GRAFT: GRAPH-TOKENIZED LLMS FOR TOOL PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly used to complete complex tasks by selecting and coordinating external tools across multiple steps. This requires aligning tool choices with subtask intent while satisfying directional execution dependencies among tools. To do this, existing methods model these dependencies as tool graphs and incorporate the graphs with LLMs through retrieval, serialization, or prompt-level injection. However, these external graph-use strategies all follow a matching paradigm, which often fails to align tool choices with the underlying subtask structure, producing semantically plausible plans that violate graph constraints. This issue is further exacerbated by error accumulation, where an early incorrect tool selection shifts the plan into an invalid graph state and causes subsequent predictions to drift away from the valid execution path. To address these challenges, we propose **GRAFT**, a graph-tokenized language model framework for dependency-aware tool planning. GRAFT internalizes the tool graph by mapping each tool node to a dedicated special token and learning directed tool dependencies within the representation space. It further introduces on-policy tool context distillation, training the model on its own sampled trajectories while distilling stepwise planning signals. Experiments show that GRAFT not only achieves state-of-the-art performance in exact sequence matching and dependency legality, but also improves end-to-end task execution on a realistic interactive agent benchmark, highlighting its potential for reliable real-world workflows.

1 INTRODUCTION

Large language models (LLMs) are increasingly extending language understanding into tool-augmented problem solving, where they interact with external tools, APIs, and expert models to perform precise computation, and support domain-specific operations beyond language generation alone (Shen, 2024; Xu et al., 2025). Despite the growing role of LLMs in task solving, multi-step tool planning remains a central challenge. Given a high-level user instruction, the model must understand the task intent, decompose it into subtasks, and generate an executable tool sequence in a valid order (Wu et al., 2024). This process is inherently dependency-aware, since tools are rarely independent and the output of one tool may serve as the input, prerequisite, or semantic context for another. Such dependencies naturally define a *tool dependency graph* (Liu et al., 2024), where nodes denote tools and directed edges specify valid execution transitions. Therefore, reliable multi-step tool planning requires generating tool sequences that satisfy both task semantics and graph-structured execution constraints, making it a sequence generation problem over a directed tool graph.

While this formulation highlights the importance of tool dependencies, existing methods still rely on external mechanisms to connect task reasoning with tool selection. A common strategy (Liu et al., 2025; Wu et al., 2024) is to first prompt the LLM to decompose the user instruction into intermediate subtasks and then retrieve tools for each subtask through graph search, as shown in Fig. 1 (a). However, this separates task decomposition from tool selection, so the generated subtasks may be incomplete or mismatched with available tools to construct a valid execution path. Another line of work, illustrated in Fig. 1 (b-c), provides graph structure as auxiliary context, either by serializing the tool graph into text (Lin et al., 2024; Wu et al., 2024) or by encoding tool relations into dense representations (Chen et al., 2026). However, both types of methods require lengthy prompts and struggle to align subtask relationships with tool transition semantics. Moreover, the core of all existing methods are still "matching", which heavily depends on the quality of query parsing and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

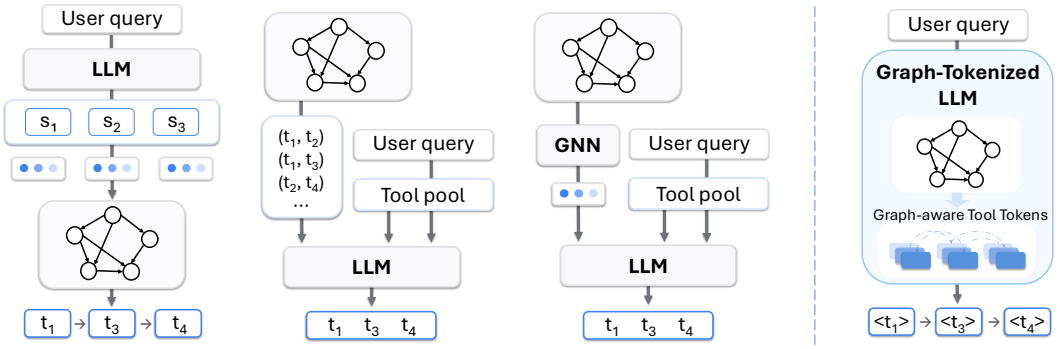

Figure 1: Existing methods use the tool graph externally: they retrieve tools t for decomposed subtasks s, serialize the graph into text, or inject graph embeddings as auxiliary LLM inputs. In contrast, our method parameterizes the tool graph as dependency-aware tool tokens, enabling the LLM to directly generate tool plans from the user query.

graph representation. Consequently, such external graph context may inform the LLM but cannot reliably constrain each next-tool decision, often producing semantically plausible plans that violate structural dependencies (Wu et al., 2024).

Beyond these structural limitations, LLM-based tool planning is also vulnerable to error accumulation. During multi-step planning, an early incorrect tool prediction changes the semantic state of the plan and moves generation away from the intended execution path. Subsequent predictions are then conditioned on an imperfect prefix, causing errors to propagate across later steps. This issue is exacerbated by supervised fine-tuning (SFT), which optimizes the model under gold prefixes during training but requires it to condition on its own generated prefixes during inference. Such train-test mismatch, known as exposure bias (Bengio et al., 2015; Ranzato et al., 2016; Lamb et al., 2016), reduces robustness to off-path states and leads to cascading failures in multi-step tool planning.

In light of these challenges, we propose Graph-Tokenized LLMs for Tool Planning (**GRAFT**), a framework that internalizes directed tool dependencies into LLMs and improves planning robustness through subtask-aware on-policy distillation. We first map each tool node to a dedicated special token and train the model to reconstruct directed successor relations from the hidden states of tool tokens. This **Graph-Token Edge Reconstruction** objective turns the tool graph from an external retrieval target into a parameterized capability of the LLM, enabling the model to jointly learn tool semantics and executable dependency structure. To mitigate exposure bias, we introduce **On-Policy Tool Context Distillation**. For each sample, the student samples a tool-token trajectory from its current policy using only the user query. Conditioned on the student-generated prefix, a frozen teacher is initialized from the same model and receives decomposed subtasks as privileged context to provide next-tool supervisions. This yields a self-distillation objective (Zhao et al., 2026; Ye et al., 2026) that trains the student on its own on-policy states while internalizing subtask-context guidance, without requiring such context at inference time. As a result, GRAFT learns to predict tools according to both the directed dependency graph and the intended reasoning process, leading to more executable and efficient multi-step planning. Our contributions are summarized as follows:

- We identify the limitations of external graph usage and SFT under exposure bias in multi-step tool planning, and propose a new formulation that internalizes the directed tool graph into LLMs for efficient dependency-aware tool planning.
- We propose a graph-token learning framework that internalizes directed tool dependencies into LLMs through tool-specific tokens and graph-token edge reconstruction, enabling the model to learn both tool semantics and executable transitions.
- We introduce on-policy tool context distillation, which uses subtasks as teacher planning context and trains the model on its own sampled trajectories to learn task decomposition signals for robust tool sequence generation.

- Experiments show that GRAFT achieves state-of-the-art performance, outperforming retrieval-, serialization- and prompt-based baselines in exact sequence matching and dependency legality. Further evaluation on the interactive agent benchmark demonstrates its effectiveness in improving end-to-end task execution, highlighting its potential for reliable real-world workflows. Our code is available at: anonymous.4open.science/r/GRAFT-24F2

2 Related Work

2.1 LLM-based Tool Planning

Early LLM-based methods mainly formulate tool planning as subtask-tool matching. They decompose a user query into subtasks and select suitable tools through prompting (Shen et al., 2023; 2024; Qin et al., 2023), retrieval (Qin et al., 2023; Yang et al., 2023), or tool-token generation (Wang et al., 2025). More recently, agentic tool-use methods (Masterman et al., 2024) formulate planning as a closed-loop decision process, where the LLM selects the next tool action based on the previous context, and execution observations, e.g., ReAct (Yao et al., 2022), AgentGym (Xi et al., 2025), and AgentFlow (Li et al., 2025). This interaction-based paradigm can refine decisions after tool execution, but it relies on the strong backbone LLM and requires repeated LLM calls with observation updates. More importantly, all these methods do not explicitly model directed dependencies among tools, making it difficult to guarantee dependency-valid tool sequences before execution.

