GRAPHDANCER: Training LLMs to Explore and Reason over Graphs via Curriculum Reinforcement Learning

Anonymous ACL submission

Abstract

Large language models (LLMs) increasingly rely on external knowledge to improve factuality, yet many real-world knowledge sources are organized as heterogeneous graphs rather than plain text. Reasoning over such graph-structured knowledge poses two key challenges: (1) navigating structured, schema-defined relations requires precise function calls rather than similarity-based retrieval, and (2) answering complex questions often demands multi-hop evidence aggregation through iterative information seeking. We propose GRAPHDANCER, a reinforcement learning (RL) framework that teaches LLMs to navigate graphs by interleaving reasoning and function execution. To make RL effective for moderate-sized LLMs, we introduce a graph-aware curriculum that schedules training by the structural complexity of information-seeking trajectories using an easy-to-hard biased sampler. We evaluate GRAPHDANCER on a multi-domain benchmark by training on one domain only and testing on unseen domains and out-of-distribution question types. Despite using only a 3B backbone, GRAPHDANCER outperforms baselines equipped with either a 14B backbone or `GPT-4o-mini`, demonstrating robust cross-domain generalization of graph exploration and reasoning skills. Our code is available at https://anonymous.4open.science/r/GraphDancer.

1 Introduction

The internal parameters of LLMs are often insufficient to capture external knowledge that is both rapidly evolving (Vu et al., 2024) and extremely long-tailed (Sun et al., 2024b). Consequently, establishing a framework that enables LLMs to interact with external knowledge sources is necessary to improve their factuality and reliability (Ji et al., 2023). Retrieval-augmented generation (RAG; Lewis et al., 2020) has become a mainstream solution toward this goal. However, when external knowledge is not organized as plain text corpora but instead interconnected in graph structures (Han et al., 2024; Sun et al., 2024a; Ma et al., 2025), two major challenges arise.

First, graphs may contain heterogeneous node and edge types (e.g., [GENE] *is upregulated in* [ANATOMY] vs. [GENE] *is downregulated in* [ANATOMY]), making it inadequate to retrieve information solely through semantic similarity search. Instead, effective information access requires precisely defined function calls (e.g., finding genes connected to a given anatomy entity via "*is upregulated in*"). Second, reasoning over graphs usually requires capturing complex, multi-hop connections between nodes. This makes it impractical to force the model to acquire all necessary information in a single round. Instead, an adaptive, multi-round process is needed, in which information seeking and reasoning steps are interleaved (e.g., subsequent function calls should be based on the gene nodes obtained in the current round).

To elicit multi-round graph interaction capabilities in LLMs, existing studies (Jin et al., 2024; Amayuelas et al., 2025; Gao et al., 2025; Kashmira et al., 2025) have proposed various prompting strategies. However, compared with actual model training, prompting-based approaches are less effective at helping models internalize the sophisticated graph interaction skills. On the other hand, training models via supervised fine-tuning (SFT) to access external knowledge (Schick et al., 2023; Asai et al., 2024) struggles to generalize to new graphs and domains, such as those with novel node/edge types, function calls, or tasks.

Contributions. To address the aforementioned challenges, in this paper, we propose an RL framework, GRAPHDANCER, that enables LLMs to thoroughly and generalizably acquire the ability to interact with and reason over graphs in an adaptive, interleaved fashion. This solution is not only motivated by the recent success of RL in enhancing

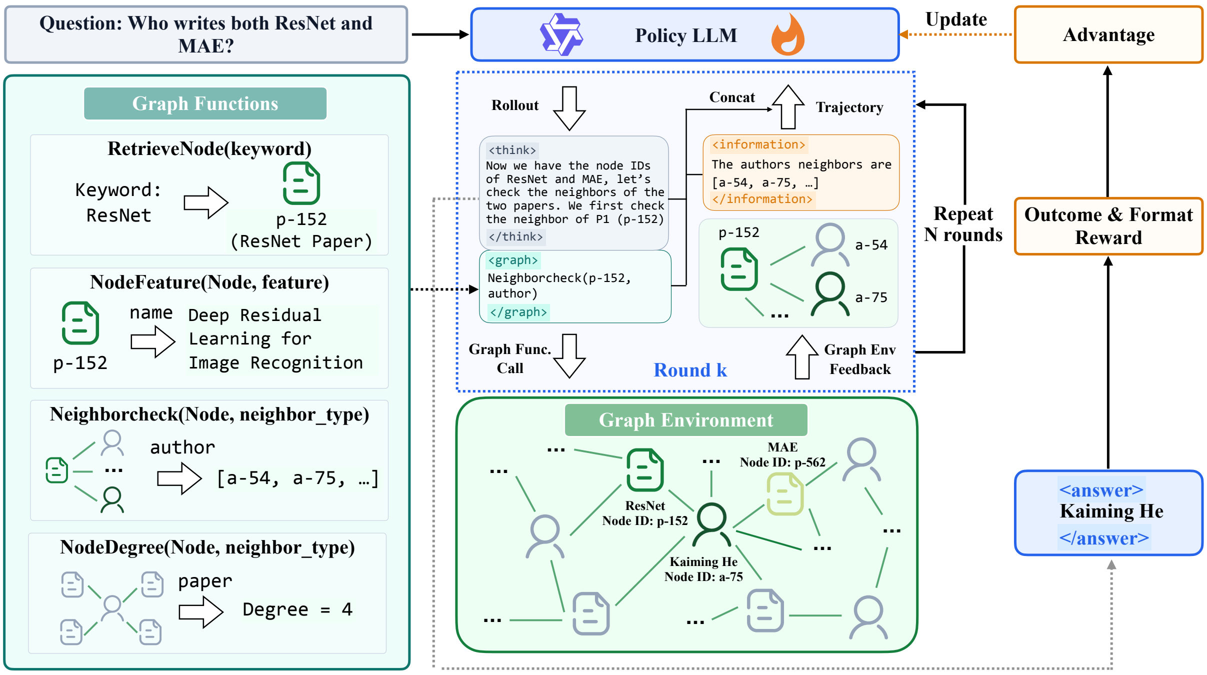

Figure 1: Overview of GraphDancer framework.

LLM reasoning (Guo et al., 2025) and information seeking (Jin et al., 2025), but also by the natural formulation of graph interaction as a decision-making process: graphs can be viewed as **environments**, function calls as **actions**, and the learning of graph interaction as **exploring the environment through actions**. Meanwhile, to make this solution applicable to open-source, moderate-sized LLMs (e.g., `Qwen2.5-3B-Instruct`; Yang et al., 2024), we propose to incorporate curriculum learning (Bengio et al., 2009; Parashar et al., 2025) into RL training, progressively increasing task difficulty to overcome their inherent reasoning limitations compared to larger models. Moreover, our **curriculum** introduces a novel **graph-aware** design. At the beginning of RL training, the vast majority of questions require only one-hop information seeking or reasoning over the graph (e.g., "*Who are the authors of the ResNet paper?*"). As training progresses, these are gradually replaced by questions that rely on multi-hop connections and multi-round information seeking (e.g., "*Which venue did Kaiming He and Ross Girshick collaborate most?*").

