

Enhanced Data Synthesis for LLM through Reasoning Structures Generated by Hierarchical GFlowNet

Tianpeng Bu*, Minying Zhang*[†], Hongtao Duan, Shurui Li, Lulu Hu, Yu Li

Alibaba Cloud Computing, Hangzhou, China

{tianpeng.btp, minying.zmy, hongtao.dht, lishurui.lsr, chudu.hll, lojze.ly}@alibaba-inc.com

Abstract

Large language models (LLMs) excel in problem-solving but require training data with diverse reasoning processes. Existing methods mainly optimize instruction-response pairs but lack a systematic design for the underlying reasoning structure. This paper proposes *RSS*: a **R**easoning **S**tructure driven data **S**ynthesis method. We first proactively develop a hierarchical GFlowNet to construct reasoning structures efficiently through a coarse-to-fine directed acyclic graph (DAG) growth process. Then these reasoning DAGs are leveraged to actively guide the instruction generation via an iterative suggester-editor workflow and enhance response quality using a structure-aware strategy. Experiments show that LLMs trained on our synthetic datasets achieve 48.50%, 84.00%, 79.90% for AlpacaEval2, GSM8K and HumanEval, outperforming existing data synthesis methods.

1 Introduction

Large language models (LLMs) have demonstrated human-like problem-solving abilities across a wide variety of tasks (OpenAI, 2023; Touvron et al., 2023; Wei et al., 2021). The reasoning process in problem-solving is often represented as graph-like reasoning structures (Besta et al., 2023). Training LLMs to master diverse reasoning structures is essential to unlock the abilities of LLMs for more general and complex tasks (Wei et al., 2022; OpenAI, 2024). However, collecting high-quality data with rich reasoning process is both costly and labor-intensive (Conover et al., 2023; Köpf et al., 2024). To address this issue, LLM-driven approaches are proposed to generate synthetic data (Wang et al., 2022; Xu et al., 2023, 2024; Taori et al., 2023). Some approaches leverage LLMs to directly augment seed data using prompts, such as Self-Instruct

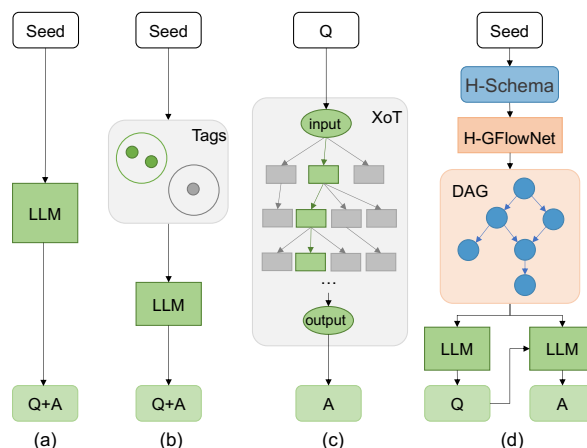


Figure 1: Comparison of data synthesis methods. a). Augment *seed* questions using prompts. b). Generate data based on the combination of *tags* extracted from the seed pool. c). *XoT* methods use trial-and-error to optimize solutions of data. d). Our method proactively generates reasoning structures as direct acyclic graphs and uses them to synthesize instructions and responses.

and Evol-Instruct (Wang et al., 2022; Xu et al., 2023). In these methods, the inherent reasoning structures to response are indirectly altered by augmented questions, as illustrated in Figure 1(a). Subsequently, methods in Figure 1(b) establish a pool of tags (knowledge points, skills, or personas) and use LLMs to generate data based on the combination of these tags (Li et al., 2024; Kaur et al., 2024; Ge et al., 2024). However, this combination influences reasoning structures in a passive manner, as logical connections among tags are not considered. Additionally, studies in Figure 1(c) focus on enhancing response quality from existing data via multi-query prompting frameworks like tree-of-thoughts (ToT) and Monte Carlo Tree Search (MCTS) (Yao et al., 2023; Zhang et al., 2024a). These methods inefficiently use trial-and-error to find accurate solutions, lacking guidance from the original questions’ reasoning structures. Therefore, we propose a **R**easoning **S**tructure

* Equal Contribution.

[†] Corresponding author: Minying Zhang.

driven data Synthesis method, abbreviated to RSS. It proactively generate reasoning structures as direct acyclic graphs and uses them to synthesize instructions and responses, as illustrated in Figure 1(d). With explicit guidance from reasoning structures, we can generate logical, real-world-like questions and high-quality responses efficiently.

Synthesizing diverse reasoning structures with complex logical connections is challenging due to the vast combination space created by the diversity of real-world questions (Newell et al., 1972). To address this, we propose a three-level hierarchical cognitive schema (H-Schema), inspired by the CHC theory (Carroll, 2005), which includes abstract-level, competency-level, and example-level cognitive nodes for representing problem-solving capabilities (see Figure 2(a)). Reasoning structures are modeled as directed acyclic graphs (DAGs) at each level, linking cognitive nodes through logical dependencies. Nodes at higher levels define candidate child nodes for finer levels, leveraging dependencies to reduce the search space. These hierarchical relationships provide an organized framework that enhances efficiency and effectiveness of the search process.

The GFlowNet excels at generating diverse DAGs through a stepwise growth process (Bengio et al., 2021a; Deleu et al., 2022), adding nodes iteratively. We developed a hierarchical GFlowNet (H-GFlowNet) to sample reasoning DAGs by connecting nodes from H-Schema (Figure 2(b)). At each generation step of H-GFlowNet, three policy networks corresponding to generating DAG for each level of H-Schema, sequentially predict the newly added node under the condition the sampled node at higher level. The policy networks undergo end-to-end a reinforcement learning training paradigm with feedback on DAG faithfulness as evaluated by LLMs, with guidance provided by the reward prompt.

Starting with reasoning DAGs, we synthesize instructions and responses as shown in Figure 2(c). Unlike existing methods, we use a prior-planned reasoning structure for direct control over the question’s reasoning process. For instruction generation, we parse the DAG into a topological node-relation sequence and use a suggester-editor workflow, similar to (Mitra et al., 2024), that refines the question step by step to accurately reflect the reasoning structure and ensure high quality. Reasoning structures have proven beneficial for producing high-quality responses. Thus, we use a rea-

soning structure-aware strategy for response. By integrating the reasoning DAG into methods like Plan-and-Solve (Wang et al., 2023), ToT (Yao et al., 2023), and MCTS (Zhang et al., 2024a), we reduce the cost of constructing these structures. Since the reasoning DAG inherently represents the problem-solving structure, a straightforward Plan-and-Solve strategy (Wang et al., 2023) can generate effective responses guided by our reasoning DAG.

We validate that our method outperforms baseline methods on mainstream tasks, such as instruction-following, math, and code. Additionally, we find that starting with complex and rich reasoning DAGs is an effective way to generate training data with high-quality.

2 Method

Reasoning Structure as Foundation

The reasoning structure, capturing human cognitive processes during problem-solving (Xiang et al., 2025), serves as our framework’s cornerstone. Building upon thought chain formalizations in GoT (Besta et al., 2023) and DoT (Zhang et al., 2024b), we formalize reasoning processes as directed acyclic graphs (DAGs) where nodes represent atomic problem-solving subprocedures and edges encode stepwise logical dependencies. This DAG abstraction illustrates the compositional nature of reasoning process. The following sections describe our methodology for (1) efficiently sampling DAGs from combinatorial spaces and (2) converting these graphs into instruction-response pairs while preserving logical integrity.

2.1 Hierarchical Cognitive Schema

To tackle the exponentially large space of reasoning DAGs for real-world questions, we propose a three-level hierarchical cognitive schema H to constrain the sampling space via coarse-to-fine abstraction: abstract-level, competency-level, and example-level cognitive nodes. For each level in H , we define corresponding nodes, parent-to-child and child-to-parent relationships extracted by LLMs from instruction-response pairs in seed data (see Appendix B.1). After de-duplication and grouping, the H-Schema organizes cognitive nodes hierarchically (Figure 2a). For instance, the abstract-level node *utilizing knowledge* includes competency-level nodes like *mathematics knowledge* and *finance knowledge*, with *mathematics knowledge* further branching into more example-level nodes.

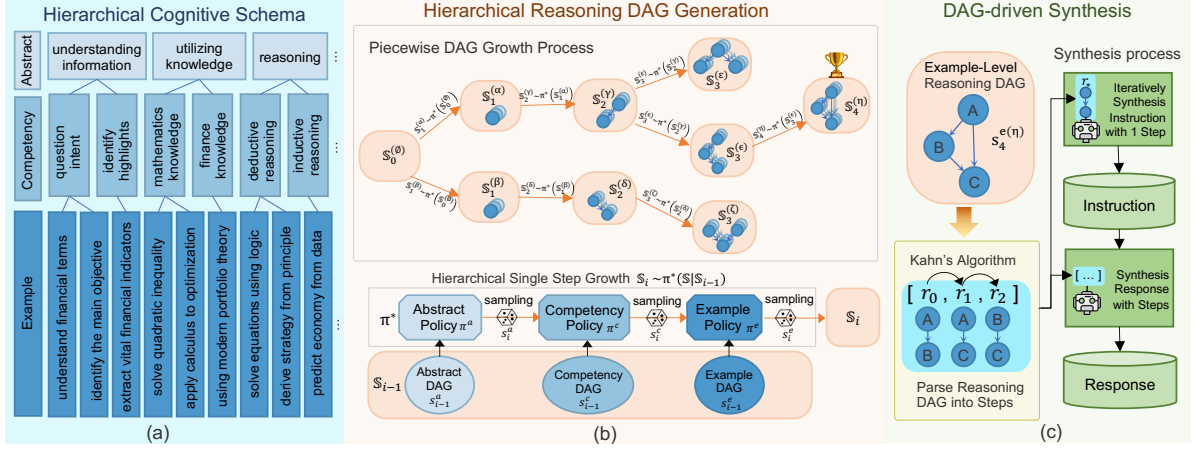


Figure 2: Overview of our method. a). The H-Schema shows the coarse-to-fine organization of cognitive nodes and their relations. b). *Top*: The illustration of Markov Decision Process for reasoning DAG modeling with H-GFlowNet. Node S_i represents a partial or complete reasoning DAG composed of three-level DAGs. The edge represents the transition probability among states modeled by policy network π^* . S_3^e , S_4^e and S_3^e are terminal nodes S_L of MDP and S_4^e receives the high reward marked with a golden cup; *Bottom*: A hierarchical single step growth of the DAG with 3 policy networks. c). The instruction-response synthesis pipeline driven by reasoning DAGs.

This schema enables efficient reasoning DAG construction through layered sampling: abstract-level DAGs define high-level connections, competency-level DAGs refine nodes into domain-specific capabilities, and example-level DAGs instantiate concrete reasoning steps. This hierarchical approach balances diversity and tractability, generating diverse reasoning structures while maintaining constrained search spaces.

