (Accuracy %)				Var ($\times 10^{-4}$)
		Canonical	Random ID	Semantic	Mixed	
		Closed Source Models				
Synthetic Graphs	gpt-5-nano	83.33	66.67	75.00	70.00	52.50
	deepseek-3.2-reason	96.61	91.53	93.22	96.67	6.54
	Qwen-plus	80.39	82.00	82.35	84.31	2.59
	gemini-2.5-flash-lite	23.33	13.33	51.67	40.00	292.00
	kimi-k2-thinking	37.21	23.26	41.86	22.73	94.80
Real-world Queries	gpt-5-nano	71.67	76.67	68.33	65.00	24.80
	deepseek-3.2-reason	86.44	94.92	91.53	90.00	12.40
	Qwen-plus	72.55	76.47	84.31	74.51	26.60
	gemini-2.5-flash-lite	56.67	68.33	65.00	56.67	35.10
	kimi-k2-thinking	20.93	13.95	25.58	18.18	23.80
		Open Source Models				
Synthetic Graphs	Qwen3_4b_ins	61.67	51.67	58.33	60.00	19.20
	Qwen3_4b	60.00	40.00	51.67	51.67	67.60
	DeepSeek-R1-Distill-Qwen-7B	10.00	3.33	8.33	13.33	17.40
Real-world Queries	Qwen3_4b_ins	50.00	45.00	50.00	48.33	5.56
	Qwen3_4b	53.33	43.33	46.67	48.33	17.40
	DeepSeek-R1-Distill-Qwen-7B	10.00	8.33	3.33	8.33	8.34

Table 10: Performance comparison across different node naming schemes on Synthetic Graphs and Real-world Queries. Accuracy is reported in percentages, and variance (Var) is scaled by 10^{-4}.

C Qualitative Case Study

To provide a granular understanding of how **GRAIN** differs from existing paradigms, we present a qualitative comparison on a representative Shortest Path instance involving a real-world narrative with semantic noise (e.g., irrelevant details about "taxis being unavailable"). We analyze the reasoning trajectories of GRAIN (Table 14), a proprietary model using CoT (GPT-5-nano, Table 15), and a multi-agent framework (MA-GTS, Table 16).

GRAIN: Decoupling Parsing from Computation. As shown in Table 14, GRAIN adopts a "parse-then-solve" approach. The model does not attempt to calculate the path distance internally. Instead, it focuses entirely on **semantic parsing**: filtering out the narrative noise and extracting the topological structure into a rigorous intermediate representation (the JSON adjacency list). Crucially, GRAIN recognizes that pathfinding is a computational task, not a linguistic one. By explicitly invoking the Dijkstra tool with the correct parameters, it guarantees arithmetic correctness. This case demonstrates GRAIN's core advantage: it uses the LLM solely for its strength (unstructured text understanding) while offloading algorithmic complexity to the external environment.

GPT-5-nano (CoT): The Fragility of Internal Simulation. Table 15 illustrates the reliance of standard LLMs on **internal simulation**. The model attempts to mimic the state transitions of Dijkstra's algorithm step-by-step within its context window (e.g., calculating "Relax TB: $0 + 1 = 1$"). While successful in this small-scale example, this approach is inherently fragile. The model is forced to act as both a parser and an arithmetic engine. In real-world scenarios with larger graphs or floating-point weights, such "mental simulation" is prone to state tracking errors and calculation hallucinations, as the model lacks an external verifier for its intermediate arithmetic steps.

MA-GTS: Redundant Procedural Fragmentation. The Multi-Agent approach (Table 16) correctly identifies the structure but suffers from **procedural fragmentation**. The task is decomposed into granular roles—an *Info Extractor* finds entities, a *Graph Builder* creates edges, and a *Solver* executes code. While this reduces the cognitive load on any single agent, the case study reveals that for fundamental graph problems, such fragmentation is often unnecessary. The strict role boundaries require the explicit serialization and re-parsing of information between agents (e.g., passing entity lists from Agent 1 to Agent 2), introducing potential information loss at each handover. GRAIN demonstrates that a single, well-optimized agentic policy can internalize this pipeline, achieving the same structural accuracy without the fragmented decision process.

