

MuSe: Multi-Stage Graph Reasoning via Vision-Language Models

Guanyu Wang[†], Xu Chu[†], Zhijie Tan, Xinrong Chen, Tong Mo^{*}, Weiping Li

School of Software and Microelectronics, Peking University, Beijing, China

{wgy2023, chuxu, besttangent, chenxinrong23}@stu.pku.edu.cn

{motong, wpli}@ss.pku.edu.cn

[†]Equal contribution

^{*}Corresponding author

Abstract

Graph-related tasks are traditionally addressed with Graph Neural Networks (GNNs) or graph transformers, but their task-specific training limits generalization. Large Language Models (LLMs) offer stronger generalization, yet encoding graphs as one-dimensional text struggles to capture multi-hop dependencies and two-dimensional topology. Vision-Language Models (VLMs) provide an alternative by visualizing graphs, but rendering large graphs in a single image causes clutter, occlusion, and distraction, hindering reasoning. We propose MuSe, a novel multi-stage graph reasoning framework based on VLMs. Instead of processing entire graphs at once, MuSe incrementally samples and visualizes task-relevant subgraphs, enabling progressive reasoning. The framework employs a two-stage training paradigm: supervised fine-tuning to acquire local sampling and reasoning skills, followed by reinforcement learning with GRPO to refine the sampling strategy and control dialog length. To support evaluation, we introduce LGVLQA, a new multimodal dataset with larger and more complex graph structures, addressing the scalability limitations of existing benchmarks. Experiments show that MuSe consistently outperforms leading LLM and VLM baselines, demonstrating improved structural understanding and reasoning ability. Our code and data are available at [this url](#).

1 Introduction

Graphs are widely used to represent complex relationships and interactions, playing a central role in domains such as social networks (Cao et al., 2020; Huang et al., 2021), knowledge graphs (Zhang et al., 2021; Liu et al., 2022; Wei et al., 2024b), and recommender systems (He et al., 2020; Wang et al., 2021). Efficiently understanding graph structures and performing reasoning over them remains a major challenge in artificial intelligence. Traditional approaches rely on Graph Neural Networks

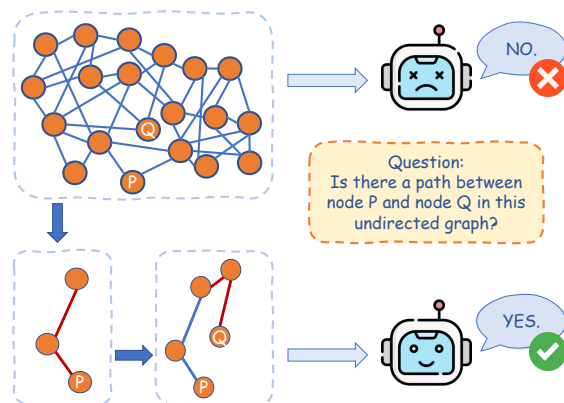


Figure 1: Existing VLM-based approaches typically visualize the entire graph within a single image, which often results in visual clutter in large graphs. Such full-graph visualization also introduces irrelevant information that hinders reasoning and increases computational cost. In contrast, performing multiple local samplings and visualizing subgraphs can mitigate these issues.

(GNNs) (Jiang et al., 2019; Xu et al., 2018) and graph transformers (Zhang et al., 2020; Kreuzer et al., 2021; Chen et al., 2022), which excel at modeling local dependencies and structural relations. However, these methods often suffer from limited generalization, task-specific architectural design and data requirements, and weak interpretability.

Recently, Large Language Models (LLMs) have shown remarkable general-purpose reasoning capabilities in natural language processing (Wei et al., 2022; Yu et al., 2023; Zhou et al., 2022; Jiang et al., 2023a,b; Li et al., 2025; Zhang et al., 2025), and have been increasingly applied to graph-related tasks. Compared to GNN-based methods, LLMs offer strong generalization and notable advantages in zero-shot/few-shot reasoning and multi-task learning. Many studies convert graph structures into textual descriptions and feed them to LLMs, achieving promising results in several graph reasoning tasks (Fatemi et al., 2023; Wang et al., 2023).

However, whether LLMs can truly capture the two-dimensional topology of graphs through text sequences remains questionable (Chu et al., 2025; Chen et al., 2024a; Zhang et al., 2024). Graphs fundamentally encode spatial relations among nodes, and mapping them into linear sequences often fails to preserve this structure. In particular, reasoning tasks involving multi-hop dependencies, where indirect relations must be inferred through intermediate nodes, are difficult to express with one-dimensional text. This mismatch hinders the accuracy and interpretability of reasoning.

To overcome these limitations, recent work has explored Vision-Language Models (VLMs), which visualize graphs as images to leverage visual perception of connectivity and topology (Wei et al., 2024a). This approach aligns with human intuition: when solving graph problems, people often rely on diagrammatic representations to better perceive structural relations. As shown in Figure 1, current VLM-based methods typically render the entire graph in a single image. When graphs are large, this leads to excessive edge crossings, overlap, and visual clutter, making the graph difficult to interpret. Moreover, full-graph visualization inevitably includes irrelevant information that may distract reasoning and increase computational overhead.

To address these issues, we propose **MuSe**, a novel **Multi-Stage** Graph Reasoning framework via Vision-Language Models. Instead of visualizing the entire graph, MuSe incrementally samples and renders task-relevant subgraphs, enabling progressive reasoning while avoiding irrelevant noise and visual clutter. We design a two-stage training framework: (i) supervised fine-tuning (SFT) to acquire local sampling and reasoning abilities, and (ii) reinforcement learning with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to further optimize the sampling strategy while constraining the dialog length.

In addition, we construct Large-scale Graph Vision-Language Question Answering (LGVLQA), a new multimodal dataset for large-scale graph reasoning. Compared to existing benchmarks, LGVLQA features larger and more complex graph structures, making full-graph visualization particularly challenging and filling the scalability gap of prior datasets. Experiments show that MuSe consistently outperforms leading LLM and VLM baselines on multiple graph reasoning tasks.

Our main contributions are as follows:

- We point out that current VLM-based graph un-

derstanding suffers from visual clutter when large graphs are rendered in a single image, which limits reasoning effectiveness.

- We propose MuSe, a novel multi-stage graph reasoning framework based on VLMs, which enables the model to progressively sample local subgraphs and integrate information.
- We construct LGVLQA to cover larger graph structures, addressing the limited scalability of existing datasets.
- MuSe achieves superior performance over LLM and VLM baselines, highlighting its effectiveness in structural understanding and general graph reasoning.

2 Related Works

2.1 Graph reasoning

Graph reasoning (Battaglia et al., 2018; Wu et al., 2020) aims to perform inference and decision-making based on structural relations in graphs, with broad applications in domains such as social network analysis (Newman, 2003; Leskovec et al., 2008), knowledge graph question answering (Bordes et al., 2013), recommender systems (Koren et al., 2009; He et al., 2017), and molecular property prediction (Gilmer et al., 2017). To address this problem, numerous task-specific models have been developed. Classic Graph Neural Networks (GNNs), such as GCN (Jiang et al., 2019), GAT (Veličković et al., 2017), GraphSAGE (Hamilton et al., 2017), MPNN (Gilmer et al., 2017), and GIN (Xu et al., 2018), leverage message-passing mechanisms to model local dependencies and topological structures. More recently, graph transformers (Zhang et al., 2020; Kreuzer et al., 2021; Chen et al., 2022) based on self-attention mechanisms have further enhanced representational power and the ability to capture long-range dependencies. Although these models achieve strong performance on specific tasks, they usually require training tailored to particular tasks or datasets, making it difficult to generalize across tasks. Moreover, their reasoning processes often lack transparency and interactivity, which significantly limits their applicability in user-centric scenarios.