To capture tool dependencies, recent graph-based studies represent tools as nodes and dependency relations as directed edges. ControlLLM (Liu et al., 2025) and PLAG (Lin et al., 2024; Wu et al., 2024) search solution paths over the tool graph, GNN4Plan (Wu et al., 2024) studies graph learning for task planning, and GTool (Chen et al., 2026) injects query-specific graph representations into LLMs. These methods demonstrate the importance of graph structure for valid tool sequencing. However, the graph is still used as an external search space, reasoning module, or auxiliary context, rather than being internalized into the LLM generation process.

In contrast, this paper focuses on single-pass pre-execution planning, where the LLM generates a complete dependency-valid tool sequence without relying on execution observations or repeated interaction. By internalizing tool dependencies into the generation process, GRAFT provides an efficient and reliable planning foundation that also extends to interactive agentic workflows.

2.2 On-policy Optimization

To adapt LLMs to task-specific generation, supervised fine-tuning (SFT) is commonly used to train the model on gold prefixes. However, during inference, the model must condition on its own generated prefixes, leading to a train-test mismatch known as exposure bias (Bengio et al., 2015; Ranzato et al., 2016; Lamb et al., 2016). On-policy learning mitigates this mismatch by optimizing the model on trajectories sampled from its current policy, thereby exposing it to inference-time states. This idea has been explored in Scheduled Sampling (Bengio et al., 2015), DAgger (Ross et al., 2011), on-policy reinforcement learning methods (e.g., REINFORCE (Williams, 1992), TRPO (Schulman et al., 2015), and PPO (Schulman et al., 2017)) and on-policy distillation methods (Song & Zheng, 2026; Ye et al., 2026; Zhao et al., 2026), all of which aim to improve robustness under policy-induced state distributions.

In multi-step tool planning, exposure bias is particularly harmful, where an early incorrect tool prediction can move the model away from the intended execution path and cause subsequent errors to accumulate. To mitigate this, we introduce on-policy tool context distillation in GRAFT, which incorporates tool context and distills planning signals from the model's own generated trajectories, improving the dependency validity of generated tool sequences.

Further comparisons with closely related methods are provided in Appendix F.

3 METHODOLOGY

To address the aforementioned challenges, we propose a graph-tokenized framework for executable multi-step tool planning. Our framework first internalizes directed tool dependencies into the LLM by representing tools as dedicated tokens and learning graph-aware tool representations. It then improves multi-step planning robustness through on-policy distillation, enabling the model to generate complete dependency-valid tool sequences under its own autoregressive planning states.

3.1 PROBLEM FORMULATION

Let $\mathcal{T} = \{t_1, t_2, \ldots, t_M\}$ denote the tool set, where each tool t_i is an executable API or function with textual metadata, such as its name and description. The dependencies among tools are represented as a directed tool graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where each directed edge $(t_i, t_j) \in \mathcal{E}$ indicates that tool t_j can be legally invoked after t_i. Such a dependency typically means that the output or precondition produced by t_i is required by t_j.

Given a user query q, the goal of tool planning is to generate an executable tool trajectory that completes the task by invoking tools in a valid order, as illustrated in Fig. 5. A training sample is denoted as $(q, s, y) \in \mathcal{D}$, where $s = (s_1, \ldots, s_L)$ denotes the pre-defined subtask decomposition provided in the training data, and $y = (y_1, \ldots, y_L)$ is the corresponding ground-truth tool trajectory. Each element $y_k \in \mathcal{T}$ denotes the tool used to solve subtask s_k. The ground-truth trajectory is executable under the directed tool graph, satisfying: $(y_k, y_{k+1}) \in \mathcal{E}$, where $k = 1, \ldots, L-1$.

We study LLM-based tool planning, where an LLM π_θ takes the user query q and the directed tool graph $\mathcal{G}$ as input, and generates a predicted tool trajectory $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_K)$ as:

$$\pi_\theta(\hat{y} \mid q, \mathcal{G}) = \prod_{k=1}^{K} \pi_\theta(\hat{y}_k \mid q, \mathcal{G}, \hat{y}_{<k}). \tag{1}$$

The expected output is a tool sequence that matches the ground-truth trajectory y and satisfies the tool dependencies specified by $\mathcal{G}$.

3.2 GRAPH TOKENIZATION AND SUBTASK GROUNDING

To internalize the tool graph into the LLM rather than using it as external context, we first treats each tool as an atomic semantic unit by representing each tool node as a dedicated token.

Tool tokenization. We map each tool $t_i \in \mathcal{T}$ to a dedicated token via an indexing function τ:

$$\tau(t_i) = a_i, \quad \mathcal{A} = \{a_1, a_2, \ldots, a_M\}, \quad \mathcal{V}' = \mathcal{V}_{\text{LM}} \cup \mathcal{A}. \tag{2}$$

Here, $\mathcal{V}_{\text{LM}}$ denotes the original LLM vocabulary, $\mathcal{A}$ is the tool-token set, and $\mathcal{V}'$ is the expanded vocabulary. We adopt an atomic token format, where each tool is represented as a single indivisible token, e.g., the `Image-to-Text` tool is mapped to `<Image_to_Text>`. The embedding of each tool token is initialized using its textual metadata.

Afterwards, for a ground-truth tool trajectory $y = (y_1, \ldots, y_L)$, we can tokenize it as token sequence $z = (z_1, \ldots, z_L)$, where $z_k = \tau(y_k) \in \mathcal{A}$. Tool planning is then converted into autoregressive generation over $\mathcal{A}$. This formulation constrains generation to the tool-token space rather than free-form text, thereby reducing hallucinated tool invocations as our empirical study in Appendix 4.6.

Subtask-to-tool grounding. Tool metadata provides only static descriptions, whereas the same tool may serve different purposes across planning scenarios. To help the LLM associate tool tokens with their practical usage in specific task contexts, we construct subtask-token pairs from each training example (q, s, y):

$$\mathcal{D}_{\text{sub}} = \{(s_k, z_k) \mid (q, s, y) \in \mathcal{D},\ z_k = \tau(y_k),\ k = 1, \ldots, |y|\}. \tag{3}$$

Here, the subtask s_k describes the local operation required at step k, while the tool token z_k represents the tool selected to perform it. The model is then trained to generate the corresponding tool token conditioned on each subtask:

$$\mathcal{L}_{\text{sub}} = -\sum_{(s_k, z_k) \in \mathcal{D}_{\text{sub}}} \log \pi_\theta(z_k \mid s_k). \tag{4}$$

This grounding stage connects natural-language subtasks with executable tool tokens, enabling the model to learn task-specific tool usage. The resulting grounded tool-token representations then provide the basis for directed edge reconstruction in the next stage.

3.3 Graph-Token Edge Reconstruction

Training sequence generation. With the constructed tool tokens, we expect them to capture dependency relations in the tool graph. A direct approach is to train each tool token independently to reconstruct its neighboring tools. However, this requires separate forward passes for individual tool nodes, which is inefficient and inconsistent with autoregressive tool planning, where tools are generated conditioned on preceding tool contexts. To align with this generation process, we construct training sequences ρ by sampling directed paths with different lengths from the tool graph $\mathcal{G}$:

$$\rho = (u_1, \ldots, u_R), \quad u_r \in \mathcal{T}, \quad (u_r, u_{r+1}) \in \mathcal{E}, \tag{5}$$

where u_r denotes a tool node and $1 \leq R \leq R_{\max}$. These variable-length paths provide sequential dependency contexts, ranging from single-tool states to multi-hop dependencies.

Directed edge reconstruction. Given a sampled path ρ, we feed the corresponding tool-token sequence into the LLM. For each position r, we extract the contextual hidden state $\mathbf{h}_r$ from the penultimate layer at the tool-token position of u_r. We then construct a candidate set $\mathcal{C}(u_r)$ for u_r containing its valid outgoing neighbors $\mathcal{N}^+(u_r)$, sampled negative tools, and reverse-edge negatives.