Summary of Paradigms. The comparison highlights a fundamental shift in the reasoning approach:

- **CoT (GPT-5)** treats graph reasoning as a *sequence prediction* problem, vulnerable to calculation errors.
- **Multi-Agent (MA-GTS)** treats it as an *organizational* problem, leading to rigid and fragmented workflows.
- **GRAIN** treats it as a *translation-and-execution* problem. By aligning natural language with symbolic intermediate representations, GRAIN achieves the robustness of tool-augmented systems while maintaining a streamlined, single-agent decision boundary.

D Empirical Study: Node-Naming Robustness on Synthetic and Real-World Graphs

Before detailing our method, we quantify the node-label sensitivity of LLMs using the Shortest Path task across varying naming schemes (Canonical, Random ID, Semantic, Mixed). As shown in Figure 2 and Table 10, three trends validate the need for robust grounding:

High Volatility. The substantial error bars across all models highlight severe instability. Mere surface-level changes in node identifiers or narrative framings induce drastic performance fluctuations, indicating that current LLMs lack consistent structural understanding.

Pattern Dependency in Open Models. Open-source models (Fig. 2, bottom) suffer a sharp performance drop on *Random IDs* compared to Semantic or Canonical names. This suggests a reliance on sequential text patterns or meaningful words rather than the underlying topology.

Semantic Overfitting in Closed Models. Surprisingly, on real-world graphs (Fig. 2, top-right), closed-source models perform slightly worse on *Canonical* names ($v_1 \dots v_n$) than on *Semantic* names. This implies an over-fitting to rich contexts, where stripping a problem down to abstract symbols paradoxically hinders performance.

Qualitative analysis confirms that these failures predominantly occur during the **grounding phase** (mapping text to graph representations), motivating GRAIN's focus on invariant decision-making.

E Zero-Shot Evaluation on the G-REAL Dataset

To strictly verify the fairness of our baseline comparisons and ensure that GRAIN's strong performance is not merely a result of overfitting to the specific distribution of our GRIT dataset, we conduct an additional zero-shot evaluation on **G-REAL**, the original dataset introduced alongside the MA-GTS multi-agent framework.

We randomly sampled 100 problems for each task from the unseen G-REAL dataset, strictly ensuring that the problems were uniformly distributed across various node sizes. Simultaneously, we also evaluated the MA-GTS framework on a subset of our GRIT Out-Of-Distribution (OOD) test set. Considering that the runtime of multi-agent systems grows exponentially when processing large-scale graph structures, we evaluated MA-GTS on a low-difficulty subset consisting of basic graphs with $10 \leq |V| \leq 20$. The comparison results are presented in Table 17.

Model / Framework	Evaluation Setup	Accuracy (%)
GRAIN-4B (Ours)	Zero-Shot on G-REAL (All sizes)	**86.60**
MA-GTS	Evaluated on GRIT OOD ($\|V\| \leq 20$)	82.79

Table 17: **Cross-Dataset Generalization Test.** GRAIN demonstrates powerful zero-shot extraction capabilities on the completely unknown G-REAL dataset, outperforming the multi-agent baseline evaluated on a simplified subset of our benchmark.

The results objectively confirm the fairness of the baseline comparisons from two independent dimensions. First, the single-agent GRAIN achieved a commanding 86.6% accuracy on the entirely unseen G-REAL dataset (with Graph Coloring at 98%, TSP at 80%, and Vertex Cover at 82%). This proves that its graph structure extraction capability is robust and universally applicable. Second, the MA-GTS framework achieved 82.79% on our OOD small-graph subset, which aligns with expectations for a strong baseline and confirms that we did not subjectively suppress its performance during evaluation.

F Comparative Failure Analysis: Single-Agent vs. Multi-Agent

To diagnose the transfer reliability of inference-only multi-agent decomposition on a compact model, we run the six-agent MA-GTS pipeline with Qwen3-4B-Instruct on the same GRIT Test and OOD inputs. Table 18 reports task-level accuracy,