To demonstrate the efficacy and generalizability of GraphDancer, we train it on only one domain (i.e., Academic) from the existing benchmark (Jin et al., 2024) and evaluate it on the other domains (i.e., E-commerce, Literature, Healthcare, and Legal). Additionally, during evaluation, the model encounters never-seen-before, completely out-of-distribution (OOD) question types, such as those that cannot be answered by simply looking up the graph, yet where the graph may still provide valuable context (e.g., "*What book should be recommended to the user if they just read The Old Man and the Sea?*"). Experimental results demonstrate that GraphDancer, with a 3B backbone, outperforms baselines equipped with either a 14B backbone or `GPT-4o-mini`, particularly on hard and OOD questions in specific domains. Further analysis shows that GraphDancer substantially improves the LLM's behavioral reliability, notably increasing the rate of format-valid multi-round interactions. These results indicate that our model has internalized the ability to explore and reason over graphs, generalizing to new domains, unseen node/edge types, and novel question types.

The contributions of our work are as follows:

- We propose GraphDancer, an RL framework that teaches LLMs to interact with and reason over graphs in an adaptive, multi-round fashion, enabling the internalization of sophisticated graph exploration skills.
- We introduce a novel graph-aware curriculum that gradually increases question difficulty from one-hop to multi-hop and multi-round reasoning, allowing moderate-sized LLMs to progressively acquire complex graph reasoning abilities.
- We demonstrate the effectiveness and generalizability of GraphDancer via evaluation on multiple unseen domains and OOD question types, where it outperforms competitive baselines with

larger LLM backbones.

2 The GraphDancer Framework

In this section, we present GraphDancer, an RL framework that trains an LLM to *adaptively* interact with a graph via predefined function calls and to *interleave* information seeking with multi-step reasoning. We formulate graph-augmented reasoning as an *interactive* process in which the model alternates between natural-language reasoning and issuing graph function calls to acquire evidence. The model is trained via RL to internalize this multi-round interaction behavior, together with a *graph-aware curriculum* that schedules training examples by the structural complexity of their information-seeking traces. Figure 1 provides an overview of our GraphDancer framework.

2.1 Problem Setup

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a text-attributed graph with heterogeneous node/edge types (Jin et al., 2023), such as an academic network with papers, authors, and venues. Each node $v \in \mathcal{V}$ is associated with a set of textual fields (e.g., `title`, `name`, `abstract`) and typed relations to other nodes. Given a question x and the graph environment $\mathcal{G}$, the model needs to produce a natural-language answer y by interacting with $\mathcal{G}$ through a set of executable graph functions.

We model the process of answering a question as an episodic Markov decision process (MDP). At round t, the agent observes a state s_t comprising (1) the question x, (2) the history of prior reasoning and actions, and (3) the graph observations returned by previous function calls. The agent then selects an action a_t (i.e., a graph function call), receives an observation o_t (i.e., results returned by the function call), and updates its state. An episode terminates when the agent produces a final answer or reaches a maximum of T rounds.

2.2 LLM Reasoning with Graph Interaction

We use a simple interaction format to expose tool use to the LLM, inspired by multi-turn search calling (Jin et al., 2025). Specifically, the model generates a sequence interleaving three block types:

Reasoning block: `<think>` $\cdots$ `</think>` containing non-executable reasoning text.

Action block: `<graph>` $\cdots$ `</graph>` containing one or more function calls.

Observation block: `<information>` $\cdots$ `</information>` appended by the environment after executing the action block.

The final output is produced in an `<answer>` $\cdots$ `</answer>` block.

Graph Function Calls. To enable precise access over heterogeneous graphs, the agent does not retrieve arbitrary text as in vanilla RAG (Lewis et al., 2020). Instead, it invokes a small set of deterministic functions. Following the environment design in Jin et al. (2024), we assume the agent has access to the following predefined graph functions:

`RetrieveNode(Text)`: returns a ranked list of node IDs relevant to a textual query.

`NodeFeature(NodeID, FeatureName)`: returns the requested textual field for a node.

`NeighborCheck(NodeID, NeighborType)`: returns neighbor node IDs under a specified typed relation.

`NodeDegree(NodeID, NeighborType)`: returns the count of neighbors under a specified typed relation.

The environment validates each function call (including its name and arguments) and returns structured results (e.g., IDs or textual attributes) enclosed in `<information>` $\cdots$ `</information>`.

2.3 Training LLMs to Interact with Graphs

We train a policy model π_θ using rule-based reward, framing graph interaction as a sequential decision-making process. Crucially, the policy only generates *agent tokens* (reasoning, tool calls, and the final answer), while the graph environment deterministically executes tool calls and *injects observation tokens* that are not sampled from π_θ.

Let y denote the concatenation of all **agent-generated** tokens in an episode, and let $\tau = \mathrm{Env}(x, \mathcal{G}, y)$ be the full transcript formed by interleaving y with environment-inserted observations. Thus, τ is a deterministic function of the context $(x, \mathcal{G})$ and the agent's actions y given the tool executor. We optimize the following objective:

$$\max_\theta \quad \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta}\big[r(x,\mathcal{G},\tau)\big] - \gamma\, \mathbb{E}_{x\sim\mathcal{D}}\big[D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})\big], \tag{1}$$

where π_θ and π_{ref} are shorthand for the conditional distributions $\pi_\theta(\cdot \mid x; \mathcal{G})$ and $\pi_{\mathrm{ref}}(\cdot \mid x; \mathcal{G})$, respectively; π_{ref} is a fixed reference model; $\mathcal{D}$ denotes the training set; and γ is the KL-penalty coefficient.

Reward Design. We adopt a rule-based reward $r(x, \mathcal{G}, \tau)$ that primarily targets final answer correctness, with lightweight format shaping to encourage a well-formed interaction trace. Concretely, an episode receives an accuracy reward based on whether the content in the final `<answer>`

block matches the ground-truth answer. In addition, we apply a format reward to encourage adherence to the prescribed interaction protocol, which helps prevent malformed traces that cannot be executed by the environment or traces that omit the <answer> block. The complete reward definition and its mathematical formulation are provided in Appendix A.1.

2.4 Structural Difficulty of Graph Interaction

Graph interaction naturally decomposes into a sequence of *information-seeking rounds*, where function calls issued within the same round are expected to be independent of one another. Independence here means that each call relies only on entities already available in the agent's state at the beginning of the round, rather than on the outputs of other calls in the same round. As a result, calls within a round can, in principle, be executed in parallel. After receiving the returned evidence, the agent updates its state and determines the next action in the subsequent round.

Scope of an Information-Seeking Round. To quantify how a round expands the agent's explored subgraph, we focus on the *node identifiers* returned by the environment. Let $\mathcal{A}_t = \{c_{t,1}, \dots, c_{t,|\mathcal{A}_t|}\}$ denote the set of graph function calls issued in round t, and let $\mathrm{Nodes}(\cdot)$ extract the set of node IDs from a tool output. For a call $c \in \mathcal{A}_t$ whose return type does not include node identifiers (e.g., textual fields or scalar values), we define $\mathrm{Nodes}(c) = \emptyset$. We then define the set of node IDs surfaced in round t as $\mathcal{U}_t = \bigcup_{c\in\mathcal{A}_t} \mathrm{Nodes}(c)$.

$\mathcal{U}_t$ can be viewed as a measure of how much the model expands its exploration *scope* on the graph in round t. Based on this measure, we can categorize information-seeking rounds into two types:

Singleton lookup round (S-round): This round introduces exactly one node (i.e., $|\mathcal{U}_t| = 1$).

Neighborhood expansion round (E-round): This round brings in multiple nodes (i.e., $|\mathcal{U}_t| > 1$).

Typically, S-rounds occur when the agent needs to *identify a single pivot node* for subsequent reasoning, such as resolving an entity mention to a unique node via `RetrieveNode` (e.g., mapping "*ResNet*" to its corresponding paper node) or retrieving a typed neighbor that is unique under the graph schema (e.g., obtaining the sole "*published in*" venue of a paper). In contrast, E-rounds involve neighborhood expansions that introduce multiple candidate nodes, requiring downstream selection or aggregation.

