Structural Chain-of-Thought: Reasoning with Executable Computational Graphs

Anonymous ACL submission

Abstract

Chain-of-thought prompting enables reasoning in large language models but produces explanations that are difficult to verify, expensive to generate, and not always faithful. We introduce Structural Chain-of-Thought framework where models generate executable computational graphs instead of free-form reasoning. The final answer is obtained by deterministically executing the graph. The model is trained using graph-based reinforcement learning that rewards structural validity, answer correctness, and node overlap with reference graphs.

Experiments across multiple model families show that our method improves accuracy over supervised graph training, while reducing token reasoning length by up to 12×. Intervention-based faithfulness experiments reveal that models trained with our approach derive their answers from the generated graphs rather than producing post-hoc rationalizations.

1 Introduction

Large language models have demonstrated strong reasoning abilities across a wide range of tasks by generating intermediate reasoning steps before producing a final answer (Wei et al., 2022; Wang et al., 2023a; Kojima et al., 2022). However, these reasoning traces can create the impression that the model arrived at its answer through the displayed reasoning process, but recent works have raised concerns about the faithfulness of such traces, showing that generated explanations do not always reflect the model's true internal decision process (Lanham et al., 2023; Turpin et al., 2023; Tutek et al., 2025; Han et al., 2026).

A likely contributing factor is the training objective itself: modern reasoning models are typically trained using rewards based solely on answer correctness and output format, without imposing any constraints on the structure or faithfulness of the reasoning process (Guo et al., 2025; Yang et al., 2025; Grattafiori et al., 2024). As a result, models generate reasoning in free-form text, making the resulting traces difficult to verify, analyze, or systematically inspect (Shojaee et al., 2025; Xiong et al., 2025; Baker et al., 2025).

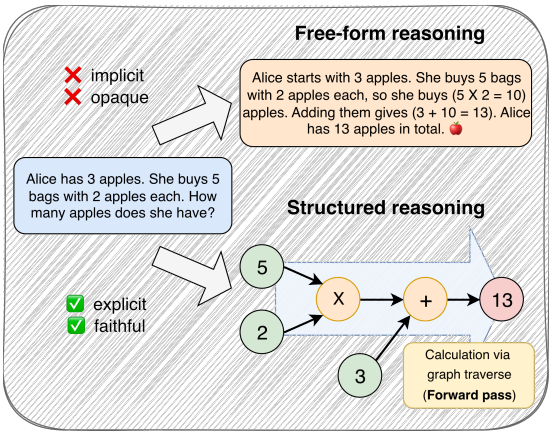

Figure 1: Comparison between free-form reasoning and computational graph representations for solving a natural-language arithmetic problem.

In this work, we address this limitation by constraining the reasoning process to follow a structured, executable format. Rather than generating free-form text, we train models to produce *computational graphs* — directed acyclic graphs in which each node represents an intermediate computation and edges encode dependencies between steps as presented in Figure 1. This representation is particularly natural for *arithmetic reasoning problems* (Cobbe et al., 2021), where solutions can be decomposed into a sequence of well-defined operations with explicit dependencies. By representing such solutions as graphs, each reasoning step becomes explicit and mechanically verifiable: the final answer is obtained by executing the graph rather than by extracting it from unstructured text. To train models to produce these graphs, we extend Group Relative Policy Optimization (Shao et al., 2024) with reward signals that evaluate not only the cor-

rectness of the final answer but also the structural validity of the generated graph. We refer to the resulting training procedure as *Graph-Based GRPO*. Our contributions are summarized as follows:

- We propose *Graph-Based GRPO*, a reinforcement learning method that optimizes structured reasoning by training models to generate executable computational graphs.
- Using arithmetic reasoning tasks, we show that reinforcement learning over graph-structured reasoning improves accuracy across multiple models and datasets while reducing reasoning length by up to $12\times$.
- We show that graph-based reasoning yields substantially higher faithfulness, indicating that model predictions are derived from the generated computational graphs rather than from implicit internal shortcuts.

Overall, our results suggest that computational graphs provide a promising intermediate representation for training reasoning-capable language models. By explicitly structuring the reasoning process, we obtain models that are more efficient, more faithful, and easier to analyze than those relying on free-form reasoning traces.

2 Related Work

Structured and Graph-Based Reasoning Representations. Chain-of-Thought prompting (Wei et al., 2022) improves performance on multi-step reasoning tasks by eliciting intermediate natural-language steps. However, such free-form traces are difficult to verify and may include steps not causally linked to the final answer. To obtain more reliable reasoning, several methods constrain the output to be an executable program. Program-of-Thought (Chen et al., 2022) and Program-aided language models (Gao et al., 2023) generate code that is run by an external interpreter, thereby separating logical reasoning from numerical computation. While effective, these representations remain sequential and do not make inter-step dependencies explicit. More recently, Hu and Wu (2025) proposed RL-Struct, which introduces a multi-component reward function that checks syntax, format, and semantic validity of outputs while optimizing models using GRPO. Similarly, (Lu et al., 2025) applied reinforcement learning to optimize structured outputs, like JSON, using reward signals that enforce structural validity. In contrast, our work focuses on improving reasoning performance and faithfulness rather than primarily optimizing output format validity. Tree-of-Thoughts (Yao et al., 2023) and Graph-of-Thoughts (Besta et al., 2024) arrange intermediate steps into branching or graph-like layouts, enabling exploration of alternative reasoning paths. However, these methods operate at inference time through structured prompting and search, and do not train the model to internalize structured generation.

Learning to Produce Faithful Reasoning. Faithfulness—the extent to which explanations reflect a model's true reasoning— is a key requirement for interpretable NLP systems (Jacovi and Goldberg, 2020; Holzinger et al., 2017). Recent work shows that reasoning traces in large language models may be post-hoc rationalizations rather than causal steps underlying their predictions (Lanham et al., 2023; Paul et al., 2024; Matton et al., 2025). To address this issue, several approaches introduce structured or verifiable reasoning formats. (Creswell and Shanahan, 2022) adopted an in-context learning approach, where language models first select relevant statements from context and then perform inference over them in a separate context window. Lyu et al. (2023) propose Faithful Chain-of-Thought, which translates natural language problems into symbolic reasoning programs that are executed by a deterministic solver, while Cobbe et al. (2021) train verifier models to rank candidate solutions based on the correctness of their intermediate reasoning. Other work explores reinforcement learning to encourage faithful reasoning: for instance, FRODO (Paul et al., 2024) introduces reward signals that favor reasoning chains aligned with the final prediction.