Naming Scheme	Definition & Characteristics	Representative Text Realization (Example)
Canonical	Uses standard graph-theoretic indices (e.g., integers $0, 1$) or generic labels (v_i). Represents the sanitized format typical of academic benchmarks.	*"There is an edge connecting **Node 0** to **Node 1** with a weight of 7. Find the path from 0 to 1."*
Random IDs	Assigns high-entropy, non-sequential integers (e.g., from $[1, 10000]$). Tests robustness against tokenization fragmentation and lack of numerical continuity.	*"Server **8392** is connected to Server **104** with latency 7ms. Start routing from **8392**."*
Semantic	Uses meaningful, domain-specific entity names (e.g., locations, proteins). Introduces semantic priors that may distract the model from topological reasoning.	*"Walk from **Xundral View** to **Knights Market**. The distance is 7 km. Identify the route departing from **Xundral View**."*
Noisy / Mixed	Simulates "dirty" real-world data by mixing aliases, **meaningless strings (e.g., hashes, garbled text)**, and heterogeneous types. Tests entity resolution and resilience to surface-form noise.	*"Link: **'0x9A_err'** $\rightarrow$ **unknown_loc** (dist: 7). Note: **'0x9A_err'** is also referred to as **Start_Pt**. Avoid nodes marked as **##@$**."*

Table 11: **Examples of Node Naming Schemes in GRIT.** We illustrate how the same underlying edge (from a source node to a target node with weight 7) is realized textually across four different naming schemes. This highlights the spectrum from clean, sanitized inputs to noisy, unstructured real-world data.

tool-call outcomes, end-to-end pipeline health, and accuracy conditional on a successful native tool call.

Component	Content / Instruction
Role Definition	You are a helpful assistant that specializes in solving graph theory problems using a **graph theory tool**. Given a question, you need to first think about the reasoning process in your mind and then provide the answer. During thinking, you can invoke the graph theory tool to compute shortest paths, minimum spanning trees, topological sorting, or other graph algorithms if needed.
Protocol & Tags	The reasoning process and answer must be enclosed within specific XML tags. Follow these guidelines strictly: 1. **Thinking:** Start with `<think>` and explain your step-by-step reasoning process. End with `</think>`. 2. **Tool Call:** When you need to use the graph theory tool, output `<graph>` followed by a JSON-formatted query, then `</graph>`. 3. **Tool Result:** After the tool call, output `<result>` followed by the tool's output, then `</result>`. 4. **Final Answer:** Finally, output `<answer>` followed by the final answer, then the final answer is `\boxed{answer here}</answer>`.

Tool Schema	The JSON query inside `<graph>` tags must follow this exact structure:
	`{`
	`  "problem_type": "problem_type_here",`
	`  "graph": {`
	`  "type": "adjacency_list",`
	`  "data": {`
	`  "node1": {`
	`  "neighbor1": weight1,`
	`  "neighbor2": weight2`
	`  },`
	`  "node2": {`
	`  "neighbor3": weight3,`
	`  "neighbor4": weight4`
	`  }`
	`  },`
	`  "directed": true_or_false,`
	`  "weighted": true_or_false`
	`  },`
	`  "parameters": {`
	`  "parameter1": "parameter1_value",`
	`  "parameter2": "parameter2_value",`
	`  "algorithm": "algorithm_name"`
	`  }`
	`}`
	Ensure that `problem_type`, `graph structure`, and `algorithm` parameters are correctly populated based on the user query.

Table 12: **System Prompt for GRAIN (SFT & RL).** This prompt instructs the model to act as a graph reasoning agent, defining the explicit "Think-Tool-Result" loop and the strict JSON schema required for interacting with the deterministic graph solver.

Table 13: **Summary of Notations.** Key symbols used in GRAIN. We update the reward notations to reflect the gated formulation and structure similarity score.

Symbol	Description	Symbol	Description
Problem Formulation & Environment		***Rewards & Optimization***	
$G = (V, E, w)$	Ground-truth weighted graph.	$r_{\text{ans}}(\tau)$	Answer correctness reward.
$\mathcal{T}$	Graph reasoning task (e.g., TSP).	$s_{\text{inv}}(\tau)$	Structure similarity score ($\hat{G}$ vs. G).
$\nu : V \to \Sigma^*$	Node naming function.	$R(\tau)$	Gated total return (Eq. 3).
ϕ	Query formulation style.	ρ_{err}	Penalty constant for format violation.
x	Input query $x = f(G, \nu, \phi, \mathcal{T})$.	λ_{inv}	Coef. for structure similarity score.
$\mathcal{G}$	Distribution of training graphs.	β	Coef. for KL divergence penalty.
$\mathcal{P}(G)$	Naming/style distribution.	$J(\theta)$	Total RLVR objective function.
$\hat{G}$	Reconstructed candidate graph.		
$\hat{z}$	Tool execution result.		
Agent & Trajectory			
π_θ	Policy (LLM) parameterized by θ.		
π_{ref}	Reference policy (SFT model).		
τ	Generated trajectory sequence.		
y	Structured output with tags.		