2.2 Hierarchical Reasoning DAG Generation

H-GFlowNet for Reasoning DAG Modeling

The essence of our method is the ability to model the distribution of reasoning DAGs and generate novel problem-solving steps. The GFlowNet excels at generating diverse DAGs through a stepwise growth process (Bengio et al., 2021a; Deleu et al., 2022), adding nodes iteratively. To model the distribution of reasoning DAGs at various H-Schema levels, we developed a hierarchical GFlowNet (H-GFlowNet), depicted in Figure 2(b). H-GFlowNet breaks down each sampling step into three hierarchical levels, employing a coarse-to-fine search via three policy networks. The H-GFlowNet’s stepwise DAG expansion is treated as a Markov Decision Process (MDP) trajectory $\tau = (S_0, \dots, S_L)$. Different from vanilla GFlowNet, each state here in MDP S_i denotes a set of reasoning DAGs (s_i^a, s_i^c, s_i^e) at different levels: abstract, competency, and example level step i , with S_0 as the initial empty state and S_L as the terminal state. Starting from S_0 , the model transits a state by adding cognitive nodes or link-

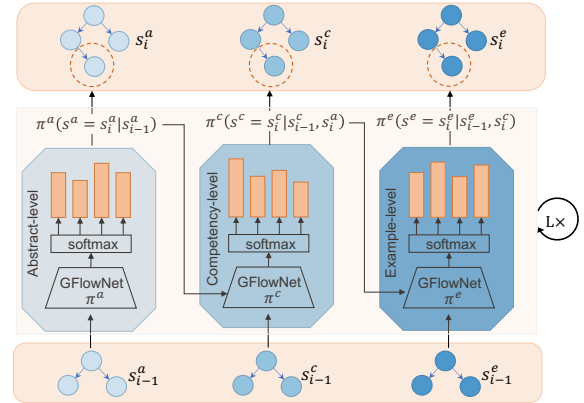


Figure 3: Hierarchical single-step growth for reasoning DAG generation of H-GFlowNet. The nodes marked with dashed-circle correspond to the single-step growth at step i for each level. This process iterates for L times.

ing existing nodes at each level of H-Schema. The transition between states S_{i-1} and S_i is expressed as sampling S_i from policy $\pi^*(S_i | S_{i-1})$. Different from original GFlowNet, H-GFlowNet consists of three policy networks $\pi^* = (\pi^a, \pi^c, \pi^e)$, which includes policies for abstract (π^a), competency (π^c), and example (π^e) levels, aligning with the H-Schema. Upon reaching the terminal state S_L , we obtain the final reasoning DAG, which is evaluated by a reward model $R(S_L)$ during training or used as the target sample during inference.

Hierarchical Single-Step Growth To address the exponential search space of example-level reasoning DAGs in each step, we implement a top-

down three-level growth mechanism coordinated by hierarchical policies (π^a, π^c, π^e) , as visualized in Figure 3. Each iteration progresses through three constrained sampling stages. 1) Abstract expansion $s_i^a \sim \pi^a(s_i^a | s_{i-1}^a)$: π^a evolves abstract-level DAG s_{i-1}^a by adding nodes/edges. 2) Competency refinement $s_i^c \sim \pi^c(s_i^c | s_{i-1}^c, s_i^a)$: π^c refines s_{i-1}^c based on s_i^a , focusing on subdomains of newly added abstract nodes. 3) Example instantiation $s_i^e \sim \pi^e(s_i^e | s_{i-1}^e, s_i^c)$: π^e generates task-specific s_i^e from s_{i-1}^e under s_i^c 's constraints. This layered progression narrows the sampling space at each level, enabling tractable construction of complex reasoning DAGs. The hierarchical constraints ensure structural coherence while maintaining diversity through localized exploration at each level. We formalize the hierarchical process to construct a complete reasoning DAG with H-GFlowNet in the following Algorithm 1.

Algorithm 1 The sampling process of H-GFlowNet

Require: Initial state $\mathbb{S}_0 = (s_0^a, s_0^c, s_0^e)$, Policy network: $\pi^* = (\pi^a, \pi^c, \pi^e)$, H-Schema: H

Ensure: Sampled DAG: $\mathbb{S}_L = (s_L^a, s_L^c, s_L^e)$

1: Initialize step: $i \leftarrow 1$, Total step: L

2: **while** $i \leq L$ **do**

3: Abstract-level: $s_i^a \sim \pi^a(s_i^a | s_{i-1}^a)$

4: Constraint space: $S^c = H.\text{find_child}(s_i^a)$

5: Competency-level: $s_i^c \sim \pi^c(s_i^c | s_{i-1}^c, S^c)$

6: Constraint space: $S^e = H.\text{find_child}(s_i^c)$

7: Example-level: $s_i^e \sim \pi^e(s_i^e | s_{i-1}^e, S^e)$

8: $i \leftarrow i + 1$

9: **end while**

10: **return** $\mathbb{S}_L = (s_L^a, s_L^c, s_L^e)$

The Training of H-GFlowNet A reinforcement learning training paradigm is often used to train GFlowNet (Bengio et al., 2021b; Malkin et al., 2022), in which a policy network generate roll-outs, and a reward model provide the feedback to the policy network. Different from other reinforcement learning algorithms where the learning objective is to optimize the policy network to maximize the reward from the reward model, the target of GFlowNet is to train the policy network to generate samples with probabilities proportional to the reward. An overview of GFlowNet for DAG generation is elaborated in Appendix A.1.

Within the framework of H-GFlowNet, we define of policy network $\pi^*(\pi^a, \pi^c, \pi^e)$ as three graph neural networks similar to (Bengio et al.,

2021a). And the reward function $R(s)$ should assign a score to the sampled example-level reasoning DAG $R(\mathbb{S}_L)$ to reflect how plausible the DAG can be used as problem-solving steps to a instruction. This requires that the reward model should be both accurate and generalizable to score all possible outcomes. Thanks to the generalization and world-knowledge of LLM, we prompt Qwen2.5-72B-Instruct (Yang et al., 2024a) to rate the sampled DAG a score from 1 to 10. The prompt for the reward model can be found in Appendix B.2. Besides, the flow-matching loss proposed in (Bengio et al., 2021a) is used and we replace the vanilla single policy network π and state s with our three-level policy networks π^* and the state collection \mathbb{S} . Specifically, the flow-matching loss for a sampled trajectory $\tau = (\mathbb{S}_0, \dots, \mathbb{S}_L)$ is expressed as follows,

$$L(\tau) = \left(\sum_{1 < i < L} \left(\sum_{\mathbb{S}_{i-1}} \pi^*(\mathbb{S}_i | \mathbb{S}_{i-1}) - \sum_{\mathbb{S}_{i+1}} \pi^*(\mathbb{S}_{i+1} | \mathbb{S}_i) \right) + \sum_{\mathbb{S}_{L-1}} \pi^*(\mathbb{S}_L | \mathbb{S}_{L-1}) - R(\mathbb{S}_L) \right)^2$$

An in-depth explanation for this equation is provided in Appendix A.2. Overall, during training, the policy network π^* hierarchically and stepwisely generates multiple reasoning DAGs, and the flow-matching loss is calculated based on these DAGs, as formalized in Algorithm 2 in Appendix A.3.

2.3 Reasoning DAG Driven Data Synthesis

Iterative Instruction Generation To synthesize instruction-question pairs that strictly adhere to the reasoning DAGs, we implement a collaborative suggester-editor framework inspired by (Mitra et al., 2024) (Figure 2c). The process begins by parsing example-level DAGs into ordered reasoning steps $[r_0, \dots, r_L]$ via Kahn’s algorithm, where each r_i encapsulates parent-child node relationships. We iteratively constructs instructions through progressive refinement. Starting from the initial reasoning step r_0 , at each iteration i , the suggester first analyzes the current instruction and partial reasoning path $[r_0, \dots, r_i]$ to propose a modification plan. This plan suggests how to integrate new reasoning step r_i into current instruction. The editor synthesizes updated instruction based on the plan, ensuring solvability through the specified reasoning path. This dual-phase mechanism propagates across all reasoning steps ($r_0 \rightarrow r_L$), sequentially embedding DAG constraints into the instruction. By decoupling exploratory suggestion from

validity enforcement, the method constructs instructions whose reasoning processes mirror the generated DAGs, ensuring diversity and adherence to reasoning structure. Prompt templates (Appendix B.3) and concrete examples (Table 15) demonstrate how the framework balances creative question variation with structural fidelity.

Reasoning Structure Aware Response Generation Research has illustrated that LLMs can provide higher-quality responses when informed by well-defined reasoning structures. These structures can be introduced through methods such as prompting, buffer retrieval, and fixed meta-cognitive prompts (Wang et al., 2023), (Yang et al., 2024b), and (Wang and Zhao, 2024). Additionally, frameworks like ToT (Yao et al., 2023) and MCTS (Zhang et al., 2024a; Qi et al., 2024) are used to discover optimal reasoning paths. Our approach leverages a reasoning DAG that can be easily combined with these advanced reasoning techniques to enhance response generation. Given that our reasoning DAG maps out the logic needed to respond to an instruction, it naturally aligns with the Plan-and-Solve strategy (Wang et al., 2023). This approach entails parsing the DAG into a sequence of steps, which then serve as a blueprint for crafting the response. Details of using the reasoning DAG in response generation are in Appendix B.4.

3 Experiments

3.1 Experimental Setup

Benchmarks and Baselines We focus on evaluating our method on instruction-following, math and code tasks. A detailed introduction for the benchmarks and baselines is in Appendix C.1.

Dataset Construction and Model Training

First, we collect 40k seed data: 15k from ShareGPT (Chiang et al., 2023), 15k from Evol-Instruct (Xu et al., 2023), 5k from MWP-Bench (Tang et al., 2024), and 5k from Code-Alpaca (Chaudhary, 2023). Using Qwen2.5-72B-Instruct, we build the H-Schema as described in Section 2.1. After deduplication and grouping, the H-Schema contains 30 abstract-level, 240 competency-level, and 5000 example-level cognitive nodes. We then construct the H-GFlowNet to perform coarse-to-fine stepwise sampling at each level. The H-GFlowNet is trained with Qwen2.5-72B-Instruct as the reward model for scoring example-level DAGs. From the trained

Core Task	Dataset Size	Base Model	
		small size	larger size
General IF	40k	LLaMA3-8B	LLaMA3-70B
Math	20k	LLaMA2-7B	LLaMA2-13B
Code	20k	DeepSeek-Coder-6.7B	DeepSeek-Coder-33B

Table 1: Dataset and model statistics. *IF* is an abbreviation for *Instruction Following*.

H-GFlowNet, we sample 40k example-level reasoning DAGs to synthesize a dataset for instruction-following tasks. For math and coding tasks, we sample the DAGs with filtering by node topics, retaining only those related to math or code, resulting in 20k reasoning DAGs respectively. Using GPT-4 and the pipeline in Section 2.3, we generate instructions and responses based on the generated DAGs to get the synthetic datasets for instruction-following, math and code tasks. For each task, we train LLMs of two sizes on the corresponding synthetic datasets via supervised fine-tuning (SFT), as detailed in Table 1. Further training details for H-GFlowNet and LLMs are provided in Appendix C.2. We denote the resulting models as *RSS-IF*, *RSS-MATH* and *RSS-CODE* for instruction-following, math and code tasks respectively.