Our work builds on these directions by enforcing an executable representation of reasoning. Instead of generating linear textual explanations, we train models to produce *computational graphs*, where nodes represent intermediate operations and edges encode dependencies between them. The final answer is obtained by executing the graph, ensuring that predictions directly follow from the generated reasoning structure. Combined with reinforcement learning via *Graph-Based GRPO*, this approach encourages compact and mechanically verifiable reasoning traces, improving the faithfulness of model explanations.

Dataset	Train Size	Test Size	# Input Nodes	Avg. # Operations
GSM8K	5,656	1,415	2–4	3–7
iGSM	2,634	658	9–23	5–18
MATH	7,959	1,992	3–5	3–8

Table 1: Datasets used in our experiments. We report dataset sizes and statistics of the corresponding computational graphs.

3 Overview

This section describes our reinforcement learning approach for training models to generate graph-based reasoning. Section 3.1 outlines the data preparation pipeline, Section 3.2 presents the proposed method, and Section 3.4 describes the baselines used for comparison.

3.1 Data and Training Format

Problem: Karl has 25 apples. Ryan has 20 more than Karl. Ben has 10 fewer than Ryan. What is the total of apples?

Computational Graph

```
[
 {"id":"karl","type":"const","value":25},
 {"id":"20","type":"const","value":20},
 {"id":"10","type":"const","value":10},
 {"id":"ryan","type":"op","op":"add","inputs
     ":["karl","20"]},
 {"id":"ben","type":"op","op":"sub","inputs
     ":["ryan","10"]},
 {"id":"total","type":"op","op":"add","inputs
     ":["karl","ryan","ben"]}
]
```

Simplified Graph

```
[
 "karl=25",
 "20=20",
 "10=10",
 "ryan=karl+20",
 "ben=ryan-10",
 "total=karl+ryan+ben"
]
```

Figure 2: Short word problem mapped to a computational graph and its simplified representation, which omits node identifiers and explicit input fields while preserving the computation structure.

Source Datasets. We evaluate on three math reasoning benchmarks of increasing difficulty. **GSM8K** (Cobbe et al., 2021) contains ∼8.5K grade-school word problems for multi-step arithmetic reasoning. **MATH** (Hendrycks et al., 2021) includes ∼11K high-school competition problems covering diverse topics and difficulty levels. **iGSM** (Zeyuan Allen-Zhu, 2023) is a synthetic dataset of ∼6K procedurally generated problems where difficulty depends on the size of the underlying reasoning graph and many objects and relations are redundant. Key dataset statistics are reported in Table 1.

Graph Construction and Verification. We synthesize graph representations from the original (question, solution) pairs in each dataset. Each graph is a directed acyclic graph where nodes represent constants or arithmetic operations and edges encode computational dependencies. Graphs are first generated in a detailed format with explicit node types and identifiers, then converted into the simplified representation in Figure 2, which preserves the computation while removing redundant metadata. Each graph is verified by executing it in topological order; if the result does not match the ground-truth answer, the graph is regenerated. After several generation–verification iterations, about 85% of examples produce valid graphs. Details are provided in Appendix B and C.

Sample Representation. For each problem we construct a training sample (x, g, y^*), where x is the natural-language problem statement, g is a *computational graph* encoding the reasoning process, and y^* is the ground-truth numerical answer. g can be also represented in a *simplified* textual format—an ordered list of assignment expressions (e.g., `ryan=karl+20`)—that preserves the full computational structure while omitting explicit node identifiers and type annotations (Figure 2, bottom). Unless otherwise stated, all experiments use this simplified format; an ablation comparing the two representations is provided in Section 6.

Training Format. During training and inference, the model receives x as input and is expected to produce a response of the form:

`<think>` g `</think> <answer>` y `</answer>`

where g is a simplified computational graph enclosed in thinking tags and y is the final numerical

answer.

Evaluation. For all methods that produce computational graphs—the final answer is obtained by executing a forward pass over the generated graph g in topological order, yielding a predicted value $\hat{y}$. For methods that produce free-form text, $\hat{y}$ is extracted as the last number in the generated response. In all cases, we compare $\hat{y}$ to the ground-truth answer y using exact match.

Data Splits. Each dataset is split into 80% training and 20% test data. For graph-based training, the training set is further split equally between supervised fine-tuning (SFT) and reinforcement learning (RL).

3.2 Method

Our method trains language models to produce structured reasoning in the form of computational graphs through a two-stage procedure: SFT on graph-annotated data, followed by reinforcement learning using GRPO with graph-specific reward signals.

Stage 1: Supervised Fine-Tuning. Given the training set of tuples (x, g, y) defined in Section 3.1, we first fine-tune the base model to generate computational graphs in the `<think>` g `</think> <answer>` y `</answer>` format. This stage exposes the model to the structured reasoning format and provides a strong initialization for the subsequent reinforcement learning stage.

Stage 2: Graph-Based GRPO. Starting from the SFT checkpoint, we further optimize the model using GRPO. For each problem x, the current policy π_θ samples a group of K candidate responses $\{(\hat{g}_1, \hat{y}_1), \ldots, (\hat{g}_K, \hat{y}_K)\}$. Each candidate is scored by a composite reward function $R(\hat{g}_i, \hat{y}_i, y)$ that evaluates answer correctness, structural graph validity, and output format (Section 3.3).

The policy is then updated to increase the probability of candidates with positive advantage and decrease the probability of those with negative advantage. Unlike standard GRPO applied to free-form reasoning, our formulation operates over structured outputs, and the reward function is designed to assess not only whether the model arrives at the correct answer but also whether the generated graph is well-formed and executable.

3.3 Reward Function Design

For the graph-based GRPO training, we design a two-component reward function that provides progressive feedback: a *Style Reward* that verifies the formal validity of the generated computational graph, a *Correctness Reward* that evaluates the numerical accuracy of the answer obtained by executing the graph, and a *Overlap Reward* that aligns the generated graph's node values to the gold graph's node values. The total reward for a candidate output is the sum of all components: $S = S_c + S_s + S_o$.