For each tool $t_j \in \mathcal{C}(u_r)$, let $\mathbf{e}_j$ be the embedding of its tool token and the directional edge score is computed as:

$$s_{rj} = \frac{\mathrm{sim}\left(\mathbf{W}_h \mathbf{h}_r, \mathbf{W}_e \mathbf{e}_j\right)}{\gamma}, \tag{6}$$

where $\mathbf{W}_h$ and $\mathbf{W}_e$ are projection matrices, $\mathrm{sim}(\cdot, \cdot)$ is normalized dot product, and γ is a temperature coefficient. The reconstruction loss for the tools in path ρ is defined as:

$$\mathcal{L}_{\mathrm{edge}}(\rho) = -\frac{1}{|\mathcal{I}_\rho|} \sum_{r \in \mathcal{I}_\rho} \frac{1}{|\mathcal{N}^+(u_r)|} \sum_{t_j \in \mathcal{N}^+(u_r)} \log \frac{\exp(s_{rj})}{\sum_{t_k \in \mathcal{C}(u_r)} \exp(s_{rk})}, \tag{7}$$

where $\mathcal{I}_\rho = \{r \mid \mathcal{N}^+(u_r) \neq \emptyset\}$ denotes the positions whose tools have outgoing edges. The final objective averages Eq. (7) over all sampled paths.

Instead of predicting only the next tool in the sampled path ρ, each node representation is trained to recover its graph-defined successors. This contrastive reconstruction objective encourages $\mathbf{h}_r$ to encode the outgoing dependency structure of u_r, allowing directed edges to be recovered from tool-token representations (Li et al., 2023; Zhang et al., 2026; Jin et al., 2024). Thus, the LLM internalizes directed tool dependencies and leverages them at inference time without explicit graph serialization or retrieval.

3.4 On-Policy Tool Context Distillation

After internalizing the tool graph, the model captures both the semantic meanings of tool tokens and the directed dependency relations among tools. Then, we adapt it to query-conditioned tool planning through a two-stage training procedure.

Query-to-tool SFT. We first warm up the graph-tokenized policy with SFT, training it to generate the tool-token sequence from the user query. For each training sample $(q, s, y) \in \mathcal{D}$ and the tokenized trajectory $z = (z_1, \ldots, z_L)$, the SFT objective is

$$\mathcal{L}_{\mathrm{sft}} = -\sum_{(q,s,y) \in \mathcal{D}} \sum_{k=1}^{L} \log \pi_\theta(z_k \mid q, z_{<k}). \tag{8}$$

This warm-up stage provides a stable query-to-tool planning policy for subsequent on-policy rollout.

On-policy tool context distillation. After SFT, we initialize both the student and teacher from the SFT checkpoint. The student is the trainable graph-tokenized policy π_θ, while the teacher is a frozen copy of the same model, denoted by $\pi_{\bar{\theta}}$. Unlike the student, the teacher is additionally conditioned on privileged planning context, including decomposed subtasks and the ground-truth tool trajectory. This forms a self-distillation setting (Song & Zheng, 2026; Zhao et al., 2026) in which the teacher provides context-aware planning guidance, while the student learns to reproduce such guidance under its own generated tool prefixes.

Specifically, we define the student and teacher prompts as

$$P_S(q) = [\text{Query} : q], \quad P_T(q, s, y) = [\text{Query} : q;\ \text{Subtasks} : s;\ \text{Tools} : z]. \tag{9}$$

The student receives only $P_S(q)$ and samples a tool-token trajectory $\hat{z} \sim \pi_\theta(\cdot \mid P_S(q))$. At step k, both policies are conditioned on the same student-generated prefix $\hat{z}_{<k}$ and produce next-token distributions:

$$p_k^S = \pi_\theta(\cdot \mid P_S(q), \hat{z}_{<k}), \quad p_k^T = \pi_{\bar{\theta}}(\cdot \mid P_T(q, s, y), \hat{z}_{<k}). \tag{10}$$

Here, the subtasks provide step-level planning guidance, while the ground-truth trajectory anchors the teacher to the correct tool sequence, enabling token-level supervision under student-generated prefixes. Finally, we minimize the reverse KL divergence (Gu et al., 2024; Ye et al., 2026; Song & Zheng, 2026) along the student rollout:

$$\mathcal{L}_{\text{opd}} = \mathbb{E}_{(q,s,y)\sim\mathcal{D}} \mathbb{E}_{\hat{z}\sim\pi_\theta(\cdot|P_S(q))} \left[\frac{1}{|\hat{z}|} \sum_{k=1}^{|\hat{z}|} D_{\text{KL}} \left(p_k^S \,\|\, p_k^T \right) \right]. \tag{11}$$

In practice, the divergence is computed only over the tool tokens and the end-of-sequence token. Specifically, we truncate the student and teacher logits to this restricted output space and re-normalize them before computing the divergence (Zhao et al., 2026), focusing distillation on executable tool-sequence generation rather than the full natural-language vocabulary.

In the second training stage, the student is optimized with the combined objective:

$$\mathcal{L} = \mathcal{L}_{\text{sft}} + \lambda \mathcal{L}_{\text{opd}}, \tag{12}$$

where λ controls the strength of on-policy distillation, and the SFT loss stabilizes policy optimization. Only the student policy is updated, while the teacher remains frozen. By doing so, the student is optimized under its own inference-time state distribution, while the teacher provides dense token-level supervision enriched with subtask reasoning knowledge.

Inference. At inference time, the model only takes the user query as input, without requiring explicit subtask decomposition, ground-truth trajectories, or additional graph context. Given only a user query q, the trained policy autoregressively generates a tool-token sequence $\hat{z} = (\hat{z}_1, \ldots, \hat{z}_K)$ over the restricted tool-token vocabulary. The sequence is then mapped back to the predicted tool trajectory $\hat{y}$ via the inverse token mapping.

The complete training procedure is summarized in Algorithm A, and the detailed student and teacher prompts are provided in Appendix G.

4 Experiments

We conduct extensive experiments across multiple datasets to evaluate GRAFT against representative baselines, followed by ablation studies, efficiency analysis, teacher prompt analysis and evaluation in interactive agentic scenarios. Notice that hallucination analysis, hyperparameter sensitivity analysis, and additional tool dependency analysis can be found in Appendix E .

4.1 Experimental Settings

Datasets. We evaluate GRAFT on four tool-planning datasets and one interactive agent benchmark. The planning datasets include HuggingFace and Multimedia from TaskBench (Shen et al., 2024; Wu et al., 2024), covering AI and multimodal tasks; UltraTool (Huang et al., 2024), covering diverse real-world scenarios; and ToolBench (Qin et al., 2023), featuring practical multi-step tool use. Each

Table 1: Comparison of our method with baselines under EM (%), NED, and ACPL. The best result is shown in **bold**, and the runner-up is underlined.