3.2 Main Results

Instruction Following Table 2 shows that *RSS-IF* achieves competitive results on general instruction-following. For smaller models, *RSS-IF-8B* achieves 48.50, 62.00 for AlpacaEval2 and IFEval respectively. These results surpass other models of similar size which are trained on synthetic datasets for instruction-following. For example, based on the same base model, *RSS-IF-8B* leads MAGPIE and Instruct-SkillMix +23.42 and +5.74 on AlpacaEval2. For larger models, despite *RSS-IF-70B* is trained on only 40k synthetic dataset, it demonstrates comparable performances to LLaMA3-70B-Instruct and Qwen2-72B-Instruct which are trained on large-scale proprietary datasets. Our method even performs on par with proprietary models like GPT-4 and Claude3.0.

Math From Table 3, we can see that our *RSS-MATH* can significantly boost the mathematical reasoning capability of LLMs. Specifically, *RSS-MATH-13B* achieves 92.0 on GSM8K, which is stronger than MathGenieLM-70B that has larger model size. For more challenging benchmark MATH, our method also shows its strength. *RSS-MATH-7B* surpasses +16.90, +25.60 and +15.00 compared with MathScale, MetaMath and Math-

Model	Data	Base Model	Instruction Following		Math		Code		Average
			AlpacaEval2 (%)	IFEval (%)	GSM8K (%)	MATH (%)	HumanEval (%)	MBPP (%)	
Proprietary Models									
Gemini Pro	-	-	24.40	-	-	67.70	84.10	72.90	-
Claude-3.0	-	-	40.50	81.00	95.00	71.10	92.00	90.50	78.35
GPT-3.5-Turbo	-	-	22.70	61.90	74.10	37.80	73.20	70.80	56.75
GPT-4	-	-	50.00	80.30	92.00	52.90	84.10	80.00	73.22
General Instruction-Tuned Models (Model Size \leq 13B)									
Orca-2-7B	Mixed	LLaMA2	-	-	55.70	10.10	17.10	28.40	-
Mistral-7B-Instruct	Mixed	Mistral	20.60	54.65	53.20	13.00	40.20	49.50	38.53
Qwen2-7B-Instruct	Mixed	Qwen2	19.37	54.70	<u>85.70</u>	<u>52.90</u>	<u>79.90</u>	67.20	59.96
LLaMA3-8B-Instruct	Mixed	LLaMA3	22.90	72.10	84.50	51.90	72.60	<u>72.80</u>	<u>62.80</u>
Tulu3-8B-SFT	Mixed	LLaMA3.1	12.40	60.10	76.20	31.50	83.90	-	-
Evol-Instruct-8B	Synthetic	LLaMA3	8.52	34.40	42.91	38.50	65.30	67.50	42.86
Magpie-Pro-300K-Filtered-8B	Synthetic	LLaMA3	25.08	38.80	47.92	43.96	70.73	66.93	48.90
Instruct-SkillMix	Synthetic	LLaMA3	<u>42.76</u>	34.75	78.17	39.62	56.71	68.09	53.35
WizardLM-13B-V1.2	Synthetic	LLaMA2	14.50	43.20	46.80	9.00	31.70	47.90	32.18
<i>RSS-IF-8B</i>	Synthetic	LLaMA3	48.50 (+5.74)	62.00	86.50 (+0.80)	56.18 (+3.28)	76.10	73.20 (+0.40)	67.08 (+4.28)
General Instruction-Tuned Models (Model Size \geq 70B)									
LLaMA2-70B-Chat	Mixed	LLaMA2	14.70	-	57.50	11.60	25.60	45.00	-
LLaMA3-70B-Instruct	Mixed	LLaMA3	34.40	<u>77.30</u>	95.10	<u>68.00</u>	80.50	86.00	73.55
Qwen2-72B-Instruct	Mixed	Qwen2	<u>38.10</u>	77.60	93.20	69.00	86.00	80.20	<u>74.02</u>
<i>RSS-IF-70B</i>	Synthetic	LLaMA3	49.80 (+1.70)	75.40	<u>94.00</u>	64.00	<u>85.10</u>	<u>83.60</u>	75.32 (+1.30)

Table 2: Results for general instruction-tuned models, grouped by parameter size (\leq 13B and \geq 70B) for fair comparison. We also provide some advanced proprietary models’ performance for reference. We also annotate the data source and base model of different methods. Best scores are bolded, second-best underlined. *RSS-IF* achieves competitive performance among general instruction-tuned models on various benchmarks in both groups. Moreover, *RSS-IF-70B* achieves performance comparable to some advanced proprietary models.

Model	GSM8K (%)	MATH (%)	Average
Proprietary Models			
Claude-3.0	95.00	71.10	78.35
GPT-4	92.00	52.90	73.22
Math Specific Models - 7B			
WizardMath-7B	54.90	10.70	32.80
MathScale-7B	66.30	31.10	48.70
MetaMath-7B	66.50	19.80	43.15
MathGenieLM-7B	71.70	33.00	52.35
<i>RSS-MATH-7B</i>	84.00 (+12.30)	48.00 (+15.00)	66.00 (+13.65)
Math Specific Models - 13B			
WizardMath-13B	63.90	14.00	38.95
MathScale-13B	71.30	33.80	53.55
MetaMath-13B	72.30	22.40	47.35
MathGenieLM-13B	80.40	43.80	61.70
<i>RSS-MATH-13B</i>	92.00 (+11.60)	58.90 (+15.10)	75.45 (+13.75)

Table 3: Results for math-specific models. Best scores are bolded, second-best underlined. Our model and compared models are fine-tuned on LLaMA2 with synthetic data. Our *RSS-MATH* outperforms other math-specific models on both the GSM8K and MATH benchmarks.

Genie on the same model size.

Code As shown in Table 4, *RSS-CODE* also demonstrates improvements on coding task. Among models trained on synthetic dataset, *RSS-CODE-33B* achieves +4.5, +4.6 increases on HumanEval and MBPP compared to WizardCoder-33B-V1.1, which is trained from the same base model as ours. Furthermore, our results remain highly competitive against DeepSeek-Coder-Instruct-33B, which uses the same base model but is fine-tuned with a much larger instruction set (approximately 2B tokens) than ours.

Model	HumanEval (%)	MBPP (%)	Average
Proprietary Models			
Claude-3.0	92.00	90.50	91.25
GPT-4	84.10	80.00	82.05
Code Specific Models - \leq 13B			
CodeLlama-Instruct-7B	34.80	44.40	39.60
CodeLlama-Instruct-13B	42.70	49.40	46.05
WizardCoder-Python-7B-V1.0	55.50	56.60	56.05
MagicCoder-7B	<u>70.70</u>	<u>68.40</u>	<u>69.55</u>
SelfCodeAlign-7B	67.10	65.20	66.15
<i>RSS-CODE-6.7B</i>	79.90 (+9.20)	76.40 (+8.00)	78.15 (+8.60)
Code Specific Models - \geq 33B			
CodeLlama-Instruct-34B	41.50	57.00	49.25
DeepSeek-Coder-Instruct-33B	79.30	70.00	74.65
WizardCoder-Python-34B-V1.0	73.20	73.20	73.20
WizardCoder-33B-V1.1	<u>79.90</u>	<u>78.90</u>	<u>79.40</u>
<i>RSS-CODE-33B</i>	84.40 (+4.50)	83.50 (+4.60)	73.95 (+4.55)

Table 4: Results for code-specific models. Best scores are bolded, second-best underlined. Our *RSS-CODE* model leads in performance on both the HumanEval and MBPP benchmarks, surpassing other code-specific models with a notable improvement.

4 Analysis

4.1 Instruction Quality Analysis

To gain insights we analyze the relation between the quality of reasoning DAGs and the quality of datasets. Specifically, we randomly sample 2000 instructions from our synthetic dataset for instruction following, Evol-Instruct, MAGPIE and Instruct-SkillMix. Given the instruction and response of each dataset, we prompt Qwen2.5-72B-Instruct to extract the reasoning structures as DAGs.

Relation between Performances and Rewards of Reasoning DAGs We first score the reasoning DAGs of each dataset with the DAG reward prompt template in Appendix B.2. We then plot the histogram of rewards for all datasets in Figure 4.

This figure shows that our method can generate reasoning DAGs with higher rewards compared with other methods, and the rewards of reasoning DAGs have a positive correlation with the model performance. This result supports our motivation that starting from explicit valuable reasoning structures, we can synthesize datasets of better quality.

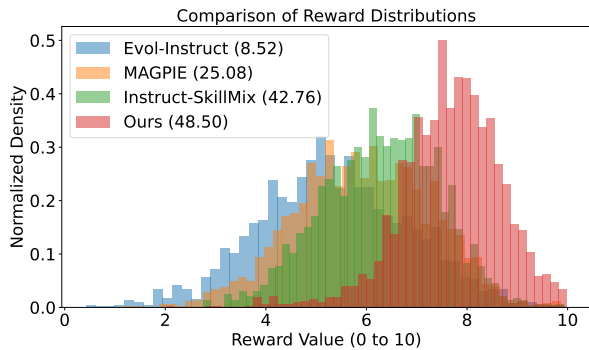


Figure 4: Rewards histogram of reasoning DAGs for different datasets. The numbers in braces are performances on AlpacaEval2 by fine-tuning LLaMA3-8B.

Relation between Performance and metrics of Reasoning DAGs

Besides, the following metrics are calculated for these reasoning DAGs: (1) Node diversity: average number of unique nodes per data; (2) Reasoning depth: average maximum depth from root to leaf; (3) In/Out degree: average number of incoming and outgoing edges per node. From Table 5, Instruct-SkillMix achieves high node diversity (0.94) but struggles with reasoning depth (5.10) compared to Evol-Instruct and MAGPIE, indicating that combining skill tags enhances diversity but falls short in deep reasoning. Evol-Instruct shows greater reasoning depth (5.51) but performs poorly in node diversity and in/out degrees, suggesting increased complexity without improving connectivity. MAGPIE achieves balanced performance across all metrics, benefiting from reward filtering and de-duplication. Our method surpasses these approaches with higher node diversity (0.99), reasoning depth (5.91), and in/out degree (1.21), demonstrating the effectiveness of integrating diverse problem-solving competencies into generated data. These results highlight richer sub-procedure interconnections, enabling our dataset to teach LLMs intricate reasoning structures.

4.2 Response Quality Analysis

In this section, we examine the effectiveness of our reasoning DAG aware response generation. We randomly select $5k$ samples from our synthetic dataset

Datasets	Node diversity	Reasoning depth	In/Out degree	AlpacaEval2
Evol-Instruct	0.89	5.51	0.84	8.52
MAGPIE	0.91	5.50	1.11	25.08
Instruct-SkillMix	0.94	5.10	1.08	42.76
Ours	0.99	5.91	1.26	48.50

Table 5: The performance correlates with reasoning DAG metrics. The performances are measured by AlpacaEval2 by fine-tuning LLaMA3-8B on each dataset.