Correctness Reward. To evaluate numerical accuracy, we execute the generated computational graph and compare the resulting value to the ground-truth answer. We use a graded reward scheme: exact matches receive the highest reward, approximate matches within a tolerance receive partial credit, and incorrect or non-executable graphs are penalized. This encourages the model to progressively converge toward correct answers.

$$S_c = \begin{cases} 3.0, & \text{if } |y_{\text{pred}} - y_{\text{true}}| < \epsilon \\ 1.0, & \text{if } 0.9 \leq y_{\text{pred}}/y_{\text{true}} \leq 1.1 \\ 0.5, & \text{if } 0.8 \leq y_{\text{pred}}/y_{\text{true}} \leq 1.2 \\ -0.5, & \text{if graph executes incorrectly} \\ -1.5, & \text{if graph cannot be executed} \\ -2.0, & \text{if no valid graph is found} \end{cases} \tag{1}$$

where y_{pred} is the value obtained by executing the generated graph, y_{true} is the ground-truth answer, and $\epsilon = 10^{-6}$.

Style Reward. We require the model to produce a valid computational graph as a JSON list of node dictionaries followed by an `ANSWER: <number>` suffix. Each node must contain an `id` and a `type` field. Nodes of type `const` must additionally include a `value` field, while nodes of type `op` must include an `op` field and an `inputs` field (a list of references to other node identifiers). If any of these structural constraints is violated—e.g. the output is not a valid JSON list, contains non-dictionary elements, or includes nodes with missing or unrecognized fields—the candidate receives a negative reward.

$$S_s = \begin{cases} 2.0, & \text{if the graph is structurally valid} \\ -2.0, & \text{otherwise} \end{cases} \tag{2}$$

Such reward ensures that the model receives informative gradients at every stage of learning: the

style reward drives the model toward producing well-formed graphs and the correctness reward can meaningfully evaluate their computational content.

Overlap reward To encourage correct graph structures, we compute the overlap between values of gold and generated nodes, considering both constant and operation nodes. The reward is defined as the sum of these overlaps. If the generated graph cannot be parsed (e.g., invalid JSON or structure), it receives a negative reward.

$$S_o = \begin{cases} |G_c \cap \hat{G}_c| + |G_o \cap \hat{G}_o|, \text{if graph is valid} \\ -1.5, \text{if node set is empty} \\ -2.0, \text{if graph is invalid} \end{cases} \tag{3}$$

where G_c and G_o denote the sets of values of gold constant and operation nodes, respectively, and $\hat{G}_c$ and $\hat{G}_o$ denote the corresponding sets for the generated nodes. This reward encourages the model to capture the logical structure of the entire solution. Instead of relying solely on the final answer, which may be obtained through shortcut strategies, we evaluate the intermediate computational steps of the generated solution.

3.4 Baselines

We compare our method (Graph SFT + Graph GRPO, Section 3.2) against five baselines spanning three training paradigms: in-context learning (ICL), supervised fine-tuning (SFT), and reinforcement learning (RL). All fine-tuned models use Low-Rank Adaptation (LoRA; Hu et al., 2022) for parameter-efficient training. Hyperparameter details are provided in Appendix A.

In-Context Learning. *ICL with Few-Shot.* As a zero-parameter-update baseline, we prompt the base model with two demonstration examples, each formatted as a (x, g, y) tuple using the simplified computational graph representation described in Section 3.1. At inference time, the model generates a graph $\hat{g}$, which is executed to obtain the predicted answer $\hat{y}$.

Supervised Fine-Tuning. We consider three SFT variants that differ in the supervision signal provided during training:

Answer-Only SFT. The model is trained on (x, y) pairs and generates a single number $\hat{y}$ at inference time, without any intermediate reasoning.

CoT + Answer SFT. The model is trained on (x, c, y) triples, where c is a free-form chain-of-thought reasoning trace obtained from the original dataset. At inference time, the predicted answer $\hat{y}$ is extracted as the last number in the generated response.

Graph SFT. The model is trained on (x, g, y) tuples using the format described in Section 3.1. This variant uses the same training data and format as our proposed method but does not include the subsequent GRPO stage, isolating the contribution of supervised graph-based training alone.

Reinforcement Learning. We consider two RL baselines, both trained using GRPO:

Default GRPO. The model is trained to produce free-form natural-language reasoning in `<think>...</think>` tags and a final answer in `<answer>...</answer>` tags. Rewards are based on answer correctness and format compliance. This baseline represents the standard RL training setup used in recent reasoning models.

Structured Default GRPO. This baseline evaluates whether a model trained with Default GRPO on free-form reasoning can generalize to graph-based generation without explicit graph supervision. At inference time, the Default GRPO checkpoint is prompted to produce a computational graph $\hat{g}$; the predicted answer $\hat{y}$ is obtained by executing the generated graph.

4 Results

Table 2 presents the main performance comparison across datasets and training strategies. Overall, structured reasoning with computational graphs provides a competitive alternative to chain-of-thought supervision. While Graph SFT alone does not consistently outperform CoT-based training across all datasets, it produces a structured intermediate representation that can be effectively optimized through reinforcement learning, as we analyze below.

Computational graphs offer a structured alternative to chain-of-thought reasoning. On average, Graph-Based GRPO yields accuracy gains of +0.03 on GSM8K, +0.07 on iGSM, and +0.01 on MATH, with improvements observed in 10 of 12 model–dataset combinations. The largest gains occur on iGSM, a compositional benchmark requiring multi-step reasoning over procedurally generated structures: Llama3.2-3B improves by +0.16,