2.2 LLMs and VLMs on Graph Reasoning

With the rise of Large Language Models (LLMs), researchers have begun exploring their general reasoning abilities for graph reasoning tasks. Existing approaches can be broadly categorized into two

directions. The first leverages LLMs as representation enhancers (Liu et al., 2023; Yang et al., 2021), where LLMs encode textual node attributes that are then fed into GNNs. While this enriches graph representations, the reasoning process still relies on conventional GNNs, and thus inherits their limitations in generalization and interpretability. The second line of work treats LLMs directly as reasoners (Li et al., 2025; Zhang et al., 2025), exploiting their strong general-purpose reasoning capabilities and zero-/few-shot adaptability for graph tasks. For example, GraphWiz (Chen et al., 2024a) and Instruct-GPT (Ye et al., 2023) transform graphs into textual descriptions and use instruction tuning to adapt to different downstream tasks. Other approaches (Fatemi et al., 2023; Guo et al., 2023) further enhance performance through prompt engineering and template-based optimization of graph descriptions. Beyond textual descriptions, approaches such as GraphGPT (Tang et al., 2024), GraphToken (Perozzi et al., 2024), and LLaGA (Chen et al., 2024b) encode graphs into specially designed token sequences mapped to the LLM’s language space, enabling reasoning within a unified text representation. However, these approaches inherently depend on sequential representations, which struggle to preserve the spatial topology of graphs, limiting their effectiveness on graph reasoning tasks.

To overcome the limitations of purely textual methods, some studies have introduced visual modalities, inspired by the human tendency to rely on diagrammatic representations when solving graph problems. GITA (Wei et al., 2024a) represents the first systematic framework of this kind, rendering graph structures as images and combining them with textual descriptions as input to a VLM. This approach achieves strong performance across multiple graph reasoning tasks, demonstrating the potential of vision-enhanced strategies. However, static visualization of the entire graph often leads to edge crossings, redundancy, and noise in large-scale graphs, which reduce structural clarity and distract reasoning. Furthermore, existing VLM-based methods lack explicit reasoning chains, limiting interpretability.

3 Method

3.1 Preliminary

In graph reasoning task, the input typically consists of two components: (i) a graph structure $G = (V, E)$, where V denotes the set of nodes and E

denotes the set of edges; and (ii) a task requirement or query Q . The objective of the model π is to generate a reasonable answer A based on (G, Q) , which can be formalized as:

$$A = \pi(G, Q) \quad (1)$$

In LLM-based graph reasoning methods, the graph G needs to be transformed into a textual representation G^T . This transformation enables reasoning within the natural language space, thereby improving interpretability and leveraging the powerful reasoning capabilities of LLMs. In this case, the reasoning process can be expressed as:

$$A = \pi(G^T, Q) \quad (2)$$

Recent studies propose to enhance graph reasoning with visual information. Specifically, the graph G is visualized as an image G^I , which is provided as an input to VLM alongside its textual form G^T . This multimodal integration facilitates a better understanding of the graph’s topology. Accordingly, the reasoning process can be formalized as:

$$A = \pi(G^I, G^T, Q) \quad (3)$$

3.2 Architecture

To address the limitations faced by current VLMs in graph reasoning, we propose a novel multi-stage graph reasoning framework based on VLMs. Instead of solving a graph reasoning problem in a single step, our approach decomposes it into a sequence of dialog interactions. A complete reasoning process is divided into multiple rounds $t = 1, \dots, T$, where at each round the model generates a response by considering the historical context and the query. The outcome of the t -th round can be represented as:

$$R_t = \pi(G^T, Q, H_{t-1}), \quad (4)$$

$$t = 1, \dots, T, \quad H_0 = \emptyset$$

where H_t denotes the historical context up to round t , which includes the model’s previous reasoning steps and the visualized images of the sampled subgraphs. The process can be expressed in a recursive form as:

$$H_t = H_{t-1} \cup \{R_t, \hat{G}_t^I\} \quad (5)$$

where \hat{G}_t^I denotes the visualized image of the subgraph sampled at round t .

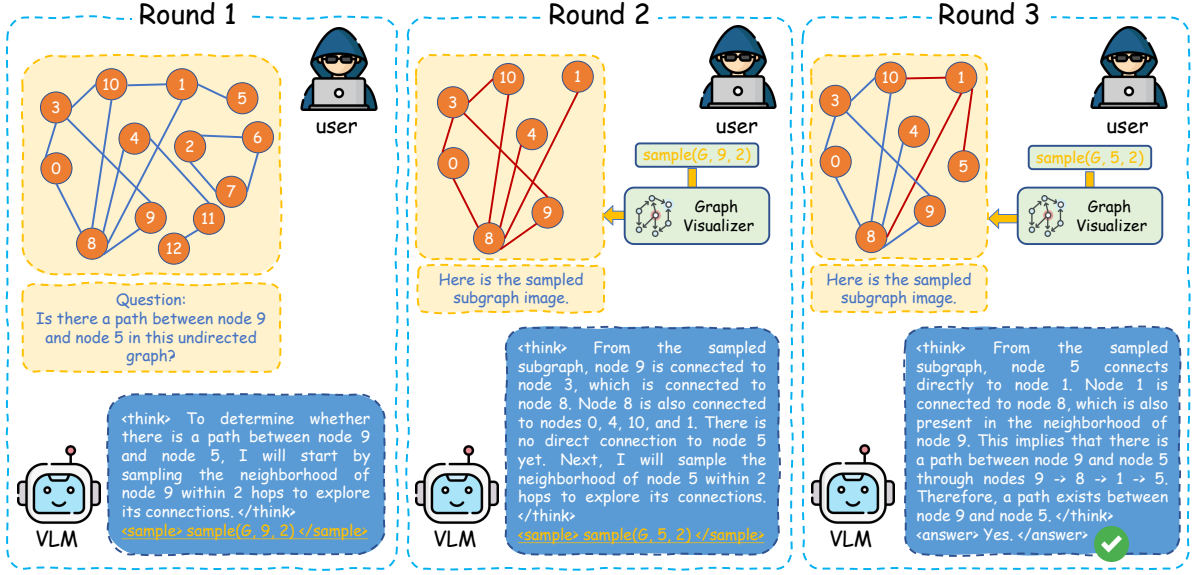


Figure 2: Overview of our method. The VLM starts reasoning from the initial inputs Q and G^T . When the information is insufficient, it outputs a sampling request. The graph visualizer samples and visualizes subgraphs based on the sampling parameters output by the model. After receiving the returned image, the model continues the reasoning process. Through iterative sampling and reasoning, the model progressively refines its understanding until it generates the final answer.

In each dialog round, the model may take one of two possible actions, either action terminates the current round of dialog:

Sample. The model can sample a subgraph from the original graph G . When sampling, the model outputs a structured text segment of the form: `<sample> sample(G, r, h) </sample>`, where r is the root node and h is the hop count. The sampling procedure is implemented by a Python utility function and adopts the standard h -hop subgraph sampling definition:

$$\begin{aligned} \text{sample}(G, r, h) &\triangleq \hat{G} = \{\hat{V}, \hat{E}\}, \\ \hat{V} &= \{v \in V \mid \text{dis}_G(v, r) \leq h\}, \\ \hat{E} &= \{(u, v) \mid u \in \hat{V}, v \in \hat{V}\} \end{aligned} \quad (6)$$

where $\text{dis}_G(v, r)$ denotes the shortest-path distance between node v and the root node r in graph G .

Answer. When the model has gathered sufficient reasoning information, it outputs the final answer in the form `<answer> ... </answer>` and terminates the multi-round dialog.

As illustrated in Figure 2, the VLM initiates reasoning based on the initial inputs Q and G^T . When the available information proves insufficient, it actively samples a subgraph and acquires its corresponding visualized representation, then incorporates the accumulated historical information to

continue the reasoning process. Through successive iterations, the VLM progressively refines its understanding and terminates the dialog once it has acquired sufficient information to produce the final answer.

3.3 Training

3.3.1 Data Construction

We employ GPT-4o to transform the original training data into Chain-of-Thought (CoT) multi-turn dialogues with a well-defined structured output format. For each question, GPT-4o engages in multiple rounds of interaction, strictly adhering to the following constraints:

- In each round, all reasoning steps must be enclosed within `<think> ... </think>` tags.
- Every sampling request must follow the format: `<sample> sample(G, r, h) </sample>`.
- Once the model outputs a sampling request, the reply must terminate immediately, awaiting the environment to return the rendered subgraph image before initiating the next round of dialogue.
- When the current information is sufficient to derive the final answer, the model should, after reasoning, output it in the format `<answer>...</answer>` and end the dialogue.

We filter the multi-turn dialogue data generated by GPT-4o, retaining only those dialogues that are

strictly format-compliant and yield correct final answers, to construct the SFT dataset D_{SFT} . This filtering step ensures that, during the SFT stage, the model learns both the rigid interaction format and the correct reasoning paradigm.