Response Type	Win-rate (%) \uparrow	Time-cost (min) \downarrow	GSM8K (%) \uparrow
Direct	50.0	0.53	70.4
PS w/o DAG	57.3	0.71	73.6
ToT w/o DAG	59.3	3.22	74.9
MCTS w/o DAG	59.5	3.65	74.8
PS w/ DAG	60.0	0.56	74.9
ToT w/ DAG	60.2	1.23	75.1
MCTS w/ DAG	60.3	1.54	75.3

Table 6: Comparison of response strategies on Qwen2.5-72B-Instruct with and without DAG guidance. Win-rate (%) indicates answer quality judged by GPT-4 relative to direct responses. Time-cost (min) measures generation time per answer. We evaluate performance of models fine-tuned on datasets generated by each strategy on GSM8K (%). Plan-and-Solve is abbreviated as PS. Reasoning DAGs can improve the performance of different answer strategies and boost the efficiency.

for math and investigate the response quality of different response generation strategies. We borrow most of the implementation for ToT and MCTS from ReST-MCTS (Zhang et al., 2024a). All responses are generated by Qwen2.5-72B-Instruct served by vLLM (Kwon et al., 2023a). Due to the efficiency concern, we restrict the breadth of expansion as 2 for ToT and MCTS. Then we use powerful GPT-4 as the judge and direct response as reference to calculate the average win-rate, along with the time-cost (minute/sample) and performance on GSM8K after fine-tuning LLaMA2-7B for each response generation strategy. From Table 6, our reasoning DAG aware response generation can boost the efficiency of existing reasoning framework by guiding the reasoning process with our prior-planned reasoning DAG. For example, we see 0.15 minutes timecost saving and for PS w/ DAG compared with naive Plan-and-Solve prompting. This is more significant for search-based methods, i.e., ToT and MCTS. In terms of the performance, simple PS w/ DAG achieves competitive response quality, i.e, 60.0 win-rate compared with direct answering and 74.9 on GSM8K after fine-tuning on the resulting dataset, costing only 0.56 minutes per sample, which is used as the answer strategy for the final dataset construction.

Reward Model	Spearman's corr.	Price(\$)/1k samples
GPT-4	0.94	15.3
Cluade-3.5	0.94	3.5
LLaMA3.1-70B-Instruct	0.91	self-deployed
Qwen2.5-72B-Instruct	0.93	self-deployed

Table 7: Spearman’s correlation between LLMs and human annotations as the reward model, along with the cost (USD) per 1k samples. Higher correlation indicates better alignment with human preferences.

4.3 Reward Model Selection for GFlowNet

The reward model is crucial in training of our H-GflowNet, which guides the policy networks to effectively explore the action space. In this section, we analyze the effectiveness of using LLMs as the reward model and the performances of different LLMs. In specific, we sample $1k$ reasoning DAGs from the randomly initialized H-GFlowNet for instruction following task. Then we use the prompt in Appendix B.2 to instruct different LLMs to score these DAGs. Moreover, we treat educated human annotations as the golden reward function and ask annotators to rate these DAGs with scores $1 \sim 10$ with the same criteria as LLMs. The agreement between human and these LLMs is measured by Spearman’s correlation as shown in Table 7. From this Table, we can see that the different LLMs perform closely and achieve a good alignment with human for scoring reasoning DAGs. Besides, we can see that Qwen2.5-72B-Instruct achieves 0.93 correlation with human, slightly lower than GPT-4 and Cluade-3.5, but it has open-source weights that can be self-deployed. Therefore, we choose the open-sourced Qwen2.5-72B-Instruct as the reward model of GFlowNet training which is a cost-effective and reliable option.

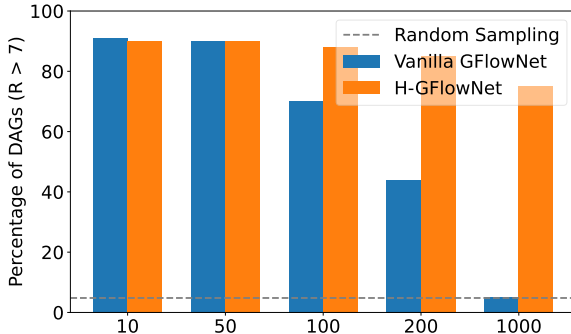


Figure 5: Comparison of proportion of high reward DAGs (reward $R > 7$) sampled from vanilla GFlowNet and H-GFlowNet that are trained on different number of example-level nodes.

4.4 Necessity of H-Schema and H-GFlowNet

To study the effectiveness of the H-Schema and H-GFlowNet, we train GFlowNets with and without hierarchical setup under different sizes of example-level nodes for instruct-following. We set the maximum sampling depth as 10 and train all models with $250k$ iterations for all experiments of this section. As shown in Figure 5, we sample reasoning DAGs from trained GFlowNets and calculate the percentage of DAGs which have reward larger than 7. From this figure, we can see that as the number of example-level nodes increases, the vanilla GFlowNet fails to find DAGs with high rewards and when the the example-level cognitive node reaches 1000, the number of high-reward DAGs found by vanilla GFlowNet is similar to random sampling. However, with the help of hierarchical design of sampling space in H-Schema, the H-GFlowNet can consistently discover high percentage of valuable DAGs, in which about 75% sampled DAGs have rewards larger than 7 even when there are 1000 nodes in example-level.

4.5 Data Scaling Analysis

We study the impact of scaling the H-Schema size (number of example-level nodes) and dataset size by fine-tuning LLaMA3-8B, evaluating model performance on AlpacaEval2. First, we generate datasets with $20k$ samples using H-Schema with example-level node sizes ranging from $1k$ to $5k$. As shown in Figure 6(a), performance on AlpacaEval2 steadily improves as the H-Schema size increases. Specifically, when the H-Schema size grows from $1k$ to $5k$, performance rises from 38.62 to 46.34. This indicates that larger H-Schema, covering a broader scope of human problem-solving capabilities, enhance dataset quality. Additionally, we analyze dataset scaling effects in Figure 6(b) by synthesizing datasets ranging from $10k$ to $40k$ samples. With an H-Schema size of $5k$, we observe a positive scaling effect, with performance increasing from 44.31 to 48.50. This demonstrates that both large H-Schema and dataset sizes contribute to improving synthetic dataset performance. However, no such improvement is observed with smaller H-Schema. For instance, with an H-Schema size of $1k$, performance saturates at 39.33 for a $30k$ dataset and cannot be further improved by generating more data.

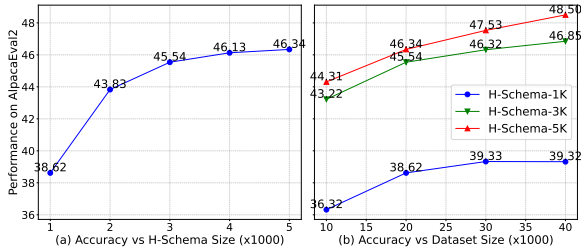


Figure 6: Data scaling effects on AlpacaEval2 for LLaMA3-8B models trained with instruction-following data. (a) Scaling curve w.r.t the size of H-Schema. A larger sampling space of H-GFlowNet improves performance due to increased diversity. (b) Scaling curve w.r.t dataset size for 3 H-Schema sizes. A larger dataset size can enhance performance.

4.6 Contamination Test

We conduct contamination test for our synthetic data as detailed in Appendix D. The results in Table 14 show that our synthetic datasets do not suffer from the data leakage problem.

4.7 Cost Analysis

We leave the cost analysis in Appendix E.

4.8 Case Study

In Appendix F, we showcase a sampled reasoning DAG from our H-GFlowNet and the iterative instruction generation process. Additionally, in Appendix G, we present some sampled data with incorrect answers generated by our method to analyze the main reason of the failures.

5 Related Works

We summarize existing synthetic data generation approaches as follows: Seed transformation-based methods, such as Self-Instruct and Alpaca, Evol-Instruct (Wang et al., 2022; Taori et al., 2023; Xu et al., 2023), use LLMs to transform seed question-answer pairs through evolving/rephrasing/back-translation and related techniques (Xu et al., 2023; Yu et al., 2023; Lu et al., 2024), yet they face limitations in data diversity due to a constrained number of prompts. In parallel, research on enhanced response generation emphasizes explicit reasoning processes, with innovations like STaR (Zelikman et al., 2022) and RFT (Yuan et al., 2023) using iterative verification and reject sampling to refine answers. While more recent works actively explore multiple reasoning paths via searching such as ToT, ReST-MCTS and rStar (Yao et al., 2023; Zhang et al., 2024a; Qi et al., 2024). However,

these methods encounter issues with inconsistent reasoning flows and high computational costs. Abstract representation approaches, such as persona-based generation (Ge et al., 2024) and hierarchical taxonomies (Li et al., 2024; Sudalairaj et al., 2024; Kaur et al., 2024), aim for scalable instruction synthesis but often produce shallow reasoning structures due to random combination of representations. Recently, graph-based solutions such as MathScale and KPSMath (Tang et al., 2024; Huang et al., 2024) primarily encode topic statistics or focus on topic co-occurrence. AlphaGeometry use theorem-proof DAGs for geometric problem generation (Trinh et al., 2024). Different from previous works, our approach is a data synthesis method driven by general and complex reasoning structures generated by a hierarchical generative model called H-GFlowNet.

6 Conclusion

This study addresses the challenge of generating synthetic data with complex reasoning capabilities for the fine-tuning of LLM through reasoning structure design. By shifting from instruction-response optimization to active reasoning structure construction, we propose a framework that integrates hierarchical cognitive modeling and efficient combinatorial sampling. Experiments across instruction-following, mathematical reasoning, and code generation benchmarks validate that complex reasoning DAGs significantly enhance the quality and controllability of synthetic data compared to existing data synthesis strategies.

Limitations

Although our method has shown excellent performance in synthesizing high-quality datasets, several directions are worth exploring in future work:

- (1) The current H-GFlowNet framework primarily relies on the capabilities of large language models (LLMs) for its reward model. A promising direction for future work is to train a customized small language model (SLM) as the reward model.
- (2) We use GPT-4 to generate instructions and responses based on reasoning DAGs. In the future, we can study the performance of using different LLMs in this process.
- (3) Although we conduct our experiments on different LLMs such as LLaMA3, LLaMA2 and DeepSeek-Coder, we can validate the performance of training more models on our synthetic datasets

in the future.

Ethics Statement

We have carefully reviewed the H-Schema to ensure it does not contain any personally identifiable information or offensive content. However, since the seed datasets are sourced from the Internet, there may still be potential biases or inappropriate information. Additionally, despite our careful review, the process of instruction generation using LLMs might inadvertently introduce inappropriate information into the synthetic data. Besides, we hire annotators to score $1k$ reasoning DAGs to study the human agreement of differently LLMs, in which we ensure that all annotators are well-informed and well-paid for this annotation task. It's also important to note that we train H-GFlowNet and LLMs on GPUs, which can have an environmental impact.