Dataset	Method	Gemma2-2B	LLama3.2-3B	Falcon3-3B	Qwen3-4B	Average
GSM8k	ICL with FS	0.02 ± 0.0	0.49 ± 0.0	0.68 ± 0.0	$\mathit{0.83} \pm 0.0$	0.50 ± 0.30
	Answer-only SFT	0.05 ± 0.05	0.16 ± 0.0	0.19 ± 0.0	0.40 ± 0.0	0.20 ± 0.13
	CoT + Answer SFT	0.41 ± 0.01	$\mathit{0.72} \pm 0.0$	$\mathit{0.72} \pm 0.0$	0.87 ± 0.0	$\mathit{0.68} \pm 0.17$
	Graph SFT	0.41 ± 0.01	0.50 ± 0.01	0.60 ± 0.0	0.76 ± 0.01	0.57 ± 0.13
	Default GRPO	$\mathbf{0.70} \pm 0.01$	$\mathbf{0.85} \pm 0.01$	$\mathbf{0.84} \pm 0.03$	$\mathbf{0.93} \pm 0.0$	$\mathbf{0.83} \pm 0.08$
	Structured Default GRPO	0.23 ± 0.01	0.48 ± 0.00	0.58 ± 0.00	0.89 ± 0.00	0.54 ± 0.23
	Graph-Based GRPO	$\mathit{0.42} \pm 0.01$	0.56 ± 0.01	0.64 ± 0.01	0.80 ± 0.0	0.60 ± 0.13
iGSM	ICL with FS	0.02 ± 0.0	0.05 ± 0.0	0.07 ± 0.0	0.14 ± 0.0	0.07 ± 0.04
	Answer-only SFT	$\mathit{0.39} \pm 0.01$	0.45 ± 0.01	0.44 ± 0.01	0.49 ± 0.01	0.44 ± 0.03
	CoT + Answer SFT	0.21 ± 0.01	$\mathit{0.56} \pm 0.01$	$\mathit{0.55} \pm 0.01$	$\mathit{0.65} \pm 0.0$	$\mathit{0.49} \pm 0.16$
	Graph SFT	$\mathbf{0.49} \pm 0.0$	0.44 ± 0.01	0.44 ± 0.01	0.55 ± 0.0	0.48 ± 0.05
	Default GRPO	0.28 ± 0.0	0.00 ± 0.0	0.26 ± 0.01	0.28 ± 0.05	0.20 ± 0.12
	Structured Default GRPO	0.09 ± 0.02	0.07 ± 0.00	0.09 ± 0.01	0.21 ± 0.02	0.11 ± 0.05
	Graph-Based GRPO	0.40 ± 0.02	$\mathbf{0.60} \pm 0.0$	$\mathit{0.55} \pm 0.01$	$\mathbf{0.67} \pm 0.0$	$\mathbf{0.55} \pm 0.10$
MATH	ICL with FS	0.03 ± 0.0	0.08 ± 0.0	0.12 ± 0.0	0.20 ± 0.0	0.11 ± 0.06
	Answer-only SFT	0.04 ± 0.0	0.32 ± 0.01	$\mathit{0.23} \pm 0.0$	$\mathit{0.43} \pm 0.0$	0.25 ± 0.14
	CoT + Answer SFT	$\mathit{0.10} \pm 0.01$	$\mathit{0.51} \pm 0.01$	$\mathbf{0.40} \pm 0.0$	$\mathbf{0.59} \pm 0.0$	$\mathit{0.40} \pm 0.19$
	Graph W Answer SFT	0.09 ± 0.0	0.12 ± 0.0	0.13 ± 0.0	0.24 ± 0.01	0.15 ± 0.06
	Default GRO	$\mathbf{0.33} \pm 0.01$	$\mathbf{0.60} \pm 0.01$	0.16 ± 0.13	0.55 ± 0.0	$\mathbf{0.41} \pm 0.18$
	Structured Default GRPO	0.05 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.26 ± 0.00	0.13 ± 0.08
	Graph-Based GRPO	0.04 ± 0.04	0.16 ± 0.0	0.17 ± 0.0	0.28 ± 0.0	0.16 ± 0.08

Table 2: Performance with standard deviations across datasets and training methods. Best results are shown in **bold** and second-best in *italic*.

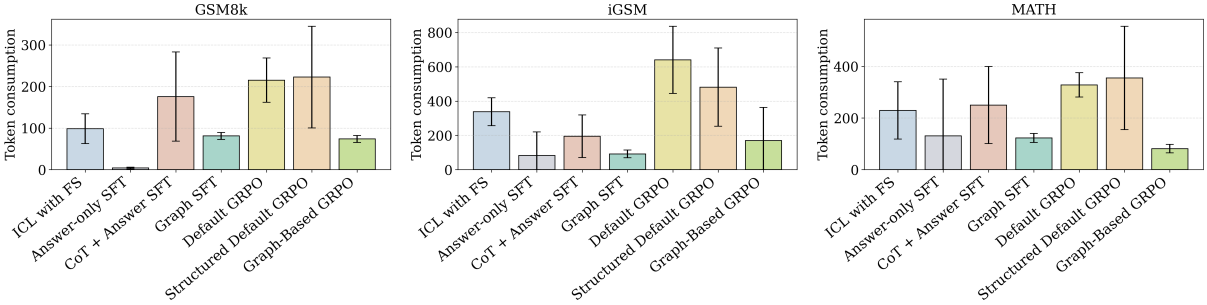

Figure 3: Mean token consumption per generation across datasets.

Qwen3-4B by +0.12, and Falcon3-3B by +0.11. These results suggest that reinforcement learning over computational graphs not only enforces well-formed output structure but also strengthens the model's capacity for compositional reasoning under distribution shift.

Larger models benefit more from RL-based graph training. The improvements are consistent for Llama, Falcon, and Qwen models, while the smallest backbone (Gemma2-2B) shows weaker gains from reinforcement learning. This may indicate that smaller models have more difficulty learning stable reasoning policies under RL optimization.

Graph reasoning is more token-efficient than free-form reasoning. Figure 3 shows the average number of tokens per solution. Graph-Based GRPO produces substantially shorter outputs than Default GRPO while maintaining comparable or higher accuracy. The effect is most pronounced on iGSM, where Graph-Based GRPO achieves higher accuracy with significantly fewer tokens. This suggests that computational graphs provide a more compact reasoning representation than natural language chain-of-thought (see Appendix D).

Direct graph training generalizes better than format transfer. We further analyze Structured Default GRPO, where models trained with GRPO on natural-language outputs are prompted to generate graphs at test time. Although not trained for graph generation, several models (Llama3.2-3B, Falcon3-3B, Qwen3-4B) still achieve non-zero accuracy on GSM8K, indicating partial transfer of reasoning behavior across formats.

However, results on the harder iGSM and MATH datasets remain far below Graph-Based GRPO.

This suggests that directly optimizing computational graphs provides a stronger inductive bias for structured and compositional reasoning.

