Graph-O-Planner: Injecting Graph Neural Tool Embeddings into LLMs for Efficient and Accurate Task Execution

Anonymous ACL submission

Abstract

Recent advancements in Large Language Models (LLMs) have enabled the development of AI agents capable of multi-step reasoning. However, deploying these agents in real-world applications requires planners that adapt to domain-specific tools and workflows, where traditional prompting frameworks often struggle to accurately represent available functional dependencies. To address this gap, we propose **Graph-O-Planner**, a novel graph-learning method that explicitly encodes tool relationships and execution sequences into LLM planning. Our approach constructs graph embeddings of available tools, enabling agents to dynamically map dependencies while minimizing context window overload. Evaluations across multiple benchmarks, including UltraTool and Task Bench, demonstrate that Graph-O-Planner achieves up to **68% higher and 60% higher performance** with our approach, compared to state-of-the-art hybrid graph+LLM based planners and LLM-finetuned planners respectively, while significantly reducing any hallucinations in LLM generation. The method's tool knowledge compression further reduces inference latency by **20%**, validating its effectiveness in resource-constrained environments and making it more compatible for real-life practical deployment. We release our code here[1].

1 Introduction

The advent of Large Language Model (LLM)-powered agents marks a paradigm shift in artificial intelligence, with transformative potential for real-world applications ranging from autonomous robotics to precision medicine. Early implementations like HuggingGPT (Shen et al., 2023a) demonstrate problem-solving flexibility in controlled benchmarks, and agents such as Voyager (Wang et al., 2023a) showcase emergent strategic reasoning in gaming environments.

[1]https://anonymous.4open.science/r/Graph-O-Planner-16B3

Effective planning modules with precise tool alignment are essential for developing practical AI agentic systems in both consumer and industrial applications. Recent advances leverage prompting strategies to decompose complex tasks: Wei et al. (2022); Yu et al. (2025) pioneered Chain-of-Thought(CoT) reasoning through sequential step generation, while Wang et al. (2023b) introduced plan-and-solve prompting for systematic task decomposition. Yao et al. (2024) later expanded these concepts with tree-based reasoning architectures. A parallel research trajectory has focused on translating these reasoning structures into executable tool operations (Schick et al., 2023; Shen et al., 2023b; Singh et al., 2023; Song et al., 2023).

These methods, however, are purely prompt-based and use LLMs deployed on the cloud. This centralization exposes sensitive data to privacy risks, incurs network-dependent latency, and complicates regulatory compliance. Another focus has been locally deployable solutions (Erdogan et al., 2024; Wu et al., 2024) where the model is finetuned for planning and tool-calling. Locally deploying preserves data privacy and minimizes inference delay, yet small to mid-sized models introduce their own constraints. We describe these constraints:

1. ***Challenge 1.*** **Tool Hallucination/ Tool Grounding** significantly drives down task-sequence determination, especially with a bigger set of tools/sub-tasks (CodeLlama13B and GPT3.5 Turbo see 60% hallucination in edge prediction with a set of 260 sub-tasks) (Wu et al., 2024).
2. ***Challenge 2.*** **Fixed Context**: The limited context window of LLMs prevents them from handling a large number of tools, often forcing tools to truncate from the context.
3. ***Challenge 3.*** **Token Overhead and Latency Issues**: Tool usage in current systems involves repeatedly injecting detailed descriptions of

tools into prompts. This token increase leads to quadratic increase in memory requirements due to attention, impacting latency during inference

4. ***Challenge 4.*** **Shallow GNN+LLM Fusion**: Existing GNN+LLM hybrid lacks deep structural grounding, limiting planning quality and scalability.

We draw upon these insights to propose a framework that effectively uses GNN for tool information injection during task-planning. Building upon the initial work done by Wu et al. (2024), we propose **Graph-O-Planner**, that represents tools as nodes in a dynamic graph, leveraging Graph Neural Networks (GNNs) to capture functional dependencies and enable scalable tool representation. By fusing GNN with Large Language Models (LLMs), we create a more grounded and hallucination-resistant decision-making process while reducing latency and memory requirements.

Our main contributions are summarized as:

- To the best of our knowledge, Graph-O-Planner is the first attempt to deeply integrate multilevel interaction of a GNN with LLMs for task planning. This setup uses the language understanding skills of LLMs in conjunction with the effective information propagation capability of GNNs.
- We show the efficacy of adding GNN based interaction by comparing against LLM-only models. Graph-O-Planner improves upon the finetuned LLM performance by nearly 60% while beating hybrid LLM+graph-based baselines by 35%.
- By incorporating dependency-aware tool graph reduces tool hallucination metrics by 13%, ensuring that model generates correct tool names instead of similar looking tool names. It further reduces the context tokens which is a major bottleneck for local deployment and speeds up inference by 1.25x.

2 Related Work

2.1 GNN-based Learning

Recent works employ task graphs to model interlinked sub-tasks and align LLMs with tool-relevant information. GNNs demonstrate strong capabilities for complex decision-making (Khalil et al., 2017; Xu et al., 2019; Dudzik and Veličković, 2022). Graph-based QA systems have successfully leveraged external KGs for factual queries through LLMs, using relevant subgraph retrieval (Luo et al., 2023; Zhang et al., 2023, 2024; Yao et al., 2022). GNNs have also been shown to enhance LLMs' ability to model textual relational structures. While transformers (Guo et al., 2025; Chung et al., 2022; Dubey et al., 2024; Yang et al., 2024) dominate sequential processing, they falter with long-range dependencies like tool relationships. GNNs overcome this by representing text as graphs (Huang et al., 2019; Pham et al., 2023; Zhu et al., 2021), improving LLMs' understanding of local and global patterns (Wu et al., 2024, 2021).