Stage	Content
User Query	My younger sibling isn't feeling well, and the nearest hospital is at **Zaphrel Summit**. Right now we're at home, near an intersection I'll call **Xundral View**. [...] I checked a city map and noted down the walking distances: Knights Market to Xundral View is 7 km... *[narrative omitted for brevity]* ... Dragons Gate to Trelvon Bay is 7 km. **Starting from Xundral View, what route should we walk to reach Zaphrel Summit while minimizing the total walking distance?**
Reasoning	<think> From an application perspective, this requires graph theory tool calculation. **Graph parameters identified:** • **Nodes:** 9 distinct locations (e.g., Knights Market, Xundral View). • **Goal:** Compute shortest path from *Xundral View* to *Zaphrel Summit*. • **Edge Extraction:** Parsing narrative to build adjacency list. Tool call explanation: I will input graph information in JSON format, including nodes, edges, start and end points. </think>

Action	<graph>
	{
	"problem_type": "shortest_path",
	"graph": {
	"type": "adjacency_list",
	"data": {
	"Knights Market": {
	"Xundral View": 7,
	"Skyline Gardens": 1,
	"Dragons Gate": 3,
	"Trelvon Bay": 6
	},
	"Xundral View": { "Trelvon Bay": 1 },
	"Draycon Arcade": {
	"Copper Clock Square": 2,
	"Vorlen Heights": 8
	},
	"Copper Clock Square": {
	"Trelvon Bay": 7,
	"Vorlen Heights": 5,
	"Zaphrel Summit": 8
	},
	"Granite Plaza": { "Trelvon Bay": 8 },
	"Dragons Gate": { "Trelvon Bay": 7 },
	"Trelvon Bay": {},
	"Zaphrel Summit": {}
	},
	"directed": false,
	"weighted": true
	},
	"parameters": {
	"start": "Xundral View",
	"end": "Zaphrel Summit",
	"algorithm": "dijkstra"
	}
	}
	</graph>

Observation	<result> Shortest path algorithm (Dijkstra) result: • **Path:** Xundral View → Trelvon Bay → Copper Clock Square → Zaphrel Summit • **Total Length:** 16 </result>
Final Response	<think> Tool call successful. Calculation result: shortest path contains 3 nodes... total weight 16. </think> **Answer:** $\boxed{16}$

Table 14: **Case Study on Shortest Path Task.** The model parses a noisy narrative, constructs a structured JSON representation, and invokes the tool to find the optimal path. The code block demonstrates the explicit structural constraints imposed by GRAIN.

Stage	Content
User Query	My younger sibling isn't feeling well, and the nearest hospital is at **Zaphrel Summit**. Right now we're at home, near an intersection I'll call **Xundral View**. [...] *(Narrative containing distances: Knights Market to Xundral View is 7 km, etc.)* ... **Starting from Xundral View, what route should we walk to reach Zaphrel Summit while minimizing the total walking distance?**
Model Response	<think> **Task restatement:** Starting from Xundral View (XV), find a walking route to Zaphrel Summit (ZS) minimizing total distance... The graph is undirected and unweighted in terms of edges with weights equal to distances. **1. Internal Graph Construction:** • Vertex set V = {XV, KM, SG, DA, GP, DG, TB, CCS, VH, ZS}. • **Edge set extraction:** – (XV, TB) = 1, (KM, XV) = 7, (KM, SG) = 1, (KM, DA) = 2 – (TB, CCS) = 7, (TB, GP) = 8, (CCS, ZS) = 8 ... *[full list extracted]* **2. Algorithm Simulation (Dijkstra):** Method: Run Dijkstra's algorithm from XV, tracking tentative distances. • *Initialize:* dist(XV) = 0; all others = ∞. • *Step 1:* From XV, relax TB with dist $0 + 1 = 1$. $\rightarrow$ **dist(TB) = 1**. • *Step 2 (Next smallest TB):* – Relax KM: $1 + 6 = 7 \rightarrow$ dist(KM) = 7 – Relax CCS: $1 + 7 = 8 \rightarrow$ dist(CCS) = 8 – ... *[relaxing other neighbors]* • *Step 3 (Next smallest KM):* Relax SG via KM: $7 + 1 = 8 \rightarrow$ dist(SG) = 8... • *[...omitting intermediate relaxation steps for brevity...]* • *Final Step:* ZS reached with dist(ZS) = 16. No other path reduces this. **Conclusion:** The minimal walking distance is 16. </think> **Answer:** 16