5 Faithfulness

Method	GSM8K	iGSM	MATH
Qwen3 Graph GRPO			
Internal	78.16%	67.93%	29.37%
External	47.07%	67.63%	29.82%

Table 3: Internal and external evaluation results for Qwen3-4B-Instruct using Graph GRPO.

Method	GSM8K	iGSM	MATH
Qwen3 ICL	2.12%	1.36%	0.45%
Qwen3 Graph GRPO	2.47%	38.90%	24.84%

Table 4: Faithfulness comparison for Qwen3 across datasets.

Table 3 reports two evaluation modes: *Internal*, where the answer is obtained by executing the generated graph, and *External*, where the answer is taken from the model's `ANSWER: <num>` field.

On iGSM and MATH, internal and external accuracies are nearly identical. This alignment suggests the model derives its answer directly from the computational graph.

On GSM8K internal accuracy (78.16%) exceeds external accuracy (47.07%) by over 31 points, indicating that the model often constructs a correct graph but outputs a different final answer. We hypothesize this discrepancy arises from reliance on memorized GSM8K patterns rather than executing the generated graph.

Intervention-based faithfulness. To directly test whether the model relies on its generated graph, we conduct an intervention study comparing Qwen3 with ICL prompting and Qwen3 trained with Graph-Based GRPO (Table 4). For each problem, the model first generates a computational graph. We then modify one operation in the graph (e.g., $a + b \rightarrow a - b$) and ask the model to regenerate the final answer based on the modified graph. A prediction is considered *faithful* if it matches the value obtained by executing the intervened graph.

The ICL baseline exhibits near-zero faithfulness across all three datasets (2.12%, 1.36%, 0.45%), confirming that it largely ignores the graph when producing an answer. The Graph-Based GRPO model shows a substantial improvement on iGSM (38.90%, +37.54) and MATH (24.84%, +24.39), indicating that the model conditions its output on the graph structure in these settings. On GSM8K, however, faithfulness remains low even after training (2.47%, +0.35), consistent with the large internal–external gap observed in Table 3: here, the model generates a plausible graph but derives its answer from a separate, likely memorized pathway.

Overall, results are consistent: when internal and external accuracies align (iGSM, MATH), faithfulness is high, indicating reliance on the computational graph; when they diverge (GSM8K), faithfulness drops, suggesting the graph is a post-hoc artifact.

6 Ablation Study

Model	Accuracy (%)	
	Mean	Max
QWEN3-4B-INSTRUCT2507	79.48 ± 0.39	79.86
- only style reward	75.88 ± 1.19 **(-3.6)**	77.24 **(-2.62)**
- only correctness reward	79.06 ± 1.04 (-0.42)	80.21 (+0.35)
- only overlap reward	78.61 ± 0.75 **(-0.87)**	79.43 (-0.43)
- default scheduler	78.99 ± 0.28 (-0.49)	79.15 **(-0.71)**
- computational graphs	83.53 ± 1.03 **(+4.05)**	84.66 **(+4.8)**
- 1/8/16 rollouts (Self-Consistency)	**79.48/85.11/86.45**	**79.86/85.44/86.78**

Table 5: Ablation results for QWEN3-4B-INSTRUCT2507 on the GSM8K test set.

To isolate the contribution of each component, we conduct ablations on Qwen3-4B using the GSM8K test set (Table 5), varying one factor at a time while keeping all other settings fixed.

Reward components. Using the style reward alone yields the largest degradation (-3.6 mean, -2.62 max) and the highest variance (± 1.19), confirming that formatting supervision alone provides no reasoning signal and destabilizes training. The correctness reward in isolation retains most of the mean accuracy (-0.42) and even slightly exceeds the full model's peak (80.21 vs. 79.86), demonstrating that answer-level feedback is the primary driver of reasoning ability. The overlap reward substantially lowers variance than either of the other single-reward settings (± 0.75), underscoring its role as a structural regularizer. Combining all three rewards achieves the best mean accuracy and the lowest variance (± 0.39), confirming that correctness and structural signals are complementary: the former drives performance while the latter stabilizes it.

Curriculum scheduling. Replacing our complexity-based curriculum (sorting by number

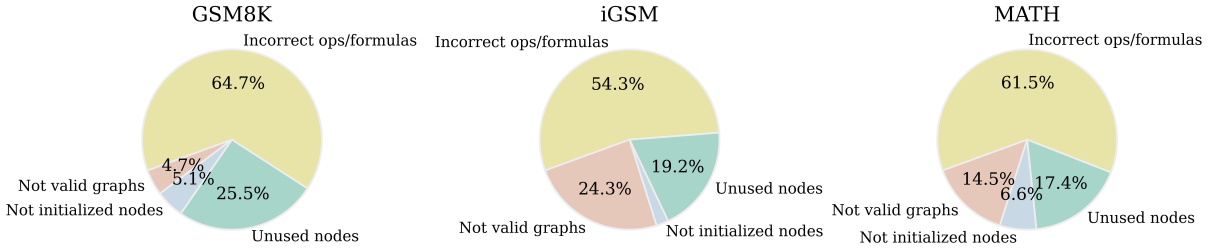

Figure 4: Error ratio across datasets averaged across models.

of nodes and edges) with random shuffling reduces both mean (-0.49) and max (-0.71) accuracy. This indicates that a simple-to-complex progression yields more effective RL training.

Graph representation. Switching from simplified graphs to full computational graphs on Qwen3-4B produces the single largest improvement in our ablations ($+4.05$ mean, $+4.80$ max), demonstrating that this model can leverage richer structural detail. In turn, for Llama, simplified graphs outperform full ones, suggesting that usable graph granularity depends on model capacity.

Number of rollouts and self-consistency. With a single rollout the model matches the default mean (79.48). Scaling to eight rollouts yields a substantial jump to 85.11 ($+5.63$), while sixteen rollouts with majority-vote self-consistency (Wang et al., 2023b) push accuracy to 86.45 ($+6.97$), the highest result in our experiments.

7 Error Analysis

Figure 4 summarizes the distribution of error categories aggregated across models for each dataset. We identify five principal failure modes, ordered from structural to semantic. Additional error statistics per model are provided in Appendix E.

Incorrect operations dominate errors. Most failures ($> 50\%$) stem from selecting incorrect arithmetic operations or composing them improperly, indicating errors in semantic reasoning.

Unused nodes indicate incomplete reasoning integration. In these cases, models generate intermediate nodes that are never used in subsequent computations, suggesting partial but incomplete problem decomposition, occurring in approximately 17% to 25.5% of cases.