2.2 Tool Graph-based Planning

Taking inspiration from KG based learning(Liu et al., 2021; Wang et al., 2021; Ye et al., 2022; Tena Cucala et al., 2022; Chen et al., 2020), training GNN using task graphs has become a powerful tool for task planning, enabling the modeling of complex dependencies between tasks, resources, and constraints. They have been applied across diverse domains, including MoE task planning (Zhou et al., 2022; Cai et al., 2024; Li et al., 2025) and multi-agent coordination (Wu et al., 2023; Chan et al., 2023; Talebirad and Nadiri, 2023; Nascimento et al., 2023) by facilitating decentralized decision making.

3 Preliminaries

In this section, we describe tool graphs, which we define as dynamically changing graphs. Subsequent paragraphs provide a detailed explanation of tool graphs, including their structure, tool descriptions & input/output formats.

Tool Graph: Let $G = (V, E, A, T, X)$ where, V is a set of tool nodes, E corresponds to edges between node embedding ($v_i \rightarrow v_j$) if output of V_i can be fed to V_j. A denotes the edge weight matrix between pair of nodes, such that $A[i, j] \in (0, 1]$, if v_i,$v_j \in$ V and $(v_i, v_j) = e_{ij} \in E$, and 0 otherwise. Tool information is defined as $T = \{n, d, i, o\}_{(k=1)}^{|V|}$, where n is k^{th} tool name, d is k^{th} tool description, i and o corresponds to k^{th} input and output format of tool respectively. Thus, $X = Emb(T)$, where Emb is the embedding function. $X = \{x_i\}_{(k=1)}^{|V|}$ contains feature embedding of tool's information for each $v_i \in V$.

Planning task definition: A task query Q can be decomposed into sub-tasks $S = s_1, s_2 \ldots s_n$, such that each sub-task s_i can be completed by an unique tool v_i. The objective is to construct

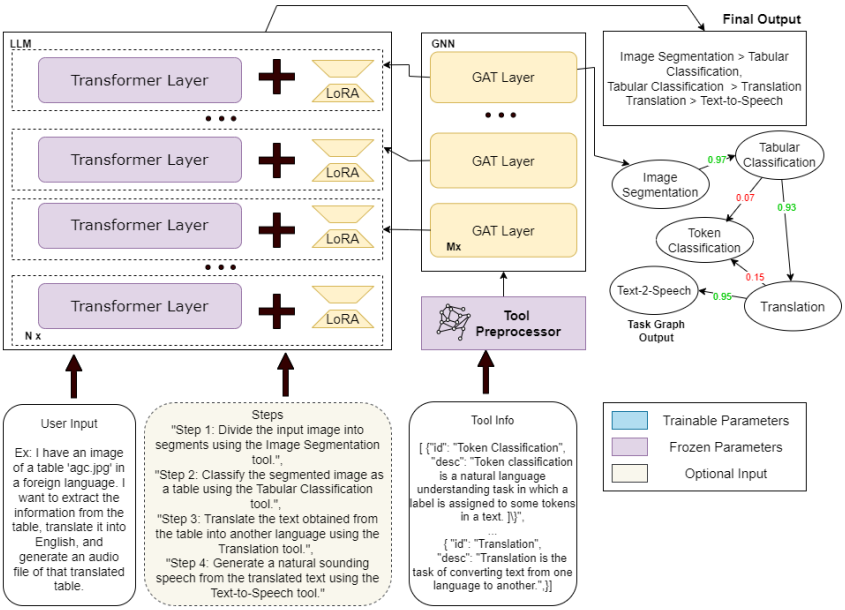


Figure 1: Overall schematic block diagram of Graph-O-Planner including data flow, key component and internal interactions. Yellow layers in the diagram represent trainable parameters, while purple indicates frozen parameters.

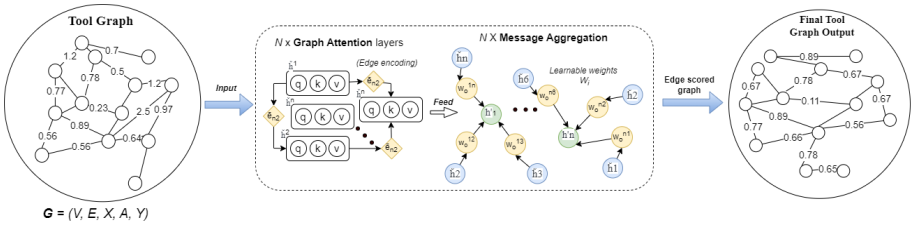


Figure 2: Creation of dependency-aware tool embeddings from GNN

a Directed Acyclic Graph (G) that represents the sequence of processes to solve query Q. This can be formally represented as:

$$DAG = (v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_P)$$

showcasing the sequence of tools that needs to be executed in order to solve the query Q. Each tool v_i is connected to the next tool v_{i+1} through a dependency edge.