Table 15: **Baseline Case Study: GPT-5-nano (Chain-of-Thought).** Unlike GRAIN which offloads computation to external tools, the baseline model attempts to **internally simulate** the execution of Dijkstra's algorithm via text generation. While successful in this instance, this approach relies heavily on the model's ability to maintain long-context state and perform precise arithmetic without external verification, which typically degrades on more complex graphs.

Agent Role	Action & Output Summary	Cost (Time/Tok)
User Query	I'm a university student living near **Lighthouse Point**... final exam at **Umbra Theater**... Yondel Crossing to Lighthouse Point is 8 km... *[...narrative omitted...]* ... **What path should I take so that the total walking distance is as short as possible?**	
Info Extractor	<action> Extract entities and definitions. <response> { "entities": ["Lighthouse Point", "Umbra Theater", "Yondel Crossing", ...], "definitions": { "shortest path": "minimize distance..." } }	2.2s620 tok
Graph Builder	<action> Construct topological structure. <response> { "nodes": [...], "edges": [("Yondel Crossing", "Lighthouse Point", {"weight": 8}), ("Brelix Strand", "Cralden Tunnel", {"weight": 9}), ...], "graph_type": "weighted undirected" }	6.2s1030 tok
Theory Agent	<action> Select algorithm and analyze complexity. <response> { "problem": "Shortest Path", "algorithm": "Dijkstra", "reasoning": "Graph size < 18 nodes, Dijkstra is optimal O((n+m)log n)..." }	12.3s3111 tok
Struct Agent	<action> Map nodes to indices and format adjacency list. <response> { "adjacency_list": { 0: [], 1: [(0,8), ...], ... }, "node_mapping": { "Lighthouse Point": 0, ... } }	18.3s801 tok
Solver Agent	<action> Execute algorithm and interpret result. <response> { "Path": [0, 1, 3, 7], "Total Distance": 15 } **Final Answer:** The shortest path is Lighthouse Point $\rightarrow$ Yondel Crossing $\rightarrow$ Cralden Tunnel $\rightarrow$ Umbra Theater. distance = 15.	23.5s1423 tok
Total Overhead		**Total Time: ~23.5s | Total Tokens: 7487**

Table 16: **Baseline Case Study: MA-GTS (Multi-Agent System).** The task is decomposed into five distinct agents. While accurate, the multi-round interaction incurs significant latency (23.5s) and token consumption (7487 tokens), illustrating the efficiency bottleneck of multi-agent frameworks compared to GRAIN's single-agent pipeline.

(A) MA-GTS Qwen3-4B-Instruct on GRIT Test

Task	#	Acc.	Native Call	Text-only Call	Alg. Error
Shortest Path	160	13.75	48.13	4.38	47.50
TSP	160	1.88	1.25	36.88	35.00
Graph Coloring	160	12.50	0.63	25.00	55.63
Vertex Cover	120	4.17	5.00	39.17	45.83
BFS / Traversal	160	0.62	2.50	51.25	45.00
Centrality	160	7.50	26.88	58.13	15.00
Micro Avg.	**920**	**6.85**	**14.46**	**35.65**	**40.43**

(B) OOD Overall

Split	#	Acc.	Native Call	Text-only Call	Alg. Error
OOD Overall	960	11.35	32.40	23.44	43.75

(C) End-to-End Pipeline Health

Split	#	Any-Agent Error	Final Native Call	Text-only Call	Terminal Failure	Avg. Time	p90 Time	Avg. Tokens
Test	920	43.04	14.46	35.65	76.09	98.46s	150.07s	25.15k
OOD	960	43.75	32.40	23.44	67.19	109.61s	165.36s	28.13k