3.3.2 Supervised Fine-Tuning

During supervised fine-tuning, the model is trained on D_{SFT} . Each training instance consists of the initial input (G^T, Q) , and the corresponding multi-turn interaction trajectory H_T . The objective is to maximize the likelihood of generating the correct response sequence conditioned on the given inputs, formulated as:

$$\mathcal{L}_{SFT}(\theta) = - \sum_{t=1}^T \log P_{\theta}(R_t | G^T, Q, H_{t-1}) \quad (7)$$

The goal of SFT is to ensure that the model adheres to the predefined structured interaction format while internalizing effective reasoning and sampling strategies distilled from GPT-4o demonstrations, thereby establishing a solid foundation for subsequent GRPO optimization.

3.3.3 Reinforcement Learning with GRPO

To further refine the sampling strategy of the SFT-trained model, particularly to prevent excessive sampling steps and potential infinite loops, we introduce Group Relative Policy Optimization (GRPO) for reinforcement learning.

GRPO follows a simple yet effective principle: comparing the relative merits of multiple candidate answers to the same problem. In our setting, given an initial input (G^T, Q) , the current policy π_{old} conducts N complete multi-turn dialogues $\{d_1, \dots, d_N\}$ and obtains the corresponding rewards $\{r_1, \dots, r_N\}$. The relative advantage of each dialogue is then computed via intra-group reward normalization:

$$A_i = \frac{r_i - \text{mean}(\{r_1, \dots, r_N\})}{\text{std}(\{r_1, \dots, r_N\})} \quad (8)$$

where A_i denotes the advantage of the i -th multi-turn dialogue relative to other dialogues within the group. Specifically, We design two complementary reward signals to guide the model toward both structural compliance and reasoning correctness:

- **Format reward** (r_{fmt}): The model receives a score of 1.0 if it strictly adheres to the predefined interaction format by using `<think>...</think>` and `<sample>...</sample>` in intermediate rounds, and

`<think>...</think>` `<answer>...</answer>` in the final round. Otherwise, the score is 0.0.

- **Accuracy reward** (r_{acc}): The model receives a score of 1.0 if the final answer exactly matches the ground truth, and 0 otherwise.

Thus the overall reward is defined as:

$$r = r_{acc} + r_{fmt} \quad (9)$$

In particular, we impose a hard limit of K interaction rounds per query during GRPO training. Dialogues exceeding K rounds are truncated, resulting in both $r_{fmt} = 0$ and $r_{acc} = 0$.

Thus the GRPO objective is defined as:

$$\begin{aligned} \mathcal{L}_{GRPO}(\theta) = & \frac{1}{N} \sum_{i=1}^N (\min(\rho_i A_i, \text{clip}(\rho_i, 1 - \epsilon, 1 + \epsilon) A_i) - \beta \mathbb{D}_{KL}), \\ \mathbb{D}_{KL} = & \frac{\pi_{\text{ref}}(d_i | G^T, Q)}{\pi_{\theta}(d_i | G^T, Q)} - \log \frac{\pi_{\text{ref}}(d_i | G^T, Q)}{\pi_{\theta}(d_i | G^T, Q)} - 1 \end{aligned} \quad (10)$$

where $\rho_i = \frac{\pi_{\theta}(d_i | G^T, Q)}{\pi_{\text{old}}(d_i | G^T, Q)}$, ϵ and β control the clipping range and the intensity of KL divergence penalty respectively. π_{ref} is the reference policy, used to prevent the optimized policy π_{θ} from deviating too far and causing catastrophic forgetting.

	GVLQA	LGVLQA
Task	Nodes	Nodes
Connect	[2,35]	[2,100]
Cycle	[2,35]	[2,100]
SP	[2,20]	[2,100]
MaxFlow	[2,20]	[2,50]
BGM	[2,33]	[2,100]
HP	[2,20]	[2,50]
TS	[2,35]	[2,50]

Table 1: Comparison of node count ranges between GVLQA and LGVLQA for different reasoning tasks.

4 Experiment

4.1 Experiment Setup

Datasets. We adopt GVLQA (Wei et al., 2024a) as our primary evaluation benchmark. GVLQA is a graph visual-language question answering dataset supporting multi-modal inputs, and covers seven representative graph reasoning tasks: connectivity judgment (**Connect**), cycle detection (**Cycle**), shortest path (**SP**), maximum flow (**MaxFlow**), bipartite graph matching (**BGM**), Hamiltonian path (**HP**), and the topological sort (**TS**). However, the

Dataset	GVLQA								
	Models	Connect	Cycle	TS	SP	MaxFlow	BGM	HP	Avg
GCN	79.6	70.8	-	-	10.3	76.7	-	-	
GAT	80.9	72.6	-	-	11.2	79.8	-	-	
GraphSAGE	82.7	73.5	-	-	11.6	81.0	-	-	
SAT	83.3	72.9	-	-	12.5	80.3	-	-	
LLaMA2-7B	97.3	94.6	33.2	26.0	9.5	90.8	23.9	53.6	
Vicuna-7B	97.5	95.0	34.4	25.9	9.3	91.0	25.5	54.1	
LLaMA3-8B	97.9	96.2	33.2	28.3	11.5	92.6	27.9	55.4	
Qwen2.5-7B	96.4	95.5	31.7	27.2	14.3	91.8	26.3	54.7	
GraphWiz	95.5	92.7	36.4	24.9	12.3	89.3	28.2	54.2	
TAPE	85.9	93.4	-	-	22.3	83.9	-	-	
LLAGA	89.2	89.4	34.4	27.6	16.7	87.9	27.6	53.3	
Qwen2.5-VL-7B	98.2	96.8	39.6	31.8	20.4	92.5	28.5	58.5	
InternVL2.5-8B	98.4	95.6	38.7	29.4	18.9	92.6	30.4	57.7	
GITA-7B	98.8	96.7	41.1	32.1	20.0	93.2	29.2	58.7	
MuSe (Ours)	99.4	97.4	42.7	35.3	22.6	94.0	33.8	60.7	

Table 2: Performance comparison of models on GVLQA tasks. We report the accuracy for each task and the overall average accuracy (**Avg**) as the primary evaluation metrics.

original GVLQA benchmark contains graphs with relatively few nodes, which may limit the evaluation of model performance on larger-scale graph structures. Thus we propose LGVLQA to evaluate the graph reasoning capability of VLMs in scenarios with larger graph sizes. LGVLQA contains graphs whose maximum number of nodes is $2\text{--}5\times$ that of GVLQA, while preserving the same multi-modal input format and covering the same seven categories of reasoning tasks. This dataset provides a more challenging benchmark to assess model performance under large-scale graph structures. The brief comparison between the two datasets are shown in the Table 1, while comprehensive statistics are available in the Appendix C.

Baselines. We compare four categories of baseline models: (1) Traditional Graph Neural Networks and Transformer-based graph models, including GCN (Jiang et al., 2019), GAT (Veličković et al., 2017), GraphSAGE (Hamilton et al., 2017), and SAT (Chen et al., 2022); (2) Large language models, such as LLaMA2-7B (Touvron et al., 2023), LLaMA3-8B (Grattafiori et al., 2024), Vicuna-7B (Zheng et al., 2023), and Qwen2.5-7B (Qwen et al., 2025); (3) Vision-language models, including Qwen2.5-VL-7B (Bai et al., 2025), GITA-7B (Wei et al., 2024a), and InternVL2.5-8B (Chen et al., 2025); (4) GraphLLMs, including GraphWiz (Chen et al., 2024a), TAPE (He et al., 2024), and LLAGA (Chen et al., 2024b).

Implement Details. We adopt Qwen2.5-VL-7B as the base model and perform full-parameter fine-tuning on eight NVIDIA A800 SXM4 80GB GPUs. During the SFT stage, we employ the DeepSpeed Zero-3 optimization strategy with a learning rate of $2e-5$ and train for 12 epochs. The GRPO stage adopts DeepSpeed Zero-2 with a learning rate of $1e-6$ and is trained for 3 epochs. The global batch size is 16 during SFT and 14 during GRPO. Other configurations remain consistent across both stages, including a warmup ratio of 0.01, and a maximum image resolution of 262,144 pixels. The graph visualizer is implemented with Graphviz (Gansner and North, 2000).