4 Graph-O-Planner

We introduce Graph-O-Planner, a novel graph-infused learning framework for task planning, to effectively align task steps to the available tools. Figure 1 illustrates the pipeline of our proposed approach. We first convert the target dataset into an aligned tool-graph, incorporating the tool name, description, required inputs and generated output (Section 4.1). This is then passed to a GNN to create its aligned node embeddings & edge scores at each layer, as defined in Section 3. Finally, a layer-wise knowledge injection method is utilized to inject knowledge from GNN layers to corresponding LLM layers (Section 4.3). This allows the LLM to effectively map the subtask to the correct tool sequence. The rough decomposed plan (attained from any global LLM) is processed sequentially through the LLM layers with injected information from the GNN to generate a sequence of tools as a DAG to be executed. With Graph-O-Planner, both LLMs and GNNs are trained in alignment with injected knowledge, to allow the model to build an understanding of how a subtask relates to a particular sequence of nodes in the available tool graph.

4.1 Tool Graph Creation

We first encode dataset's Tool information using a text encoder. We chose ModernBERT Large (Warner et al., 2024) which outperforms other encoder-based language models on Natural Language Understanding (NLU) and retrieval tasks. Using the encoder model we generate a tool's em-

bedding representation x.

$$x_i = Emb(ToolName, ToolDesc., ToolInputs, ToolOutput) \quad (1)$$

where x_i is i^{th} tool embedding and Emb is embedding model. For a given set of task steps, some nodes and edges are more semantically relevant than others. To effectively model this by leveraging core semantic information, the edge connections between the tools are scored. Alignment with the requirements of the decomposed task steps is obtained by using a bilinear layer to estimate the relevance score for each edge given the embedding of the sub-task. Finally, these embeddings are normalized to increase computational efficiency. Motivated by Jang et al. (2017), we used Gumbel softmax approach to model the output as soft labels with a stop gradient mechanism to address the problem of gradient propagation of hard labels during backward pass. The node and the scored edge embeddings together comprise the required Tool Graph (defined in Section 3).

4.2 Dependency-Aware Tool Graph

The scored tool graph is then passed through a graph network, to obtain its graphical embedding representation at every layer. This is pictorially shown in Figure 2. We use Graph Attention (GAT) layers for encoding the tool info graph representations $X = \{x_1 x_2, \ldots . x_n\}$, via iterative convolutional operations between neighboring nodes of the graph network.

1. **Edge Encoding**
   Given a graph, $G = (V, E, A, T, X)$ (refer section 3), with node features $h_i^l \in R^d$ (initially $h_i^0 = x_i, x_i \in X$), and auxiliary node features $\phi \in R^{|K| \times d}$, our goal is to learn node representations that captures structural neighborhood patterns and edge semantic relationship. For each edge $e_{ij} \in E$, we obtain its encoding ϵ_{ij} as:

   $$\epsilon_{ij} = f_{edge}(\phi_{e_{ij}} \oplus \tau_{k_i} \oplus \tau_{k_j}) \quad (2)$$

   where f_{edge} is a multi-layer MLP, $\phi_{e_{ij}}$ represents the Gumbel Softmax of edge e_{ij}, and τ_{k_i} and τ_{k_j} represents the Gumbel Softmax of node k_i and k_j respectively, with $\oplus$ as the concatenate function.

2. **Multi-Head Heterogeneous Attention**
   Each edge's representation ϵ_{ij} is then utilized to transform node representations using a multi-head heterogeneous attention, M. Specifically, for each graph node, we obtain the normalized attention of the pre-head computations. The node embedding $\tilde{h}_i$ is obtained by concatenating ($\oplus$) node features and its features of its neighboring edges $N(i)$, to better capture neighborhood patterns every iteration:

   $$\tilde{h}_i = h_i \oplus \bigoplus_{j \in N(i)} \epsilon_{ij} \quad (3)$$

   The pre-heads of Key, Query and Value for the k-th node are thus computed as:

   $$Q_i^k = W_Q^k \tilde{h}_i \quad (4)$$

   $$K_i^k = W_K^k \tilde{h}_i \quad (5)$$

   $$Q_i^k = W_V^k \tilde{h}_i \quad (6)$$

   which is then normalized based on the degree of the node:

   $$\alpha_{ij}^k = \frac{d_j}{Z_i} * \exp \frac{\langle q_j^k, k_i^k \rangle}{\sqrt{d}} \quad (7)$$

   where $Z_i = \sum_{j \in N(i)} d_j * \exp \frac{\langle q_j^k, k_i^k \rangle}{\sqrt{d}}$ and $d_i = |N(i)|$ is the out degree of node j.

3. **Message Aggregation**
   The multi head attention output are aggregated through a two stage process to synthesize neighborhood information. First for each attention head k, messages from neighboring nodes are weighted by their normalized coefficients α_{ij}^k producing head-specific representations

   $$m_i^k = \sum_{j \in N(i)} (\alpha_{ij}^k v_{ij}^k) \quad (8)$$

   These head embeddings are then concatenated across all k heads and linearly projected to the original dimension d using learnable weights W_o to obtain M - edge aware attention. This ensures structured fusion of heterogeneous relation patterns. The hybrid approach retains structural information while allowing the model to learn incremental feature updates, balancing neighborhood influence with node-specific characteristics.