References

- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. 2021a. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394.
- Yoshua Bengio, Tristan Deleu, J. Edward Hu, Salem Lahlou, Mo Tiwari, and Emmanuel Bengio. 2021b. [Gflownet foundations](#). *ArXiv*, abs/2111.09266.
- Maciej Besta, Nils Blach, Ale Kubíek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2023. [Graph of thoughts: Solving elaborate problems with large language models](#). In *AAAI Conference on Artificial Intelligence*.
- Nicola De Cao and Thomas Kipf. 2018. [Molgan: An implicit generative model for small molecular graphs](#). *ArXiv*, abs/1805.11973.
- John B Carroll. 2005. The three-stratum theory of cognitive abilities.
- Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. <https://github.com/sahil280114/codealpaca>.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. [Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality](#).
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm. *Company Blog of Databricks*.
- Tristan Deleu, Ant'onio G'ois, Chris C. Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. 2022. [Bayesian structure learning with generative flow networks](#). *ArXiv*, abs/2202.13903.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. 2024. Length-controlled alpacaEval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*.
- Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.
- 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*.
- Edward J. Hu, Moksh Jain, Eric Elmoznino, Younesse Kaddar, Guillaume Lajoie, Yoshua Bengio, and Nikolay Malkin. 2023. [Amortizing intractable inference in large language models](#). *ArXiv*, abs/2310.04363.
- Yiming Huang, Xiao Liu, Yeyun Gong, Zhibin Gou, Yelong Shen, Nan Duan, and Weizhu Chen. 2024.

- Key-point-driven data synthesis with its enhancement on mathematical reasoning. *arXiv preprint arXiv:2403.02333*.
- Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.
- Simran Kaur, Simon Park, Anirudh Goyal, and Sanjeev Arora. 2024. Instruct-skillmix: A powerful pipeline for llm instruction tuning. *arXiv preprint arXiv:2408.14774*.
- Thomas Kipf and Max Welling. 2016. [Variational graph auto-encoders](#). *ArXiv*, abs/1611.07308.
- Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. 2024. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023a. [Efficient memory management for large language model serving with pagedattention](#). *Preprint*, arXiv:2309.06180.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023b. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.
- Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong Zhang, et al. 2024. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*.
- Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. 2023. Tinygsm: achieving > 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*.
- Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2024. [MathGenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of LLMs](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2732–2747, Bangkok, Thailand. Association for Computational Linguistics.
- Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023a. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.
- Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023b. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.
- Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. 2019. [Graphnvp: An invertible flow model for generating molecular graphs](#). *ArXiv*, abs/1905.11600.
- Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. 2022. [Trajectory balance: Improved credit assignment in gflownets](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 5955–5967. Curran Associates, Inc.
- Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Coda, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, et al. 2023. Orca 2: Teaching small language models how to reason. *arXiv preprint arXiv:2311.11045*.
- Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Coda, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. 2024. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*.
- Allen Newell, Herbert Alexander Simon, et al. 1972. *Human problem solving*, volume 104. Prentice-hall Englewood Cliffs, NJ.
- Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. 2020. [Permutation invariant graph generation via score-based generative modeling](#). In *International Conference on Artificial Intelligence and Statistics*.
- OpenAI. 2024. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>. Accessed: 2024-12-20.
- R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5).
- Mariya Popova, Mykhailo Shvets, Junier B. Oliva, and Olexandr Isayev. 2019. [Molecularrrn: Generating realistic molecular graphs with optimized properties](#). *ArXiv*, abs/1905.13372.

- Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lina Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*.
- Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.
- Shivchander Sudalairaj, Abhishek Bhandwadar, Aldo Pareja, Kai Xu, David D Cox, and Akash Srivastava. 2024. Lab: Large-scale alignment for chatbots. *arXiv preprint arXiv:2403.01081*.
- Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024. Mathscales: Scaling instruction tuning for mathematical reasoning. *arXiv preprint arXiv:2403.02884*.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.
- Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shrubti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.
- Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.
- Yuqing Wang and Yun Zhao. 2024. Metacognitive prompting improves understanding in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1914–1926, Mexico City, Mexico. Association for Computational Linguistics.
- Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.
- 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.
- Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro Von Werra, Arjun Guha, and Lingming Zhang. 2024a. Selfcodealign: Self-alignment for code generation. *arXiv preprint arXiv:2410.24198*.
- Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024b. Magicoder: Empowering code generation with OSS-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 52632–52657. PMLR.
- Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, et al. 2025. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-thought. *arXiv preprint arXiv:2501.04682*.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.
- Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *arXiv preprint arXiv:2406.08464*.
- An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. 2024a. Qwen2 technical report. *ArXiv*, abs/2407.10671.

- Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. 2024b. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. [Tree of thoughts: Deliberate problem solving with large language models](#). *ArXiv*, abs/2305.10601.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.
- Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.
- Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.
- Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.
- Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. 2024b. On the diagram of thought. *arXiv preprint arXiv:2409.10038*.
- Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A More Details for Methods

A.1 Methods Overview for DAG Generation

Method for DAG Generation There are two main deep generative approaches for graphs modeling. One is training deep neural networks to learn the distribution of graphs from observations and then sample new graphs from the learned distribution, such as VAE-(Kipf and Welling, 2016), normalizing flow- (Madhawa et al., 2019) and diffusion-based methods(Niu et al., 2020). The other treats graph generation as a sequence of decision-making processes and formulates a reinforcement learning (RL) paradigm, which trains a policy network to explore the action space in order to maximize the rewards from the environment (Bengio et al., 2021a; Popova et al., 2019; Cao and Kipf, 2018). Deep generative flow networks (GFlowNet) (Bengio et al., 2021a; Deleu et al., 2022) pertain to this category, which trains a policy network to generate a sequence of stochastic actions which construct targets with probabilities proportional to the defined reward function.

Comparison between GFlowNet and Other Methods on DAG Generation (1) **GFlowNet can be used to model the sequential process of thoughts.** The nature of sequential probabilistic inference that GFlowNets can perform make it a powerful candidate in to our reasoning DAG modeling. Similar idea has been practiced in (Hu et al., 2023). (2) **GFlowNet can be trained without reasoning DAG dataset.** Compared with learning a distribution from a pre-built dataset like the usual objective for VAE-, normalizing flow- and diffusion-based generative models, GFlowNet can be sufficiently trained without pre-built reasoning DAG dataset as long as the reward model can provide feedback supervision to the policy network. (3) **Better coverage of the reasoning DAG distribution.** Through leveraging the world-knowledge and great generalization ability of LLM as a reward model, the training quality of the policy network can be improved by exploring the action space and requesting feedback from the powerful reward model as many times as possible without concerning with overfitting. This is the key to generate reasoning DAGs beyond the distribution defined by current available instruction datasets. (4) **Better diversity of generated reasoning DAGs.** RL-based method learns the generation policies with the highest reward, which may result in the non-diversity failure

mode. However, our aim is to find diverse modes with high rewards in the distribution of reasoning DAGs. Benefit from the property of GFlowNet that it samples a distribution of trajectories whose probability is proportional to the given reward, it has been proven that GFlowNet can discover more diverse high reward modes than RL-based methods in molecule design in previous research (Bengio et al., 2021a). This motivates us to use GFlowNet as the generative model for reasoning DAGs.

A.2 In-depth Background of H-GFlowNet

The training of H-GFlowNet shares similar paradigm to traditional reinforcement learning, where the policy model π^* generates samples \mathbb{S}_L (terminal state of MDP) and the reward model R provides the feedbacks $R(\mathbb{S}_L)$, but with different interpretation and loss objectives.

The reason to call GFlowNet "flow" is because of the analogy to water flowing through pipes with multiple nodes. The pipes and nodes are structured as a DAG topology as shown in Figure 2(b), in which the pipe (π^*) connects nodes (\mathbb{S}). The water enters the flow network through initial state \mathbb{S}_0 , flowing through multiple intermediate nodes $\mathbb{S}_{i-1}, \dots, \mathbb{S}_i$ and finally flowing into different sinks (terminal nodes) \mathbb{S}_L . The percentage of water reaching a specific terminal node \mathbb{S}_L is proportional to $R(\mathbb{S}_L)$. And the amount of water flowing from node \mathbb{S}_{i-1} to \mathbb{S}_i is modeled by the policy network π^* . With this analogy, one can derive the loss function based on the following two constraints in (Bengio et al., 2021a):

- Intermediate node does not save water. The amount of water entering a specific intermediate node \mathbb{S}_i , i.e., $\sum_{\mathbb{S}_{i-1}} \pi^*(\mathbb{S}_i|\mathbb{S}_{i-1})$, should be equal to the water flowing out of it $\sum_{\mathbb{S}_{i+1}} \pi^*(\mathbb{S}_{i+1}|\mathbb{S}_i)$.
- The amount of water flowing into a specific sink \mathbb{S}_L is: $\sum_{\mathbb{S}_{L-1}} \pi^*(\mathbb{S}_L|\mathbb{S}_{L-1})$ should be equal to its reward $R(\mathbb{S}_L)$.

These two constraints collectively define the following loss function:

$$L(\tau) = \left(\sum_{1 < i < L} \left(\sum_{\mathbb{S}_{i-1}} \pi^*(\mathbb{S}_i|\mathbb{S}_{i-1}) - \sum_{\mathbb{S}_{i+1}} \pi^*(\mathbb{S}_{i+1}|\mathbb{S}_i) \right) + \sum_{\mathbb{S}_{L-1}} \pi^*(\mathbb{S}_L|\mathbb{S}_{L-1}) - R(\mathbb{S}_L) \right)^2$$

In the following, we explain this flow-matching loss term-by-term with an intuitive example.

Clarification to $\sum_{\mathbb{S}_{i-1}} \pi^*(\mathbb{S}_i|\mathbb{S}_{i-1})$: This term represents the in-flow to \mathbb{S}_i from all possible previous state \mathbb{S}_{i-1} . The $\sum_{\mathbb{S}_{i-1}}$ exists because: There potentially exist more states \mathbb{S}_{i-1} that can arrive \mathbb{S}_i by performing one-step growth. For example, if \mathbb{S}_i is a DAG represented as:

node: [0, 1, 2, 3]
 edge: [(0, 1), (0, 2), (1, 3)]

It is straightforward that \mathbb{S}_i has two possible parental states \mathbb{S}_{i-1} . One is that removing node_3 and edge (1, 3), which leads to:

node: [0, 1, 2]
 edge: [(0, 1), (0, 2)]

And another is that removing node_2 and edge (0, 2) to have:

node: [0, 1, 3]
 edge: [(0, 1), (1, 3)]

In other words, by performing the policy model π^* to both these two possible states \mathbb{S}_{i-1} (since the states are statistical), we can both arrive state \mathbb{S}_i . Therefore, to calculate the inflow for \mathbb{S}_i we should sum over all possible parental states \mathbb{S}_{i-1} , which can lead to \mathbb{S}_i .

Clarification to $\sum_{\mathbb{S}_{i+1}} \pi^*(\mathbb{S}_{i+1}|\mathbb{S}_i)$: Similarly, The DAG \mathbb{S}_i may also have multiple successive states \mathbb{S}_{i+1} , which means that to calculate outflow for \mathbb{S}_i , we need to sum over all of its successive states \mathbb{S}_{i+1} .

Clarification to $\sum_{1 < i < L}(\dots)$: The sampling of reasoning DAG is performed stepwisely. Thus we need to sum over all the difference between the in- and out-flows for each intermediate state \mathbb{S}_i before reaching the final state.