4.2 Main Results

Evaluation on GVLQA. As shown in Table 2, our method consistently surpasses various existing baseline models across all graph reasoning tasks included in the GVLQA dataset. These results clearly indicate that the proposed approach can stably capture critical visual-topological structures in graphs while effectively incorporating textual instructions during the reasoning process, thereby demonstrating stronger cross-modal collaboration capabilities. Furthermore, our method maintains high generalization and robustness across multiple tasks, which further validates the significant advantages and broad applicability of the proposed multi-stage reasoning framework in diverse graph reasoning scenarios.

Dataset	LGVLQA								
	Models	Connect	Cycle	TS	SP	MaxFlow	BGM	HP	Avg
GCN	72.2	81.2	-	-	9.4	80.2	-	-	-
GAT	78.4	83.9	-	-	10.3	83.3	-	-	-
GraphSAGE	77.3	82.7	-	-	10.0	82.1	-	-	-
SAT	78.0	82.6	-	-	11.7	81.7	-	-	-
LLaMA2-7B	86.9	78.4	16.7	13.4	24.6	74.3	28.4	45.7	
Vicuna-7B	87.2	80.6	17.8	12.9	23.9	75.1	29.2	46.2	
LLaMA3-8B	88.8	82.8	18.9	15.0	25.2	78.0	30.5	48.0	
Qwen2.5-7B	87.3	77.9	19.4	14.2	26.9	72.1	30.7	45.6	
GraphWiz	89.5	71.1	17.0	12.8	28.2	73.3	38.2	47.2	
TAPE	72.9	81.3	-	-	11.8	81.2	-	-	-
LLAGA	89.2	75.4	14.4	18.6	31.0	74.9	36.6	48.6	
Qwen2.5-VL-7B	91.2	82.8	19.6	16.8	24.4	75.8	38.5	50.0	
InternVL2.5-8B	90.4	81.6	18.7	15.4	23.9	74.6	37.4	48.9	
GITA-7B	91.6	82.5	19.1	16.2	24.8	77.9	38.9	50.4	
MuSe (Ours)	93.6	85.4	22.3	20.9	31.8	84.5	40.7	54.2	

Table 3: Performance comparison of models on LGVLQA tasks. We report the accuracy for each task and the overall average accuracy (**Avg**) as the primary evaluation metrics.

Evaluation on LGVLQA As shown in Table 3, when the experimental setting is shifted to the LGVLQA dataset with larger node scales, the proposed method not only maintains its superiority over all baseline models across all tasks, but also exhibits an even greater performance margin compared to its results on GVLQA. In this large-scale graph scenario, conventional VLMs no longer demonstrate a performance advantage over other approaches. The primary reason is that existing baselines often rely on single-shot visualization of the entire graph, when the graph size grows significantly, node overlapping and information redundancy make it difficult for the model to extract salient topological features from cluttered visual inputs. In contrast, our method employs a dynamic subgraph sampling strategy to precisely select task-relevant topological structures, effectively avoiding the interference induced by large-scale graph visualizations and reducing the burden of processing redundant information. Consequently, the proposed approach sustains stable reasoning performance as node scales increase, highlighting its stronger adaptability and scalability, which fully aligns with the original design goals.

5 Ablation

5.1 Impact of Training Strategies

To evaluate the impact of different training strategies on model performance, we re-train our model

using three distinct strategies and conduct evaluation on GVLQA dataset:

- Strategy 1: Full SFT. Performing SFT on the entire training dataset as the baseline approach.
- Strategy 2: Two-stage training with SFT as the primary phase. Performing SFT on 80% of the data to allow the model to fully acquire the fundamental knowledge and reasoning patterns required for graph reasoning tasks, followed by GRPO on the remaining 20% to further enhance strategic decision-making capabilities.
- Strategy 3: Two-stage training with GRPO as the primary phase. Conducting a cold-start SFT on only 20% of training data, then applying GRPO on the remaining 80% of training data.

As shown in Table 4, Strategy 3 yields the poorest performance, Strategy 1 achieves intermediate results, and Strategy 2, which is adopted in this work, delivers the best overall performance.

From a training dynamics perspective, the degraded performance of Strategy 3 is primarily due to the limited SFT phase. Graph reasoning tasks are inherently challenging, requiring the model to understand complex topological structures and conduct complex reasoning. With insufficient exposure to the foundational patterns and knowledge during SFT, the model enters the GRPO stage without a strong initial policy. Consequently, even with numerous rollouts, it struggles to produce correct answers consistently and fails to establish a relative

Strategy	MaxFlow	SP	TS	Avg
Strategy 1	21.1	33.6	40.9	59.1
Strategy 2	22.6	35.3	42.7	60.7
Strategy 3	15.6	18.2	24.5	34.8

Table 4: Comparison of three training strategies. We report the average accuracy across all tasks based on the GVLQA as the primary evaluation metric, and further present detailed results on three representative tasks: MaxFlow, SP, and TS.

advantage within GRPO groups, leaving limited optimization potential. In contrast, Strategy 1 allows the model to acquire a complete foundation through full SFT, but without introducing policy optimization, its performance remains constrained by the inherent limitations of purely supervised strategy selection. Strategy 2 combines the advantages of both approaches: leveraging abundant SFT data to construct a robust base model, and using a smaller proportion of GRPO data to refine strategy selection and decision-making capabilities. This ensures stable task comprehension while effectively improving performance in complex reasoning scenarios, thereby validating the effectiveness of our proposed two-stage training framework for graph reasoning tasks.

5.2 Impact of Maximum Interaction Rounds

To investigate how the maximum number of interaction rounds K in the GRPO training stage affects the performance of our model, we keep all other training parameters fixed while setting K to 2, 3, 4, 5, and 10, respectively and conduct evaluation on GVLQA dataset.

The experimental results, as shown in Table 5, demonstrate a rapid initial improvement in accuracy, which subsequently stabilizes: when K increase from 2 to 3, accuracy improves significantly; further increasing from 3 to 4 yields another notable gain; however, when K increases from 4 to 5, and from 5 to 10, only slight fluctuations in accuracy are observed.

These findings suggest a clear trade-off between the round limit and model performance: while a moderate increase in interaction rounds allows the model to gather more task-relevant subgraph information, thereby improving answer quality in complex reasoning scenarios, in most graph reasoning tasks approximately five rounds are sufficient to collect all necessary information to derive the correct answer. Limiting rounds to around this value also

K	MaxFlow	SP	TS	Avg
2	14.3	29.7	25.4	48.4
3	19.8	34.4	39.2	56.8
4	22.1	35.3	42.4	60.0
5	22.6	35.3	42.7	60.7
10	22.9	35.3	42.7	60.9

Table 5: Performance Variation Across Different Maximum Interaction Rounds K . We report the average accuracy across all tasks based on the GVLQA as the primary evaluation metric, and further present detailed results on MaxFlow, SP and TS.

encourages the model to identify an optimal reasoning path within the constraint rather than engaging in excessive sampling of little benefit to problem-solving. Beyond five rounds, additional increases in K fail to produce substantial performance gains, and may instead lead to higher training costs and longer inference time.

6 Conclusion

We propose MuSe, a novel multi-stage graph reasoning framework via vision-language models. MuSe progressively samples and visualizes task-relevant subgraphs to enable incremental reasoning, effectively mitigating the interference caused by node overlap, occlusion, and irrelevant information in large-scale graphs. To further assess its scalability, we construct LGVLQA, a new multimodal graph reasoning dataset that encompasses larger and more complex graph structures. Experimental results demonstrate that MuSe consistently and significantly outperforms LLM and VLM baselines, showcasing superior structural understanding and reasoning capabilities.

7 Limitations

Our method depends on GPT-generated large-scale dialogue data for SFT, since GPT has low accuracy on graph reasoning tasks, we must generate reasoning with the correct answer provided, however, this can result in oversimplified reasoning for complex problems. Enhancing the quality and completeness of such data remains an open challenge. In addition, the current method is tailored for graph reasoning tasks, in future work, we plan to extend the proposed framework to broader application scenarios, including text-attributed graphs, knowledge graphs, and other graph-based reasoning tasks, in order to further enhance its generality and applicability.

8 Acknowledgments

This work is supported by the National Key R&D Program of China [2023YFC3304902].