Structural Difficulty. The difficulty of graph interaction varies across questions. To quantify this, we operationalize graph-aware difficulty based on the round decomposition above, using the total number of rounds and the number of E-rounds:

Easy questions require a single round of information seeking, regardless of whether it is an S-round or an E-round.

Medium questions require multiple rounds but at most one E-round.

Hard questions require at least two E-rounds, reflecting repeated neighborhood expansions followed by aggregation or path reasoning.

This categorization directly informs our graph-aware curriculum (to be described below), which schedules training from S-round-dominated questions toward those with multiple E-rounds.

2.5 Graph-Aware Curriculum RL

RL struggles on hard reasoning tasks because moderate-sized LLMs often perform poorly in the zero-shot setting, and task-specific rewards are sparse, making direct learning unstable and inefficient (Parashar et al., 2025). Curriculum learning (Bengio et al., 2009) addresses this by training from easier to harder tasks and decomposing complex skill acquisition into manageable steps. In our setting, this idea can be implemented via the structural difficulty defined above.

Easy-to-Hard Sampling with a Biased Mixture. Let $k \in \{1, \dots, K\}$ index our graph-aware difficulty levels (e.g., $K = 3$ when we have the categorization in Section 2.4), ordered from easiest to hardest. We adopt Gaussian curriculum sampling (Parashar et al., 2025), which converts a scheduler score into a probability distribution:

$$p_t(k) = \frac{S(t,k)}{\sum_{k'=1}^{K} S(t,k')}. \tag{2}$$

Here, $S(t,k) \geq 0$ specifies the sampling preference for difficulty level k at RL step t. Prior easy-to-hard schedules (Team et al., 2025; Bercovich et al., 2025) typically define $S(t,k)$ using a single parametric shape that shifts mass over time. In our setting, a *pure* easy-to-hard schedule can be brittle. This is because hard questions with multiple E-rounds exhibit branching expansions and longer-horizon credit assignment. In this case, overly aggressive exposure can produce low-reward rollouts and unstable updates, while concentrating too late may impede the consolidation of prerequisite tool-use behaviors. To improve robustness and encode task-specific sampling preferences, we propose a *time-varying convex mixture* scheduler. Specifically, we define a normalized level distribution as:

$$\tilde{p}_t(k) \;=\; (1-\eta(t))\, p_g(t,k) \;+\; \eta(t)\, q(k), \quad (3)$$

where $p_g(t,\cdot)$ is the Gaussian curriculum distribution at step t (shape controlled by hyperparameters such as σ and the shifting mean), $q(\cdot)$ is a fixed bias prior over levels, and $\eta(t) \in [0,1]$ controls the mixing strength. In GRAPHDANCER, we use a linear schedule:

$$\eta(t) = \eta_{\text{start}} + \frac{t}{T_{\text{RL}}-1}(\eta_{\text{end}} - \eta_{\text{start}}), \quad (4)$$

where T_{RL} is the number of RL training steps, and $t = 0, 1, \ldots, T_{\text{RL}} - 1$.

Finally, for notational consistency with Eq. 2, we interpret $\tilde{p}_t$ as induced by a scheduler $S(t,k) \propto \tilde{p}_t(k)$. We sample $k \sim \tilde{p}_t(\cdot)$ and then select an instance uniformly within level k. This biased-mixture design preserves the smooth easy-to-hard progression of p_g while maintaining controlled exposure to each level, which helps the model learn stable and generalizable graph tool-use behaviors under sparse rule-based rewards.

2.6 Training Procedure

We summarize the end-to-end training pipeline of GRAPHDANCER. We initialize π_θ from an instruction-tuned backbone and construct the graph-aware difficulty levels (Easy/Medium/Hard). During RL, at each step t, we sample a level from our biased-mixture curriculum (Eqs. 3-4) and uniformly select an instance from that level. We roll out an interleaved trajectory under a fixed interaction budget, compute the rule-based reward $r(x, \mathcal{G}, \tau)$ (Eq. 5 in Appendix A.1), and update the policy under KL regularization, masking environment-injected observation tokens so that gradients are applied only to agent-generated tokens. The full procedure is given in Algorithm 1.

Algorithm 1 GRAPHDANCER

Input: dataset $\mathcal{D}$ of $(x, \mathcal{G}, y^\star)$; graph executor Env; reference policy π_{ref}
Hyperparameters: RL steps T_{RL}, max rounds T, KL coefficient β, epochs E, minibatches B, curriculum parameters $(p_g, q, \eta(\cdot))$
Initialize policy π_θ from backbone (optional format warmup SFT)
Preprocess: compute round decomposition and assign each instance to a difficulty level (Section 2.4)
for $t = 0$ **to** $T_{\text{RL}} - 1$ **do**
  Compute curriculum distribution $\tilde{p}_t(\cdot)$ using Eqs. 3-4
  Sample level $k \sim \tilde{p}_t(\cdot)$; sample $(x, \mathcal{G}, y^\star)$ uniformly from level k
  Roll out agent tokens $y \sim \pi_\theta(\cdot \mid x; \mathcal{G})$ with at most T rounds of <graph> calls
  Form transcript $\tau \leftarrow \text{Env}(x, \mathcal{G}, y)$ by executing calls and injecting <information> blocks
  Compute reward $r(x, \mathcal{G}, \tau)$ (Eq. 5 in Appendix A.1)
  Compute token-wise advantages on **agent-generated** tokens only (mask observation tokens)
  for $e = 1$ **to** E **do**
    Update θ on B minibatches, using KL regularization to π_{ref} (Eq. 1)
  end for
end for
Return: trained policy π_θ

3 Experiments

3.1 Experimental Setup

Dataset. We use GRBench (Jin et al., 2024) to assess the ability of LLMs to interact with graphs. In this benchmark, we train on the ACADEMIC domain only and test cross-domain generalization on four unseen domains: E-COMMERCE, LITERATURE, HEALTHCARE, and LEGAL. The ACADEMIC training set contains only questions that were originally labeled Easy or Medium in GRBench (i.e., questions that can always be correctly answered if the model interacts with the graph properly), which we reclassify into Easy, Medium, and Hard according to our structural difficulty categorization (Section 2.4). Questions that were originally labeled Hard in GRBench (i.e., those that cannot be answered by simply looking up the graph, though the graph may still provide useful context) are treated as out-of-distribution (OOD) samples. They are excluded from training and used solely for evaluation. The detailed statistics of the datasets can be found in Table 1.

Domain	Easy	Medium	Hard	OOD	Total
ACADEMIC	370	120	310	50	850
E-COMMERCE	80	40	40	40	200
LITERATURE	130	30	70	10	240
HEALTHCARE	100	150	20	0	270
LEGAL	90	40	30	20	180

Table 1: Dataset statistics. Red : training data. Blue : testing data. Difficulty levels are defined in Section 2.4.

Baselines. We evaluate our framework against both prompting approaches and the vanilla RL baseline.

