GRAPHDANCER: Training LLMs to Explore and Reason over Graphs via Two-Stage Curriculum Post-Training

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 graphs requires models to follow schema-defined relations through precise function calls and to aggregate evidence across multiple rounds of interaction. We propose GRAPHDANCER, a two-stage post-training framework that teaches LLMs to reason over graphs by interleaving natural-language reasoning with graph function execution. The first stage teaches the model how to interact with the graph under rule-based rewards, while the second stage further teaches it to prefer more grounded and efficient interaction trajectories. The key novelty of GRAPHDANCER is a graph-aware curriculum that organizes both stages by the structural complexity of information-seeking trajectories, progressively increasing task difficulty during training. 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 larger/stronger backbones, demonstrating robust cross-domain generalization of graph exploration and reasoning skills. Our code can be found at `https://anonymous.4open.science/r/GraphDancer-Anonymous`.

1 Introduction

LLMs often need external knowledge to answer factual questions, especially when the relevant facts are rapidly changing (Vu et al., 2024) or long-tailed (Sun et al., 2024b). Retrieval-augmented generation (RAG; Lewis et al., 2020) addresses this problem when knowledge is stored as text. However, many important knowledge sources are heterogeneous graphs, where information is encoded not only in node attributes but also in typed relations between entities (Han et al., 2024; Sun et al., 2024a; Ma et al., 2025). This graph-structured setting creates two challenges that standard text retrieval does not directly solve.

First, accessing graph knowledge requires schema-aware actions in addition to similarity search. For example, in a biomedical graph, [GENE] *is upregulated in* [ANATOMY] and [GENE] *is downregulated in* [ANATOMY] are distinct relations. Retrieving ANATOMY nodes that are semantically similar to a GENE node is not enough to decide which relation should be followed. Second, answering graph questions often requires multi-hop evidence gathering. A model may need to identify a pivot node, expand to its neighbors, select relevant candidates, and then query their attributes before producing an answer. Thus, graph reasoning is naturally an interactive process in which reasoning and information seeking must be interleaved.

Prior work has explored prompting-based graph agents (Luo et al., 2024; Wang et al., 2024; Jin et al., 2024; Amayuelas et al., 2025; Gao et al., 2025; Kashmira et al., 2025), but prompting alone is limited: the model is asked to mimic graph traversal at inference time without internalizing robust interaction behavior. On the other hand, supervised fine-tuning for tool use (Schick et al., 2023; Asai et al., 2024) may not generalize well to new graph schemas, domains, or question types. We instead view graph interaction as a decision-making problem: the graph is an environment, function calls are actions, and successful reasoning requires learning when to retrieve, expand, stop, and answer.

Contributions. To address the aforementioned challenges, we propose GRAPHDANCER, a two-stage post-training framework that enables LLMs to interact with and reason over graphs in an adaptive and interleaved fashion. The first stage uses proximal policy optimization (PPO; Schulman et al., 2017) with executable rule-based rewards to teach the model *how to interact with the graph*:

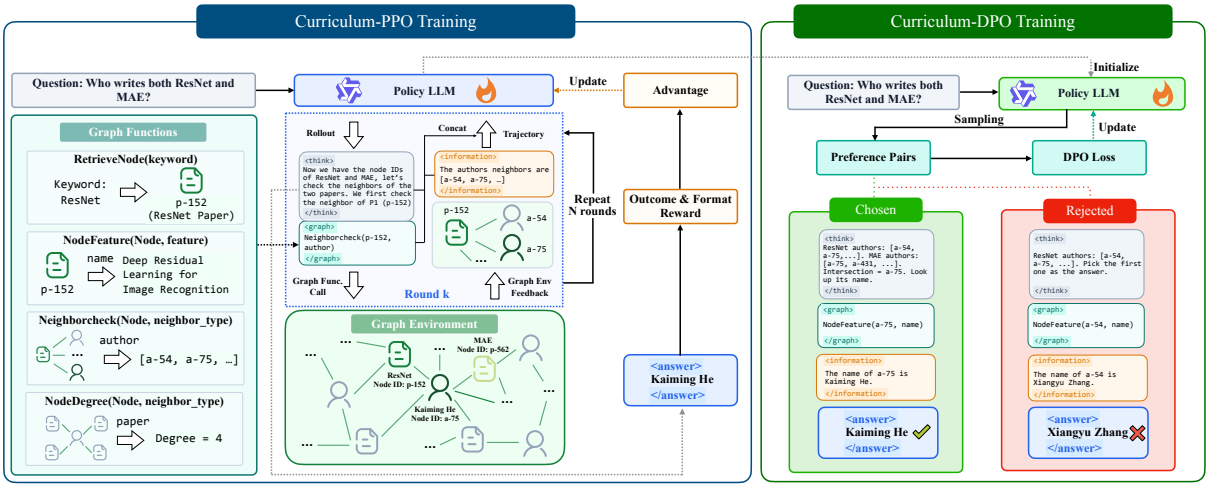

Figure 1: Overview of GRAPHDANCER. **Left**: Stage 1 (Curriculum-PPO, Section 2.2) interleaves natural-language reasoning with executable graph function calls; rule-based outcome and format rewards drive PPO updates. **Right**: Stage 2 (Curriculum-DPO, Section 2.3) samples $M = 8$ trajectories from the PPO checkpoint, ranks them by six trajectory-level keys with fixed tie-break priority (Section 2.3), and trains on the chosen/rejected pair. Both stages are organized by a graph-aware curriculum based on the S-round / E-round structure of information-seeking trajectories (Section 2.4).

issue valid graph calls, condition on returned observations, chain evidence across rounds, and produce an accurate answer. The second stage uses direct preference optimization (DPO; Rafailov et al., 2023) on self-generated preference pairs to teach the model *how to interact with the graph better*: among plausible trajectories, prefer those that reach the right evidence, avoid invalid calls or loops, and use fewer interaction rounds. This design separates absolute online feedback from comparative offline feedback, making PPO and DPO complementary rather than redundant.