References

- Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. [Qwen2.5-vl technical report](#). *Preprint*, arXiv:2502.13923.
- Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, and 1 others. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.
- Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.
- Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. 2020. Popularity prediction on social platforms with coupled graph neural networks. In *Proceedings of the 13th international conference on web search and data mining*, pages 70–78.
- Dexiong Chen, Leslie O’Bray, and Karsten Borgwardt. 2022. Structure-aware transformer for graph representation learning. In *International conference on machine learning*, pages 3469–3489. PMLR.
- Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024a. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364.
- Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. 2024b. Llaga: Large language and graph assistant. *arXiv preprint arXiv:2402.08170*.
- Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, Lixin Gu, Xuehui Wang, Qingyun Li, Yiming Ren, Zixuan Chen, Jiapeng Luo, Jiahao Wang, Tan Jiang, Bo Wang, and 23 others. 2025. [Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling](#). *Preprint*, arXiv:2412.05271.
- Xu Chu, Hanlin Xue, Zhijie Tan, Bingce Wang, Tong Mo, and Weiping Li. 2025. Graphsos: Graph sampling and order selection to help llms understand graphs better. *arXiv preprint arXiv:2501.14427*.
- Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *arXiv 2023. arXiv preprint arXiv:2310.04560*.
- Emden R Gansner and Stephen C North. 2000. An open graph visualization system and its applications to software engineering. *Software: practice and experience*, 30(11):1203–1233.
- Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. Pmlr.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.
- Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.
- Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648.
- Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182.
- Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024. [Harnessing explanations: Llm-to-llm interpreter for enhanced text-attributed graph representation learning](#). In *International Conference on Learning Representations*, volume 2024, pages 5711–5732.
- Chao Huang, Huance Xu, Yong Xu, Peng Dai, Lianghao Xia, Mengyin Lu, Liefeng Bo, Hao Xing, Xiaoping Lai, and Yanfang Ye. 2021. Knowledge-aware coupled graph neural network for social recommendation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4115–4122.
- Bo Jiang, Ziyang Zhang, Doudou Lin, Jin Tang, and Bin Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11313–11320.

- Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T Kwok. 2023a. Forward-backward reasoning in large language models for mathematical verification. *arXiv preprint arXiv:2308.07758*.
- Weisen Jiang, Yu Zhang, and James Kwok. 2023b. Effective structured prompting by meta-learning and representative verbalizer. In *International Conference on Machine Learning*, pages 15186–15199. PMLR.
- Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.
- Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629.
- Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470.
- Hanqing Li, Kiran Sheena Jyothi, Henry Liang, Sharika Mahadevan, and Diego Klabjan. 2025. Zero-shot graph reasoning via retrieval augmented framework with llms. *arXiv preprint arXiv:2509.12743*.
- Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023. One for all: Towards training one graph model for all classification tasks. *arXiv preprint arXiv:2310.00149*.
- Xiao Liu, Shiyu Zhao, Kai Su, Yukuo Cen, Jiezhong Qiu, Mengdi Zhang, Wei Wu, Yuxiao Dong, and Jie Tang. 2022. Mask and reason: Pre-training knowledge graph transformers for complex logical queries. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 1120–1130.
- Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM review*, 45(2):167–256.
- Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862*.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, and 25 others. 2025. *Qwen2.5 technical report*. Preprint, arXiv:2412.15115.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shrutu Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861.
- Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning intents behind interactions with knowledge graph for recommendation. In *Proceedings of the web conference 2021*, pages 878–887.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits its reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James T. Kwok, and Yu Zhang. 2024a. *GITA: graph to visual and textual integration for vision-language graph reasoning*. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- Yanbin Wei, Qiushi Huang, James T Kwok, and Yu Zhang. 2024b. Kicgpt: Large language model with knowledge in context for knowledge graph completion. *arXiv preprint arXiv:2402.02389*.
- Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.

- Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.
- Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2021. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Advances in Neural Information Processing Systems*, 34:28798–28810.
- Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2023. Language is all a graph needs. *arXiv preprint arXiv:2308.07134*.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhengguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.
- Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*.
- Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. 2024. [Can LLM graph reasoning generalize beyond pattern memorization?](#) In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2289–2305, Miami, Florida, USA. Association for Computational Linguistics.
- Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xinyun Liu, and Yulia Tsvetkov. 2025. Generalizable llm learning of graph synthetic data with reinforcement learning. *arXiv preprint arXiv:2506.00845*.
- Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. 2021. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. *Advances in Neural Information Processing Systems*, 34:19172–19183.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.
- Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A Multi-Turn Dialog Data Generation

As illustrated in Figure 3, we present the system prompt template employed to guide GPT-4o in generating the SFT training data.

B Quality Inspection of Synthetic Data

The synthetic data will be filtered and processed through three stages: rule-based verification of format and answer correctness, inspection for overly simplified reasoning, and logical consistency validation.

Rule-based verification The rule-based checking consists of three steps, including: 1) checks on format compliance: check whether all thinking processes are within the `<think>` `</think>` tags, whether sampling is invoked in the form of `<sample>` `sample(r, h)` `</sample>`, and whether the answer is within the `<answer>` `</answer>` tags. 2) correctness of function-call arguments: `r` must be a node existing in the graph, and `h` must be an integer. 3) accuracy of the final answers. Samples that did not meet these criteria were regenerated.

Inspection for overly simplified reasoning Then, for samples with high reasoning difficulty but with dialog rounds ≤ 2 , we conducted manual inspection. If we observed overly simplified reasoning or insufficient explanation in responses, we prompted GPT-4o to regenerate more fine-grained reasoning paths.

Logical consistency validation We further employed GPT-4o as judge to detect whether generated dialogues contain logical contradictions. Any problematic samples identified by this automated consistency check were regenerated.

Through this multi-stage filtering and refinement process, we have done our best to ensure that, at the current dataset scale, the synthetic training data maintains a high level of quality and reasoning soundness.

C Dataset Statistics

We present a detailed statistical comparison between the GVLQA and LGVLQA datasets in Table 6, including the range of node counts (**Nodes**) and the range of edge counts (**Edges**) for each task, as well as the average number of nodes (**Avg. N**) and the average number of edges (**Avg. E**). This provides a comprehensive quantification of the data complexity and graph structure scale.

Dataset	GVLQA				LGVLQA			
	Task	Nodes	Avg. N	Edges	Avg. E	Nodes	Avg. N	Edges
Connect	[2,35]	25.0	[1,363]	95.4	[2,100]	49.9	[1,384]	106.8
Cycle	[2,35]	23.4	[4,38]	23.6	[2,100]	49.7	[1,301]	80.8
TS	[2,35]	21.8	[0,323]	114.1	[2,50]	24.6	[3,361]	168.3
SP	[2,20]	13.6	[4,63]	23.9	[2,100]	49.6	[3,217]	107.7
MaxFlow	[2,20]	13.9	[2,113]	49.1	[2,50]	26.0	[2,147]	55.5
BGM	[2,33]	21.1	[1,129]	51.0	[2,100]	49.7	[0,363]	117.6
HP	[2,20]	13.2	[4,132]	45.0	[2,50]	25.3	[4,383]	158.8

Table 6: Statistical comparison of GVLQA and LGVLQA datasets across different tasks, including node and edge ranges and their averages.

Dataset	GVLQA								
	Models	Connect	Cycle	TS	SP	MaxFlow	BGM	HP	Avg
InternVL2.5-8B		98.4	95.6	38.7	29.4	18.9	92.6	30.4	57.7
MuSe-InternVL2.5-8B		99.3	96.8	41.8	33.9	21.5	94.1	34.2	60.2
Qwen2.5-VL-7B		98.2	96.8	39.6	31.8	20.4	92.5	28.5	58.5
MuSe-Qwen2.5-VL-7B		99.4	97.4	42.7	35.3	22.6	94.0	33.8	60.7

Table 7: Performance comparison of models on different base models

D Generalization Evaluation

We further applied our method to InternVL2.5-8B and evaluated it on GVLQA. As shown in Table 7, the results show consistent improvements across all tasks, indicating that our method generalizes beyond a single model.

E Latency analysis

We conducted latency analysis on GVLQA using MuSe (trained on Qwen2.5-VL-7B) and other LLM-based methods with the same family of base models. As shown in Table 8, we report three metrics: average inference latency (**AIL**), first-token latency (**FTL**), and average generation length (**AGL**), where **dir** means the model directly outputs the final answer, and **cot** means that the model is prompted to think step by step before giving the answer. Our method indeed exhibits higher latency compared to other methods. This is mainly because MuSe performs multi-turn reasoning with explicit visual-text interactions. Such multi-stage inference enables the model to generate a high-quality and transparent reasoning process, which leads to substantial accuracy improvements over other methods. We acknowledge this latency-accuracy trade-off, and we consider it meaningful in scenarios where reliable and interpretable reasoning is preferred over minimal response time.