Malformed graph syntax accounts for few errors on simple tasks but increases with complexity. These errors correspond to invalid graph structures (e.g., malformed JSON or missing nodes). They are rare on GSM8K (4.7%) but become more frequent on iGSM(24.3%) and MATH(14.5%) as reasoning complexity increases.

Uninitialized node references. This error occurs when an operation references a node that has not been defined, preventing graph execution even if parts of the reasoning are correct. It is relatively rare, accounting for less than 7% of cases across all datasets.

Incorrect constant extraction reflects natural-language grounding failures. These errors arise when constants are incorrectly extracted from the problem statement, especially for expressions such as *twice* or *half*.

8 Conclusion

In this work, we explored structured reasoning as an alternative to free-form chain-of-thought explanations by training language models to generate executable computational graphs.

Our experiments show that reinforcement learning over graph-structured reasoning improves both accuracy and efficiency. Graph-Based GRPO consistently outperforms supervised graph training and yields particularly strong gains on compositional reasoning tasks, while producing substantially shorter reasoning traces.

Importantly, intervention experiments reveal significantly higher faithfulness: the final answers change when the reasoning graph is modified, indicating that models genuinely rely on the generated structure rather than producing post-hoc explanations.

Overall, these results suggest that executable reasoning structures provide a promising path toward more reliable, efficient, and verifiable reasoning in language models.

Limitations

While our Graph-Based GRPO method demonstrates consistent improvements over Graph SFT, it still underperforms compared to CoT + Answer SFT and Default GRPO across most settings. This suggests that, despite the advantages of structured reasoning, there remains substantial room for improvement, particularly in the design of reward functions and output representations, as highlighted by our ablation study.

Additionally, our experiments are limited to mathematical reasoning datasets with well-defined numerical answers. It remains an open question how well the proposed approach generalizes to other domains, especially those requiring more diverse or less structured forms of reasoning.

References

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. 2025. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Antonia Creswell and Murray Shanahan. 2022. Faithful reasoning using large language models. *arXiv preprint arXiv:2208.14271*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International conference on machine learning*, pages 10764–10799. PMLR.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Yunseok Han, Yejoon Lee, and Jaeyoung Do. 2026. Rfeval: Benchmarking reasoning faithfulness under counterfactual reasoning intervention in large reasoning models. *arXiv preprint arXiv:2602.17053*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Andreas Holzinger, Chris Biemann, Constantinos S Pattichis, and Douglas B Kell. 2017. What do we need to build explainable ai systems for the medical domain? *arXiv preprint arXiv:1712.09923*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Ruike Hu and Shulei Wu. 2025. Rl-struct: A lightweight reinforcement learning framework for reliable structured output in llms. *arXiv preprint arXiv:2512.00319*.

Alon Jacovi and Yoav Goldberg. 2020. Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4198–4205.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamile Lukosiute, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, and 11 others. 2023. Measuring faithfulness in chain-of-thought reasoning. *CoRR*, abs/2307.13702.

Yaxi Lu, Haolun Li, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Zhiyuan Liu, Fangming Liu, and Maosong Sun. 2025. Learning to generate structured output with schema reinforcement learning. In

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 4905–4918.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329.

Katie Matton, Robert Osazuwa Ness, John Guttag, and Emre Kıcıman. 2025. Walk the talk? measuring the faithfulness of large language model explanations. *arXiv preprint arXiv:2504.14150*.

Debjit Paul, Robert West, Antoine Bosselut, and Boi Faltings. 2024. Making reasoning matter: Measuring and improving faithfulness of chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15012–15032.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. 2025. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *SuperIntelligence-Robotics-Safety & Alignment*, 2(6).

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965.

Martin Tutek, Fateme Hashemi Chaleshtori, Ana Marasović, and Yonatan Belinkov. 2025. Measuring chain of thought faithfulness by unlearning reasoning steps. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9946–9971.

Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2023a. Towards understanding chain-of-thought prompting: An empirical study of what matters. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2717–2739.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zidi Xiong, Shan Chen, Zhenting Qi, and Himabindu Lakkaraju. 2025. Measuring the faithfulness of thinking drafts in large reasoning models. *arXiv preprint arXiv:2505.13774*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Yuanzhi Li Zeyuan Allen-Zhu. 2023. Physics of language models. *Computing Research Repository*, arXiv:2309.14402. Version 2.

A Model's Hyperparameters

All models were trained with LoRA, 3 epochs for SFT pipelines, 600 iterations of GRPO for default GRPO pipeline and 1 epoch for pre-GRPO SFT checkpoint and 600 iterations for Graph GRPO pipeline. Models trained for 3 seed runs - 0, 42, 2026.

LoRA $\alpha = 32, r = 32$, attention (q_proj, k_proj, v_proj, o_proj) and mlp (gate_proj, up_proj, down_proj) modules are tuned.

GRPO num_genrations=4 for Default GRPO and num_generations=16 for Graph GRPO pipelines.

Other lr=5×10^{-6} for SFT and GRPO pipelines, lr=3×10^{-4} for pre-GRPO checkpoint. paged_adamw_8bit optimizer (Kingma and Ba, 2014), batch_size=2, gradient_accumulation=2.

B Computational graph generation algorithm

To generate computational graphs for the datasets, we use a few-shot prompting strategy. Specifically, two example problems from the same domain are included in the system prompt to demonstrate the desired graph format and guide the model toward producing structured outputs.

We use a low sampling temperature ($T = 0.2$) to ensure stable graph generation and reduce variability in the produced structures. All graphs are generated using GPT-4o.

To ensure the validity of the generated graphs, we execute a forward pass through each graph and compare the computed answer with the ground-truth solution. We additionally enforce that the final node of the graph must be an operation node rather than a constant node. This constraint prevents degenerate cases in which the model outputs the correct answer directly while the preceding nodes and edges do not contribute to the final computation.

The graph generation algorithm is identical for all three datasets.

C Conversion to simplified graphs algorithm

To simplify processing and analysis, we convert the generated computational graphs into a simplified representation. In this format, each node is represented as a string describing either a constant assignment or an arithmetic operation.

The conversion procedure preserves the original order of nodes in the graph, ensuring that dependencies remain valid and that the computational structure can be executed sequentially.