The primary novelty of GRAPHDANCER is the **graph-aware curriculum**, where the difficulty of the task is progressively increased during training. Rather than relying on external difficulty annotations (Parashar et al., 2026), we define difficulty through the structure of information search. At the beginning of RL training, the vast majority of questions require only one-hop information seeking 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?*"). The same easy-to-hard curriculum is used during PPO and DPO: it first stabilizes online exploration under sparse rewards, then shapes preference learning toward increasingly complex graph-use behaviors.

We evaluate GRAPHDANCER on a multi-domain benchmark (Jin et al., 2024). Following a strict generalization setting, we train only on the ACADEMIC domain and evaluate on unseen domains including E-COMMERCE, LITERATURE, HEALTHCARE, and LEGAL. We also test on out-of-distribution (OOD) question types that cannot be answered by simply looking up the graph, but 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?*"). Despite using a 3B backbone, GRAPHDANCER significantly outperforms graph-agent baselines built on larger/stronger backbones, including `Qwen3-14B`, `Mixtral-8x7B-Instruct` and `GPT-4o-mini`. Further analysis shows that GRAPHDANCER substantially improves the LLM's behavioral reliability, especially by increasing valid multi-round interactions and reducing unstable tool-use failures. These results suggest that the model learns transferable graph exploration skills rather than memorizing domain-specific patterns.

The contributions of our work are as follows: (1) We propose GRAPHDANCER, a two-stage post-training framework that teaches LLMs to interact with graphs using both absolute online feedback and comparative offline feedback. (2) We introduce a graph-aware curriculum based on the structural complexity of information-seeking trajectories, and use it to organize both PPO and DPO stages. (3) We demonstrate strong cross-domain and OOD generalization with a 3B backbone, outperforming graph-agent baselines built on substantially larger or stronger models.

2 The GRAPHDANCER Framework

GRAPHDANCER is a two-stage post-training framework for graph reasoning with executable function calls. We first define the graph interaction problem and tool interface (Section 2.1), then describe how PPO learns basic graph-interaction behavior (Section 2.2) and how DPO refines it using self-generated preferences (Section 2.3). Finally, we introduce the graph-aware curriculum that organizes both stages (Section 2.4). Figure 1 provides an overview.

2.1 Problem Setup

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a text-attributed heterogeneous graph (Jin et al., 2023), where each node has textual fields (e.g., `name`, `title`, `abstract`) and typed relations to other nodes. Given a question x, the model needs to produce a natural-language answer by interacting with $\mathcal{G}$ through graph functions.

We formulate this process as an episodic Markov decision process (Puterman, 2014). At round t, the state s_t contains the question, the previous reasoning/action history, and all graph observations returned so far. The model selects an action a_t, the environment executes it and returns an observation o_t, and the episode ends when the model outputs an answer or reaches a maximum of $T_{\max}$ rounds.

We use a simple interleaved interaction format, following multi-turn search calling (Jin et al., 2025). To be specific, the model emits reasoning in **`<think> </think>`** blocks, graph calls in **`<graph> </graph>`** blocks, and the final response in an **`<answer> </answer>`** block. The graph environment executes each action block and inserts the returned evidence in **`<information> </information>`** blocks.

Following Jin et al. (2024), the action space consists of a small set of deterministic graph functions:

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

`NodeFeature(NodeID, FeatureName)`: returns the requested textual field of 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.

This function-call interface is better suited to text-attributed heterogeneous graphs than vanilla text retrieval, because the model must follow schema-defined relations rather than retrieve arbitrary passages by semantic similarity as in classic RAG (Lewis et al., 2020).

2.2 Learning to Interact with Graphs via PPO

The first stage uses PPO (Schulman et al., 2017) with rule-based rewards to make graph interaction executable and useful. Intuitively, this stage teaches the policy to carry out the basic loop of graph reasoning: identify relevant nodes, expand along typed relations, read node attributes, and turn the acquired evidence into an answer.

Learning Signal. Let y denote the concatenation of all *agent-generated* tokens in an episode, including reasoning, graph calls, and the final answer. Let $\tau = \mathrm{Env}(x, \mathcal{G}, y)$ be the full transcript obtained after the graph executor inserts observation tokens. Let $\hat{y}$ be the content extracted from the final **`<answer> </answer>`** block and $y^\star$ be the ground-truth answer. Let x denote the input question (Section 2.1) and $\mathcal{D}$ the training distribution over $(x, \mathcal{G}, y^\star)$ triples. We use a rule-based reward that combines exact-match correctness, format validity, and answer presence:

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

Here, EM (Exact Match) checks if $\hat{y}$ and $y^\star$ are the same, VF (Valid Format) checks whether the trace follows the interaction protocol, and AP (Answer Presence) checks whether the answer is non-empty. The first term is the primary correctness signal, the second mildly penalizes structurally invalid traces even when the answer is correct, and the third gives a small reward to well-formed non-empty attempts to discourage degenerate outputs.

Optimization Objective. We optimize the policy with a KL-regularized RL objective:

$$\begin{aligned} \mathcal{J}_{\mathrm{PPO}}(\theta) = \; & \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta}\big[r(x, \mathcal{G}, \tau)\big] \\ & - \gamma\, \mathbb{E}_{x\sim\mathcal{D}}\big[D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})\big], \end{aligned} \tag{2}$$

where π_{ref} is a fixed reference model. Note that we use PPO rather than group-relative variants such as GRPO (Shao et al., 2024) or DAPO (Yu et al., 2025) because our pipeline separates two learning signals cleanly: PPO optimizes *each* on-policy trajectory against an *absolute* executable reward, while DPO later performs the comparative optimization over sampled trajectories. This makes the two stages symmetric but non-redundant: PPO

learns *whether* an interaction succeeds under the graph environment, and DPO learns *which* successful or partially successful interaction is preferable. In fact, we also experimented with replacing PPO by GRPO or DAPO in our framework, but did not observe performance gains. This further supports the rationale behind our two-stage design.