	AIL (s)	FTL (s)	AGL (tokens)
Qwen2.5-7B-dir	1.97	1.13	16.42
Qwen2.5-7B-cot	8.31	1.13	315.76
Qwen2.5-VL-7B-dir	3.82	2.18	15.91
Qwen2.5-VL-7B-cot	8.78	2.18	297.08
MuSe	19.88	1.37	624.35

Table 8: Latency analysis across different methods on GVLQA.

F Training Details

All experiments are conducted on eight NVIDIA A800 SXM4 GPUs with 80GB memory each. The system environment is configured with CUDA 12.4, Python 3.10.15, PyTorch 2.7.1, and Transformers 4.57.0. The random seed is fixed to 42 for reproducibility. We adopt full-parameter fine-tuning for the SFT stage, using Qwen2.5VL-7B-Instruct as the base model. The training dataset is preprocessed with 4 workers and a maximum sequence length of 10,000 tokens. We also restrict the maximum input resolution to 262,144 pixels. Optimization is performed using the AdamW optimizer with an initial learning rate of $2e-5$ with a warmup ratio of 0.01. The model is trained for 13 epochs with mixed-precision training in bfloat16 format. The batch size is set to 1 per device with a gradient accumulation of 2 steps, resulting in an effective batch size of 16 (2×8 GPUs). Gradient clipping is applied with a maximum norm of 1.0. Model checkpoints are saved every 500 steps, and training

loss is logged every 5 steps.

We also adopt full-parameter fine-tuning for the GRPO stage. We use the Qwen2.5VL-7B-Instruct model fine-tuned with SFT as the base model. During the GRPO stage, we deploy VLLM on a single GPU to generate rollouts for each sample, with the rollout number set to 8. The remaining seven GPUs are used for training, each configured with a batch size of 1 per device and a gradient accumulation of 2 steps, resulting in an effective batch size of 14 (2 x 7 GPUs). We restrict the maximum input resolution to 262,144 pixels and the maximum number of output rounds to 5. The learning rate is set to 1e-6 with a warmup ratio of 0.01, and the maximum output length (including all model outputs throughout the entire dialogue) is limited to 2048 new tokens. For multi-turn dialogues, the model outputs from each round are incorporated into the GRPO logit computation.

During evaluation, we use deterministic greedy decoding (equivalent to temperature = 0) to ensure the reproducibility of the results.

G Case Study

In this section, we present detailed output examples of MuSe when handling each type of task. Figure 4 illustrates Connect, Figure 5 illustrates Cycle, Figure 6 illustrates MaxFlow, Figure 7 illustrates SP, Figure 8 illustrates TS, Figure 9 illustrates BGM, Figure 10 illustrates HP.

Following the approach of GITA, the textual description of the graph structure is formulated according to the specific task objective. After being converted into a task-adaptive description G^T , it is concatenated with the problem Q and provided as the first-round input. The model then performs reasoning based on the given question and the initial textual description of the graph. When it becomes necessary to retrieve task-relevant information from a local region of the graph, the model invokes the operation `<sample> sample(G, r, h) </sample>` to sample and visualize a local subgraph. The resulting visualization is subsequently fed back as the next-round user input, enabling the model to continue reasoning based on both its previous thoughts and the visualized subgraph. This process is repeated iteratively until the model outputs the final answer in the format `<answer> ... </answer>`, thus concluding the multi-turn dialog.

Note: In the figures, some lengthy problem descriptions are truncated for clarity.

```

SYSTEM_PROMPT = ""You are a graph reasoning assistant. Your task is to solve problems on
graphs described in text.
You must always follow this strict output format:
1. Reasoning process:
  * You need to provide your reasoning and must wrap it all in <think> ... </think>.
  * Explain why you need to sample certain nodes and hops.
2. Function calls:
  * When you need to inspect part of the graph (at least once), call the function in the form:
  ...
  <sample> sample(G, root, hops) </sample>
  ...
  * `G` is always the graph, `root` is the starting node (must be an integer), and `hops` is the
  number of hops (must be an integer).
  * After each `<sample> ... </sample>` you must stop generating until the system returns
  the result.
3. Final answer:
  * Once you have enough information, output your answer using the format:
  <answer> <your final answer> </answer>
4. Ground-truth enforcement:
  * The user provides the correct final answer here.
  * You must follow the sampling steps and reasoning, and you must finish by outputting
  the given correct final answer exactly.
  * The correct final answer is :
  Answer: {ground_truth}
  * Do not deviate from the provided ground-truth.
---
### Example of whole process

{"messages = [
  {"role": "user",
   "content": [{"type": "text", "text": "In an undirected graph, (i,j) means that node i and node j
are connected with an undirected edge.\n\nThe nodes are numbered from 0 to 9, and the edges
are: (1,2) (1,5) (2,8) (2,5) (3,5) (3,7) (4,9) (5,6) (5,8) (5,7) (6,7).\n\nIs there a path between node 1
and node 6 in this undirected graph?"}]
  ],
  {"role": "assistant",
   "content": "<think> I need to check whether node 1 and node 6 are connected.
First, I will sample the neighborhood of node 1 within 2 hops to see its connections. </think>
<sample> sample(G, 1, 2) </sample>"}
  {"role": "user",
   "content": [
     {"type": "image_url", "image_url": {"url": "https://example.com/graph1.png"}}
   ]
  },
  {"role": "assistant",
   "content": "<think> From the sample I see that node 1 connects to node 5, and node 5
connects to node 6.
This shows that a path exists. </think>
<answer> Yes. </answer>"}]"}
"""

```

Figure 3: Here is the system prompt template to guide GPT-4o in generating the SFT training data.

"user": "In an undirected graph, (i,j) means that node i and node j are connected with an undirected edge.\n\nThe nodes are numbered from 0 to 26, and the edges are: (0,2) (0,10) (0,17) (0,15) (0,4) (0,12) (0,24) (0,9) (0,3) (0,13) (0,23) (0,7) (0,19) (0,16) (0,6) (0,25) (0,8) (1,10) (1,17) (1,4) (1,12) (1,24) (1,3) (1,13) (1,23) (1,26) (1,22) (1,19) (1,6) (1,14) (1,25) (1,8) (1,20) (2,10) (2,17) (2,11) (2,4) (2,24) (2,9) (2,3) (2,13) (2,23) (2,26) (2,7) (2,22) (2,19) (2,6) (2,14) (2,25) (2,8) (3,10) (3,17) (3,11) (3,15) (3,4) (3,12) (3,24) (3,9) (3,13) (3,26) (3,7) (3,19) (3,16) (3,14) (3,25) (3,8) (3,20) (4,10) (4,17) (4,11) (4,15) (4,12) (4,24) (4,9) (4,7) (4,19) (4,16) (4,6) (4,25) (4,20) (5,18) (6,10) (6,17) (6,15) (6,12) (6,24) (6,9) (6,23) (6,7) (6,16) (6,25) (6,8) (7,10) (7,11) (7,15) (7,12) (7,13) (7,22) (7,19) (7,16) (7,25) (7,8) (7,20) (8,10) (8,17) (8,11) (8,15) (8,12) (8,24) (8,9) (8,23) (8,22) (8,19) (8,16) (8,14) (8,25) (8,20) (9,10) (9,17) (9,11) (9,12) (9,24) (9,13) (9,23) (9,26) (9,22) (9,19) (9,16) (9,20) (10,17) (10,11) (10,15) (10,12) (10,24) (10,13) (10,23) (10,26) (10,19) (10,14) (10,20) (11,15) (11,12) (11,24) (11,13) (11,23) (11,26) (11,19) (11,16) (11,14) (11,20) (12,15) (12,24) (12,23) (12,22) (12,19) (12,16) (12,14) (12,25) (12,20) (13,15) (13,24) (13,23) (13,19) (13,16) (13,14) (13,25) (13,20) (14,17) (14,15) (14,24) (14,23) (14,26) (14,22) (14,19) (14,16) (14,25) (14,20) (15,17) (15,24) (15,23) (15,26) (15,16) (15,20) (16,24) (16,23) (16,26) (16,19) (16,25) (16,20) (17,23) (17,20) (18,21) (19,23) (19,22) (19,25) (19,20) (20,24) (20,23) (20,22) (20,25) (22,26) (22,25) (23,24) (23,26) (23,25) (24,26) (24,25) (25,26).\n\nIs there a path between node 21 and node 22 in this undirected graph?"