Finally, we use a three phase computation over the edge-attention M to obtain the embeddings for every time-stamp:

$$H^{l+1} = MLP(Agg(M(H^l, E, \epsilon))) \quad (9)$$

We use an additional node, master node which connects to all the nodes in the graph to pool the representations of all the nodes and obtain representation of the graph.

4.3 Deep Fusion of GNN with LLM

Finally, we inject the GNN knowledge into LLM layers for effective tool-aligned subtask creation. For a chosen subset of LLM layers, the standard multi-head self-attention and feed-forward layer is extended to fuse the modalities between text and graph domain. The pooled representation of the graph embeddings i.e. representation of the master node H from equation 9 is fused by concatenating ($\oplus$) with LLM decoder module to obtain an intermediate representation I_c formulated as follows:

$$I_c = \{\theta_{key} \oplus \psi_{key}, \theta_{query} \oplus \psi_{query}, \theta_{value} \oplus \psi_{value}\} \quad (10)$$

where ψ_i are aligned values obtained from GNN after passing through two Feed Forward Layers. The injection pipeline is aligned with the LoRA layers of the LLM to avoid re-training of the complete LLM. As the knowledge injection methodology only needs aligned layer output fusion, it is independent of the architecture of the LLM in use.

5 Experimental Setup and Results

In this section, we describe the training setup and datasets used. We also enumerate a detailed set of experiments to evaluate the performance of Graph-O-Planner against state-of-the-art graph-based baselines and LLMs finetuned on four open-source datasets. For our primary pipeline, we choose Flan-T5-XL (3B) as the LLM with a LoRA of rank 8 and alpha 16. All models are trained using Adam Optimizer, with the batch size of 32 and learning rate of $1e-4$ and $3e-4$ for the LLM and GNN respectively. All models are trained using A-6000 GPU in a Pytorch framework and CUDA 12.6. We provide specific hardware and software versions information in appendix A.2

5.1 Datasets

We train and evaluate our model on four open source datasets, UltraTool (Huang et al., 2024), HuggingFace, Multimedia and TaskBench-Daily Life (Shen et al., 2023b). Each dataset is converted to a Tool Graph as described in Section 4.1. Detailed dataset description has been provided in the Appendix A.6.

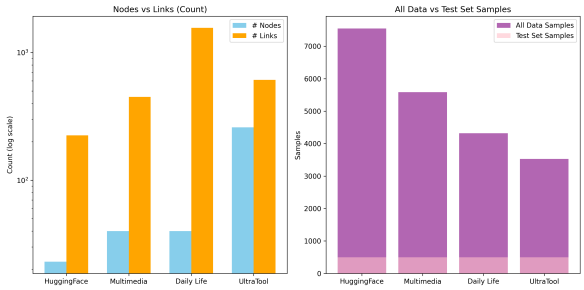


Figure 3: Statistics of tool graph and samples of datasets.

5.2 Results & Analysis

We present our model performance across accuracy, hallucination and latency metrics, detailed in the section below. In all experiments, our base pipeline uses FlanT5-XL as LLM and a GAT GNN. We compare our model performance against various existing SOTA models. We also demonstrate the impact of a graph based knowledge injection by comparing the performance against traditional LLM only approaches, shallow GNN interaction methods and reasoning only based settings. We report tool and sequence performances along with hallucination and model latency for all four target datasets in below sections.

5.2.1 Tool and Sequence Detection

The primary requirement of a tool aligned planner is to ensure that the model is able to correctly pick from the provided set of available tools. A planner that provides a "somewhat-correct" output is not scalable in a real-life application. We thus measure node prediction F1-score as a primary metric of performance evaluation. Node-F1 score is estimated as the correctness of predicted tool nodes required to complete a given task.

Another crucial performance metric is to ensure that the correct nodes are detected in the correct sequence. Thus, the edges between various task nodes and their relative sequence is of utmost importance. Accordingly, we design Edge-F1 score which compares the predicted links with the ground truth edges, using the tool network topology populated adjacency matrices. We present the edge and node F1 algorithm in Appendix A.9.

For F1 scores, the set of predicted nodes/edges and set of ground truth nodes/edges is used for each