LLM	Method	HuggingFace			Multimedia			UltraTool			ToolBench		
		EM	NED	ACPL	EM	NED	ACPL	EM	NED	ACPL	EM	NED	ACPL
Qwen2.5-1.5B	BeamSearch	1.75	0.76	0.36	3.52	0.71	0.41	5.60	0.79	0.35	10.59	0.73	0.51
	Direct	4.85	0.70	0.44	5.21	0.65	0.61	3.62	0.87	0.34	9.15	0.79	0.32
	HuggingGPT	4.81	0.74	0.33	5.41	0.63	0.67	3.83	0.88	0.35	9.21	0.85	0.36
	PLaG	4.86	0.76	0.36	5.21	0.66	0.67	4.85	0.76	0.36	15.29	0.62	0.65
	GNN4Plan	13.43	0.52	0.99	23.80	0.42	1.59	12.02	0.63	0.65	6.40	0.71	0.70
	GTool	21.80	0.43	1.33	48.00	0.24	1.67	41.60	0.37	1.25	10.80	0.82	0.51
	ToolGen	50.00	0.19	2.28	55.20	0.18	2.61	53.00	0.28	1.48	31.40	0.41	1.03
	Ours	**53.20**	**0.18**	**2.33**	**64.00**	**0.16**	**2.75**	**60.60**	**0.22**	**1.63**	**39.40**	**0.35**	**1.28**
Llama-3.2-3B	BeamSearch	6.54	0.62	0.65	5.58	0.69	0.54	3.29	0.82	0.20	11.70	0.76	0.57
	Direct	8.69	0.61	0.71	11.30	0.58	0.89	4.90	0.85	0.37	15.58	0.66	0.57
	HuggingGPT	7.59	0.63	0.65	11.51	0.55	0.88	4.99	0.88	0.38	15.15	0.70	0.57
	PLaG	7.17	0.71	0.59	12.84	0.52	0.88	5.24	0.92	0.44	16.86	0.64	0.67
	GNN4Plan	18.40	0.47	1.20	37.65	0.33	2.05	16.84	0.59	0.74	6.61	0.78	0.70
	GTool	37.00	0.30	1.85	56.80	0.21	2.55	55.60	0.25	1.55	11.80	0.67	0.52
	ToolGen	57.20	0.17	2.42	67.80	0.15	2.83	58.60	0.24	1.63	33.00	0.38	1.18
	Ours	**62.40**	**0.13**	**2.62**	**74.80**	**0.11**	**3.09**	**65.20**	**0.18**	**1.75**	**41.80**	**0.34**	**1.33**
Mistral-7B	BeamSearch	12.53	0.49	0.93	29.12	0.37	1.59	15.45	0.59	0.65	13.26	0.66	0.57
	Direct	7.82	0.56	0.74	16.06	0.47	1.10	5.02	0.67	0.44	15.47	0.99	0.52
	HuggingGPT	7.39	0.56	0.53	10.02	0.56	0.82	8.38	0.74	0.40	16.01	0.94	0.60
	PLaG	8.53	0.54	0.56	11.07	0.55	0.78	10.52	0.71	0.48	21.95	0.50	0.90
	GNN4Plan	12.22	0.51	1.06	34.27	0.36	1.95	15.43	0.56	0.80	6.70	0.79	0.72
	GTool	45.20	0.23	2.04	59.20	0.19	2.60	63.80	0.20	1.69	13.80	0.66	0.58
	ToolGen	60.00	0.14	2.55	68.40	0.14	2.94	62.00	0.20	1.71	35.00	0.37	1.21
	Ours	**63.20**	**0.13**	**2.63**	**74.20**	**0.11**	**2.98**	**66.20**	**0.17**	**1.78**	**42.60**	**0.32**	**1.35**

dataset provides user queries, ground-truth subtasks, tool sequences, and directed tool graphs. To further assess end-to-end execution, we use the retail domain of τ^2-bench, an interactive benchmark with 15 tools and a dependency graph constructed from training trajectories. Further details are provided in Appendix B.

Baselines. We compare GRAFT with seven representative baselines. BEAMSEARCH (Wu et al., 2024; Liu et al., 2025) performs graph-constrained search with LLM-based scoring. DIRECT prompts the LLM to generate tool sequences directly from user queries and the tool pool. HUGGINGGPT (Shen et al., 2023) decomposes queries into subtasks and assigns corresponding tools. PLAG (Lin et al., 2024) serializes the tool graph into text for LLM-based planning. GNN4PLAN (Wu et al., 2024) uses graph neural networks to encode tools for subtask-level retrieval. GTOOL (Chen et al., 2026) injects graph-aware representations into the LLM for tool generation. TOOLGEN (Wang et al., 2025) represents tools as dedicated tokens for generative tool retrieval without explicitly modeling tool dependencies.

Evaluation metrics. We evaluate tool-planning performance using three primary metrics. Exact Match (EM) measures the fraction of predicted sequences that exactly match the ground truth. Normalized Edit Distance (NED) measures sequence-level differences using Levenshtein distance normalized by the maximum sequence length, where lower is better. Average Correct Prefix Length (ACPL) measures the number of correctly predicted tools before the first error. Additional metrics, including Edge Legality Rate (ELR), Tool F1, and Prefix Accuracy@K, are defined in Appendix B, with results reported in Appendix E.

Implementation details. We evaluate all methods with three open-source LLM backbones: Qwen2.5-1.5B-Instruct (Yang et al., 2024), Llama-3.2-3B-Instruct (Meta AI, 2024), and Mistral-7B-Instruct (Jiang et al., 2023). All models are fine-tuned with rank-16 LoRA (Hu et al., 2021). To sample training sequences for edge reconstruction, the maximum path length $R_{\max}$ is set to the maximum tool-sequence length of each dataset, and path lengths follow the empirical length distribution. We use a positive-to-negative edge ratio of 1:10 and set the edge-scoring temperature to $\tau = 0.1$. The distillation loss weight is selected from $\{0, 0.1, 0.3, 0.5, 0.7, 1, 10\}$. All experiments are conducted on a server with Intel Xeon Gold 6326 CPUs at 2.90GHz and NVIDIA A40 GPUs with 48GB memory.

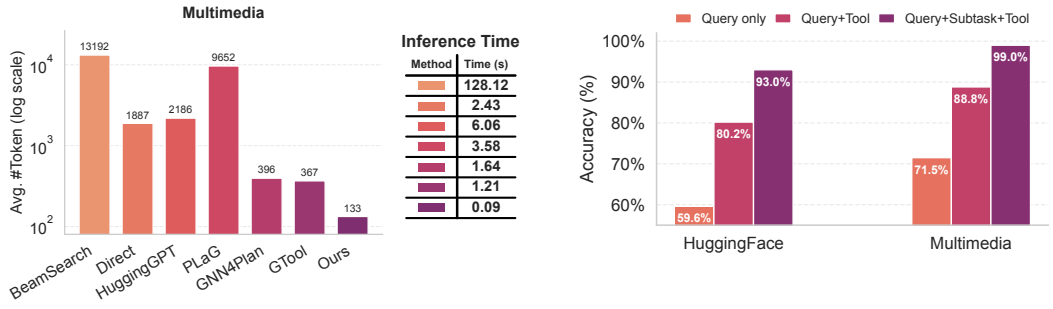

Figure 2: Efficiency and teacher prompt analysis. (a) compares total token cost and average inference time per test sample using Qwen2.5-1.5B. (b) reports validation accuracy under different teacher prompt variants using Llama-3.2-3B.

4.2 Main Results

Table 1 compares all methods in terms of EM, NED, and ACPL. GRAFT consistently achieves the best performance across all datasets and LLM backbones, outperforming both conventional planning methods and graph-based baselines. Compared with ToolGen, GRAFT achieves higher EM, lower NED, and longer correct prefixes, demonstrating improvements in both sequence accuracy and robustness against error accumulation. These results highlight the effectiveness of explicitly modeling tool dependencies and mitigating error propagation in multi-step planning.

4.3 Ablation Study

Table 2 reports the ablation results on Llama-3.2-3B. Here, *w/o distill.* removes the on-policy distillation loss, *w/o graph* removes edge reconstruction, *w/o subtask* removes the subtask context from the teacher prompt, and *w/o STT* removes subtask-to-tool grounding training. Removing distillation leads to the largest drop on both datasets, particularly in ACPL, indicating that on-policy learning is important for preserving longer correct prefixes. Removing graph reconstruction also degrades EM and increases NED, confirming that learning directed tool dependencies improves prediction accuracy. The performance drops from *w/o subtask* and *w/o STT* further show that subtask-level supervision helps align task decomposition with tool selection.

Further analysis of learned tool dependencies and the graph reconstruction loss is provided in Appendix E.2.

4.4 Efficiency Analysis

We compare the token consumption and inference time of different methods in Figure 2(a). BeamSearch requires multiple iterative LLM calls, leading to the largest token cost and inference time. PLaG serializes the tool graph into the prompt, which substantially increases the input length and reasoning complexity. HuggingGPT also introduces additional subtask-level reasoning, resulting in relatively high token usage. GNN4Plan and GTool reduce token cost by encoding the graph with GNNs, but they still need to include the tool descriptions in the prompt. In contrast, our method internalizes tool graph knowledge through tokens and only requires the user query at inference time, achieving the lowest token consumption and significant reduction in per-query inference time.