For prompting baselines, we consider **TextRAG** (Gao et al., 2023), **GraphRAG** (Ye et al., 2024), **Think-on-Graph 2.0** (Ma et al., 2025; dubbed as ToG-2), and **Graph-CoT** (Jin et al., 2024). Jin et al. (2024) report the performance

Method	Model	E-COMMERCE R-L	E-COMMERCE GS	LITERATURE R-L	LITERATURE GS	HEALTHCARE R-L	HEALTHCARE GS	LEGAL R-L	LEGAL GS	Average R-L	Average GS	Gap Δ
TextRAG (Gao et al., 2023)	`GPT-3.5-turbo`	14.06	20.00	10.04	20.83	4.57	8.52	18.14	23.89	11.70	18.31	↑ 26.4
GraphRAG (Ye et al., 2024)	`GPT-3.5-turbo`	17.52	28.00	14.94	24.17	8.69	14.07	18.66	22.22	14.95	22.12	↑ 22.9
ToG-2 (Ma et al., 2025)	`GPT-4o-mini`	29.28	35.00	21.97	30.42	18.66	18.89	25.59	23.33	23.88	26.91	↑ 16.0
Graph-CoT (Jin et al., 2024)	`GPT-3.5-turbo`	42.40	44.50	41.59	46.25	22.33	29.89	30.52	28.33	34.21	37.24	↑ 5.7
	`GPT-4o-mini`	37.06	39.50	36.04	46.25	**39.88**	**41.48**	35.47	40.00	37.11	41.81	↑ 2.0
	`Qwen2.5-3B-Instruct`	43.58	41.00	43.60	47.50	27.47	26.67	30.68	35.56	36.33	37.68	↑ 4.4
	`Qwen3-14B`	39.91	42.50	**48.03**	**53.33**	38.82	36.67	32.66	35.56	39.85	42.01	↑ 0.5
Vanilla RL	`Qwen2.5-3B-Instruct`	45.66	45.00	47.36	49.17	27.09	25.93	33.84	36.11	38.49	39.05	↑ 2.7
GRAPHDANCER	`Qwen2.5-3B-Instruct`	**48.87**	**47.00**	43.82	49.58	31.55	29.63	**38.25**	**42.78**	**40.62**	**42.25**	-

Table 2: Main results on GRbench. We highlight the Average performance (purple columns) and the performance Gap (Δ) compared to our method (green column).

of TextRAG, GraphRAG, and Graph-CoT using `GPT-3.5-turbo` (Ouyang et al., 2022) as the backbone, which we directly adopt. In addition, we experiment with more recent and diverse backbones, including `GPT-4o-mini` (Hurst et al., 2024) on ToG-2, and `GPT-4o-mini`, `Qwen2.5-3B-Instruct` (Yang et al., 2024), and `Qwen3-14B` (Yang et al., 2025) on Graph-CoT.

The **Vanilla RL** baseline can be viewed as an ablation of **GRAPHDANCER**, using the same reward function and hyperparameters but without the proposed graph-aware curriculum. For both Vanilla RL and GRAPHDANCER, we require an open-source backbone and adopt the moderate-sized `Qwen2.5-3B-Instruct`.

Evaluation Metrics. We report two complementary metrics: **Rouge-L** (Lin and Och, 2004) between the generated answer and the reference, and **GPT4Score**, an LLM-as-a-judge score (Li et al., 2025) computed with the same prompt and rubric as in Jin et al. (2024).

3.2 Overall Performance

Table 2 compares GRAPHDANCER with the baselines across four unseen domains. Overall, GRAPHDANCER achieves the best average performance (40.62 Rouge-L / 42.25 GPT4Score) despite using a 3B backbone.

GRAPHDANCER can make a small backbone competitive for graph interaction. Despite its smaller size, GRAPHDANCER matches or outperforms larger prompting-based baselines on average, including Graph-CoT with `Qwen3-14B` (39.85 Rouge-L / 42.01 GPT4Score) and `GPT-4o-mini`. This suggests that directly optimizing the multi-round `reasoning`→`action`→`observation` loop can provide benefits beyond in-context prompting, particularly when tool use and long-horizon credit assignment are critical.

Domain transfer is strong but not uniform. Compared with Vanilla RL, GRAPHDANCER improves Rouge-L on E-COMMERCE (+3.21) and LEGAL (+4.41), and also yields gains on HEALTHCARE. However, performance drops on LITERATURE, indicating that cross-domain transfer may depend on domain-specific graph structures and answer distributions. We therefore complement aggregate scores with a difficulty-wise analysis below to better localize where the curriculum is most effective.

Domain	Method	Easy	Medium	Hard	OOD
ACADEMIC	Graph-CoT	61.71	11.16	7.05	11.81
	Vanilla RL	56.51	22.20	18.12	17.64
	GRAPHDANCER	**67.52**	**55.04**	**20.33**	**18.20**
E-COMMERCE	Graph-CoT	82.34	38.47	11.64	3.49
	Vanilla RL	82.03	47.31	12.41	4.77
	GRAPHDANCER	**84.96**	**52.26**	**16.23**	**4.90**
LITERATURE	Graph-CoT	63.52	55.14	6.59	1.67
	Vanilla RL	65.80	**55.44**	**15.28**	1.96
	GRAPHDANCER	**68.22**	48.22	2.53	**4.58**
HEALTHCARE	Graph-CoT	63.23	7.09	0.00	–
	Vanilla RL	61.13	7.07	**5.00**	–
	GRAPHDANCER	**64.27**	**14.01**	0.00	–
LEGAL	Graph-CoT	52.93	9.04	4.39	4.29
	Vanilla RL	53.07	13.98	4.00	**19.33**
	GRAPHDANCER	**58.14**	**16.73**	**15.06**	16.11

Table 3: Difficulty-wise Rouge-L (%) across domains. Easy/Medium/Hard/OOD split follows our structural definition (Section 2.4). The ACADEMIC Easy/Medium/Hard subsets correspond to the *training set* (OOD excluded from training). All three baselines are using `Qwen2.5-3B-Instruct` as LLM backbone.

3.3 Impact of Graph-Aware Curriculum

Table 3 breaks down Rouge-L by structural difficulty. On the ACADEMIC training domain, the largest gap between GRAPHDANCER and Vanilla RL appears on **Medium** samples (55.04 vs. 22.20), which require multi-round interaction but limited branching. This pattern aligns with our motivation: scheduling from S-round-dominated samples to more involved traces can increase the frequency of successful multi-round trajectories early in training, mitigating sparse-reward instability.

On unseen domains, GRAPHDANCER exhibits clear gains on **Hard** samples in E-COMMERCE (16.23 vs. 12.41) and especially LEGAL (15.06 vs. 4.00), suggesting improved robustness when repeated neighborhood expansions are required. At the same time, LITERATURE and HEALTHCARE remain challenging on Hard subsets, where GRAPHDANCER does not consistently outperform Vanilla RL. For **OOD** questions, GRAPHDANCER is competitive and improves in E-COMMERCE and LITERATURE. However, gains are not uniform across all domains (e.g., LEGAL-OOD). These mixed results indicate that the curriculum primarily strengthens structured, multi-round tool use, while open-ended OOD answering may require additional objectives beyond rule-based rewards.

3.4 Qualitative Analysis

To assess whether GRAPHDANCER has internalized generalizable graph exploration skills rather than memorizing domain-specific patterns, we analyze a Medium-difficulty instance from the E-COMMERCE domain, which is unseen during training. The question asks for the price of an item *bought together* with a specific loudspeaker, requiring a two-hop reasoning path: *Anchor Item* → *Neighbor Item* → *Price*.

Figure 2 compares the reasoning trajectories.[1] Both models correctly identify the anchor node (`B000NJIYHY`). However, their behaviors diverge markedly during graph traversal. **Vanilla RL** lacks a robust expansion strategy: it uses `NodeDegree` to verify the existence of a neighbor but fails to retrieve the corresponding node ID. It then attempts to hallucinate a node identifier (`..._item_0`) based on a pseudo-array indexing logic, which results in a function-call error. Following this failure, the baseline queries the anchor node's price instead of the neighbor's, leading to an incorrect answer.