Clarification to $\sum_{\mathbb{S}_{L-1}} \pi^*(\mathbb{S}_L|\mathbb{S}_{L-1}) - R(\mathbb{S}_L)$: $\sum_{\mathbb{S}_{L-1}} \pi^*(\mathbb{S}_L|\mathbb{S}_{L-1})$ means the total amount of in-flow to the final state (the final DAG \mathbb{S}_L). This term should be optimized to be equal to the reward $R(\mathbb{S}_L)$.

Overall, by minimizing $L(\theta)$, we can enforce the total in- and out-flow for intermediate states being equal while ensure the probability of the final state being proportional to the reward.

A.3 The Training Process for H-GFlowNet

We formalize the training process of H-GFlowNet for reasoning DAG generation in Algorithm 2.

B Prompt Templates

B.1 Prompt for Hierarchical Cognitive Node Extraction

The prompt for tagging seed data with three-level cognitive nodes is shown in Table 8.

B.2 Prompt for Reward Model in GFlowNet

The prompt template for reward model of GFlowNet is demonstrated in Table 9.

B.3 Prompt for Question Generation

The prompt templates for iterative question generation can be found in Table 10 for editor and Table 11 for suggester.

B.4 Prompts for Response Generation

The prompt for reasoning structure aware response generation with a Plan-and-Solve style is shown in Table 12.

C Experimental Setup Details

C.1 Benchmarks and Baselines

Benchmarks We conduct comprehensive experiments on various benchmarks focusing on the core capabilities of LLMs, including instruction following, math and code. For instruction following, we choose AlpacaEval2 (Dubois et al., 2024) and IFEval (Zhou et al., 2023) as benchmarks. For math task, we include GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) are selected to evaluate code capabilities.

Baselines For each task, we compare our method with models from various sources. They include:

- **Proprietary Models** These include leading LLMs like OpenAI’s GPT-3.5 and GPT-4 (OpenAI, 2023), Gemini Pro (Team et al., 2024) and Claude 3.0.
- **General Instruction-Tuned Models** We compare our methods with popular instruction-finetuned models such as LLaMA3 (Dubey et al., 2024), Tulu3 (Lambert et al., 2024) and Qwen2 (Yang et al., 2024a), Mistral (Jiang et al., 2023), which are trained on non-disclosed large scale post-training datasets. Besides, we also include models in which synthetic datasets play a role, such as MAGPIE (Xu et al., 2024), Orca-2 (Mitra et al.,

Algorithm 2 Training Process of H-GFlowNet with Flow-matching Loss

Require: Policy network: $\pi^* = (\pi^a, \pi^c, \pi^e)$, DAG growth step limits: (L_{\min}, L_{\max}) , Training iterations: num_iter, Reward model: $R(\mathbb{S})$

Ensure: Trained policy network: $\pi^* = (\pi^a, \pi^c, \pi^e)$

```
1: Iteration counter: iter  $\leftarrow$  1
2: while iter  $\leq$  num_iter do ▷ Start training iterations
3:   Growth steps:  $L \sim \text{randint}(L_{\min}, L_{\max})$ 
4:   Growth step counter:  $i \leftarrow 0$ 
5:   # Lists for saving intermediate in- and out-flows:
6:   current_state_list  $\leftarrow$  [], parent_state_list  $\leftarrow$  [], reward_list  $\leftarrow$  []
7:   while  $i < L$  do ▷ Start to sample one reasoning DAG
8:     Sample next state:  $\mathbb{S}_i \sim \pi^*(\mathbb{S}_i | \mathbb{S}_{i-1})$ 
9:     Find all possible parent states of  $\mathbb{S}_i$ :  $\bar{\mathbb{S}}_{i-1} = \text{find\_all\_possible\_parent\_to\_s\_i}(\mathbb{S}_i)$ 
10:    if  $i == L$  then
11:      reward =  $R(\mathbb{S}_L)$  ▷ Calculate reward for final state
12:    else
13:      reward = 0 ▷ Intermediate node reward equals to 0
14:    end if
15:    # Append to lists:
16:    current_state_list.append( $\mathbb{S}_i$ )
17:    parent_state_list.append( $\bar{\mathbb{S}}_{i-1}$ )
18:    reward_list.append(reward)
19:    Increment growth step counter:  $i \leftarrow i + 1$ 
20:  end while
21:  # Calculate all possible next states for recorded current states (out-flow):
22:  next_state_list =  $\pi^*(\text{current\_state\_list})$ 
23:  # Calculate total out-flows:
24:  total_out_flows = calculate_out_flow(current_state_list, next_state_list)
25:  # Calculate total in-flows:
26:  total_in_flows = calculate_in_flow(current_state_list, parent_state_list)
27:  # Compute loss:
28:  loss = (total_in_flows - total_out_flows +  $R(\mathbb{S}_L)$ )2
29:  # Backpropagate gradients:
30:  loss.backward()
31:  Increment iteration counter: iter  $\leftarrow$  iter + 1
32: end while
33: return  $\pi^* = (\pi^a, \pi^c, \pi^e)$ 
```

Prompt template for tagging three-level cognitive node for seed data.

You are given a <question, answer> pair and your task is to identify what capabilities of human are utilized underlying this <question, answer> pair.

The capabilities in problem solving can be represented by a coarse-to-fine H-Schema of the nodes, which are defined as abstract-level, competency-level and example-level nodes. In specific,

- abstract-level node: defined as the general cognitive ability of human in problem solving, such as: perception, understanding, decision_making, verification, utilizing_knowledge, reasoning, critical_thinking, creativity, planning and so on.
- competency-level node: defined as the more specific abilities within each abstract-level node. For example, mathematics, physics, programming are competency-level nodes for utilizing_knowledge.
- example-level node: defined as how a competency-level node is used in the context of a specific question. For example, the competency-level node mathematics can have addition, multiplication, bayes_theorem_for_medical_test, law_of_large_number_for_insurance_price_estimation. They must be related to a specific question.

Overall, the abstract-level node contributes to a general capability of human in problem solving, and each abstract-level has multiple competency-level nodes that are corresponding to more specific abilities and knowledge. Furthermore, the example-level node indicates how specific ability is applied in a specific question.

You are given the following <question, answer> pair:

Question: <<question>>

Answer: <<answer>>

Steps by Step Guidance:

- First you need have an overall understanding of the given question.
- Second, identify individual steps in the answer to solve the question.
- You need to tag each identified step with an example-level node with lower case phrases joined with underscores. The example-level nodes should be specific to the question.
- Categorize each example-level node to its corresponding competency-level node and more general abstract-level node with lower case phrases joined with underscores.

Please think step by step and put these three-level nodes for all steps in a JSON list: [{"example_level": str, "competency_level": str, "abstract_level": str}].

Give your thinking process and the final answer in this format.

Thought

Step1:

<understand_question>

Step2:

<identify_steps>

Step3:

<tag_instance_level_nodes>

Step4:

<categorize_competency_and_abstract_level_nodes>

Result

<your_final_json>

Table 8: Prompt for tagging seed data with three-level cognitive nodes. <<question>> and <<answer>> will be replaced with question and answer from the seed data.

Prompt template for reward model to train GFlowNet.

The reasoning steps required to solve a question can be formulated by Directed Acyclic Graphs (DAGs). You are an advanced language model tasked with analyzing DAGs to assess the quality of questions they represent. Your objective is to evaluate and score the DAG based on the educational and cognitive value of the question it represents, assigning a score from 0 to 10. A score of 0 implies a low-quality question, and a score of 10 indicates a high-quality question.

Criteria for Evaluating Question Quality:

- Breadth of Inquiry (Score 0-10): Assess how well the DAG reflects diverse aspects of the question, showcasing a broad range of concepts or topics.
- Depth of Understanding Required (Score 0-10): Evaluate the complexity and depth of the DAG. A high-quality question should involve multi-layered reasoning or comprehension.
- Logical Structure and Cohesion (Score 0-10): Consider the clarity and cohesiveness of the problem-solving steps. A well-structured DAG implies a clear, logical approach to the question.
- Critical Thinking and Key Steps (Score 0-10): Analyze the importance and role of pivotal nodes in the DAG. High-quality questions often require critical thinking at key steps.
- Innovative Thinking Potential (Score 0-10): Evaluate the balance between convergent and divergent paths in the DAG. A stimulating question encourages exploration and synthesis of ideas.
- Potential for Insight and Generalization (Score 0-10): Consider whether the DAG suggests patterns or insights that could be generalized to other contexts, reflecting a question's broader applicability or innovation.
- Complex Problem Solving (Score 0-10): Examine opportunities within the DAG for engaging with complex, parallel processes, reflecting questions that challenge conventional thinking.

Here is the given DAG: <<the_given_dag>>

Step by Step Guidance:

- Step1: First, you need to understand how the main intent and the logic flows in the given DAG.
- Step2: Based on the understanding, please examine each above criteria one by one and give a score for each criteria based on the statements in the criteria.
- Step3: Calculate the average score.

Please think step by step and put the final result in a JSON: {"score": float, "summary": <concise paragraph summarizing the strengths and weaknesses of the question based on the DAG analysis.> }
Formulate your thinking process and the final answer in this format:

Thought

Step1:

<understand_the_dag>

Step2:

<examine_each_criteria_and_score_dag>

Step3:

<calculate_average_score>

Result

<your_final_json>

Table 9: Prompt template for scoring the reasoning DAG for GFlowNet training. <<the_given_dag>> will be replaced by the nodes and edges of a reasoning DAG.

2023), Instruct-SkillMix (Kaur et al., 2024) and Evol-Instruct (Xu et al., 2023).

• **Instruction-Tuned Models for Math** For math capabilities, we mainly compare the

Editor prompt template for the initial question generation.

Design a question that requires one step of problem solving:

Step: <<steps>>

Requirements:

- First, you need to replace the step with more specific one based on its meaning.
- Generate a question that need only one step to be solved.
- The step information should not explicitly appear in the question.
- You should incorporate specific data in your question if necessary.

Return the question as the following format:

Analysis:

<your_analysis>

Question:

<generated_question>

Editor prompt template for the iterative question generation.

The thinking steps of a LLM for a question can be formulated by a directed acyclic graph (DAG). The node of the DAG indicates the thought and action in a particular step and the edge of DAG shows the dependency between nodes.

You are given a question and its corresponding problem-solving step as a DAG as follows:

Question: <<question>>

DAG: <<dag>>

Task:

The user wants to add one more step to the original DAG and the new DAG becomes: <<new_dag>>

Your task is to evolve the original question to a new question which uses the new DAG as the problem-solving steps. The user give you some suggestion on how to achieve this evolving:

Suggestions: <<suggestion>>

Step by Step Guidance:

- You need to understand original question and original DAG.
- Understand the logic flow of the new DAG and what is the difference between original DAG and the new DAG.
- Choose the best suggestion based on your above analysis.
- Based on this suggestion, you need to evolve the original question into a new question. The problem-solving steps of new question should coincide with the new DAG.

Requirements:

- You should incorporate specific data into the question if necessary.
- You can change most content of the original question such that the evolved question should be exactly solved under the guidance of the logic flow of the new DAG.
- Do not show any DAG's information or logic flow of the new DAG in the evolved question.
- Try not to make the evolved question too verbose.