Table 2: Ablation study on Llama-3.2-3B.

Method	HuggingFace			Multimedia		
	EM	NED	ACPL	EM	NED	ACPL
w/o distill.	59.81	0.17	2.41	71.76	0.15	2.88
w/o graph	60.25	0.16	2.44	72.90	0.15	2.90
w/o subtask	61.90	0.16	2.57	73.23	0.14	2.93
w/o STT	62.14	0.14	2.59	74.10	0.13	3.00
Ours	62.40	0.13	2.62	74.80	0.11	3.09

4.5 Teacher Prompt Analysis

Figure 2(b) compares different teacher prompt designs to examine how planning context affects the quality of distillation supervision. Compared with the query-only prompt, providing the ground-truth

tool sequence improves teacher prediction accuracy on both datasets. However, the accuracy is still far from perfect, indicating that the teacher does not simply copy the provided answer. Instead, the ground-truth sequence serves as useful planning guidance, while the teacher still needs to interpret the query and predict the next tool under the given context. More importantly, adding subtask information leads to a substantial further improvement. This suggests that subtasks provide explicit intermediate intent and help the teacher better understand the stepwise structure of the task. Overall, these results confirm the effectiveness of subtask-aware teacher prompting and support our design of using privileged planning context to guide on-policy distillation.

4.6 Hallucination Analysis

Figure 3 reports the hallucination ratio, defined as the proportion of generated tool names outside the tool pool among all generated tool predictions. Direct prompting yields the highest hallucination ratio, indicating that plain-text tool descriptions are insufficient for reliable tool grounding. BeamSearch reduces hallucination but still generates invalid tools, since its search space relies on both graph and LLM-estimated candidate tools. HuggingGPT, PLaG, and GTool also suffer from hallucination because they remain generative retrieval-based methods that select tools from prompt-provided descriptions. In contrast, GRAFT achieves a hallucination ratio of 0.00%. This is because each tool is represented as a dedicated token, restricting generation to the tool-token space, while subtask grounding and query-to-tool SFT further align tool tokens with practical usage and task-level planning.

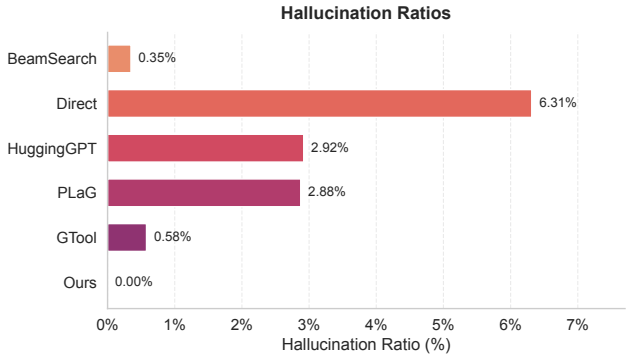

Figure 3: Hallucination ratios on Multimedia using Llama-3.2-3B.

4.7 Interactive Agentic Evaluation

We further evaluate GRAFT on the retail domain of τ^2-bench to examine whether improved tool planning translates into better end-to-end task execution. We compare GRAFT with PLaG and ToolGen using the same Llama-3.2-3B backbone, while keeping the executor and user agent fixed. As shown in Table 3, GRAFT achieves the highest task completion rate under both single-attempt (40.0%) and five-attempt (85.0%) settings, while maintaining 98.7% valid graph edges. These results demonstrate its effectiveness in interactive agentic workflows. Further implementation details are provided in Appendix D.

Table 3: Interactive agentic evaluation on τ^2-bench (Retail). All values are percentages.

Method	TCR (1)	VGER	TCR (5)
PLaG	12.5	75.0	52.5
ToolGen	30.0	92.9	80.0
GRAFT	**40.0**	**98.7**	**85.0**

5 Conclusion

We presented GRAFT, a framework for multi-step tool calling that internalizes directed tool dependencies through graph-token learning and on-policy tool context distillation. By reconstructing successor edges from tool-token hidden states and learning from student-generated trajectories under a subtask-aware privileged teacher, GRAFT significantly improves both exact planning accuracy and dependency legality than baselines on multiple real-world benchmarks.

A potential limitation of GRAFT is its reliance on a predefined tool-dependency graph, which may require updates as tools and their dependencies evolve. Future work will explore dynamic graph adaptation to support more flexible and evolving agentic workflows.

AI USE STATEMENT

Generative AI tools were used solely to improve the grammar, clarity, and readability of the manuscript. The authors reviewed all suggested edits and take full responsibility for the final content.

REFERENCES

Anonymous. Graphtoolbench: Benchmarking llms for sequential graph comprehension and conflict identification in tool learning, 2026. URL https://openreview.net/forum?id=zABFKZKtjK. OpenReview.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2015.

Wenjie Chen, Di Yao, Wenbin Li, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph enhanced tool planning with large language model. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bn47cqGQ7l.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *The twelfth international conference on learning representations*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, et al. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4363–4400, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):8622–8642, 2024.

Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems*, 2016.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. A survey of graph meets large language model: Progress and future directions. *arXiv preprint arXiv:2311.12399*, 2023.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*, 2025.

Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony G. Cohn, and Janet B. Pierrehumbert. Graph-enhanced large language models in asynchronous plan reasoning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 30108–30134. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/lin24k.html.

Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. Toolnet: Connecting large language models with massive tools via tool graph. *arXiv preprint arXiv:2403.00839*, 2024.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. Controllllm: Augment language models with tools by searching on graphs. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 89–105, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73254-6.

Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. *arXiv preprint arXiv:2404.11584*, 2024.

Meta AI. Llama 3.2 3b instruct model card. https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct, 2024. Accessed: 2026-05-07.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The twelfth international conference on learning representations*, 2023.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*, 2016.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36: 38154–38180, 2023.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574, 2024.

Zhuocheng Shen. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*, 2024.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models. *arXiv preprint arXiv:2604.00626*, 2026.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=XLMAMmowdY.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, et al. Can graph learning improve planning in llm-based agents? *Advances in Neural Information Processing Systems*, 37:5338–5383, 2024.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, et al. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*, 2025.

Weikai Xu, Chengrui Huang, Shen Gao, and Shuo Shang. Llm-based agents for tool learning: A survey: W. xu et al. *Data Science and Engineering*, pp. 1–31, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

Zhongjian Zhang, Xiao Wang, Mengmei Zhang, Jiarui Tan, and Chuan Shi. Toward graph-tokenizing large language models with reconstructive graph instruction tuning. In *Proceedings of the ACM Web Conference 2026*, pp. 430–441, 2026.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Appendix

CONTENTS

A Algorithm **13**

B Datasets **13**

C Evaluation Metrics **14**

D Evaluation Setting for Agentic system **15**

E Additional Results **16**

E.1 Other Metric Results 16

E.2 Analysis of Learned Tool Dependencies 17

E.3 Hyperparameter Sensitivity 17

E.4 Incremental Adaptation to New Tools 17

E.5 Robustness to Imperfect Tool Graphs 18

F Related Work **18**

G Prompts **19**

A ALGORITHM

Algorithm 1 outlines the optimization pipeline of GRAFT. The first stage constructs graph-tokenized tool representations and grounds them with subtask-level supervision. The second stage samples directed tool paths from $\mathcal{G}$ and optimizes edge reconstruction to internalize tool dependencies. The third stage warms up the planner with query-to-tool SFT. The final stage performs on-policy distillation, where the student is updated with the combined SFT and distillation objective under its own sampled trajectories.

Algorithm 1: Overall optimization pipeline of GRAFT.