Interaction-Aware Masking. Following Jin et al. (2025), only agent-generated tokens are sampled from π_θ. Observation tokens inside **<information> </information>** blocks are deterministic outputs of the graph executor, so they are masked out when computing policy gradients. This prevents the model from being trained to imitate environment feedback and focuses optimization on reasoning, tool use, and answer generation.

2.3 Learning to Interact with Graphs Better via DPO

After PPO, the model can usually navigate the graph, but trajectories may still differ in evidence quality and tool efficiency. The second stage therefore applies DPO (Rafailov et al., 2023) to refine the PPO checkpoint using self-generated preference pairs. Where PPO teaches the model to *perform* graph interaction, DPO teaches it to *prefer better* graph interaction.

Learning Signal. For each training question x, we sample M trajectories from the PPO checkpoint $\pi_\theta^{\mathrm{PPO}}$ using the same interaction budget as PPO. We rank the trajectories by six keys:

$$(\mathrm{EM}, \mathrm{EH}, \mathrm{VF}, -L, -I, -N).$$

Here, EM, EH, and VF are Boolean indicators for Exact Match, Evidence Hit (the gold answer appears in some returned `<information>` block), and Valid Format, respectively, with 1 preferred over 0; L, I, and N count loop-limit halts, invalid tool calls, and interaction rounds, respectively, where smaller values are preferred. Trajectories are compared by fixed tie-break priority: if two trajectories tie on one key, the next key is used to break the tie. Thus, the ranking first favors correctness and evidence grounding, then valid and efficient interaction. For each question, the best trajectory becomes the chosen response y_w for DPO and the worst trajectory becomes the rejected response y_l. Questions with uniformly perfect or uniformly failed candidate pools are discarded.

Optimization Objective. We initialize both the trainable policy and the frozen reference from $\pi_\theta^{\mathrm{PPO}}$ and optimize the standard DPO objective:

$$\mathcal{J}_{\mathrm{DPO}}(\theta) = \mathbb{E}\Bigg[\log \sigma\Bigg(\beta \log \frac{\pi_\theta(y_w)}{\pi_{\mathrm{ref}}(y_w)} - \beta \log \frac{\pi_\theta(y_l)}{\pi_{\mathrm{ref}}(y_l)}\Bigg)\Bigg]. \tag{3}$$

This objective is naturally paired with PPO in our setting: both compare the policy to a frozen reference through log-probability ratios, but PPO uses online scalar rewards while DPO uses offline pairwise preferences.

Interaction-Aware Masking. As in PPO, the DPO loss is applied only to agent-generated tokens. Environment-inserted tokens inside **<information> </information>** blocks are present in the transcript as context, but they do not contribute to the preference objective.

2.4 Graph-Aware Curriculum

The primary novelty of GRAPHDANCER is the curriculum that organizes both PPO and DPO. Generic curricula often rely on external annotations or surrogate signals (Qu et al., 2018; Parashar et al., 2026). In contrast, graph interaction exposes a structural notion of difficulty: a question is hard when answering it requires repeated expansion of the explored subgraph and long-horizon decisions about which evidence to keep. We exploit this structure to schedule training from simple lookup behavior toward expansion-heavy reasoning.

Information-Seeking Rounds. We decompose each trajectory into rounds of graph function calls. Calls in the same round are independent: each call depends only on information available before the round begins, so their outputs can be consumed together before the next reasoning step. Let $\mathcal{U}_t$ be the set of node IDs returned by all calls in round t. We distinguish two round types:

Singleton lookup round (S-round): $|\mathcal{U}_t| = 1$. The model identifies exactly one node, such as resolving an entity mention (e.g., mapping "*ResNet*" to its corresponding paper node) or retrieving a unique typed neighbor (e.g., obtaining the sole "*published in*" venue of a paper).

Neighborhood expansion round (E-round): $|\mathcal{U}_t| > 1$. The model brings in multiple nodes (e.g., obtaining all authors of a paper) and will later select, aggregate, or chain evidence from them.

Structural Difficulty. We assign each training question to one of three levels. *Easy* questions require one information-seeking round. *Medium*

questions require multiple rounds but at most one E-round. *Hard* questions require at least two E-rounds, reflecting repeated neighborhood expansion and downstream aggregation or path reasoning. This taxonomy is graph-specific: it measures the interaction burden imposed on the policy rather than surface-level question length or answer type. *OOD* questions that fall outside this taxonomy (i.e., those that cannot be answered by simply looking up the graph, though the graph may still provide useful context) are excluded from training and used only for evaluation, as described in Section 3.1.

Easy-to-Hard Sampling. Let $k \in \{1, 2, 3\}$ index the Easy, Medium, and Hard levels. We propose a scheduler $p(t, k)$ built upon the Gaussian scheduler $g(t, k)$, which gradually shifts focus from easy to hard tasks (Parashar et al., 2026). However, a pure easy-to-hard schedule can be brittle for graph interaction: hard questions contain branching expansions and sparse rewards, while removing them for too long delays learning multi-hop behavior. We therefore use a time-varying biased mixture:

$$p(t, k) \;=\; (1 - \eta(t))\, g(t, k) \;+\; \eta(t)\, q(k), \quad (4)$$

where $q(k)$ is a fixed level prior and $\eta(t)$ controls the strength of the bias. In GRAPHDANCER, $\eta(t)$ follows a linear schedule:

$$\eta(t) = \eta_{\text{start}} + \frac{t}{T_{\text{stage}} - 1}(\eta_{\text{end}} - \eta_{\text{start}}), \quad (5)$$

where T_{stage} is the number of steps in the current PPO or DPO stage. At each PPO or DPO step, we sample a difficulty level according to $p(t, k)$ $(k = 1, 2, 3)$ and then sample an instance uniformly from that level. In this way, the same structural curriculum first stabilizes online exploration under sparse rewards, then shapes offline preference learning from simpler graph-use preferences toward harder expansion-rich preferences.

The full training procedure of GRAPHDANCER is summarized in Algorithm 1.