Please think step by step and give your thinking process and the final answer in this format.

Thought:

Step1: <understand_question_and_dag>

Step2: <understand_logicflow_of_new_dag_and_difference>

Step3: <the_best_suggestion>

Step4: <rephraing_the_question>

Result:

<final_evolved_question>

Table 10: Prompt template for the editor for question generation.

Suggester prompt template for the question generation.

The thinking steps of a LLM for a question can be formulated by a directed acyclic graph (DAG). The node of the DAG indicates the thought and action in a particular step and the edge of DAG shows the dependency between nodes.

You are given a question and its corresponding problem-solving step as a DAG as follows:

Question: <<question>>

DAG: <<dag>>

Task:

The user wants to add one more step to the original DAG and the new DAG becomes:

<<new_dag>>

Your task is to make suggestions on how to the evolve original question, such that the new DAG can be used as the problem-solving steps to solve the evolved question.

Step by Step Guidance:

- You need to understand the original question and original DAG.
- Understand the logic flow of the new DAG and what is the difference between original DAG and the new DAG.
- Based on the differences between previous and new logic flow of DAG, make 3 ~ 5 specific suggestions on how to evolve the original question such that the problem-solving steps of evolved question can be represented as the new DAG.

Output Format:

Briefly return the final suggestions in a List: ["<first_suggestion>", "<second_suggestion>", ...].

Please think step by step and give your thinking process and the final answer in this format.

Thought:

Step1: <understand_question_and_dag>

Step2: <understand_logicflow_of_new_dag_and_difference>

Step3: <evolve_suggestions>

Result:

<your_final_list_without_comments>

Table 11: Prompt template for the suggester for question generation.

Plan-and-Solve style prompt template for the response generation.

Give the question: <<question>>

Here is a plan to solve the question: <<plan>>

Please follow the first understand the question and the plan. Then solve the question step by step, and show the answer.

Table 12: Generate the response based on the plan derived from the reasoning DAGs.

models especially trained on synthetic mathematical reasoning dataset, which includes MathGenie (Lu et al., 2024), MathScale (Tang et al., 2024), MetaMath (Yu et al., 2023) and WizardMath (Luo et al., 2023a). These models are trained based on LLaMA2 (Touvron et al., 2023) with supervised finetuning.

- **Instruction-Tuned Models for Code**
This series of models include CodeLlama-Instruct (Roziere et al., 2023) and DeepSeek-

Coder-Instruct (Guo et al., 2024) which are fine-tuned on large scale coding instructions. Besides, we also present the performance of WizardCoder (Luo et al., 2023b), MagicCoder (Wei et al., 2024b) and SelfCodeAlign (Wei et al., 2024a) which are post-trained on synthetic datasets especially focusing on programming capabilities.

C.2 Model training details

GFlow Training Details Table 13 presents the parameters for constructing and training the hierarchical GFlowNet. We train H-GFlowNet on NVIDIA Tesla A100 GPUs and the reward model in H-GFlowNet is Qwen2.5-72B-Instruct, which is deployed on the same cluster through vllm (Kwon et al., 2023b). We adopt most parameters in vanilla GFlowNet (Bengio et al., 2021a) with the some adaptations.

Hyper-parameter	Value
Learning Rate	5×10^{-4}
Iterations	250k
Maximum Dag Depth	10
Adamw Optimizer	$\beta_s = (0.9, 0.999)$ $\epsilon = 10^{-8}$
Batch Size	4
Model-Structures	Value
# Hidden	256 for each hierarchy
# GNN Layers	10 for each hierarchy
Loss ϵ	$2.5e^{-5}$
Reward T	10
Reward β	10
Exploration Prob.	0.05
Leaf loss coeff. λ_T	10
R_{min}	0.01

Table 13: The detailed parameters for constructing and training of the H-GflowNet.

LLM Training Details All experiments are conducted on NVIDIA Tesla A100 GPUs. All models are open-sourced, which are trained through fully supervised fine-tuning for all these 3 tasks.

- **Fine-tuning LLaMA3 (Dubey et al., 2024) Series Models (Instruction following task):**
 - Optimizer: AdamW
 - LR Schedule: Cosine
 - LR: $2e^{-5}$
 - Maximum Sequence Length: 4096
 - Effective Batch Size: 32 for 8B; 4 for 70B
 - Training Epochs: 2
- **Fine-tuning LLaMA2 (Touvron et al., 2023) Series Models (Math task):**
 - Optimizer: AdamW
 - LR Schedule: Cosine
 - LR: $2e^{-5}$, with a 3% learning rate warm-up
 - Maximum Sequence Length: 2048
 - Effective Batch Size: 128

Datasets	13-Gram Match	8-Gram Match	Total Size
AlpacaEval2	1	4	40k
GSM8K	5	44	20k
HumanEval	2	32	20k

Table 14: The contamination test of our datasets to evaluation benchmarks measured by n-gram matches.

- Training Epochs: 3
- **Fine-tuning DeepSeek-Coder (Guo et al., 2024) Series Models (Code task):**
 - Optimizer: AdamW
 - LR Schedule: Cosine
 - LR: $2e^{-5}$
 - Maximum Sequence Length: 2048
 - Effective Batch Size: 128
 - Training Epochs: 3

D Contamination Test

We use the method proposed in (Liu et al., 2023) to measure the 8- and 13-gram matches between our synthetic datasets and evaluation benchmarks for instruction-following (AlpacaEval2), math (GSM8K) and coding (HumanEval) tasks. The experimental results are shown in the Table 14. From this table, we can see that our synthetic datasets do not suffer from data leakage problem.

E Cost Analysis

In our data synthesis pipeline, for building the H-Schema and the reward function for H-GFlowNet training, we use the self-deployed Qwen2.5-72B-Instruct, which is cost-effective. Moreover, because the H-Schema and H-GFlowNet are built once, we can effectively generate reasoning DAGs for down-stream tasks with filtering by node names. Furthermore, we use GPT-4 to generate instructions and responses, in which we estimate that it costs \$18 for generating $1k$ samples.

F Case Study

From Figure 7 we showcase a sampled DAG from H-GFlowNet. It can be found that the generated DAG can cover diverse skills in human problem-solving, such as in mathematics, finance and reasoning. Besides, these skills are logically connected with rich divergent and convergent connections. For example, *hypothesis testing* and *quadratic inequality* both contribute to the *find causal pattern in sequence* node, which indicates

that *find causal pattern in sequence* has to digest the information from both of its parental node and then conduct corresponding causal pattern recognition. Moreover, Table 15 shows how the iterative question construction pipeline is performed based on this reasoning DAG. For instance, at iteration 4, we need to append *find causal pattern in sequence* after *hypothesis testing*. The suggester provides 5 specific options, and editor incorporates time-series data of past allocations into the original question.

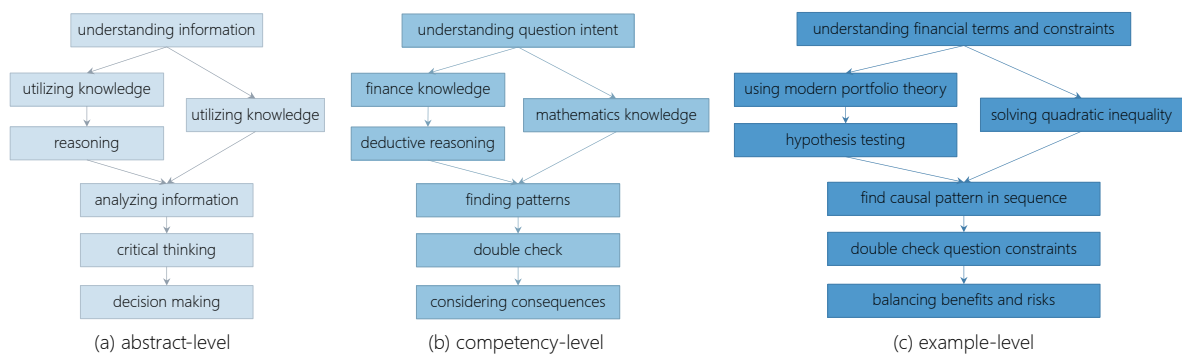


Figure 7: A case study of sampled DAGs from H-GFlowNet corresponding to the each-level of H-Schema.

Iteration: 0	
DAG:	node: ["understanding_financial_terms_and_constraints"], edge: []
Initial Question	A company has a marketing budget of \$50,000 for the next quarter. If they allocate 40% of this budget to digital advertising, how much money will be left for other marketing activities?
Iteration: 1	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory"] edge: [[0, 1]]
Suggester:	["Add a risk assessment component to the budget allocation process.", " Include expected ROI for different marketing activities to optimize allocation. ", "Introduce the need for diversifying marketing efforts across multiple channels.", "Ask how to strategically allocate the remaining budget among various activities.", "Incorporate scenario analysis affecting budget decisions."]
Editor:	A company has a marketing budget of \$50,000 for the next quarter. After allocating 40% to digital advertising, they want to distribute the remaining budget across three channels: social media campaigns (12% risk, 8% ROI), influencer partnerships (18% risk, 10% ROI), and email marketing (6% risk, 4% ROI). To minimize overall risk while ensuring a weighted average ROI of at least 6.5%, how much should they allocate to each channel?
Iteration: 2	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory", "quadratic_inequality"] edge: [[0, 1], [0, 2]]
Suggester:	[" Include quadratic risk variance minimization alongside linear ROI constraints ", "Add a quadratic constraint on the combined risk variance of allocations", "Introduce a penalty term in the risk calculation requiring quadratic programming", "Require balancing ROI and risk with quadratic weighting"]
Editor:	A company has a marketing budget of \$50,000 for the next quarter. After allocating 40% to digital advertising, they want to distribute the remaining budget across three channels social media campaigns (historical risk variance of 0.015, 8% ROI), influencer partnerships (historical risk variance of 0.025, 10% ROI), and email marketing (historical risk variance of 0.005, 4% ROI). The covariance between social media and influencer risks is 0.008, social media and email risks is -0.002, and influencer and email risks is 0.004. To minimize the total risk variance of the allocation while ensuring a weighted average ROI of at least 6.5%, how much should they allocate to each channel?
Iteration: 3	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory", "quadratic_inequality", "hypothesis_testing"] edge: [[0, 1], [0, 2], [1, 3]]
Suggester:	["Add a requirement to statistically validate if the optimal allocation's risk-return profile significantly outperforms a previous quarter's allocation using historical data.", "Introduce a need to test the significance of the ROI difference between influencer partnerships and social media campaigns to justify their allocation proportions.", "Require verification that the covariance assumptions between channels are statistically valid based on historical risk data before finalizing the allocation.", " Include a step to compare the optimized portfolio's risk variance against a benchmark variance (e.g., industry standard) using hypothesis testing to ensure compliance. "]