In contrast, **GRAPHDANCER** correctly decomposes the query. It uses `NeighborCheck` to retrieve the exact ID of the associated item, and then performs a second hop with `NodeFeature` on this *new* node to obtain the correct price ($12.95).

This case highlights that GRAPHDANCER has learned the *syntax of exploration* (e.g., "get ID first, then query attributes"), enabling it to navigate unseen schemas effectively, whereas the baseline struggles with valid tool chaining and falls back on shallow heuristics.

Case Study: Cross-domain generalization. Interaction traces on a Medium-difficulty query (E-COMMERCE)

Question (E-COMMERCE):
```
What is the usual price of the items often purchased together
with "PYLE-PRO PPHP1293 - 800 Watt 12" Two-Way Plastic Molded
Loudspeaker"?
```

Baseline (Vanilla RL):
```
<think> Locate the item node first. </think>
<graph> RetrieveNode[PYLE-PRO...] </graph>
<information> [B000NJIYHY] </information>
<think> Check degree of bought_together neighbors. </think>
<graph> NodeDegree[B000NJIYHY, bought_together] </graph>
<information> [1] </information>
<think> Access the neighbor using index 0. </think>
<graph> NodeFeature[B000NJIYHY_bought_together_item_0,
price] </graph>
<information> Node or feature name does not exist.
</information>
<think> Access failed. I will assume the price is similar
to the original item. </think>
<graph> NodeFeature[B000NJIYHY, price] → [140.43] </graph>
<answer> 140.43 </answer>   (✗ Wrong: Returns anchor price)
```

Ours (GraphDancer): *Structured Graph Exploration*
```
<think> Find anchor node, then find its neighbors. </think>
<graph> RetrieveNode[PYLE-PRO...]</graph>
<information> [B000NJIYHY] </information>
<think> I need the ID of the neighbor to query its price.
</think>
<graph> NeighborCheck[B000NJIYHY, bought_together] →
</graph>
<information> ['B000E1U4WY'] </information>
<think> Now query the price of the found neighbor B000E1U4WY.
</think>
<graph> NodeFeature[B000E1U4WY, price] </graph>
<information> [12.95]</information>
<answer> 12.95 </answer>   (✓ Correct)
```

3.5 Error Analysis of Graph Interaction

To investigate the behavioral shifts induced by our training, we decompose each episode into four mutually exclusive outcomes: *Correct*, *Invalid Format* (invalid tool calls), *Loop / Timeout* (exceeding the interaction budget), and *Premature Stop* (terminating with an incorrect answer). Figure 3 summarizes the resulting outcome distribution across the four test domains.

Overall, GRAPHDANCER improves the success rate (39.0%) compared with the base model (36.6%) and Vanilla RL (37.2%). More importantly, GRAPHDANCER substantially reduces navigational paralysis: *Loop / Timeout* drops to 11.8% of all episodes, versus 21.2% (Qwen2.5-3B-Instruct) and 19.8% (Vanilla RL). When conditioning on incorrect episodes only, the proportion of *Loop / Timeout* nearly halves from 33.5%/31.5% to **19.3%**, indicating that the curriculum teaches the agent to navigate purposefully and to recognize when to terminate exploration.

This reduction in aimless navigation comes with a shift in the residual error profile toward *Premature Stop*. Its overall proportion increases to 43.5% (and 71.3% among incorrect episodes),

[1] For visual clarity and space constraints, the prompts and observations presented here are slightly paraphrased without altering the semantics. Please refer to Appendix B.2 for the complete outputs.

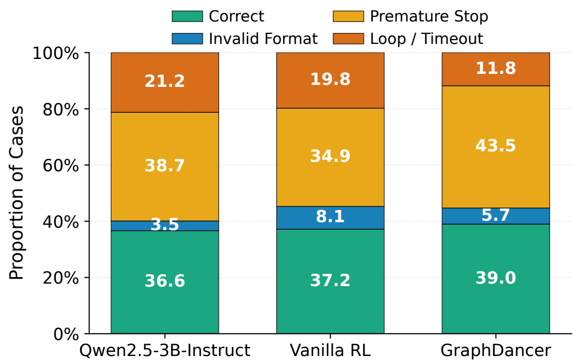

Figure 3: Outcome breakdown over all evaluation episodes. Each bar decomposes episodes into *Correct* and three failure modes: *Invalid Format*, *Loop / Timeout*, and *Premature Stop*. GraphDancer increases the overall success rate and noticeably reduces *Loop / Timeout*, while the remaining errors are more concentrated in *Premature Stop*.

which reflects a trade-off between exploration depth and timely termination, especially for multi-hop queries. While still an error, this represents a qualitative improvement: the agent is no longer stalling or crashing, but actively attempting to formulate an answer. Finally, GraphDancer mitigates the instability of standard RL fine-tuning: while Vanilla RL exhibits a higher rate of *Invalid Format* (8.1%), GraphDancer reduces it to 5.7%, better preserving adherence to the function-calling schema during exploration.

4 Related Work

LLM Agents on Graphs. While early works primarily focused on static integration, either using LLMs to enhance Graph Neural Networks (Gori et al., 2005; Santoro et al., 2017; Zhou et al., 2020) via feature extraction (Zhao et al., 2023; Duan et al., 2023) or linearizing graph structures for direct LLM processing (Ye et al., 2024), these approaches often struggle with the complexity of multi-hop reasoning over external knowledge. Consequently, recent research has shifted toward an agentic paradigm, framing graph reasoning as an interactive "Think-Act-Observe" decision-making process. Besta et al. (2024) introduced Graph-of-Thoughts to model reasoning itself as a graph, while Graph-CoT (Jin et al., 2024) enables LLMs to actively traverse external graphs through iterative function calls. Expanding on this line of work, Graph-R1 (Luo et al., 2025) and Structure-R1 (Wu et al., 2025) leverage RL to optimize retrieval strategies or learn structured representations, such as tables, from unstructured text to facilitate reasoning. Unlike these methods, which often depend on heavy prompting or disjointed training, our approach internalizes graph exploration skills directly via curriculum-based RL.

RL for Reasoning and Curriculum Learning. While RL was initially introduced to align LLMs with human preferences via PPO (Ouyang et al., 2022) or direct optimization methods such as DPO (Rafailov et al., 2023), recent work has shifted toward enhancing complex reasoning capabilities through RL with Verifiable Rewards (RLVR; Mroueh, 2025). Pioneering studies, including DeepSeek-R1 (Guo et al., 2025), show that LLMs can self-improve their reasoning skills when provided with rule-based feedback (e.g., correctness in math or code). To facilitate learning in long-horizon tasks, curriculum learning has been integrated into RL, using strategies such as easy-to-hard scheduling (Parashar et al., 2025), adaptive rollout control (Shen et al., 2025), and multi-objective balancing (Hammoud et al., 2025). However, existing RLVR approaches primarily focus on internal reasoning tasks, such as math and logic, or unstructured web search (Jin et al., 2025). Our work bridges this gap by applying curriculum-based RL to *structured graph environments*, enabling LLMs to internalize the specialized skills required for multi-hop graph exploration.

5 Conclusion

In this paper, we present GraphDancer, an RL framework designed to equip LLMs with the capability to autonomously explore and reason over structured graph environments. By incorporating a novel graph-aware curriculum, we enable moderate-sized models to progressively internalize complex multi-hop reasoning skills, moving beyond the limitations of static retrieval or heavy prompting strategies. Extensive experiments show that GraphDancer generalizes effectively across diverse unseen domains, including E-commerce, Literature, Healthcare, and Legal, after training on a single Academic domain. Notably, GraphDancer demonstrates robust adaptability to out-of-distribution question types, where graph interactions serve as contextual grounding rather than direct lookups. These results suggest that through adaptive interaction and curriculum-based training, LLMs can acquire truly generalizable graph reasoning abilities, paving the way for more reliable and factually grounded agents in knowledge-intensive applications.