Algorithm 1 GRAPHDANCER: two-stage training

Input: dataset $\mathcal{D}$ of $(x, \mathcal{G}, y^\star)$; graph executor Env; reference policy π_{ref}
Initialize π_θ from an LLM backbone
Preprocess: compute round decomposition; assign each instance a difficulty level (Section 2.4)
`// Stage 1: Curriculum-PPO`
for $t = 0$ **to** $T_{\text{PPO}} - 1$ **do**
  Compute difficulty distribution $p(t, k)$ $(k = 1, 2, 3)$ via Eqs. 4-5
  Sample level $k \sim p(t, \cdot)$; sample $(x, \mathcal{G}, y^\star)$ uniformly from level k
  Roll out $y \sim \pi_\theta(\cdot \mid x; \mathcal{G})$ with at most $T_{\max}$ rounds; form transcript τ
  Compute reward $r(x, \mathcal{G}, \tau)$ via Eq. 1; update θ via Eq. 2 with interaction-aware masking
end for
`// Stage 2: Curriculum-DPO`
$\pi_{\text{ref}} \leftarrow \pi_\theta$
Sample M trajectories per training question from π_θ
Construct preference pairs (y_w, y_l)
for $t = 0$ **to** $T_{\text{DPO}} - 1$ **do**
  Compute difficulty distribution $p(t, k)$ $(k = 1, 2, 3)$ via Eqs. 4-5
  Sample level $k \sim p(t, \cdot)$; draw a preference minibatch from pairs at level k
  Update θ via Eq. 3 with interaction-aware masking
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. See Table 1 for dataset statistics.

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 compare GRAPHDANCER with three groups of baselines. (1) Single-round RAG methods: **TextRAG** (Gao et al., 2023) and **GraphRAG** (Ye et al., 2024). Their results with `GPT-3.5-turbo` (Ouyang et al., 2022) are reported by Jin et al. (2024), and we directly adopt those numbers. (2) Prompting-based graph agents: **Graph-CoT** (Jin et al., 2024) and **Graph-**

Method	Model	E-COMMERCE		LITERATURE		HEALTHCARE		LEGAL		Average		Gap
		R-L	GS	R-L	GS	R-L	GS	R-L	GS	R-L	GS	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**	↑ 30.5
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**	↑ 26.9
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**	↑ 9.7
	`GPT-4o-mini`	37.06	39.50	36.04	46.25	39.88	41.48	35.47	40.00	37.11**	41.81**	↑ 6.0
	`Qwen2.5-3B-Instruct`	43.58	41.00	43.60	47.50	27.47	26.67	30.68	35.56	36.33**	37.68**	↑ 8.5
	`Qwen3-14B`	39.91	42.50	48.03	53.33	38.82	36.67	32.66	35.56	39.85**	42.01**	↑ 4.5
Graph-Counselor (Gao et al., 2025)	`GPT-3.5-turbo`	41.18	40.00	44.27	50.42	30.87	41.48	16.09	23.33	33.10**	38.81**	↑ 11.4
	`GPT-4o-mini`	39.42	38.50	43.64	48.33	**46.52**	**43.33**	36.53	38.33	41.53**	42.12**	↑ 2.9
	`Qwen2.5-3B-Instruct`	42.19	41.00	41.93	48.33	30.29	31.11	15.06	22.78	32.37**	35.81**	↑ 12.1
	`Qwen3-14B`	38.29	41.00	**49.75**	54.58	38.53	40.37	21.44	26.67	37.00**	40.66**	↑ 7.5
Graph-MCTS (Liu, 2025b)	`Mixtral-8x7B-Instruct`	33.71	37.22	37.52	45.83	26.79	32.84	26.23	27.78	31.06**	35.92**	↑ 12.0
Graph-o1 (Liu, 2025a)	`Mixtral-8x7B-Instruct`	34.61	38.76	38.42	45.83	29.54	33.70	27.36	32.20	32.48**	37.62**	↑ 10.4
GRAPHDANCER	`Qwen2.5-3B-Instruct`	**49.92**	**49.00**	49.25	**55.83**	32.89	33.70	**45.82**	**47.22**	**44.47**	**46.44**	–

Table 2: Main results on the four unseen GRBench domains. We highlight the Average performance (purple columns) and the performance Gap (Δ) compared to our method (green column). Bold values denote the best score in each column. ** $p < 0.01$ vs. GRAPHDANCER.

Counselor (Gao et al., 2025). Since both methods provide public code, we evaluate them with four representative backbones: `GPT-3.5-turbo` (Ouyang et al., 2022), `GPT-4o-mini` (Hurst et al., 2024), `Qwen2.5-3B-Instruct` (Yang et al., 2024), and `Qwen3-14B` (Yang et al., 2025). (3) RL-based graph agents: **Graph-MCTS** and **Graph-o1**. Their results with `Mixtral-8x7B-Instruct` (Jiang et al., 2024) are reported by Liu (2025a), and we directly adopt the reported performance.

GRAPHDANCER uses `Qwen2.5-3B-Instruct` as its backbone. For more details on the baselines and GRAPHDANCER, please refer to Appendix A.1 and Appendix A.2, respectively.

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. We run GRAPHDANCER five times, compute its average performance, and conduct a two-tailed Z-test to compare it with each baseline in terms of average Rouge-L and average GPT4Score. The significance level is also marked in Table 2.

GRAPHDANCER enables a 3B backbone to significantly outperform baselines with larger backbones. Despite its smaller size, GRAPHDANCER outperforms all baselines on average, including Graph-CoT and Graph-Counselor on `Qwen3-14B` and `GPT-4o-mini`, as well as Graph-o1 on `Mixtral-8x7B-Instruct`. In terms of average Rouge-L and average GPT4Score, the advantage of GRAPHDANCER is always statistically significant. This suggests that directly optimizing the multi-round `reasoning`→`action`→`observation` loop yields gains beyond in-context prompting and multi-agent prompting, particularly when tool use and long-horizon credit assignment are required.