Input: Training data $\mathcal{D} = \{(q, s, y)\}$, tool graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, LLM π_θ
Output: Trained planner π_θ
Map tools to dedicated tokens by Eq. (2);
Construct $\mathcal{D}_{\text{sub}}$ by Eq. (3) and update π_θ with $\mathcal{L}_{\text{sub}}$ in Eq. (4) ; `// Tokenization & Subtask SFT`
Construct a path set $\mathcal{R}$ by sampling ρ from $\mathcal{G}$ via Eq. (5);
for $epoch = 1, \ldots, E_{\text{edge}}$ **do**
    foreach *mini-batch* $\mathcal{B} \subset \mathcal{R}$ **do**
        Compute successor scores by Eq. (6);
        Update π_θ with $\mathcal{L}_{\text{edge}}$ in Eq. (7) ; `// Graph Reconstruction`
for $epoch = 1, \ldots, E_{\text{sft}}$ **do**
    foreach *mini-batch* $\mathcal{B} \subset \mathcal{D}$ **do**
        Compute $\mathcal{L}_{\text{sft}}$ on $\mathcal{B}$ by Eq. (8);
        Update π_θ with $\mathcal{L}_{\text{sft}}$; `// Query-to-tool Warm-up`
Initialize student π_θ and frozen teacher $\pi_{\bar{\theta}}$ from the SFT checkpoint;
for $epoch = 1, \ldots, E_{\text{distill}}$ **do**
    foreach *mini-batch* $\mathcal{B} \subset \mathcal{D}$ **do**
        Sample student trajectories $\hat{z} \sim \pi_\theta(\cdot \mid P_S(q))$;
        Compute $\mathcal{L}_{\text{sft}}$ by Eq. (8) and $\mathcal{L}_{\text{opd}}$ by Eq. (11);
        Update π_θ with $\mathcal{L} = \mathcal{L}_{\text{sft}} + \lambda \mathcal{L}_{\text{opd}}$ in Eq. (12) ; `// Distillation + SFT`
Return: Optimized π_θ

B DATASETS

We evaluate GRAFT on four publicly available datasets with directed tool graphs: HuggingFace (Wu et al., 2024), Multimedia (Wu et al., 2024), UltraTool (Huang et al., 2024), and ToolBench (Qin et al., 2023). To ensure consistent evaluation across different benchmarks, we process all datasets into a unified sequential tool-planning format. In the original graph-structured annotations, nodes represent tools and directed edges represent dependency relations between tools. During preprocessing, we use these original task nodes and graph edges to recover the ordered tool invocation sequence. After reconstruction, each final dataset instance contains the sample ID, user query, decomposed subtasks, and the ordered tool sequence. Fig. (5) shows representative examples from the experimental datasets.

For HuggingFace, Multimedia, and UltraTool, we use the datasets adopted in GNN4Plan (Wu et al., 2024) as our base datasets and retain the original tool pool and tool graph of each dataset. We first reformat the original data into a unified sequential structure and retain multi-tool invocation samples with chain-structured dependencies. For each retained sample, we recover the tool invocation path from its original dependency annotations and extract the ordered tool sequence associated with the given user query as the ground truth.

As for ToolBench, we follow the GraphToolBench (Anonymous, 2026) and use the publicly released data construction methodology and code. Specifically, 612 high quality tools are selected and GPT 5.4 is used to generate the dependency-aware user query, together with intermediate task nodes and dependency links. These intermediate annotations are used to recover the ordered tool invocation sequence.

Table 4: Statistics of the evaluated datasets. "Max Len." denotes the maximum length of the ground truth tool sequence.

Dataset	#Tools	#Edges	#Train	#Val	#Test	Max Len.
HuggingFace	23	225	2351	500	500	8
Multimedia	40	449	1835	500	500	8
UltraTool	260	611	2185	500	500	5
ToolBench	612	3140	1908	500	500	10

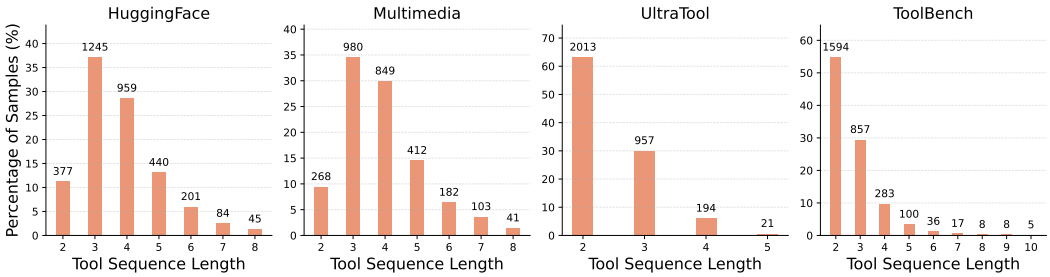

Figure 4: The distribution of tool sequence length for different datasets.

Each dataset is randomly split into training, validation, and test sets, with 500 samples for validation, 500 samples for testing, and the remaining samples for training. The detailed statistic of dataset and the tool sequence length distribution are shown in Table 4 and Fig. (4), respectively.

C Evaluation Metrics

Evaluation metrics. We report the following metrics to evaluate our proposed method.

Exact Match (EM). Exact Match measures whether the predicted tool sequence exactly matches the ground-truth trajectory:

$$\mathrm{EM} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left[\hat{y}_i = y_i\right]. \tag{13}$$

EM is the strictest metric, requiring both tool identities and their order to be fully correct.

Edge Legality Rate (ELR). Edge Legality Rate measures whether predicted adjacent tool transitions satisfy the directed dependency constraints in the tool graph. For each example, we compute

$$\mathrm{ELR}_i = \frac{1}{\hat{L}_i - 1}\sum_{k=1}^{\hat{L}_i-1} \mathbb{I}\left[(\hat{y}_{i,k}, \hat{y}_{i,k+1}) \in \mathcal{E}\right], \tag{14}$$

The final score is averaged over all test examples: $\mathrm{ELR} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{ELR}_i$. ELR evaluates whether the generated trajectory is executable under the tool graph, rather than whether it exactly matches the reference.

Average Correct Prefix Length (ACPL). Average Correct Prefix Length measures the number of correctly predicted tools before the first mismatch:

$$c_i = \max\left\{k : 0 \le k \le \min(L_i, \hat{L}_i),\ \hat{y}_{i,1:k} = y_{i,1:k}\right\}, \qquad \mathrm{ACPL} = \frac{1}{N}\sum_{i=1}^{N} c_i. \tag{15}$$

ACPL reflects how far the model can follow the ground-truth trajectory before making its first error.

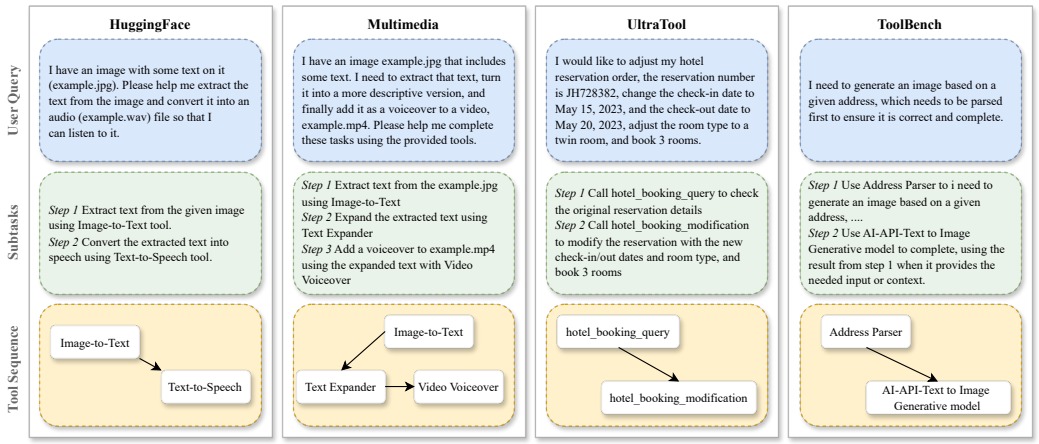

Figure 5: Representative examples from the experimental datasets.