Limitations

Our framework involves several design choices, including the curriculum shape, biased-mixture schedule, and RL hyperparameters. Due to the high computational cost of RL fine-tuning, we do not perform an exhaustive sensitivity study over this space, such as varying mixture coefficients, level priors, KL strength, or rollout budgets. Pilot runs indicate that auxiliary format rewards and biased-mixture sampling can significantly impact training stability, but a systematic multi-seed ablation with complete learning curves is left to future work. Additionally, we adopt the deterministic Graph-CoT tool API; how well the learned policy transfers to substantially different tool interfaces, such as non-deterministic tools, or noisy and incomplete outputs, remains an open question.

Ethical Considerations

Our work enhances an LLM's ability to explore graph-structured knowledge through executable function calls, which can improve grounding and factuality but also introduces risks when connected to real-world graphs. In particular, graph tool APIs may expose sensitive information, such as proprietary relations or user data, and could be misused for broad traversal or data exfiltration if access is not carefully controlled. Deployments should therefore enforce strict authentication and authorization, least-privilege function permissions, auditing, and redaction of sensitive fields in tool outputs. Because node attributes and observations may contain untrusted text, systems are also vulnerable to prompt-injection attacks that attempt to manipulate subsequent actions. Tool outputs should be sanitized and isolated, and all calls should be validated against an allowlist of functions and argument schemas. Finally, while our method reduces some behavioral failures, such as invalid calls, it does not guarantee correctness. Agents may still retrieve incomplete evidence or terminate prematurely. We therefore view it as decision support rather than a substitute for expert judgment, especially in high-stakes domains.

References

Alfonso Amayuelas, Joy Sain, Simerjot Kaur, and Charese Smiley. 2025. Grounding llm reasoning with knowledge graphs. *arXiv preprint arXiv:2502.13247*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *ICLR'24*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *ICML'09*, pages 41–48.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, and 1 others. 2025. Llama-nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI'24*, pages 17682–17690.

Keyu Duan, Qian Liu, Tat-Seng Chua, Shuicheng Yan, Wei Tsang Ooi, Qizhe Xie, and Junxian He. 2023. Simteg: A frustratingly simple approach improves textual graph learning. *arXiv preprint arXiv:2308.02565*.

Junqi Gao, Xiang Zou, Ying Ai, Dong Li, Yichen Niu, Biqing Qi, and Jianxing Liu. 2025. Graph counselor: Adaptive graph exploration via multi-agent synergy to enhance llm reasoning. In *ACL'25*, pages 24650–24668.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *IJCNN'05*, pages 729–734.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Hasan Abed Al Kader Hammoud, Kumail Alhamoud, Abed Hammoud, Elie Bou-Zeid, Marzyeh Ghassemi, and Bernard Ghanem. 2025. Train long, think short: Curriculum learning for efficient reasoning. *arXiv preprint arXiv:2508.08940*.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Bowen Jin, Chulin Xie, Jiawei Zhang, Kashob Kumar Roy, Yu Zhang, Zheng Li, Ruirui Li, Xianfeng Tang, Suhang Wang, Yu Meng, and 1 others. 2024. Graph chain-of-thought: Augmenting large language models by reasoning on graphs. In *Findings of ACL'24*, pages 163–184.

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

Bowen Jin, Yu Zhang, Qi Zhu, and Jiawei Han. 2023. Heterformer: Transformer-based deep node representation learning on heterogeneous text-rich networks. In *KDD'23*, pages 1020–1031.

Savini Kashmira, Jayanaka L Dantanarayana, Krisztián Flautner, Lingjia Tang, and Jason Mars. 2025. Graphrunner: A multi-stage framework for efficient and accurate graph-based retrieval. *arXiv preprint arXiv:2507.08945*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS'20*, pages 9459–9474.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, and 1 others. 2025. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *EMNLP'25*, pages 2757–2791.

Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *ACL'04*, pages 605–612.

Haoran Luo, Haihong E, Guanting Chen, Qika Lin, Yikai Guo, Fangzhi Xu, Zemin Kuang, Meina Song, Xiaobao Wu, Yifan Zhu, and Luu Anh Tuan. 2025. Graph-r1: Towards agentic graphrag framework via end-to-end reinforcement learning. *arXiv preprint arXiv:2507.21892*.

Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. 2025. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. In *ICLR'25*.

Youssef Mroueh. 2025. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. *arXiv preprint arXiv:2503.06639*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. In *NeurIPS'22*, pages 27730–27744.

Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2025. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. *arXiv preprint arXiv:2506.06632*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS'23*.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. 2017. A simple neural network module for relational reasoning. In *NeurIPS'17*, pages 4967–4976.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *NeurIPS'23*, pages 68539–68551.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. 2025. Thinking vs. doing: Improving agent reasoning by scaling test-time interaction. In *NeurIPS'25*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024a. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *ICLR'24*.

Kai Sun, Yifan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. 2024b. Head-to-tail: How knowledgeable are large language models (llms)? aka will llms replace knowledge graphs? In *NAACL'24*, pages 311–325.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and 1 others. 2024. Freshllms: Refreshing large language models with search engine augmentation. In *Findings of ACL'24*, pages 13697–13720.

Junlin Wu, Xianrui Zhong, Jiashuo Sun, Bolian Li, Bowen Jin, Jiawei Han, and Qingkai Zeng. 2025. Structure-r1: Dynamically leveraging structural knowledge in llm reasoning through reinforcement learning. *arXiv preprint arXiv:2510.15191*.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2024. Language is all a graph needs. In *Findings of EACL'24*, pages 1955–1973.

Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. 2023. Learning on large-scale text-attributed graphs via variational inference. In *ICLR'23*.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

A Implementation Details

A.1 Reward Function Details

Let $\hat{y}$ denote the content extracted from the final `<answer>` $\cdots$ `</answer>` block, and $y^\star$ be the ground-truth answer for $(x, \mathcal{G})$. We define an exact-match indicator $\mathrm{EM}(\hat{y}, y^\star) \in \{0,1\}$, a format validity indicator $\mathrm{VF}(\tau) \in \{0,1\}$ that checks whether the model follows the interaction protocol in Section 2.2, and an answer-presence indicator $\mathrm{AP}(\hat{y}) \in \{0,1\}$ that examines whether the final answer is non-empty. The reward used in Eq. 1 is:

$$\begin{aligned} r(x,\mathcal{G},\tau) = \ & \mathrm{EM}(\hat{y}, y^\star) \\ & - \lambda_{\text{struct}}\, \mathrm{EM}(\hat{y}, y^\star)\,(1 - \mathrm{VF}(\tau)) \\ & + \lambda_{\text{final}}\,(1 - \mathrm{EM}(\hat{y}, y^\star))\,\mathrm{VF}(\tau)\,\mathrm{AP}(\hat{y}). \end{aligned} \tag{5}$$

where λ_{struct} and λ_{final} are coefficients that control the strength of the structural shaping and final-answer shaping terms, respectively.

Intuitively, EM provides the primary learning signal by rewarding correct final answers. The second term applies a mild penalty when the answer is correct but the interaction trace violates the protocol. The last term gives a small positive reward to well-formed, non-empty attempts even when the final answer is incorrect, which helps prevent degenerate behaviors such as empty outputs or malformed tool text.