Domain-level wins are positive across the board, but not uniform. On E-COMMERCE, LITERATURE, and LEGAL, GRAPHDANCER achieves the highest GPT4Score. The one domain where larger-backbone baselines still outperform GRAPHDANCER is HEALTHCARE: GRAPHDANCER scores 32.89 Rouge-L, behind both Graph-CoT (39.88) and Graph-Counselor (46.52) on `GPT-4o-mini`; the gap narrows on GPT4Score but does not close. A difficulty-wise breakdown (Appendix B.1) localizes the gains: GRAPHDANCER achieves the largest absolute gains on LEGAL across all difficulty levels and on LITERATURE Medium, while HEALTHCARE Hard remains the hardest cell for all methods with near zero Rouge-L.

3.3 Ablation Analysis

To isolate the contribution of each training-stage component, we evaluate three ablations of GRAPHDANCER: **NoPPO** skips the PPO stage so that DPO trains directly from `Qwen2.5-3B-Instruct`; **NoDPO** removes the DPO stage and directly outputs the Curriculum-PPO checkpoint; **NoCurriculum** replaces the curriculum with uniform sampling in both training stages. Table 3 reports the per-domain Rouge-L results.

PPO is the load-bearing component: removing it drops the average Rouge-L by 7.90 and collapses HEALTHCARE by 9.46 (32.89 → 23.43). DPO

Setting	E-com.	Literature	Healthcare	Legal	Average
GRAPHDANCER	**49.92**	**49.25**	**32.89**	**45.82**	**44.47**
NoPPO	43.19	43.23	23.43	36.42	36.57
NoDPO	48.87	43.82	31.55	38.25	40.62
NoCurriculum	48.15	47.60	31.44	38.50	41.42

Table 3: Ablation analysis (Rouge-L) on the four unseen GRBench domains.

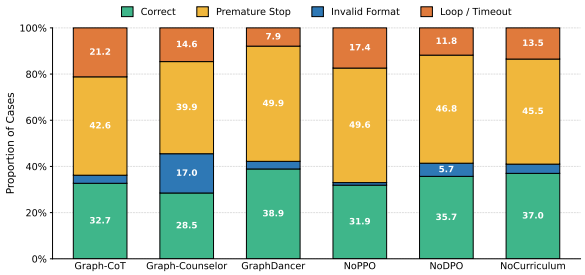

Figure 2: Outcome breakdown pooled over 890 questions across the four unseen GRBench domains. Each bar decomposes episodes into *Correct* and three failure modes: *Invalid Format*, *Loop / Timeout*, and *Premature Stop*. All six methods use `Qwen2.5-3B-Instruct` as the LLM backbone.

contributes 3.85 Rouge-L on average, concentrated on LEGAL (+7.57, 38.25 → 45.82) and LITERATURE (+5.43, 43.82 → 49.25). The graph-aware curriculum contributes 3.05 Rouge-L on average, where the gains are more pronounced on LEGAL (+7.32, 38.50 → 45.82).

3.4 Error Analysis of Graph Interaction

To further analyze why GRAPHDANCER performs better, we decompose each episode into four outcomes: *Correct*, *Invalid Format* (invalid tool calls), *Loop / Timeout* (exceeding the interaction budget), and *Premature Stop* (terminating with an incorrect answer). Figure 2 reports the distribution for Graph-CoT and Graph-Counselor (both with the `Qwen2.5-3B-Instruct` backbone, the same as GRAPHDANCER), full GRAPHDANCER, and three ablation versions.

PPO drives most of the *Loop / Timeout* reduction: NoPPO retains 17.4% Loop / Timeout against Graph-CoT's 21.2% and GRAPHDANCER's 7.9%, and its *Correct* rate (31.9%) falls below Graph-CoT. DPO instead targets schema adherence: NoDPO nearly doubles *Invalid Format* (3.3% → 5.7%) while leaving *Loop / Timeout* close to the full system. The curriculum sits between the two: NoCurriculum compresses Loop / Timeout only to 13.5% and trails the full system by 1.9 points on *Correct*, indicating that the easy-to-hard schedule mainly helps the agent learn when to terminate. In other words, PPO produces the navigation behavior that

Method	EH↑	P(EM|EH)↑	P(EM| ¬EH)	EM↑
Graph-CoT	40.5	69.8	7.4	32.7
Graph-Counselor	39.0	73.2	0.0	28.5
GRAPHDANCER	**43.8**	**77.4**	8.8	**38.9**

Table 4: Evidence-grounded accuracy (%) pooled over 890 questions across the four unseen GRBench domains. EH is the fraction of episodes that surface answer-supporting evidence in tool observations; $\mathrm{P}(\mathrm{EM} \mid \mathrm{EH})$ and $\mathrm{P}(\mathrm{EM} \mid \neg\mathrm{EH})$ condition exact-match accuracy on the presence or absence of retrieved evidence. All methods use `Qwen2.5-3B-Instruct` as the LLM backbone.

suppresses Loop / Timeout, DPO produces the format compliance that suppresses Invalid Format, and the graph-aware curriculum amplifies both. Residual errors in the full system concentrate on *Premature Stop* (49.9%), where the agent fails decisively rather than wandering. Per-domain behavioral diagnostics (VF, VC, EH) are reported in Appendix B.2.

3.5 Evidence-Grounded Accuracy

Rouge-L and GPT4Score reward accurate final answers but cannot tell whether the model actually retrieved supporting evidence or guessed. We complement them with an evidence-grounded view in Table 4. Each episode is decomposed by exact-match correctness (EM) and evidence hit (EH; whether the model's tool observations contain the gold answer). $\mathrm{P}(\mathrm{EM} \mid \mathrm{EH})$ then measures evidence-to-answer conversion, and $\mathrm{P}(\mathrm{EM} \mid \neg\mathrm{EH})$ characterizes the probability that the model guesses correctly without retrieving good evidence.