Tool F1. Tool F1 evaluates tool selection accuracy regardless of order. For each example, we compare the set of predicted tools with the set of ground-truth tools, and compute the harmonic mean of node precision and node recall. This metric captures whether the model chooses the correct tools, but ignores their ordering.

Normalized Edit Distance (NED). Normalized Edit Distance measures the sequence-level distance between prediction and ground truth. We compute the Levenshtein edit distance with unit insertion, deletion, and substitution costs, and normalize it by the maximum sequence length:

$$\mathrm{NED}_i = \frac{\mathrm{EditDist}(\hat{y}_i, y_i)}{\max(\hat{L}_i, L_i)}. \tag{16}$$

A lower NED indicates a prediction closer to the ground-truth trajectory.

Prefix Accuracy@K (PA@K). Prefix Accuracy@K measures whether the model correctly predicts the first K tools of the ground-truth trajectory. Using the correct prefix length c_i, it is computed as

$$\mathrm{PA@}K = \frac{1}{|\mathcal{D}_K|} \sum_{i \in \mathcal{D}_K} \mathbb{I}[c_i \geq K], \qquad \mathcal{D}_K = \{i \mid L_i \geq K\}. \tag{17}$$

This metric evaluates prefix correctness at different planning depths.

D Evaluation Setting for Agentic System

Evaluation setup. We evaluate GRAFT on the retail domain of τ^2-bench, which features interactive customer-service tasks involving user authentication, order management, and product returns or exchanges. Our evaluation framework consists of three agents: a planner that predicts the next tool, an executor that validates the selected tool and generates its arguments, and a user agent that provides information or confirmation when required. We use the default model GPT-4.1 for both the executor and user agent, varying only the planner to isolate the effect of planning. All planning methods use the same Llama-3.2-3B backbone.

Tool-dependency graph. The retail environment contains 15 tools. We construct a directed dependency graph from tool transitions in the gold training trajectories, yielding 51 directed edges. These edges capture implicit information dependencies among tools. For example, `get_order_details` returns item IDs that can be used by `get_item_details`, forming the directed edge `get_order_details` $\rightarrow$ `get_item_details`.

Interactive inference. To adapt our sequence-trained planner to interactive execution, we use autoregressive tool-chain prefix continuation. At each step, the planner predicts the next tool

Table 5: Comparison of our method with baselines under Node F1 (%), ELR, PA@1 and PA@3. The best result is shown in **bold**.

LLM	Method	HuggingFace				Multimedia			
		Node F1	ELR	PA@1	PA@3	Node F1	ELR	PA@1	PA@3
Qwen2.5-1.5B	BeamSearch	35.59	0.92	0.28	0.02	37.72	0.87	0.22	0.06
	Direct	38.21	0.53	0.28	0.05	42.99	0.45	0.37	0.07
	HuggingGPT	32.68	0.48	0.20	0.03	44.62	0.49	0.38	0.09
	PLaG	29.92	0.49	0.21	0.05	43.24	0.46	0.35	0.07
	GNN4Plan	58.82	**1.00**	0.47	0.16	68.13	**1.00**	0.59	0.36
	GTool	67.21	0.78	0.64	0.25	77.39	0.87	0.60	0.49
	ToolGen	85.98	0.91	0.78	0.54	88.36	0.91	0.84	0.62
	Ours	**86.50**	0.96	**0.79**	**0.57**	**90.52**	0.94	**0.85**	**0.66**
Llama-3.2-3B	BeamSearch	49.18	0.93	0.38	0.08	40.70	0.79	0.26	0.09
	Direct	48.50	0.64	0.39	0.10	54.51	0.76	0.44	0.15
	HuggingGPT	48.55	0.66	0.36	0.10	55.62	0.76	0.44	0.13
	PLaG	48.18	0.67	0.35	0.08	56.58	0.79	0.46	0.14
	GNN4Plan	63.66	**1.00**	0.53	0.23	74.44	**1.00**	0.69	0.48
	GTool	77.81	0.91	0.69	0.41	85.59	0.93	0.79	0.62
	ToolGen	86.75	0.96	0.81	0.57	89.45	0.91	0.86	0.69
	Ours	**89.93**	0.99	**0.84**	**0.65**	**92.97**	0.99	**0.90**	**0.76**
Mistral-7B	BeamSearch	61.78	0.84	0.46	0.14	74.00	0.91	0.57	0.45
	Direct	55.39	0.67	0.45	0.08	65.80	0.64	0.52	0.19
	HuggingGPT	55.29	0.62	0.46	0.08	53.48	0.55	0.41	0.13
	PLaG	57.08	0.67	0.48	0.09	55.62	0.62	0.40	0.13
	GNN4Plan	58.78	**1.00**	0.51	0.18	72.63	**1.00**	0.68	0.44
	GTool	82.85	0.97	0.72	0.49	87.53	0.94	0.80	0.62
	ToolGen	88.04	0.98	0.82	0.62	91.18	0.99	0.88	0.72
	Ours	**89.99**	**1.00**	**0.85**	**0.65**	**92.67**	0.99	**0.89**	**0.74**

conditioned on the user query, completed tool sequence, and previous execution feedback. The executor then validates the predicted tool and generates its arguments. Successful tool executions extend the completed sequence, while failures trigger replanning with updated feedback. We evaluate two settings: a single attempt and up to five prediction attempts per step.

E Additional Results

E.1 Other Metric Results

Table 5 reports the comparison under Tool F1, ELR, PA@1, and PA@3. Tool F1 measures whether the predicted tool set matches the required tools, reflecting tool selection quality. ELR measures whether predicted adjacent tool transitions satisfy the directed dependency constraints in the tool graph. PA@1 and PA@3 measure whether the generated plan matches the ground-truth sequence within the first one and first three steps, respectively.

BeamSearch and GNN4Plan are graph search-based methods and thus achieve relatively high ELR. In particular, GNN4Plan strictly selects each next tool from graph-constrained neighbors, leading to an ELR of 1.00 in all cases. However, its much lower F1 suggest that local dependency validity alone is insufficient for accurate multi-step planning. In contrast, our method consistently achieves the best F1 across all backbones and datasets, while maintaining competitive ELR. Across all methods, PA@1 is consistently higher than PA@3, indicating that predicting the first tool is relatively easier, while maintaining correctness over later tool steps becomes more challenging due to the increasing difficulty of long-horizon dependency modeling. Across both datasets and all three LLM backbones, GRAFT consistently achieves the best performance on all metrics. Compared with the strongest baseline ToolGen, GRAFT further improves Tool F1 and PA, showing that graph-tokenized dependency learning and on-policy distillation improve both tool selection and sequence-level planning accuracy.

Table 6: Link prediction performance (ROC-AUC) under different training objectives. The best results are highlighted in bold.

Dataset	Base	+SFT	+Distill	+Graph
HuggingFace	0.5968	0.7124	0.7212	**0.7373**
Multimedia	0.5858	0.7413	0.7496	**0.7605**

E.2 Analysis of Learned Tool Dependencies

We further investigate whether graph-token edge reconstruction learns tool dependencies beyond standard sequence supervision. While sequence supervision observes only the next tool in each training trajectory, graph reconstruction encourages each tool representation to predict all its outgoing neighbors, providing complementary structural supervision across trajectories.

To assess the learned dependencies, we conduct link prediction experiments on HuggingFace and Multimedia using Qwen2.5-1.5B. Starting from a common base model trained to recognize tool tokens and their associated subtasks, we separately apply supervised fine-tuning (SFT), on-policy distillation, and graph reconstruction. We evaluate the resulting representations using ROC-AUC for directed tool-dependency prediction.

As shown in Table 6, graph reconstruction achieves the highest ROC-AUC on both datasets, outperforming SFT and distillation alone. These results indicate that graph reconstruction more effectively captures directed tool dependencies by providing structural supervision beyond individual training trajectories. Together with the ablation results, this finding suggests that graph reconstruction contributes complementary structural knowledge, while on-policy distillation primarily addresses error accumulation during sequential planning.