Editor:	A company has a marketing budget of \$50,000 for the next quarter. After allocating 40% to digital advertising, they want to distribute the remaining budget across three channels social media campaigns (historical risk variance of 0.015, 8% ROI), influencer partnerships (historical risk variance of 0.025, 10% ROI), and email marketing (historical risk variance of 0.005, 4% ROI). The covariance between social media and influencer risks is 0.008, social media and email risks is -0.002, and influencer and email risks is 0.004. To minimize the total risk variance of the allocation while ensuring a weighted average ROI of at least 6.5%, how much should they allocate to each channel? Additionally, the company requires statistical validation at a 95% confidence level that the optimized allocation's risk variance is lower than the industry benchmark variance of 0.012.
Iteration: 4	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory", "quadratic_inequality", "hypothesis_testing", "find_causal_pattern_in_sequence"] edge: [[0, 1], [0, 2], [1, 3], [3, 4]]
Suggester:	["Require identifying causal relationships between historical allocation sequences and risk outcomes after hypothesis testing.", " Incorporate time-series analysis of past budget allocations to detect causal patterns influencing risk and ROI. ", "Add a causal inference step to validate that the optimized allocation strategy directly causes reduced risk variance.", "Introduce a requirement to analyze sequential quarterly data to find causal drivers of marketing channel performance.", "Mandate causal discovery from historical data to ensure allocations' effects are not merely correlational but causally linked to outcomes."]
Editor:	A company has a marketing budget of \$50,000 for the next quarter. After allocating 40% to digital advertising, they want to distribute the remaining budget across three channels social media campaigns (historical risk variance of 0.015, 8% ROI), influencer partnerships (historical risk variance of 0.025, 10% ROI), and email marketing (historical risk variance of 0.005, 4% ROI). The covariance between social media and influencer risks is 0.008, social media and email risks is -0.002, and influencer and email risks is 0.004. Historical quarterly data shows sequences of past allocations (e.g., 2022-Q1 [25%, 35%, 40%], 2022-Q2 [30%, 30%, 40%]) and corresponding risk variances. To minimize total risk variance while ensuring a weighted average ROI of at least 6.5%, how much should they allocate to each channel, and which allocation patterns from past data causally drive reduced risk?
Iteration: 5	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory", "quadratic_inequality", "hypothesis_testing", "find_causal_pattern_in_sequence"] edge: [[0, 1], [0, 2], [1, 3], [3, 4], [2, 4]]
Suggester:	["Require comparing optimized allocations with historical patterns to identify which past allocations align with the optimal risk-return tradeoff derived from quadratic constraints.", "Introduce a need to validate if historical allocations satisfy the quadratic inequality conditions derived for the current ROI constraint.", " Ask to integrate the quadratic optimization results with causal pattern discovery to filter past allocations that are mathematically feasible under the new constraints. ", "Add a requirement to reconcile discrepancies between theoretical optimal allocations (via quadratic programming) and empirically successful historical patterns through causal analysis."]

Editor:	A company has a \$50,000 marketing budget, allocating 40% to digital ads first. The remaining funds go to social media (0.015 risk variance, 8% ROI), influencer partnerships (0.025 risk variance, 10% ROI), and email marketing (0.005 risk variance, 4% ROI). Covariances between risks are 0.008 (social-influencer), -0.002 (social-email), and 0.004 (influencer-email). Historical allocations include 2022-Q1 [25%, 35%, 40%], 2022-Q2 [30%, 30%, 40%], and 2023-Q1 [20%, 40%, 40%], with observed risk variances of 0.012, 0.014, and 0.011 respectively. Determine the optimal allocations to minimize total risk variance while guaranteeing 6.5% ROI. Additionally, identify which historical allocations satisfy the ROI constraint's quadratic inequality conditions and explain how these mathematically compliant patterns causally correlate with lower historical risks.
Iteration: 6	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory", "quadratic_inequality", "hypothesis_testing", "find_causal_pattern_in_sequence", "double_check_question_constraints"] edge: [[0, 1], [0, 2], [1, 3], [3, 4], [2, 4], [4, 5]]
Suggester:	["Introduce an additional constraint requiring verification of budget reallocation limits per channel post-analysis", "Require cross-validation of ROI constraints against multiple historical periods after identifying causal patterns", "Add a secondary risk tolerance threshold that must be confirmed after initial optimization", "Include a regulatory compliance check step after establishing causal patterns to ensure allocations meet new disclosure laws", "Introduce a dynamic ROI target adjustment mechanism that needs post-optimization verification"]
Editor:	A company has a \$50,000 marketing budget, allocating 40% to digital ads first. The remaining funds go to social media (0.015 risk variance, 8% ROI), influencer partnerships (0.025 risk variance, 10% ROI), and email marketing (0.005 risk variance, 4% ROI). Covariances between risks are 0.008 (social-influencer), -0.002 (social-email), and 0.004 (influencer-email). Historical allocations include 2022-Q1 [25%,35%,40%], 2022-Q2 [30%,30%,40%], 2022-Q3 [28%,32%,40%], 2023-Q1 [20%,40%,40%], and 2023-Q2 [22%,38%,40%], with observed risk variances of 0.012, 0.014, 0.013, 0.011, and 0.010 respectively. Determine the optimal allocations to minimize total risk variance while guaranteeing 6.5% ROI. Identify which historical allocations satisfy the ROI constraint's quadratic inequality conditions and explain their correlation with lower risks. Finally, validate whether these compliant patterns consistently hold when cross-referenced against Q3 2022 and Q2 2023 performance data, specifying any deviations requiring constraint relaxation.
Iteration: 7	
DAG:	node: ["understanding_financial_terms_and_constraints", "modern_portfolio_theory", "quadratic_inequality", "hypothesis_testing", "find_causal_pattern_in_sequence", "double_check_question_constraints", "balancing_benefits_and_risks"] edge: [[0, 1], [0, 2], [1, 3], [3, 4], [2, 4], [4, 5], [5, 6]]
Suggester:	["Require maximizing ROI alongside minimizing risk, necessitating a balance between the two objectives in the optimal allocation.", "Introduce a trade-off parameter between risk and ROI, asking for allocations that optimize a combined utility function considering both factors.", "Add a secondary constraint on ROI variability, requiring the solution to balance between achieving consistent returns and minimizing risk variance.", "Include a scenario where the company seeks a Pareto-optimal solution, explicitly demanding analysis of risk-benefit equilibria post-constraint verification."]

Editor:	<p>A company has a \$50,000 marketing budget, allocating 40% to digital ads first. The remaining funds go to social media (0.015 risk variance, 8% ROI), influencer partnerships (0.025 risk variance, 10% ROI), and email marketing (0.005 risk variance, 4% ROI). Covariances between risks are 0.008 (social-influencer), -0.002 (social-email), and 0.004 (influencer-email). Historical allocations include 2022-Q1 [25%,35%,40%], 2022-Q2 [30%,30%,40%], 2022-Q3 [28%,32%,40%], 2023-Q1 [20%,40%,40%], and 2023-Q2 [22%,38%,40%], with observed risk variances of 0.012, 0.014, 0.013, 0.011, and 0.010 respectively. Determine Pareto-optimal allocations balancing risk minimization and ROI maximization. Identify which historical allocations lie on the Pareto frontier and explain their risk-return trade-offs. Validate if these allocations maintain Pareto efficiency when cross-verified with Q3 2022 and Q2 2023 performance data, specifying any deviations necessitating adjustments to the risk-return equilibrium.</p>
----------------	---

Table 15: The example to show the iterative mechanism of suggester-editor workflow for question generation. The outputs marked with red color are chosen suggestions which are used to evolve the question by the editor.

G Error Analysis

In this section, we inspect the failures of the answers in our synthetic method and analyze the causes of errors. Most of the failed cases come from the subsequent solution-solving steps. Even with a reasonable question and plan generated by a high-quality DAG sampled from our H-GFlowNet, LLM can not solve the question correctly due to its limited capabilities to solve difficult questions. The following are two typical examples that provide insight into the details of errors. Those errors can be potentially addressed by combining reasoning DAGs with more advanced inference methods such as MCTS.

In error Case 1, the plan and the solution have already divided the question into smaller steps using the Product Rule, and they further adjust the strategy to use the Chain Rule after realizing that $v(x)$ is a composition function. However, the response model still gives an incorrect derivative for $v(x)$, although our reasoning DAG provides a correct solution plan.

The error Case 2 generated question asks respondents to compute the probability that an incoming request is blocked. By examining the plan derived from our DAG and the intermediate results, we can verify its correctness to lead to the final solution. However, the LLM struggles with complex computations, particularly when calculating the exact value in scientific notation in the final step.

Error Case 1	
DAG:	node: ["access_external_resources","identifying_patterns_or_series","identify_differences","detailing_tasks_into_smaller_steps","replicating_text_without_alteration","identify_keywords_or_highlights","matching_information_to_question","adjusting_strategy_based_on_new_insights","confirming_accuracy_and_correction","mathematics_calculus"], edge: [[0,1],[1,2],[2,3],[3,4],[4,5],[4,6],[6,7],[4,8],[3,9]]
Question	Find the derivative of the function $f(x) = x^3 \cos e^{2x}$.
Plan	... we will use the product rule. ... Moreover, for v(x), we need to use the Chain Rule ... We will calculate the derivatives of u(x) and v(x) separately and then apply the product rule and the chain rule for v(x).
Answer	Certainly! Let's go through the problem step by step. Problem: Find the derivative of the function $f(x) = x^3 \cdot \cos e^{2x}$. Step-by-Step Solution: ... Step 4: Apply the Chain rule formula For the function $v(x) = \cos e^{2x}$, using the chain rule, = $-2\sin(e^{2x})$ Final Answer: The derivative of $f(x) = x^3 \cdot \cos e^{2x}$ is: $\frac{d(f(x))}{dx} = -2x^3 \sin e^{2x} + 3x^2 \cos e^{2x}$
Error Case 2	
DAG:	node: ["brainstorm_innovative_designs","apply_advanced_network_troubleshooting_techniques","assess_the_problem_statement","utilize_STEM_knowledge_Mathematics","considering_infrastructure_compatibility","deductive_reasoning","aggregating_data","matching_information_to_question","follow_and_apply_specific_formats_in_the_answer"] edge: [[0,1],[1,2],[2,3],[3,4],[2,5],[2,6],[2,7],[1,8]]
Question	An online shopping website wants to improve their trading service network. It receives an average of 1800 trading requests per hour. Each request takes an average of 2 minutes to process on its CPU servers. The website has 70 CPUs available. Suppose the trading service network is an $M/M/k/k$ system. Calculate the probability that an incoming request is blocked step by step. Provide your answer in scientific notation.
Plan	... use Erlang B formula to calculating the probability that all servers are busy in such a system. Erlang B is given by: $P_{block} = \frac{A^k/k!}{\sum_{n=0}^k (A^n/n!)}$ where offered traffic $A = \frac{\lambda}{\mu}$, λ is the arrival rate and μ is the service rate per server. By substituting the formula with the given value, we can calculate the probability that an incoming request is blocked.
Answer	... Plug these into the Erlang B formula: ... calculating $\frac{60^{70}}{70!}$ and $\sum_{n=0}^7 0(A^n/n!)$ requires handling very large numbers computation-ally, often assisted by specialized software or calculators. $P_{block} = 1.5 \times 10^{-5}$

Table 16: 2 examples to show the error cases in our generated data. The outputs marked with red color are where the incorrectness happen.