Representation format. Constant nodes are written in the form `"id=value"`, where `id` is the node identifier and `value` is the numeric constant. Operation nodes are represented as expressions of the form "`id=input`$_1$ `op input`$_2$", where the operator symbol is placed between the input nodes.

During conversion, textual operation names are mapped to standard arithmetic symbols. In particular, `add`, `sub`, `mul`, `div`, `pow`, and `mod` are converted to `+`, `-`, `*`, `/`, `**`, and `%`, respectively.

For operations with multiple inputs, the operator symbol is inserted between all inputs. For example, an `add` operation with inputs `["a", "b", "c"]` is converted to the expression `"a+b+c"`.

D Token consumption

Table 6 reports the average number of tokens generated per solution across datasets, models, and training strategies. This analysis provides additional insight into the efficiency of different reasoning approaches.

Natural language reasoning vs structured reasoning. Methods that rely on natural language reasoning traces, such as CoT + Answer SFT and Default GRPO, consistently produce the longest generations. For example, on GSM8K the CoT + Answer SFT approach requires up to 360 tokens on average for Gemma2-2B and more than 130 tokens for Falcon3-3B. Similar patterns can be observed on iGSM and MATH, where natural language reasoning traces often exceed 400 tokens. This reflects the verbose nature of chain-of-thought explanations, where intermediate reasoning steps are expressed in full sentences.

In contrast, graph-based approaches represent reasoning steps as compact computational structures. As a result, Graph SFT and Graph-Based GRPO produce substantially shorter outputs across all datasets.

Effect of reinforcement learning on generation length. One of the most notable trends in Table 6 is that Graph-Based GRPO consistently reduces generation length compared to Graph SFT. For example, on GSM8K the average token count for Falcon3-3B decreases from 89.53 tokens (Graph SFT) to 80.36 tokens (Graph-Based GRPO), while on iGSM it decreases dramatically from 131.18 to 51.70 tokens. Similar reductions can be observed for Qwen3-4B on iGSM (76.43 $\rightarrow$ 68.69 tokens) and MATH (126.77 $\rightarrow$ 82.84 tokens).

This suggests that reinforcement learning not only improves reasoning accuracy but also encourages the model to generate more concise computational graphs. Intuitively, GRPO rewards solutions that lead to correct answers, which implicitly discourages unnecessary intermediate nodes and redundant operations.

Comparison with Default GRPO. The difference between Default GRPO and Graph-Based GRPO is particularly striking in terms of token efficiency. Default GRPO often produces very long outputs, especially on more difficult datasets such as iGSM. For example, Default GRPO with Falcon3-3B generates on average 693 tokens on iGSM, compared to only 51.70 tokens for Graph-Based GRPO. A similar pattern can be observed for Qwen3-4B (887 vs 68 tokens).

These results indicate that graph-based reasoning provides a much more compact representation of the reasoning process than natural language chain-of-thought. Instead of describing intermediate steps in natural language, the model directly

Dataset	Method	Gemma2-2B	LLama3.2-3B	Falcon3-3B	Qwen3-4B
GSM8k	ICL with FS	150.53 ± 65.12	58.88 ± 50.77	72.87 ± 34.67	112.58 ± 129.79
	Answer-only SFT	7.35 ± 3.08	2.10 ± 0.33	3.46 ± 1.04	3.39 ± 1.04
	CoT + Answer SFT	360.42 ± 182.43	93.91 ± 42.62	130.30 ± 67.32	118.91 ± 57.69
	Graph SFT	88.47 ± 47.92	69.74 ± 27.08	89.53 ± 45.44	76.85 ± 32.61
	Default GRPO	170.27 ± 67.75	169.92 ± 54.76	299.99 ± 87.93	221.28 ± 98.31
	Structured Default GRPO	175.33 ± 138.43	222.10 ± 100.27	78.45 ± 36.22	414.82 ± 213.87
	Graph-Based GRPO	82.67 ± 38.94	61.49 ± 28.78	80.36 ± 36.02	71.11 ± 38.06
iGSM	ICL with FS	326.33 ± 117.56	361.42 ± 160.62	218.36 ± 135.57	444.99 ± 137.17
	Answer-only SFT	319.60 ± 164.40	4.00 ± 0.00	4.60 ± 0.49	4.61 ± 0.98
	CoT + Answer SFT	410.17 ± 166.68	127.37 ± 144.87	131.41 ± 149.65	112.03 ± 122.22
	Graph SFT	83.59 ± 101.61	76.55 ± 96.42	131.18 ± 123.14	76.43 ± 87.87
	Default GRPO	341.38 ± 139.05	640.95 ± 297.01	693.38 ± 265.08	887.43 ± 246.89
	Structured Default GRPO	343.25 ± 307.45	437.66 ± 207.11	279.67 ± 256.10	864.80 ± 266.57
	Graph-Based GRPO	505.45 ± 122.76	50.55 ± 39.85	51.70 ± 43.75	68.69 ± 61.98
MATH	ICL with FS	279.42 ± 123.85	164.77 ± 177.27	91.79 ± 89.83	383.12 ± 168.09
	Answer-only SFT	511.44 ± 16.35	4.17 ± 0.41	5.64 ± 16.08	3.20 ± 0.98
	CoT + Answer SFT	507.23 ± 63.34	149.80 ± 130.21	189.37 ± 156.75	153.98 ± 137.38
	Graph SFT	148.62 ± 183.45	114.15 ± 149.91	101.76 ± 124.76	126.77 ± 168.18
	Default GRPO	285.49 ± 126.45	278.94 ± 134.75	364.36 ± 126.88	385.02 ± 138.67
	Structured Default GRPO	222.89 ± 201.15	436.59 ± 259.77	120.88 ± 172.65	640.59 ± 305.70
	Graph-Based GRPO	105.16 ± 132.62	59.24 ± 93.37	78.29 ± 108.38	82.84 ± 115.33

Table 6: Token consumption comparison per generation across datasets and training methods. Standard deviations are shown in smaller font for readability.

produces executable reasoning graphs.

Gemma token anomaly. An interesting anomaly can be observed for the Gemma2-2B model. Despite being the smallest model in our study (2B parameters), it often produces unusually long outputs, particularly when trained with CoT-based methods. For example, on GSM8K the CoT + Answer SFT approach generates more than 360 tokens on average, significantly exceeding the generation length of larger models such as Llama3.2-3B or Falcon3-3B.