Consistent with intuition, evidence hit is the dominant driver of correctness: $\mathrm{P}(\mathrm{EM} \mid \mathrm{EH})$ is roughly an order of magnitude above $\mathrm{P}(\mathrm{EM} \mid \neg\mathrm{EH})$ in every row, and Graph-Counselor is the limiting case at $\mathrm{P}(\mathrm{EM} \mid \neg\mathrm{EH}) = 0$ (it never returns a correct answer without first retrieving the gold entity). Among compared methods, GRAPHDANCER achieves the highest EH (43.8%) and the highest evidence-conversion rate (77.4%), indicating that DPO post-training improves both retrieval coverage and the conversion of retrieved evidence into a correct answer. Graph-Counselor's clean grounding behavior is offset by its lower EH, capping its EM at 28.5%.

3.6 Case Study

We analyze a Medium-difficulty instance from the unseen E-COMMERCE domain to inspect whether GRAPHDANCER learns transferable graph exploration behavior. The question asks for the price of an item *bought together* with a specific loud-

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"?

Graph-CoT (Qwen2.5-3B-Instruct):
<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)**

GRAPHDANCER
<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)**

Figure 3: Case study. Graph-CoT finds the anchor item but invents a neighbor ID and returns the anchor price, while GRAPHDANCER explicitly retrieves the neighbor through NeighborCheck before querying its price.

speaker, requiring the two-hop path *Anchor Item* → *Neighbor Item* → *Price*.

Figure 3 compares two trajectories. Both models find the anchor node (B000NJIYHY), but Graph-CoT (with the Qwen2.5-3B-Instruct backbone, the same as GRAPHDANCER) fails at the expansion step: it checks that a *bought together* neighbor exists with NodeDegree, then invents a neighbor ID instead of retrieving it, causing an invalid function call. It subsequently queries the anchor item's price and returns the wrong answer.

GRAPHDANCER instead follows the required graph protocol: it retrieves the neighbor ID with NeighborCheck, then queries that node's price with NodeFeature, yielding the correct answer ($12.95). This illustrates a transferable exploration pattern: first obtain the relevant node ID, then query its attributes.

4 Related Work

LLM Agents on Graphs. Early work integrates LLMs with graphs through feature extraction for graph neural networks (Santoro et al., 2017; Zhao et al., 2023; Jin et al., 2023; Duan et al., 2023) or single-pass retrieval over textualized graph context (Gao et al., 2023; Ye et al., 2024). These methods are effective for exposing graph-derived information to LLMs, but they do not explicitly train models to conduct multi-round graph exploration. Recent prompting-based graph agents address this by interleaving reasoning with graph actions (Luo et al., 2024; Wang et al., 2024): Graph-CoT (Jin et al., 2024) traverses external graphs through iterative function calls, Graph-Counselor (Gao et al., 2025) uses a multi-agent prompting framework, and GraphRunner (Kashmira et al., 2025) separates planning, verification, and execution to reduce traversal errors. Another line introduces RL into graph agents, including Graph-MCTS (Liu, 2025b), Graph-o1 (Liu, 2025a), GraphScout (Ying et al., 2026), Graph-R1 (Luo et al., 2025), and Structure-R1 (Wu et al., 2025). These methods improve inference-time search or post-training, but typically do not couple online executable rewards and offline trajectory preferences under the same structural curriculum. GRAPHDANCER differs by using a unified post-training pipeline in which PPO, DPO, and a graph-aware curriculum jointly internalize transferable graph exploration behavior.

LLM Post-Training and Curriculum Learning. Preference optimization (Schulman et al., 2017; Rafailov et al., 2023; Shao et al., 2024; Yu et al., 2025) has become central to LLM post-training (Ouyang et al., 2022; Guo et al., 2025), while curriculum learning (Bengio et al., 2009) improves long-horizon training through easy-to-hard scheduling (Parashar et al., 2026), adaptive rollout control (Shen et al., 2025), and multi-objective balancing (Hammoud et al., 2025). Most existing applications focus on math, code, or unstructured web search (Jin et al., 2025). We instead combine PPO, DPO, and curriculum learning in structured graph environments mediated by typed function calls.

5 Conclusion

We present GRAPHDANCER, a two-stage post-training framework for graph reasoning with LLMs. PPO teaches executable graph interaction under rule-based rewards, DPO refines the policy with self-generated preferences, and a graph-aware curriculum organizes both stages by information-seeking complexity. Training only on ACADEMIC, GRAPHDANCER generalizes to four unseen domains and OOD question types with a 3B backbone. The results suggest that transferable graph exploration requires training signals matched to graph interaction structure, beyond stronger prompting or larger backbones.

Limitations

Our framework involves several design choices, including the curriculum shape, biased-mixture schedule, PPO hyperparameters, and DPO preference-pair construction. Due to the high computational cost of two-stage post-training, 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, biased-mixture sampling, and preference-pair quality can substantially affect training stability and final performance, but a systematic ablation with complete learning curves is left to future work.

Additionally, we adopt the deterministic Graph-CoT tool API and the GRBench benchmark, both of which provide clean, well-typed function signatures and exact retrieval. Real knowledge graphs do not look like this: schemas change between snapshots, entity mentions are ambiguous, coverage is uneven, and tool calls go through a learned retriever rather than a deterministic lookup. We leave the transfer of our policy to such settings as an open question for future work. Our evaluation also focuses on factual question answering over the four GRBench unseen domains, which exercise multi-hop traversal and aggregation but do not stress open-ended generation, long-horizon planning beyond the $T_{\max}$=10 budget, or interactive multi-turn dialogue.

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 PPO+DPO post-training pipeline reduces some behavioral failures, such as invalid calls, it does not guarantee correctness. Agents may still retrieve incomplete evidence, prefer a flawed trajectory, or terminate prematurely. We therefore view it as decision support rather than a substitute for expert judgment, especially in high-stakes domains. We also use an existing public benchmark (GRBench) and do not collect any new user data. We inspected the released graphs for sensitive personal information and did not find private identifiers (e.g., emails, phone numbers, addresses); person names that appear (e.g., academic authors) are part of publicly available entity records. The benchmark is not sourced from user-generated content, and we did not observe offensive/toxic text; we do not release any additional personal information beyond the benchmark.