Method/Dataset	Huggingface			Ultratool			Multimedia			Dailylife		
	Acc.	Edge-F1	Node-F1	Acc.	Edge-F1	Node-F1	Acc.	Edge-F1	Node-F1	Acc.	Edge-F1	Node-F1
Mistral7b + GraphToken*	20.08	32.55	62.15	64.57	68.57	85.63	35.06	74.57	63.71	69.42	73.57	92.50
e5-335M + GraphSAGE*	24.74	40.60	66.52	37.56	42.36	71.23	62.37	70.25	88.86	86.57	85.80	97.42
e5-335M + GCN*	16.36	30.23	60.06	46.23	32.54	63.54	61.25	50.76	73.34	75.49	65.49	86.39
e5-335M + GIN*	23.31	39.37	65.77	57.84	37.25	69.25	62.55	69.84	88.74	84.49	82.65	93.36
e5-335M + GAT*	23.39	40.66	66.44	60.25	40.23	80.23	63.69	70.24	88.90	89.91	89.32	97.21
e5-335M + GraphTransformer*	24.54	41.64	66.92	46.42	57.68	76.38	NA	70.24	88.90	NA	85.80	97.43
Qwen 2.5 Coder 3B‡	81.32	68.23	87.77	74.80	78.57	91.87	21.48	26.96	56.94	89.86	91.83	**99.24**
Deepseek R1 1.5B ‡	79.28	80.22	84.33	85.28	89.27	87.33	82.25	83.24	81.27	87.23	88.23	86.24
Flan T5 XL‡	49.0	63.48	74.11	73.8	73.87	83.87	43.40	58.25	58.25	63.87	48.77	66.45
RAG + Qwen 2.5 Coder 3B (not fintuned)ψ	32.8	42.86	40.09	56.8	46.84	38.36	30	32.89	36.65	30.13	32.56	36.36
RAG + Qwen 2.5 Coder 3B (finetuned) ψ	75.00	76.33	78.26	95.56	95.87	96.56	75.6	76.82	78.76	96.8	**96.00**	96.41
Graph-O-Planner(Ours)	**82.6**	**89.73**	**97.36**	**97.39**	**97.73**	**99.19**	**92.0**	**93.55**	**99.13**	**96.81**	95.59	99.82

Table 1: Accuracy, Edge-F1 and Node-F1 scores across all four datasets. All result are in (%) with the best bold and runner-up underlined. * indicates hybrid LLM+graph based approaches and that the numbers are sourced from Wu et al. (2024). ‡ indicates LLM-only models finetuned for each dataset. ψ indicated Retrieval Augmented Generation (RAG) based results. The information about the baseline models are described in Appendix A.7.

sample i among N total samples.

$$\text{Precision} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\text{Predicted}_i \cap \text{Ground Truth}_i|}{|\text{Predicted}_i|} \tag{11}$$

$$\text{Recall} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\text{Predicted}_i \cap \text{Ground Truth}_i|}{|\text{Ground Truth}_i|} \tag{12}$$

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

We also evaluate the success rate at the task level using the Accuracy (Acc) metric. The Accuracy metric is defined as:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} I(\text{F1}_i = 1) \tag{14}$$

Here, $I(\text{F1}_i = 1)$ is an indicator function that assigns a value of 1 if the F1 score for a given task i is perfect (i.e., all nodes are correctly predicted), and 0 otherwise. This binary evaluation allows us to assess the model's ability to accurately predict all nodes in a task.

Table 1 shows our performance across four different datasets, which included a variety of task types and complexities. This suggests that our method is flexible and can be applied to different problems and datasets. Specifically, across the more voluminous **ultratool dataset** (with 260+ tools), Graph-O-Planner shows a **33% improvement** over older graph interaction methods and 8% improvement against Deepseek, **$\tilde{54}$% improvement** againt retrieval+prompt based settings and **6.4% accuracy improvemen**t against retrieval+finetune. even when the pipeline using a lesser competent llm (Flan-t5). This improvement is even more significant when considering the more convoluted **huggingface dataset**, with Graph-O-Planner seeing nearly **68% improvement** over previous models as presented in Table 1.

5.2.2 Tool Hallucination Reduction

We also demonstrate the efficacy of integrating GNNs in reducing hallucination in LLMs. We use two hallucination metrics: micro hallucination and macro hallucination. These metrics are designed to quantify the extent of hallucination in the predicted sets of nodes compared to the ground truth sets.

Let N be the total number of samples, P_i be the predicted set of nodes for the i^{th} sample, and let V be the set of valid nodes.

Micro hallucination calculates the fraction of predicted nodes that are absent in the ground truth, averaged over all samples, represented as:

$$\text{Micro Hallucination} = \frac{1}{N}\sum_{i=1}^{N}\frac{|P_i \setminus V|}{|P_i|} \tag{15}$$

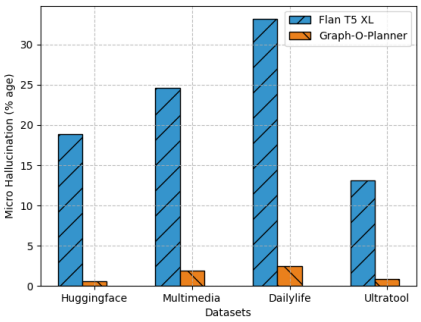
where $|P_i \setminus V|$ represents the number of nodes in P_i that are not in V, essentially the number of hallucinated nodes in the prediction.

Macro hallucination checks if any of the predicted nodes are absent from the ground truth and assigns 1 if at least one node is absent, 0 otherwise, and then averages over all samples:

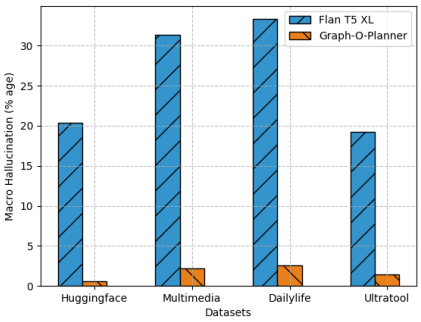
$$\text{Macro Hallucination} = \frac{1}{N}\sum_{i=1}^{N} I(P_i \setminus V \neq \emptyset) \tag{16}$$

where $I(P_i \setminus V \neq \emptyset)$ equals 1 if there are any nodes in P_i not in V (i.e., $P_i \setminus V$ is not empty), and 0 otherwise.