E.3 Hyperparameter Sensitivity

Figure 6 analyzes the sensitivity to the distillation weight λ on the Multimedia dataset with Llama-3.2-3B. As λ increases from 0 to 0.7, all three metrics improve consistently, indicating that on-policy distillation benefits both task-level tool selection and dependency-aware sequence generation. However, performance declines when λ becomes too large, suggesting that an overly strong soft distillation loss may dominate the SFT objective and weaken the model's alignment with ground-truth tool trajectories. Overall, moderate distillation is most effective: both insufficient and excessive distillation lead to suboptimal performance.

E.4 Incremental Adaptation to New Tools

GRAFT represents tools as dedicated tokens and therefore requires additional training to incorporate unseen tools. However, its token-based design allows newly introduced tools to be incorporated through incremental updates. To evaluate this capability, we randomly hold out 10% of the tools as newly introduced tools and compare two settings. In the original setting, all tools are trained jointly, whereas in the incremental setting, the model is first trained on existing tools and subsequently updated using trajectories involving the new tools and their graph dependencies.

Table 7: Exact match (%) under joint and incremental training.

Dataset	Original	Incremental
HuggingFace	53.2	53.0
Multimedia	64.0	63.6

As shown in Table 7, incremental training achieves performance comparable to joint training, with EM decreasing by only 0.2 and 0.4 percentage points on HuggingFace and Multimedia, respectively. This indicates that GRAFT can incorporate new tools through incremental updates without retraining all tools jointly. Nevertheless, direct zero-shot generalization to unseen tools remains beyond the scope of this work.

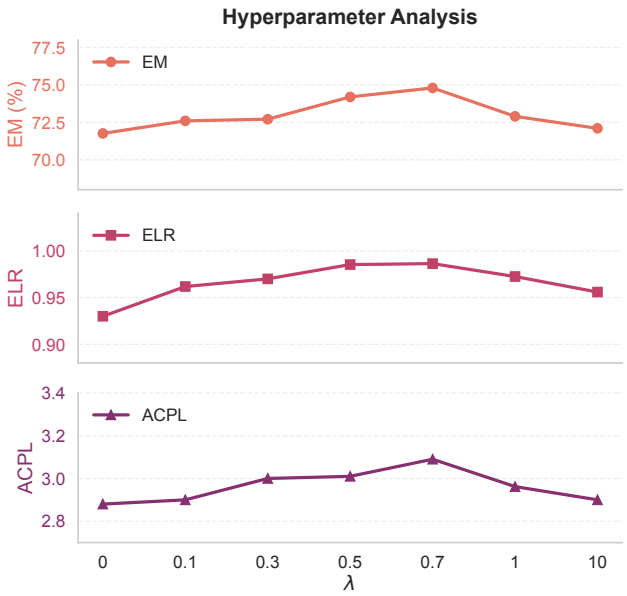

Figure 6: Sensitivity analysis of hyperparameter λ using Llama-3.2-3B on the Multimedia dataset.

E.5 Robustness to Imperfect Tool Graphs

We further examine GRAFT's robustness to imperfect dependency graphs by introducing two types of perturbations: edge corruption, which modifies existing dependencies, and edge removal, which removes valid dependencies. We randomly perturb 20% and 40% of the graph edges and evaluate the resulting planning performance.

Table 8: Exact match (%) under different graph perturbation levels.

Dataset	Edge Corruption			Edge Removal		
	0%	20%	40%	0%	20%	40%
HuggingFace	62.4	62.0	61.4	62.4	62.2	61.8
Multimedia	74.8	74.2	73.4	74.8	74.2	73.6

As shown in Table 8, GRAFT maintains relatively stable performance as graph quality degrades. Even with 40% edge corruption or removal, EM decreases by at most 1.4 percentage points. Edge corruption generally causes greater performance degradation than edge removal, suggesting that incorrect dependencies are more disruptive than missing ones. This robustness may be attributed to joint supervision from ground-truth tool sequences, which complements structural supervision when the dependency graph is imperfect.

F Related Work

We further discuss the distinctions between GRAFT and two closely related paradigms, graph tokenization and tool tokenization, to clarify our contributions to dependency-aware tool planning.

Graph Tokenization for LLMs. Existing graph-tokenization methods, such as RGLM (Zhang et al., 2026), incorporate graph structure into LLM representations to support graph understanding tasks, including node classification and link prediction. These methods typically encode graphs as additional input tokens, preserving explicit graph information during inference. GRAFT instead internalizes the tool graph during training, enabling dependency-aware planning directly from user queries without additional graph inputs. Beyond this difference, graph representation methods generally emphasize node-level semantics and structural information, whereas tool planning requires modeling

the execution order of actions. For instance, RGLM aggregates hidden states of repeated node occurrences into a unified representation. In contrast, GRAFT preserves the contextual representation of each tool occurrence, as the same tool can have different valid successors depending on preceding actions. Its graph-token edge reconstruction objective therefore learns directed dependencies at individual planning steps rather than static node representations.

Tool Tokenization for Planning. Existing tool-tokenization methods, such as ToolGen (Wang et al., 2025), represent tools as dedicated tokens and learn to generate tool sequences through language modeling. While this enables efficient autoregressive tool selection, directed dependencies among tools are not explicitly supervised. Moreover, training on gold trajectories introduces exposure bias, allowing early prediction errors to accumulate during inference. GRAFT addresses these limitations by introducing graph-token edge reconstruction to explicitly learn directed tool dependencies and on-policy tool context distillation to mitigate error accumulation. The latter trains the model on its own generated prefixes while distilling stepwise planning signals from a teacher conditioned on privileged trajectory and subtask information. Together, these components extend tool tokenization toward dependency-aware and robust sequential planning.

G PROMPTS

In on-policy distillation, the student and teacher models receive different prompt inputs. The student prompt contains only the user query and asks the model to directly predict the ordered tool-token sequence. The teacher prompt additionally receives a reference solution constructed from the ground-truth reasoning subtasks and gold tool sequence.

Student prompt template.

System:
Predict the exact ordered tool token sequence needed to complete the user query. A query may require multiple tools. Output only tool tokens in order, with no spaces, commas, or other separators between them. Do not explain, do not restate the query, and do not output any extra words or punctuation.

User:
Query: `{query}`

Teacher prompt template.

System:
Predict the exact ordered tool token sequence needed to complete the user query. A query may require multiple tools. Output only tool tokens in order, with no spaces, commas, or other separators between them. Do not explain, do not restate the query, and do not output any extra words or punctuation. You are given a reference solution to help determine the final tool sequence.

User:
Query: `{query}`
Here is a reference solution:
`{reference_solution}`
Based on the reference solution above, output the exact correct tool sequence only.

Example from the multimedia dataset. For the multimedia sample with ID 25717055, the user query is:

I have a video file named 'example.mp4', please extract the audio track from it, then change the pitch to a higher tone according to my instruction 'Increase pitch'. After that, create a colorful spectrogram image from the modified audio.

The teacher prompt is:

System:
Predict the exact ordered tool token sequence needed to complete the user query. A query may require multiple tools. Output only tool tokens in order, with no spaces, commas, or other separators between them. Do not explain, do not restate the query, and do not output any extra words or punctuation. You are given a reference solution to help determine the final tool sequence.

User:
Query: I have a video file named 'example.mp4', please extract the audio track from it, then change the pitch to a higher tone according to my instruction 'Increase pitch'. After that, create a colorful spectrogram image from the modified audio.

Here is a reference solution:
Step 1: Extract the audio track from the input video. Use tool: `<Video_to_Audio>`
Step 2: Modify the extracted audio's characteristics according to the user's instruction. Use tool: `<Voice_Changer>`
Step 3: Generate a visual representation of the modified audio track. Use tool: `<Audio_to_Image>`
Step 4: Add color to the visual representation using deep learning techniques. Use tool: `<Image_Colorizer>`

Correct tool sequence:
`<Video_to_Audio><Voice_Changer><Audio_to_Image><Image_Colorizer>`

Based on the reference solution above, output the exact correct tool sequence only.

The expected target sequence for both prompts is:

```
<Video_to_Audio><Voice_Changer><Audio_to_Image><Image_Colorizer>
```