Manual inspection suggests that this behavior is frequently associated with hallucinated reasoning traces, where the model generates excessively verbose or redundant intermediate steps that do not contribute to the final computation. This phenomenon is consistent with previous observations that smaller models tend to rely on longer and less structured reasoning traces when attempting to solve complex tasks.

Summary. Overall, the results show that graph-based reasoning significantly reduces token consumption while maintaining or improving performance. Reinforcement learning further enhances this effect by encouraging more compact reasoning graphs. These findings suggest that computational graphs provide a more efficient intermediate representation for reasoning compared to natural language explanations.

E Error research

Table 7 presents a detailed breakdown of error categories across datasets and models.

Several trends can be observed from the table.

Dominance of reasoning errors. Across most models and datasets, the largest proportion of failures is caused by incorrect operations or formulas. This indicates that the primary difficulty lies in selecting and composing the correct arithmetic operations rather than generating the graph structure itself.

Dataset difficulty effects. The distribution of error types varies substantially across datasets. GSM8K shows a relatively balanced distribution of errors, while iGSM contains a larger proportion of structural errors such as invalid graphs. This is likely due to the higher compositional complexity of iGSM problems.

Model size effects. Smaller models struggle more with producing valid structured outputs. In particular, Gemma2-2B exhibits a very high rate of invalid graphs on iGSM and MATH, suggesting that maintaining strict structural constraints is difficult for smaller models.

Unused and missing nodes. Errors related to unused nodes and missing node initialization occur when the model partially decomposes the problem

Dataset	Model	Not valid graphs	Not initialized nodes	Unused nodes	Incorrect ops/formulas
GSM8K	Falcon3-3B	2.68 ± 1.34	7.80 ± 1.49	24.72 ± 2.42	64.81 ± 3.53
	Gemma2-2B	7.55 ± 2.45	3.09 ± 0.43	30.27 ± 3.39	59.08 ± 0.62
	Llama3.2-3B	3.82 ± 0.33	4.82 ± 0.66	26.73 ± 0.17	64.63 ± 0.72
	Qwen3-4B	4.59 ± 0.66	4.80 ± 1.43	20.45 ± 1.71	70.16 ± 0.64
iGSM	Falcon3-3B	0.57 ± 0.33	1.79 ± 0.58	12.82 ± 3.76	84.83 ± 3.29
	Gemma2-2B	94.49 ± 3.02	2.28 ± 1.57	1.52 ± 0.56	1.71 ± 1.56
	Llama3.2-3B	0.51 ± 0.36	2.26 ± 0.93	26.37 ± 8.84	70.86 ± 9.83
	Qwen3-4B	1.67 ± 1.30	2.45 ± 1.16	36.01 ± 2.80	59.87 ± 2.69
MATH	Falcon3-3B	0.87 ± 0.11	4.78 ± 0.10	18.48 ± 2.48	75.87 ± 2.32
	Gemma2-2B	51.72 ± 32.39	13.01 ± 6.59	9.68 ± 4.58	25.60 ± 33.19
	Llama3.2-3B	2.44 ± 0.62	3.16 ± 0.44	22.92 ± 0.92	71.48 ± 1.07
	Qwen3-4B	3.06 ± 1.29	5.32 ± 0.47	18.65 ± 1.89	72.98 ± 1.65

Table 7: Error category distribution (%) across datasets and models (mean and std over 3 seeds, incorrect constant nodes init included in the last category).

but fails to correctly connect intermediate steps. These errors highlight the difficulty of maintaining consistent dependency structures within the generated graphs.

Overall, these results suggest that while models are generally capable of generating the correct graph format, the main challenge remains learning correct computational compositions within the graph structure.

F Prompt for generating computational graph

```
You are an expert in computational graphs and mathematical reasoning.

Given a math word problem, generate a computational graph as a JSON
    array where each node represents either:
1. A constant value (type: "const")
2. An arithmetic operation (type: "op")

Rules:
1. Each constant node has: {"id": <name>, "type": "const", "value": <
    number>}
2. Each operation node has: {"id": <result_name>, "type": "op", "op":
    <operation>, "inputs": [<input_ids>]}
3. Available operations: "add", "sub", "mul", "div", "pow", "mod"
4. Operation nodes are named after their result
5. List nodes in computational order (dependencies before usage)
6. Use descriptive names for node IDs
7. Constants should represent actual numbers from the problem
8. Each operation produces a new value
9. Each operation could only use previous const and op nodes as inputs
10. After the end of the graph, write "ANSWER: <number>"

EXAMPLE:
Question: Natalia sold clips to 48 of her friends in April, and then
    she sold half as many clips in May. How many clips did Natalia
    sell altogether in April and May?

[
  {"id": "april_clips", "type": "const", "value": 48},
  {"id": "half", "type": "const", "value": 0.5},
  {"id": "may_clips", "type": "op", "op": "mul", "inputs": ["
      april_clips", "half"]},
  {"id": "total_clips", "type": "op", "op": "add", "inputs": ["
      april_clips", "may_clips"]}
]
ANSWER: 72
```

Figure 5: Example of prompt used for generating structured computational graphs with answer.

G Prompt for generating simplified computational graph

```
You are an expert in computational graphs and mathematical reasoning.

Given a math word problem, generate:
1) A simplified computational graph as a JSON list of strings.
2) A final answer line in the format: ANSWER: <number>

Graph format rules:
1. Constant node format: "node_id=value"
2. Operation node format: "node_id=input1<op>input2" (example: "c=a-b
    ")
3. Available operations: "+", "-", "*", "/", "**", "%"
4. Operation nodes are named after their result
5. List nodes in computational order (dependencies before usage)
6. Use descriptive names for node IDs
7. Constants should represent actual numbers from the problem
8. Each operation produces a new value
9. Each operation could only use previous const and op nodes as inputs

EXAMPLE:
Question: Natalia sold clips to 48 of her friends in April, and then
    she sold half as many clips in May. How many clips did Natalia
    sell altogether in April and May?

[
  "april_clips=48",
  "half=0.5",
  "may_clips=april_clips*half",
  "total_clips=april_clips+may_clips"
]
ANSWER: 72
```

Figure 6: Example of prompt used for generating simplified computational graphs with answer.