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.

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, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

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*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, and 1 others. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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.

Lihui Liu. 2025a. Graph-o1: Monte carlo tree search with reinforcement learning for text-attributed graph reasoning. *arXiv preprint arXiv:2512.17912*.

Lihui Liu. 2025b. Monte carlo tree search for graph reasoning in large language model agents. In *CIKM'25*, pages 4966–4970.

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*.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *ICLR'24*.

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*.

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 1 others. 2026. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. In *ICLR'26*.

Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Meng Qu, Jian Tang, and Jiawei Han. 2018. Curriculum learning for heterogeneous star network embedding via deep reinforcement learning. In *WSDM'18*, pages 468–476.

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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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*.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Peter Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and 1 others. 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.

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.

Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *AAAI'24*, pages 19206–19214.

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.

Yuchen Ying, Weiqi Jiang, Tongya Zheng, Yu Wang, Shunyu Liu, Kaixuan Chen, and Mingli Song. 2026. Graphscout: Empowering large language models with intrinsic exploration ability for agentic graph reasoning. *arXiv preprint arXiv:2603.01410*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. In *NeurIPS'25*.

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*.

A More Experimental Details

A.1 Baselines Details

- **TextRAG** (Gao et al., 2023) treats graph-associated textual fields as an unstructured corpus. For each question, it retrieves relevant text units and appends them to the prompt before a single answer-generation step. It does not expose typed graph relations or executable graph functions to the LLM.

- **GraphRAG** (Ye et al., 2024) augments retrieved text with graph context by collecting and linearizing the local subgraph associated with the retrieved entry. The resulting context is still consumed in a single forward pass, so the model cannot adaptively issue follow-up graph actions based on intermediate observations.

- **Graph-CoT** (Jin et al., 2024) performs iterative graph reasoning through a reasoning-interaction-execution loop. At each step, the LLM reasons about what information is needed, emits graph interactions such as node retrieval or neighbor checking, and then conditions on the executor's returned observations before deciding the next step. We keep the original system instructions and few-shot examples from Jin et al. (2024).

- **Graph-Counselor** (Gao et al., 2025) decomposes graph reasoning across multiple prompted agents. It coordinates planning, thought, and execution agents to gather textual, structural, and degree information, and it performs backward checking to improve semantic consistency. We use the released implementation with its default settings, including `max_steps` = 10, `max_reflect` = 2, and sampling temperature 0.7.

- **Graph-MCTS** (Liu, 2025b) uses Monte Carlo Tree Search to guide graph exploration. The search procedure evaluates alternative graph traversal actions through selection, expansion, simulation, and backpropagation, helping the LLM choose more promising graph paths than a purely greedy prompting strategy.

- **Graph-o1** (Liu, 2025a) extends MCTS-style graph exploration with an agent-environment formulation and reward-driven optimization over text-attributed graphs. It selectively explores and retrieves relevant subgraph elements through multi-turn interaction, rather than encoding a large retrieved subgraph as static context.

A.2 Hyperparameters

Graph Executor and Interaction Budget. All models interact with the same graph executor and function API. We set `max_turns` = 10. Within each turn, the model may emit a batch of graph calls inside a `<graph>` block, which are executed deterministically. Table 8 shows the full prompt template used for all rollouts.

PPO Training. We use the same decoding configuration for inference-time generation and PPO rollout generation: temperature 0.7, top-p 0.8, and top-k 20. We perform PPO for 200 steps with a global rollout buffer and minibatch size 128 (micro-batch size 8 per GPU). The KL penalty coefficient is fixed to $\gamma = 0.001$ relative to the reference policy. We use policy clip ratio 0.2 and value clip range 0.5. The actor and critic learning rates are 10^{-6} and 10^{-5}, respectively. For the reward in Eq. 1, we set $\lambda_{\text{struct}} = 0.2$ and $\lambda_{\text{final}} = 0.1$.

DPO Training. We sample $M = 8$ trajectories per ACADEMIC training question from the PPO checkpoint, using temperature 1.0, top-p 0.95, and the same $T_{\max} = 10$ interaction budget. We perform DPO with $\beta = 0.1$, learning rate 2×10^{-7}, linear warmup over the first 5% of steps, and linear decay. Training runs for 100 optimization steps with global batch size 64 (8 GPUs $\times$ per-device batch size 2 $\times$ gradient accumulation 4). We use AdamW in `bf16`, gradient checkpointing, and FSDP full-shard with parameter and optimizer offload.

Curriculum Configuration. For both PPO and DPO, we use the Gaussian curriculum scheduler with shape parameter $\kappa = 3$ and $\sigma = 0.75$. The time-varying mixture in Eq. 4 uses $\eta_{\text{start}} = 0.2$ and $\eta_{\text{end}} = 0.8$. The fixed level-bias prior is $q = [0.5, 0.5, 0]$ over {Easy, Medium, Hard}.

B More Analysis

B.1 Difficulty-wise Breakdown of Cross-Domain Gains

Table 5 breaks down Rouge-L by structural difficulty (Section 2.4), contrasting GRAPHDANCER against Graph-CoT and Graph-Counselor on the four unseen GRBench domains.