"assistant": "<think> To determine whether there is a path between node 21 and node 22, I will first need to check the connections of node 21. Since node 21 is mentioned only in the edge (18,21), it's important to check from node 18 as well. I will sample the graph starting from node 21 with 1 hop to see its immediate connections. </think>\n\n<sample> sample(G, 21, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> Node 21 is only connected to node 18. Now, I need to determine if node 18 is connected to node 22. I will sample the graph starting from node 18 with 1 hop to see if it gives any connections to node 22 or any path that eventually leads to it. </think>\n\n<sample> sample(G, 18, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sample, node 18 is connected to nodes 21 and 5. Since node 22 does not appear among these immediate connections of node 18, I need to explore further from node 5 to see if there is any connection leading to node 22. I will sample the graph starting from node 5 with 1 hop. </think>\n\n<sample> sample(G, 5, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> Node 5 only connects back to node 18 and does not extend any further towards node 22. Since there are no other connections available from nodes 5, 18, and 21, I can conclude that there is no path connecting node 21 to node 22 in this graph. </think>\n\n<answer> No. </answer>"

Figure 4: This example illustrates how MuSe handles a Connect problem.

"user": "In an undirected graph, (i,j) means that node i and node j are connected with an undirected edge.\n\nThe nodes are numbered from 0 to 25, and the edges are: (1,21) (23,2) (7,14) (9,24) (4,0) (12,18) (17,24) (3,22) (12,23) (5,22) (6,20) (11,13) (8,14) (17,3) (6,10) (14,23) (25,10) (15,5) (21,20) (4,14) (11,19) (1,8) (13,16) (3,2) (16,25).\n\nIs there a cycle in this undirected graph?"

"assistant": "<think> To determine if there is a cycle in the graph, I need to explore the connections between nodes and see if any path returns to the same starting node without repeating any edge. \n\nA cycle requires at least 3 nodes. I will start sampling from node 1 since it is part of two edges (1,21) and (1,8). Sampling within 3 hops should provide a good understanding of potential cycles involving node 1. </think>\n\n<sample> sample(G, 1, 3) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, I can see the following paths:\n- 1 connects to 21, 8, and 4.\n- 21 connects to 20.\n- 8 connects to 14.\n- 4 connects to 14.\n- 14 connects to 23.\n\nHowever, there is no path that forms a cycle (a closed loop where no edge is repeated). To further verify, I would need to explore other parts of the graph, as this subgraph does not provide a complete view of all nodes or connections.\n\nNext, I will sample another node that is part of multiple edges. Node 14 connects to multiple nodes (4, 7, 23, 8). I will sample from node 14 within 2 hops to check for cycles involving this node. </think>\n\n<sample> sample(G, 14, 2) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the expanded subgraph, I observe the following:\n- 14 connects to 4, 7, 23, 8.\n- 23 connects to 12 and 2.\n- 12 connects to 14.\n- 8 connects to 14.\n- 1 connects to 4.\n- 0 connects to 4.\n\nDespite seeing more connections, there still is no evident cycle formed in this part of the graph. Since a cycle requires a return to the starting node without repeating edges, the current sampling does not show any such formation.\n\nTo ensure thoroughness, I will sample another node with multiple connections. Node 23 connects to nodes 12, 2, and 14. I will sample from node 23 within 2 hops to investigate further potential connections leading to a cycle. </think>\n\n<sample> sample(G, 23, 2) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> In the newly sampled subgraph around node 23, I observe:\n- 23 connects to 12, 14, and 3.\n- 12 connects to 18.\n- 3 connects to 2 and 14.\n- 14 connects to 4, 7, 8, and 23.\n\nFrom this expanded view, there are now several connections, but still, no clear cycle is visible. There are overlapping connections (e.g., 14 to 23 and back to 14), but they do not form a closed loop where new nodes are visited without repetition of edges.\n\nGiven the current exploration, there seems to be no cycle in the graph. Further sampling might not be necessary since the observed structures do not indicate any cycle formation. </think>\n\n<answer> No. </answer>"

Figure 5: This example illustrates how MuSe handles a Cycle problem.

"user": "In a directed graph, the nodes are numbered from 0 to 14, and the edges are:\nan edge from node 0 to node 13 with capacity 8,\nan edge from node 0 to node 7 with capacity 20,\nan edge from node 0 to node 11 with capacity 12,\n Q: What is the maximum flow from node 14 to node 13:"

"assistant": "<think> To calculate the maximum flow from node 14 to node 13, I need to analyze the connectivity and capacity of the paths from node 14 (source) to node 13 (sink). \nFirst, I will sample the neighborhood of the source node (14) to identify its immediate outgoing edges and their capacities to analyze possible paths. </think> \n<sample> sample(G, 14, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, node 14 has direct edges to node 9 with a capacity of 16 and to node 10 with a capacity of 13. These allow initial flow from the source (14). \nFrom node 9, there are edges leading to node 4 and node 11 (which do not directly lead to the sink node 13). \nFrom node 10, there is a direct edge to node 13 with a capacity of 15. \nTo determine the paths leading to sink 13 and confirm flow possibilities, I need to explore the neighborhoods of nodes directly connected to 14 (i.e., node 9 and node 10) further within 2 hops. \nFirst, I'll sample from node 9. </think> \n<sample> sample(G, 9, 2) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled graph, I see the following important observations: \n- Node 9 does not have any direct connections leading to the sink node 13. Thus, node 9 doesn't contribute to the flow directly toward node 13. \n- Node 10 has a direct connection to node 13 with capacity 15, which can allow a flow. \nHence, to maximize flow, the path will focus on utilizing node 10's capacity to 13 of 15 units. \nThe maximum flow from node 14 to node 13 is derived solely based on the direct edge from node 10 to node 13. </think> \n<answer> 25. </answer>"

Figure 6: This example illustrates how MuSe handles a MaxFlow problem.

"user": "In a undirected graph, the nodes are numbered from 0 to 11, and the edges are:\nan edge between node 0 and node 9 with weight 7,\nan edge between node 5 and node 0 with weight 10,\nan edge between node 0 and node 11 with weight 5,\nan edge between node 0 and node 10 with weight 8,\nan edge between node 3 and node 1 with weight 5,\nan edge between node 1 and node 5 with weight 6,\nan edge between node 1 and node 2 with weight 10,\nan edge between node 2 and node 7 with weight 1,\nan edge between node 5 and node 3 with weight 9,\nan edge between node 11 and node 3 with weight 4,\nan edge between node 3 and node 7 with weight 2,\nan edge between node 4 and node 6 with weight 5,\nan edge between node 4 and node 8 with weight 5,\nan edge between node 5 and node 8 with weight 8,\nan edge between node 8 and node 7 with weight 3,\nan edge between node 10 and node 11 with weight 5.\nThis graphic illustrates an undirected graph, with each edge's distance or length indicated by a numerical label in close proximity.\nQ: What is the shortest path from node 10 to node 6?"

"assistant": "<think>\nTo determine the shortest path from node 10 to node 6 in an undirected graph, I should first explore the connections starting from node 10. I know the edges involved and their weights, so I will begin by identifying immediate connections emanating from node 10.\n\nFirst, let me sample the neighborhood of node 10 within 2 hops, which will help me trace possible routes from node 10. Through this, I can identify viable paths and calculate their respective total weights to determine the shortest path effectively. I may have to sample further to ensure a complete path from node 10 to node 6 is identified.\n\n</think>\n<sample> sample(G, 10, 2) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think>\n\nIn the sampled subgraph, node 10 is connected directly to node 11 with a weight of 5. Node 11 is connected to node 3 with a weight of 4, and node 3 is connected to node 7 with a weight of 2. \n\nNode 7 is connected to node 8 with a weight of 3, and node 8 is connected to node 4 with a weight of 5. Finally, node 4 is connected to node 6 with a weight of 5. By summing up all these edge weights, we can determine the shortest path from node 10 to node 6.\n\nThe path 10->11->3->7->8->4->6 has a total weight of $5 + 4 + 2 + 3 + 5 + 5 = 22$. \n\nThis seems to be a direct and viable path. However, I need to check if there's a shorter path by exploring other possibilities or shorter segments.\n\nI will sample further from node 11 to check for potential shortcuts.\n\n</think>\n<sample> sample(G, 11, 2) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think>\n\nIn the newly sampled subgraph, node 11 connects directly to node 6 with a weight of 5. This was not visible in the previous sample. The path 10->11->6 has a total weight of $5 (10 \text{ to } 11) + 5 (11 \text{ to } 6) = 10$. \n\nThis appears to be the shortest path from node 10 to node 6, as it does not involve additional traversals through other nodes which would only add more weight. \n\nThus, the shortest path from node 10 to node 6 is determined to be 10->11->6.\n\n</think>\n<answer> 10->11->3->7->8->4->6. </answer>"

Figure 7: This example illustrates how MuSe handles a SP problem.