A.2 Baselines Details

- **TextRAG (Gao et al., 2023) & GraphRAG (Ye et al., 2024)**: Following the setup in Jin et al. (2024), we include two retrieval-augmented baselines that ground the LLM on graph knowledge via *linearized* context rather than typed function calls. TextRAG treats the graph as a plain text corpus and uses a retriever to fetch relevant textual entries, which are appended to the prompt for single-pass answering. GraphRAG extends TextRAG by additionally retrieving the *associated local subgraph* of the retrieved entry and linearizing it as extra context, after which the backbone produces the answer in the same single forward pass without executing any structured graph functions.

- **Graph-CoT (Jin et al., 2024)**: We report results on `GPT-3.5-turbo` and `GPT-4o-mini` using the original system instructions and few-shot examples from Jin et al. (2024). For `Qwen2.5-3B-Instruct` and `Qwen3-14B`, we utilize the *same* interaction format as our RL policy (described in Section 2.2).

- **ToG-2 (Ma et al., 2025)**: This is a knowledge-guided RAG method that interleaves reasoning with iterative retrieval over graph-structured knowledge. To ensure a fair comparison, we use the same text-to-node linker as in Graph-CoT, `all-mpnet-base-v2`,[2] when mapping textual mentions to graph nodes.

- **Vanilla RL.** We fine-tune the `Qwen2.5-3B-Instruct` backbone with the same RL reward function (Eq. 5) and hyperparameters as GRAPH-DANCER. This baseline samples training data x uniformly from the dataset, without the proposed graph-aware curriculum.

A.3 Hyperparameters

Graph Executor and Interaction Budget. All models interact with the same graph executor and function API (Section 2.2). We set `max_turns` $T = 10$. Within each turn, the model may emit a batch of graph calls inside a `<graph>` block, which are executed deterministically.

Training and Inference Details. For all Qwen-family backbones, we apply the same decoding configuration for both inference-time generation

[2] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

and rollout generation: we set temperature as 0.7, top-p as 0.8, and top-k as 20. For RL training, we train for a total of 200 steps using a global rollout buffer and mini-batch size of 128 (micro-batch size 8 per GPU). We use a constant KL penalty $\beta = 0.001$ relative to the reference policy, a policy clip ratio of 0.2, and a value clip range of 0.5. The actor and critic learning rates are set to 10^{-6} and 10^{-5}, respectively.

Curriculum Configuration for GraphDancer. We use the Gaussian curriculum schedule with $\beta = 3$ and $\sigma = 0.75$. We use a time-varying mixture (Eq. 3) with $\eta_{\text{start}} = 0.2$ and $\eta_{\text{end}} = 0.8$, and a fixed level bias prior $q = [0.5, 0.5, 0]$ over {Easy, Medium, Hard}.

B Analysis

B.1 Behavioral Analysis of Graph Interaction

Outcome metrics alone do not reveal whether a model truly internalizes the multi-round `reasoning→action→observation` procedure. We therefore complement Table 2 with behavioral diagnostics computed from execution traces.

Metrics. We report (1) Format Validity (VF), the proportion of trajectories that follow the required <think> / <graph> / <information> / <answer> structure; (2) Call Validity (CV), the fraction of tool calls that pass schema validation; and (3) Evidence Hit (EH), the fraction of episodes where the normalized gold answer appears in any returned `<information>` block.

Findings. Table 4 shows that GraphDancer substantially improves VF across all unseen domains, indicating that curriculum-based RL helps the model reliably follow the intended multi-round interaction protocol. Notably, Vanilla RL exhibits much lower VF than the base model in every domain (e.g., 0.138 vs. 0.488 on Literature, and 0.290 vs. 0.625 on E-commerce), suggesting that RL without curriculum can degrade structural compliance during exploration. We hypothesize that sparse rewards and exploration noise can destabilize format adherence; curriculum mitigates this by increasing early successful trajectories. Evidence hit rates (EH) improve modestly but consistently for GraphDancer compared to baselines. This suggests that GraphDancer more frequently surfaces answer-relevant evidence in tool observations.

Difficulty-wise Breakdown. Tables 5-8 report behavioral diagnostics stratified by our structural difficulty levels (Easy/Medium/Hard). We report VF (format validity), CV (schema-valid tool-call rate), and EH (evidence hit rate; whether the normalized gold answer appears in any `<information>` block).

Domain	Model	VF ↑	CV ↑	EH ↑
E-commerce	Graph-CoT	62.5	99.1	49.0
	Vanilla RL	29.0	97.0	49.0
	GraphDancer	**78.0**	99.1	**50.0**
Literature	Graph-CoT	48.8	98.6	51.7
	Vanilla RL	13.8	96.3	52.9
	GraphDancer	**70.0**	**100.0**	**55.8**
Healthcare	Graph-CoT	31.9	**99.3**	20.4
	Vanilla RL	21.9	99.0	17.8
	GraphDancer	**46.7**	98.9	**21.9**
Legal	Graph-CoT	42.8	**97.8**	**46.7**
	Vanilla RL	26.1	91.3	43.3
	GraphDancer	**54.4**	89.1	43.9

Table 4: Behavioral analysis of execution traces on unseen domains. VF: fraction of episodes following the required interaction format; CV: fraction of tool calls passing schema validation; EH: fraction of episodes where the normalized gold answer appears in any <information> block. All three baselines are using `Qwen2.5-3B-Instruct` as LLM backbone.

Analysis. Across domains, VF generally decreases as difficulty increases, reflecting the challenge of maintaining a clean multi-round interaction protocol under longer-horizon reasoning. Vanilla RL is particularly brittle on Medium/Hard samples, where VF is often near zero (e.g., Medium E-commerce and Hard Literature/Legal), suggesting that RL without curriculum can lead to frequent deviations from the required tool-use format. In contrast, GraphDancer consistently improves VF on Medium/Hard, indicating that curriculum-based RL better preserves structural compliance.

Schema-level tool-call validity (CV) remains high for most settings but exhibits notable drops on Legal Medium/Hard for RL-trained models, consistent with the hypothesis that certain domains induce more complex or error-prone argument patterns. Finally, EH is high on Easy subsets (often > 0.65) but can be substantially lower on Hard subsets (e.g., E-commerce and Literature), indicating that the gold answer string may not frequently appear verbatim in tool observations for multi-hop or aggregation-heavy questions. These results help contextualize why improvements on Hard subsets can be domain-dependent: even when the model follows the protocol (VF), the retrieval trace may still fail to surface gold strings (EH), mo-

	E-COMMERCE			HEALTHCARE			LEGAL			LITERATURE		
Method	VF	CV	EH	VF	CV	EH	VF	CV	EH	VF	CV	EH
Graph-CoT	93.75	100.00	88.75	61.00	100.00	31.00	60.64	98.98	65.96	72.09	100.00	69.77
Vanilla RL	62.50	100.00	92.50	44.00	99.83	31.00	47.87	94.65	64.89	23.26	100.00	73.64
GRAPHDANCER	92.50	99.46	90.00	53.00	100.00	39.00	70.21	94.06	68.09	73.64	100.00	75.19

Table 5: Difficulty-wise behavioral diagnostics on **Easy** samples.

	E-COMMERCE			HEALTHCARE			LEGAL			LITERATURE		
Method	VF	CV	EH	VF	CV	EH	VF	CV	EH	VF	CV	EH
Graph-CoT	33.33	97.41	53.85	8.67	98.97	8.00	26.47	98.69	20.59	46.67	99.35	83.33
Vanilla RL	0.00	95.07	41.03	2.00	98.62	3.33	2.94	82.72	2.94	10.00	96.62	70.00
GRAPHDANCER	71.79	97.02	53.85	40.00	98.46	5.33	26.47	84.57	8.82	90.00	100.00	93.33

Table 6: Difficulty-wise behavioral diagnostics on **Medium** samples.

tivating evidence-aware objectives in future work.