GRAPHDANCER wins Easy and Medium across all four domains. The largest absolute gains over Graph-CoT concentrate on LEGAL, where Easy improves +7.92, Medium +21.87, Hard +17.27, and OOD +24.38. LITERATURE Medium also moves substantially (+3.12 over Graph-CoT). The two

Domain	Method	Easy	Medium	Hard	OOD
E-COMMERCE	Graph-CoT	82.34	38.47	11.64	3.49
	Graph-Counselor	77.73	40.76	9.11	**4.64**
	GRAPHDANCER	**87.86**	**54.31**	**16.33**	2.85
LITERATURE	Graph-CoT	63.52	55.14	6.59	1.67
	Graph-Counselor	61.97	52.38	6.48	3.45
	GRAPHDANCER	**73.52**	**58.26**	**6.68**	**6.50**
HEALTHCARE	Graph-CoT	63.23	7.09	0.00	–
	Graph-Counselor	63.12	11.45	**5.00**	–
	GRAPHDANCER	**66.51**	**15.17**	0.00	–
LEGAL	Graph-CoT	52.93	9.04	4.39	4.29
	Graph-Counselor	26.26	0.00	0.00	3.92
	GRAPHDANCER	**60.85**	**30.91**	**21.66**	**28.67**

Table 5: Difficulty-wise Rouge-L (%) on the four unseen GRBench domains. "–" marks splits with no test samples (see Table 1); the only such split is HEALTHCARE OOD. All three methods use `Qwen2.5-3B-Instruct` as the LLM backbone.

cells where Graph-Counselor outperforms GRAPHDANCER are both at the corners of the benchmark: HEALTHCARE Hard (5.00 vs. 0.00), a 20-question bucket of extreme-aggregation queries, and E-COMMERCE OOD (4.64 vs. 2.85). Graph-Counselor itself collapses on LEGAL (Easy 26.26, Medium 0.00), reflecting the volatility of multi-agent prompting at the 3B scale.

B.2 Behavioral Analysis of Graph Interaction

Outcome metrics alone do not reveal whether a model has internalized the multi-round `reasoning`→`action`→`observation` procedure. We therefore complement Table 2 with behavioral diagnostics computed from execution traces, comparing GRAPHDANCER against Graph-CoT and Graph-Counselor (both on `Qwen2.5-3B-Instruct`).

We report the following metrics: (1) Valid Format (VF), the proportion of trajectories that follow the required `<think>` / `<graph>` / `<information>` / `<answer>` structure; (2) Valid Call (VC), 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 tool observation. Graph-Counselor uses a free-form Plan / Thought / Action / Observation protocol that does not map onto the `<think>`/`<graph>` block structure, so VF and VC are not applicable; EH is well-defined regardless of protocol since it inspects retrieved observations rather than format compliance.

Table 6 shows that GRAPHDANCER substantially improves VF over Graph-CoT across all domains (e.g., 31.9 → 46.7 on HEALTHCARE, 42.8 → 54.4 on LEGAL), confirming that curriculum-based post-training strengthens adherence to the multi-round interaction protocol. In terms of EH, the three methods cluster within ∼3 points on E-COMMERCE and LITERATURE; Graph-Counselor wins HEALTHCARE EH (31.11 vs. 21.9 for GRAPHDANCER) but collapses on LEGAL (17.78 vs. 43.9), reflecting the volatility of multi-agent prompting at the 3B scale.

Domain	Model	VF ↑	VC ↑	EH ↑
E-COMMERCE	Graph-CoT	62.5	99.1	49.0
	Graph-Counselor	–	–	49.5
	GRAPHDANCER	**78.0**	99.1	**50.0**
LITERATURE	Graph-CoT	48.8	98.6	51.7
	Graph-Counselor	–	–	55.0
	GRAPHDANCER	**70.0**	**100.0**	**55.8**
HEALTHCARE	Graph-CoT	31.9	**99.3**	20.4
	Graph-Counselor	–	–	**31.1**
	GRAPHDANCER	**46.7**	98.9	21.9
LEGAL	Graph-CoT	42.8	**97.8**	**46.7**
	Graph-Counselor	–	–	17.8
	GRAPHDANCER	**54.4**	89.1	43.9

Table 6: Behavioral analysis of execution traces on the four unseen GRBench domains. VF: fraction of episodes following the required interaction format; VC: fraction of tool calls passing schema validation; EH: fraction of episodes where the normalized gold answer appears in any returned tool observation. All methods use `Qwen2.5-3B-Instruct` as the LLM backbone. Graph-Counselor's multi-agent Plan / Thought / Action / Observation protocol does not match the `<think>`/`<graph>` format definition of VF/VC; EH is computed from the concatenation of every observation returned to the agents.

	Simple Lookup n=37	Aggregation n=29	Multi-hop n=21	Set Ops. n=5
	EM↑ / R-L↑	EM↑ / R-L↑	EM↑ / R-L↑	EM↑ / R-L↑
Graph-CoT	43.24 / 49.34	55.17 / 55.17	28.57 / 32.02	20.00 / 40.45
Graph-Counselor	48.65 / 55.87	48.28 / 48.28	28.57 / 29.32	0.00 / 16.26
GRAPHDANCER	**51.35 / 61.24**	**58.62 / 58.62**	**33.33 / 35.46**	**40.00 / 40.99**

Table 7: Question-type breakdown on Literature (%). Categories are derived by regex-matching the canonical question templates from Jin et al. (2024). All methods use `Qwen2.5-3B-Instruct` as the LLM backbone. Set Ops. = Set Operations.

B.3 Question-Type Breakdown on LITERATURE

To localize the gains of GRAPHDANCER on LITERATURE, we group test questions into four categories using a regex matcher over the canonical question templates from Jin et al. (2024). Table 7 reports exact match (EM) and Rouge-L for each category.

GRAPHDANCER achieves the best performance in all four categories under both metrics. The

largest gain appears on **Multi-hop** questions, where GRAPHDANCER reaches 33.33% EM, compared with 28.57% for both Graph-CoT and Graph-Counselor ($+4.76$). On **Set Operations**, GRAPHDANCER doubles Graph-CoT's EM from 20.00% to 40.00%, while Graph-Counselor falls to 0.00% on this small 5-question subset. For **Aggregation**, GRAPHDANCER improves over Graph-CoT by $+3.45$ EM and over Graph-Counselor by $+10.34$ EM. **Simple Lookup** is the closest category: GRAPHDANCER still edges Graph-Counselor by 2.70 EM and Graph-CoT by 8.11 EM, while Graph-Counselor's relatively strong 48.65% suggests that multi-agent prompting is effective for single-step retrieval when no multi-hop chaining is required.

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