"user": "In a directed graph with 32 nodes numbered from 0 to 31:\nnode 0 should be visited before node 16\nnode 0 should be Q: The topological order of the directed graph is:"

"assistant": "<think> To determine the topological order of the directed graph, I need to perform a sampling to explore how nodes are connected and ensure nodes in the graph follow the precedence rules as described. Since the graph has 32 nodes, sampling from the graph will allow me to progressively understand the dependencies and constraints between the nodes. Starting from node 0, I will sample its connections to infer relationships and dependencies. Then I will broaden the scope to include dependencies stemming from other nodes while adhering to the directed edges represented. Several samples will be required to establish the complete topological order. </think> \n<sample> sample(G, 0, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, I can see a portion of the directed graph with its dependencies. Nodes such as 0, 1, 9, 10, 24, and others have their outgoing edges respected. However, to ensure the entire graph's topological order, additional samples are needed to fill in the rest of the graph and verify all dependencies. I will sample from node 1 next to understand its connections and further dependencies. </think> \n<sample> sample(G, 1, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> The sampled subgraph reveals more details about the directed graph's structure. Nodes like 28, 9, and others are shown with their directed edges respecting the precedence rules. However, to conclusively determine the topological order, additional samples are still required to fully understand all dependencies within the graph. I will sample from node 9 next to uncover further connections and dependencies. </think> \n<sample> sample(G, 9, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> The sampled subgraph uncovers additional dependencies from node 9 and others, providing a richer view of the directed graph's structure. However, to ensure the completion of the topological order, more samples are still necessary to respect all dependencies within the graph. I will sample from node 28 next, as it has an influence on nodes like 1, 30, and others. </think> \n<sample> sample(G, 28, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> The sampled subgraph sampled from node 28 broadens the view of the graph's structure, revealing dependencies involving nodes like 1, 30, 24, and others. However, to finalize the topological order and ensure all dependencies are respected, the remaining portions of the graph need sampling. Since the preceding samples have inferred much of the graph structure, I can now determine the topological order by compiling the information systematically. </think> \n:
<answer>9,10,28,1,24,30,18,5,11,29,26,13,7,19,27,21,17,23,4,25,14,31,15,0,16,3,22,6,12,8,20,2.</answer>"

Figure 8: This example illustrates how MuSe handles a TS problem.

"user": "There are 15 hosts numbered from 0 to 14, and 13 tasks numbered from 0 to 12. Each host has a set of tasks that it is interested in: \nHost 1 is interested in task 12.\nHost 8 is interested in Q: What is the maximum number of hosts that can be assigned a task they are interested in?"

"assistant": "<think> This problem can be modeled as a bipartite graph where hosts are on one side and tasks are on the other. Each host is connected to a task if they are interested in it. To find the maximum number of hosts that can be assigned a task, we need to determine the maximum matching in this bipartite graph.\n\nTo start, I need to inspect the graph by sampling nodes from one host and its connections to tasks. Let's begin by sampling the neighborhood of host 1 to explore its connections. </think>\n<sample> sample(G, 1, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, host 1 is connected to tasks 4, 12, 2, and 3. Since each host can solve only one task and each task can be solved by one host, I need more information about how other hosts are connected to these tasks to investigate potential matchings. I will now sample the neighborhood of host 8 to look at its connections. </think> \n<sample> sample(G, 8, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, host 8 is connected to tasks 4, 10, 9, 1, 7, 2, 11, and 5. These tasks overlap with some tasks connected to host 1 (e.g., task 2, task 4, etc.). To determine the maximum matching, I need more information about other hosts' connections to these tasks. I will next sample the neighborhood of host 4 to understand its connections. </think> \n<sample> sample(G, 4, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, host 4 is connected to tasks 0, 2, 1, and 3. This adds more connections to tasks that were also seen in previous samples (e.g., task 2, task 3). \n\nTo continue building towards finding the maximum matching, I need to explore connections for other hosts. I will now sample the neighborhood of host 6 to understand its connections. </think> \n<sample> sample(G, 6, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, host 6 is connected to tasks 9, 6, 12, 2, and 5. This adds more overlapping connections with tasks seen in previous samples (e.g., task 9, task 6, task 2, etc.).\n\nTo summarize, I've been sampling connections for hosts 1, 8, 4, and 6. The next step would be to analyze the graph structure and perform a maximum bipartite matching to determine how many hosts can be assigned tasks they are interested in. However, I notice that sampling connections from different hosts reveals repetitive visits to certain tasks, highlighting the need for a matching algorithm implementation.\n\nLet me now sample the neighborhood of host 3 to explore additional overlaps. </think> \n<sample> sample(G, 3, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, host 3 is connected to tasks 4, 5, 10, 2, 0, 6, and 8. This adds further overlaps with tasks connected to previously sampled hosts (e.g., task 4, task 5, task 2, etc.). \n\nThe graph structure is becoming clearer, and I'll soon need to perform a maximum bipartite matching to determine the maximum number of hosts that can be assigned tasks they're interested in. At this point, I should sample the neighborhood of host 5 to continue uncovering connections. </think> \n<sample> sample(G, 5, 1) </sample>"

"user": "<image>Here is the sampled subgraph image"

"assistant": "<think> From the sampled subgraph, host 5 is connected to tasks 4, 10, 1, 7, 2, 11, 0, and 8. This adds more overlapping connections with tasks seen in previous samples (e.g., task 4, task 10, task 7, task 2, etc.).\n\nWith sufficient information sampled from various hosts, I can now determine the maximum bipartite matching. The goal is to find the maximum number of hosts that can be assigned tasks they are interested in, ensuring no task is assigned to more than one host and vice versa. Using graph matching algorithms like the Hopcroft-Karp algorithm or maximum flow, this can be computed effectively. Based on the sampled graph structure, the answer should be rounded to the nearest valid number of matchings. </think>\n<answer> 13. </answer>"

Figure 9: This example illustrates how MuSe handles a BGM problem.

"user": "In an undirected graph, (i,j) means that node i and node j are connected with an undirected edge.\n\nThe nodes are numbered from 0 to 4, and the edges are: (2,1) (4,1) (1,3) (3,4) (0,2).\n\nQ: Begin with node 0, what is the path in this graph that visits every node exactly once?"

"assistant": "<think> I need to find a path starting from node 0 that visits every node exactly once. This calls for a Hamiltonian path investigation.\n\nFirst, I will see the connections of node 0, which will be the starting point, to understand potential directions for a path.\n\nI will sample node 0 with 1 hop initially. </think>\n\n<sample> sample(G, 0, 1) </sample>"

"user": "Here is the sampled subgraph image"

"assistant": "<think> From the sample, node 0 is directly connected to node 2. Now I will explore node 2 to find further connections and continue building the path. Node 2 should lead us to new nodes. I will sample node 2 with 1 hop to see its immediate connections. </think>\n\n<sample> sample(G, 2, 1) </sample>"

"user": "Here is the sampled subgraph image"

"assistant": "<think> Node 2 connects to node 1. Therefore, from node 0, the path goes to node 2 and then to node 1.\n\nNext, I need to check the connections of node 1 to find a path to new nodes.\n\nI will sample node 1 with 1 hop to identify further potential connections. </think>\n\n<sample> sample(G, 1, 1) </sample>"

"user": "Here is the sampled subgraph image"

"assistant": "<think> Node 1 connects to nodes 3 and 4. Thus, we can continue the path from node 1 to node 3.\n\nNode 3 then connects to node 4, allowing us to visit every node while following the edges.\n\nAll nodes have now been accounted for in the path: 0 -> 2 -> 1 -> 3 -> 4.\n\nThis ensures we visit every node exactly once. </think>\n\n<answer> 0->2->1->3->4. </answer>"

Figure 10: This example illustrates how MuSe handles a HP problem.