As shown in figure 4a and 4b, our proposed GNN-based approach achieves a substantial reduction in prediction hallucination, with a **13% decrease in incorrect edge predictions**.



(a) Micro hallucination in Flan T5 XL vs Graph-O-Planner



(b) Macro hallucination in Flan T5 XL vs Graph-O-Planner

Figure 4: Tool Prediction Hallucination in Flan T5 XL vs Graph-O-Planner

The results suggest that the GNN's ability to model complex structural relationships between tasks is instrumental in mitigating hallucination. By representing task sequences as graphs and leveraging the strengths of GNNs, we can better capture the nuances of task dependencies and generate more accurate and contextually relevant responses.

5.2.3 Model Latency

A complementary benefit of our proposed approach is the reduction in input context size during both training and inference as presented in more detail in Appendix A.5. In the current literature, training Large Language Model (LLM) planners pass all the tool information, including name, description, input/output format, directly to the prompt resulting into longer context with a large number of tools, as seen in the Ultratool dataset. Even with context length of 8192, we observed a spill-over of input tokens. Since we inject tool knowledge as tool embeddings directly into the Graph Neural Network (GNN) layers, while the LLM focuses on the input query and the steps needed to execute the task. This makes it a more scalable and reliable solution for handling complex task sequences and a large number of tools.

This also significantly reduces inference time latency, as shown in Figure 5, making it more suitable for real-time applications due to the reduced input size, computational requirements, and focused input context. Our method achieves faster inference than encoder-decoder, reasoning, and prompt-based LLMs. Compact inputs and rapid generation enable more efficient, reliable outputs with fewer hallucinations.

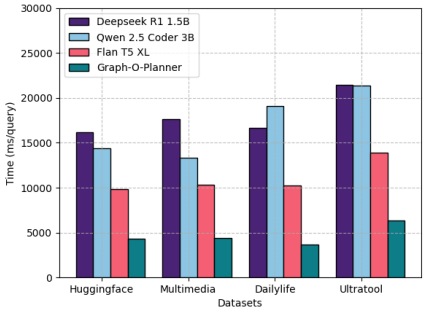


Figure 5: Inference time comparison of different models. The bar plot illustrates the average inference time (in ms) for each model, highlighting performance variations.

5.3 Ablation Study

1. Effect of Number of GNN Layers. We further conducted a study to evaluate the impact of varying GNN layer depths (2, 5, 10) across tool graph datasets of different structures and reported the result in Table 3. We observe that for smaller tool graphs such as huggingface, multimedia and dailylife, lesser layers of GNN suffice. Increasing the number of layers can cause a negative effect due to over-smoothing effect. For UltraTool with 260 tools, 5 GNN layers perform the best. For the Ultratool graph increasing depth from 2 to 5 layers yielded statistically significant improvements in edge prediction (edge-F1: 95.79% $\rightarrow$ 97.73%), accompanied by a marginal yet consistent gain in node prediction (node-F1: 98.83% $\rightarrow$ 99.19%). Performance degradation at 10 layers (edge-F1: 97.26%) aligns with established phenomena of over-smoothing in excessively deep architectures. Conversely, datasets with dense connections but less number of nodes such as dailylife and hug-

Method/Dataset	Huggingface		Ultratool		Multimedia		Dailylife	
	Edge-F1	Node-F1	Edge-F1	Node-F1	Edge-F1	Node-F1	Edge-F1	Node-F1
Message Passing	88.25	96.21	97.56	98.88	**93.55**	97.11	95.06	97.51
GCN	**90.15**	**97.50**	**97.73**	98.99	92.80	99.05	94.96	98.85
SGC	89.73	97.23	97.20	98.99	91.43	99.00	95.43	**99.82**
GAT	89.73	97.36	**97.73**	**99.19**	91.49	**99.13**	**95.59**	**99.82**

Table 2: Effect of different GNN networks used in Graph-O-Planner, evaluated on Node -f1 and Edge-f1. All result are in (%) with the best bold and runner-up underlined.

Layers/Dataset	Huggingface		Ultratool		Multimedia		Dailylife	
	Edge-F1	Node-F1	Edge-F1	Node-F1	Edge-F1	Node-F1	Edge-F1	Node-F1
2	**89.73**	97.12	95.79	98.83	93.52	98.87	**95.59**	**99.84**
5	88.74	97.36	**97.73**	**99.19**	93.18	**99.13**	92.78	99.71
10	88.8	**98.96**	97.26	98.72	**93.55**	99.04	94.54	99.82

Table 3: Performance comparison across datasets by GNN layer depth. For each dataset best performing epoch in 15 epochs is reported in above table. Best numbers were highlighted in bold.

gingface exhibited peak edge-F1 scores at 2 layers (95.59% and 89.73%, respectively), with deeper configurations inducing performance declines (dailylife edge-F1: 92.78% at 5 layers). Node prediction metrics remained stable across depths for these datasets (node-f1 $\Delta < 0.2\%$), suggesting limited utility of extended neighborhood aggregation in locally cohesive graphs.

2. Comparison of different GNN approaches. We experiment with different GNN approaches, namely Message Passing Neural Networks (MPNNs), Simplified Graph Convolution (SGC), standard Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT). The experimental results presented (Table 2) demonstrate that GAT outperforms other GNNs for Ultratool, Multimedia and Dailylife and second best performance on Ultratool.