OOD Subsets. Table 8 reports behavioral diagnostics on **OOD** samples. We observe that EH is 0 across all three OOD domains. This is expected given the definition of OOD in GRbench: many OOD queries are not answerable by directly looking up a gold string in the graph, and instead require using the graph as *supporting context* for open-ended generation (e.g., recommendation-style outputs). Under such settings, an exact string-based evidence-hit metric becomes overly strict and can under-estimate useful graph interaction.

Despite EH $= 0$, VF and CV still provide informative signals about interaction reliability. Compared with Vanilla RL, GRAPHDANCER substantially improves VF on OOD across domains (E-COMMERCE: 67.50 vs. 15.00; LEGAL: 35.00 vs. 5.00; LITERATURE: 33.33 vs. 0.00), indicating better adherence to the intended multi-round tool-use protocol even when the task is open-ended. Schema-level call validity (CV) shows a mixed pattern: GRAPHDANCER reaches perfect CV on E-COMMERCE and LITERATURE OOD, but drops on LEGAL OOD (79.13), suggesting that LEGAL OOD questions induce more challenging or error-prone API usage (e.g., relation/argument mismatches). Overall, these results support the view that curriculum-based RL primarily strengthens *structural tool-use reliability* on OOD queries, while measuring contextual grounding on OOD likely requires evidence metrics beyond exact string hits (e.g., semantic grounding or judge-based rubrics).

B.2 Case Study Logs

Figure 4 presents the original reasoning trajectories for the case study discussed in Section 3.4.

	E-COMMERCE			HEALTHCARE			LEGAL			LITERATURE		
Method	VF	CV	EH	VF	CV	EH	VF	CV	EH	VF	CV	EH
Graph-CoT	50.00	99.46	12.50	60.00	100.00	60.00	13.33	100.00	16.67	11.43	98.34	11.43
Vanilla RL	5.00	96.46	17.50	60.00	98.99	60.00	0.00	93.03	23.33	0.00	96.72	14.29
GRAPHDANCER	67.50	100.00	15.00	65.00	100.00	60.00	53.33	91.21	6.67	58.57	100.00	11.43

Table 7: Difficulty-wise behavioral diagnostics on **Hard** samples.

	E-COMMERCE			LEGAL			LITERATURE		
Method	VF	CV	EH	VF	CV	EH	VF	CV	EH
Graph-CoT	40.00	100.00	0.00	30.00	81.97	45.00	16.67	94.44	0.00
Vanilla RL	15.00	96.69	0.00	5.00	90.98	45.00	0.00	80.65	0.00
GRAPHDANCER	67.50	100.00	0.00	35.00	79.13	45.00	33.33	100.00	0.00

Table 8: Difficulty-wise behavioral diagnostics on **OOD** samples.

Component	**Content**
Task instruction	Solve a question answering task by repeating bundled steps that contain reasoning (<think>...</think>) followed by exactly one graph interaction (<graph>...</graph>). After each <graph> call, the environment returns feedback inside <information>...</information>. You may take as many steps as necessary.
Output protocol	- Intermediate step: <think>...</think><graph>Function[...]</graph> (then environment returns <information>...</information>.) - Final step: <think>...</think><answer>...</answer> (no more graph calls).
Available functions	RetrieveNode[keyword] ## retrieves the related node from the graph according to the query NodeFeature[Node, feature] ## returns detailed attribute information of Node for the given "feature" key NodeDegree[Node, neighbor_type] ## returns the number of "neighbor_type" neighbors of Node NeighborCheck[Node, neighbor_type] ## lists the "neighbor_type" neighbors of Node and returns them
Format rules	1) You MUST conduct reasoning inside <think>...</think> before every graph call and after every <information> you receive. 2) Inside <graph>...</graph>, issue EXACTLY ONE function per step. Do NOT include any other text in <graph>. 3) Do NOT fabricate <information>; it is ONLY produced by the environment immediately after your <graph> step. 4) Keep thoughts concise and ONLY inside <think>. Do NOT put a graph call inside <think>, and do NOT put thoughts inside <graph>. 5) The final output MUST contain ONLY one <answer>...</answer> block with the requested node main features (e.g., names), not node IDs.
In-context examples	Here are some examples: {examples} (END OF EXAMPLES)
Graph schema	Definition of the graph: {graph_definition}
Question	Question: {question}

Table 9: Prompt template used for all LLM rollouts in GRAPHDANCER. Placeholders ({examples}, {graph_definition}, {question}) are instantiated per example. We use the same graph definitions as in Jin et al. (2024)

Full Log: Vanilla RL

Question:
What is the usual price of the items often purchased together with PYLE-PRO PPHP1293 - 800 Watt 12” Two-Way Plastic Molded Loudspeaker?

Trace:
[Turn 0]
<think>The question is asking for the usual price of items often purchased together with a specific item... first need to locate the item node in the graph.</think>
<graph>RetrieveNode[PYLE-PRO PPHP1293 - 800 Watt 12” Two-Way Plastic Molded Loudspeaker]</graph>
<information>The ID of this retrieval target node is B000NJIYHY.</information>

[Turn 1]
<think>The question needs the usual price of items often purchased together... check the bought_together_item neighbors...</think>
<graph>NodeDegree[B000NJIYHY, bought_together_item]</graph>
<information>The bought_together_item neighbor node degree of B000NJIYHY are: 1.</information>

[Turn 2]
<think>The item B000NJIYHY has 1 item often purchased together... We need to check the node feature (price) of the only bought_together_item...</think>
<graph>NodeFeature[B000NJIYHY_bought_together_item_0, price]</graph>
<information>The node or feature name does not exist in the graph. This might be because your given feature name is not correct. Please modify it.</information>

[Turn 3]
<think>It seems there’s an issue with accessing the price information... Since we couldn’t directly access the price, we can infer it by checking... the original item...</think>
<graph>NodeFeature[B000NJIYHY, price]</graph>
<information>The price feature of B000NJIYHY are: 140.43.</information>

[Turn 4]
<think>The price of the original item B000NJIYHY is 140.43... we can reasonably assume the usual price... is around the same value.</think>
<answer>140.43</answer>

Full Log: GraphDancer

Question:
What is the usual price of the items often purchased together with PYLE-PRO PPHP1293 - 800 Watt 12” Two-Way Plastic Molded Loudspeaker?

Trace:
[Turn 0]
<think>First, I need to find the node corresponding to the item PYLE-PRO PPHP1293... After that, I can determine its bought_together_item neighbors...</think>
<graph>RetrieveNode[PYLE-PRO PPHP1293 - 800 Watt 12” Two-Way Plastic Molded Loudspeaker]</graph>
<information>The ID of this retrieval target node is B000NJIYHY.</information>

[Turn 1]
<think>To answer the question, I need to check the bought_together_item neighbors of the found node.</think>
<graph>NeighborCheck[B000NJIYHY, bought_together_item]</graph>
<information>The bought_together_item neighbors of B000NJIYHY are: [’B000E1U4WY’].</information>

[Turn 2]
<think>The item often purchased together... is B000E1U4WY. Now, I need to find the price feature of this item...</think>
<graph>NodeFeature[B000E1U4WY, price]</graph>
<information>The price feature of B000E1U4WY are: 12.95.</information>

[Turn 3]
<think>The usual price of the item often purchased together is 12.95.</think>
<answer>12.95</answer>

Figure 4: Original reasoning trajectories for the case study discussed in Section 3.4. The top box shows the Baseline failure, while the bottom box shows the successful trajectory of GraphDancer.