(D) Accuracy Conditional on a Successful Native Tool Call

Split / Task	Native Calls	Correct Tool Results	Conditional Acc.
Test / Shortest Path	77	42	54.55
Test / Centrality	43	32	74.42
OOD / Shortest Path	199	147	73.87
OOD / Centrality	112	90	80.36

Table 18: **MA-GTS Transfer Diagnostic with Qwen3-4B-Instruct.** Results on the same GRIT Test/OOD inputs. The panels report task accuracy, tool-call outcomes, pipeline health, and accuracy conditional on an executed native tool call. All rates and accuracies are percentages. This diagnostic evaluates end-to-end transfer reliability rather than an isolated agent-count effect.

Qwen3-4B-Instruct is not completely unable to solve graph tasks: when MA-GTS produces and executes a native tool call, the selected subset can be correct. The main observed weakness is end-to-end reliability. On Test, final native-call coverage is only 14.46% and terminal failures reach 76.09%; on OOD, the corresponding values are 32.40% and 67.19%. Text-only tool calls and Algorithm-Agent errors therefore often prevent a complete executable graph-solving trajectory.

On the same 960 OOD inputs, MA-GTS Qwen3-4B-Instruct obtains 11.35% overall accuracy, whereas GRAIN obtains 81.25%. Because training, backbone initialization, and agent organization differ, this result is a transfer diagnostic rather than an isolated single- versus multi-agent comparison. It shows that inference-only multi-agent decomposition does not automatically repair entity grounding, graph-state preservation, and native tool-call reliability in this compact-model setting.

G Intermediate Structure Error Decomposition

Table 19 decomposes GRAIN's exact structure recovery outcomes on the 2,760-example GRIT Test set.

Structural Outcome	# Examples	Percent
Exact graph recovered	**2,685**	**97.28**
Graph parse / format failure	33	1.20
Node-set mismatch	23	0.83
Extra edges only	9	0.33
Missing edges only	7	0.25
Edge-weight mismatch only	2	0.07
Mixed edge errors	1	0.04

Table 19: **Intermediate Structure Error Decomposition.** Counts and percentages over all 2,760 GRIT Test examples.

Exact graph recovery is deliberately stricter than edge-set overlap: any graph parse or format failure, node-set mismatch, extra or missing edge, or edge-weight mismatch counts as non-exact. The decomposition therefore exposes failure types that high Jaccard or Edge-F1 scores can mask, while the six-task final-answer evaluation independently checks whether the recovered representation supports computation. These 2,760 examples contain connected undirected graphs; we do not claim directed-graph robustness or a separate parameter-level error analysis.

H Graph Complexity Statistics and Structure Recovery

We further characterize the 2,760 connected, undirected graphs in the GRIT Test set using graph size, density, average degree, and diameter. Density is defined as $2|E|/(|V|(|V|-1))$, and diameter is the unweighted shortest-path diameter of the gold graph.

(A) Graph Size Buckets

Size	#	$\lvert V\rvert$	$\lvert E\rvert$	Dens.	Deg.	Diam.	Exact
10–15	536	12.6	34.3	0.457	5.33	2.98	99.63
16–20	460	18.0	71.9	0.467	7.93	2.67	99.35
21–30	920	25.5	146.3	0.463	11.33	2.59	98.15
31–40	844	35.1	284.6	0.470	16.08	2.46	93.72

(B) Graph Density Tertiles

Density	#	Avg.	Parse	Edge-F1	Exact
Low (< 0.314)	912	0.310	96.71	96.69	95.07
Mid (0.314–0.415)	920	0.359	99.67	99.39	98.15
High (≥ 0.415)	928	0.722	100.00	99.91	98.60

Table 20: **Graph Complexity and Structure Recovery on GRIT.** Panel A groups the 2,760 Test examples by node count; Panel B groups them by density. Edge-F1 denotes weighted edge recovery.

Exact graph recovery decreases from 99.63% for graphs with 10–15 nodes to 93.72% for graphs with 31–40 nodes, while remaining at least 95.07% across density tertiles. Within GRIT, degradation is therefore associated more strongly with graph size than with density. The dense Erdős–Rényi construction also produces small diameters, so these results should not be interpreted as covering the full range of graph topological complexity.