3. Other Design Choices - In addition to the type and size of GNN used, Graph-O-Planner reuses some of modules and techniques that have been proven as state-of-art in earlier works. As discussed earlier, we use frozen ModernBERT to encode the task graph information. It provides strong semantic representations while maintaining a lightweight and efficient architecture in comparison to previous work such as BERT and DeBERTa (discussed in Warner et al. (2024)). Our decision to freeze the encoder ensures that the entire learning capacity is focused on the GNN and LLM layers, keeping the pipeline modular and resource-efficient. Additionally, instead of normal softmax, we choose to use Gumbel Softmax. The recent work of (Zhang et al., 2024) validates the increased efficacy of using gumbel softmax for graphical architectures. As these choices are derivative usages of proven SOTA techniques, we do not include detailed insights into variations caused by these design choices. We provide more details of multi-layer and multi-level interactions in Graph-O-Planner in Appendix A.8.

6 Conclusion

In this work, we propose Graph-O-Planner, a graph-based task selection method for generalized agent planning. Traditional prompt based methods of LLM based agent creation are hindered by concerns related to ever increasing tool context length, hallucinations and inductive biases. We propose using a GNN based network to effectively embed the information of the available tools, and use a knowledge-injection methodology in Graph-O-Planner to empower the LLM to map the sub-tasks to the appropriate tool sequence. Our method enables a more modular and flexible architecture by decoupling tool knowledge from the input prompt and injecting it into GNN layers, allowing for seamless integration of new tools and task sequences complementing the LLM for better performance as presented in Appendix A.3. We evaluate our model across 4 open-source datasets, comparing with multiple existing SOTA methodologies. As noted in results, we beat existing benchmarks by significant levels, enforcing the efficacy of the proposed model. The impact of tool information compression is also seen in inference latency, with a 1.25x improvement in inference speed. To the best of our knowledge, proposed work is the first of its kind, exploring a deeply integrated GNN-LLM framework for effective task planning.

Limitations

Despite encouraging performance, this work is only the beginning of exploring the GNN-LLM interaction in-depth. We also want to extend the pipeline to make the planning truly generalizable across any unseen tool-graph, task type, ambiguous data and erroreneous formats/descriptions. In real-life application scenarios, the available tools and user preferences will be constantly evolving and varied from person-to-person. A truly intelligent agent should be able to effectively generalize across all such interactions without the need of any fine-tuning or adaption. We aim to look into more details on these in future works.

References

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Xiaojun Chen, Shengbin Jia, and Yang Xiang. 2020. A review: Knowledge reasoning over knowledge graph. *Expert systems with applications*, 141:112948.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *arXiv preprint*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Andrew J Dudzik and Petar Veličković. 2022. Graph neural networks are dynamic programmers. *Advances in neural information processing systems*, 35:20635–20647.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2024. Tinyagent: Function calling at the edge. *Preprint*, arXiv:2409.00608.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. 2019. Text level graph neural network for text classification. *arXiv preprint arXiv:1910.02356*.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *Preprint*, arXiv:2401.17167.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. *Preprint*, arXiv:1611.01144.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30.

Jiachen Li, Xinyao Wang, Sijie Zhu, Chia-Wen Kuo, Lu Xu, Fan Chen, Jitesh Jain, Humphrey Shi, and Longyin Wen. 2025. Cumo: Scaling multimodal llm with co-upcycled mixture-of-experts. *Advances in Neural Information Processing Systems*, 37:131224–131246.

Shuwen Liu, Bernardo Grau, Ian Horrocks, and Egor Kostylev. 2021. Indigo: Gnn-based inductive knowledge graph completion using pair-wise encoding. *Advances in Neural Information Processing Systems*, 34:2034–2045.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2023. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061*.

Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Self-adaptive large language model (llm)-based multiagent systems. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 104–109. IEEE.

Phu Pham, Loan TT Nguyen, Witold Pedrycz, and Bay Vo. 2023. Deep learning, graph-based text representation and classification: a survey, perspectives and challenges. *Artificial Intelligence Review*, 56(6):4893–4927.

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. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023a. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Preprint*, arXiv:2303.17580.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2023b. Taskbench: Benchmarking large language models for task automation. *arXiv preprint arXiv:2311.18760*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*.

DJ Tena Cucala, B Cuenca Grau, Egor V Kostylev, and Boris Motik. 2022. Explainable gnn-based models over knowledge graphs.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *Preprint*, arXiv:2305.16291.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023b. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Yu Wang, Zhiwei Liu, Ziwei Fan, Lichao Sun, and Philip S Yu. 2021. Dskreg: Differentiable sampling on knowledge graph for recommendation with relational gnn. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3513–3517.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *Preprint*, arXiv:2412.13663.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. 2021. Graph neural networks for natural language processing: A survey. *CoRR*, abs/2106.06090.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, et al. 2024. Can graph learning improve task planning? *arXiv preprint arXiv:2405.19119*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2019. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*.

Peng Xu, Xinchi Chen, Xiaofei Ma, Zhiheng Huang, and Bing Xiang. 2021. Contrastive document representation learning with graph attention networks. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3874–3884, Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Yunzhi Yao, Shaohan Huang, Li Dong, Furu Wei, Huajun Chen, and Ningyu Zhang. 2022. Kformer: Knowledge injection in transformer feed-forward layers. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 131–143. Springer.

Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. 2022. A comprehensive survey of graph neural networks for knowledge graphs. *IEEE Access*, 10:75729–75741.

Junwei Yu, Yepeng Ding, and Hiroyuki Sato. 2025. Dyntaskmas: A dynamic task graph-driven framework for asynchronous and parallel llm-based multi-agent systems. *Preprint*, arXiv:2503.07675.

Qinggang Zhang, Junnan Dong, Hao Chen, Xiao Huang, Daochen Zha, and Zailiang Yu. 2023. Knowgpt: Black-box knowledge injection for large language models. *arXiv preprint arXiv:2312.06185*.

Yu Zhang, Kehai Chen, Xuefeng Bai, Quanjiang Guo, Min Zhang, et al. 2024. Question-guided knowledge graph re-scoring and injection for knowledge graph question answering. *arXiv preprint arXiv:2410.01401*.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. 2022. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114.

Jason Zhu, Yanling Cui, Yuming Liu, Hao Sun, Xue Li, Markus Pelger, Tianqi Yang, Liangjie Zhang, Ruofei Zhang, and Huasha Zhao. 2021. Textgnn: Improving text encoder via graph neural network in sponsored search. In *Proceedings of the Web Conference 2021*, pages 2848–2857.

A Appendix

A.1 Notations

The symbolic notations used in the paper are summarized in Table 4.

A.2 Implementation Details

Detailed information on the experimental setup section 5.

Hardware. All the models are trained using PyTorch 2.3.1 framework in Python 3.11 conda environment. Ubuntu server equipped with four 48GB Nvidia-RTX A6000 with driver version 560.35.03 and CUDA 12.6 are utilized to perform the study.

Hybrid GNN+LLM Baselines: We used the repo by Wu et al. (2024) to reproduce the results.

LLM-Only: We used the Transformer library's Trainer with LoRA similar to Graph-O-Planner to train the piepeline.

RAG Baselines We utilize open-source ChromaDB framework which uses a hierarchical configuration architecture, employing the all-MiniLM-L6-v2 transformer model as its default text embedding engine, which produces 384-dimensional semantic vectors optimized for cosine similarity computations through PyTorch-based inference. Input text undergoes preprocessing via whitespace normalization and lexical truncation at 256 tokens prior to vectorization, with document batches processed at 32-sample granularity to balance memory efficiency and throughput. The system implements an in-memory HNSW (Hierarchical Navigable Small World) index with empirically tuned parameters (ef_construction=200, M=16) to optimize approximate nearest neighbor search latency. **Graph-O-Planner**: **GNN.** The dimensions of GNN module is converted from 1024 embedding size to 200, which can be modified as per user's choice and a variable number of layer of GNN modules between 5 and 7 both inclusive with a dropout of 0.18 applied within each consecutive layer. **Training.** All models are trained using Adam optimizer. Learning rate of LLM module and GNN module is kept 1e-4 and 3e-4 respectively, with batch size of 32. We choose FLAN-T5-XL(3B) model as LLM for Low-Rank Adaption (LoRA) training with rank 8 and alpha 16. The maximum token length across tokenizer is kept variable as per the requirement of dataset.

A.3 Additional aid to LLM with Graph-O-Planner

In the figure 6, 7, 8, 9, 10, 11 shown Dev, test and loss for only LLM approach in comparison with Graph-O-Planner approach. From figure 6 and 7, it is observed that Graph-O-Planner approach with the help of tool embeddings from GNN through multilevel interaction learn meaningful insights about the tool graph and surpass 80% edge-f1 in merely 5 epochs and settles at >90% after 10 epochs. While in T5 only training approach we find that the overall edge-f1 cannot surpass 60% even after 30 epochs as shown in figure 8 and 11.

We also observed a much reliable training with Graph-O-Planner approach. As seen from Figure 10, the loss curve much cohesively justifies the overall loss when compared with the improvement seen from test eval curve. While for T5 only approch in Figure 11 we can observe that the loss drops drastically till 40th epoch, but soon reaches a stagnant curve, however as can be observed from 9, the test edge-f1 has a lot of scope for improvement. From these result we can come to conclusion that Knowledge fusion between LLM and GNN can lead to benefits listed below:

- **Unified Task Perspective.** In our approach, the LLM can be directly leveraged to produce outputs for multiple tasks. For varying tasks, it can either operate in a masked mode using precomputed embeddings—eliminating the need for re-computing task graph. This underscores our core contribution: the LLM+GNN functions as a flexible, "plug-and-play" mod-

Notations	Definition
G, V, E	Tool graph with set of nodes V and edges E
T_i, A_i	Features embeddings i-th tool in graph G, A represents adjacency matrix
T	Tool information in graph G
Q_i	i-th query of dataset
$S_i = s_1, \ldots s_n$	Subtasks of query Q_i
$Emb(.)$	Embedding function representing tool info in graph space for GNN training
$h^{(l)}$	GNN node representation at step l
$M(.)$	Edge aware attention function
$1_{\phi(e_{ij})}$	Encoded edge features obtained after applying Gumbel softmax transform
$1_{\tau(v_i)}$	Encoded node features obtained after applying Gumbel softmax transform
(e_{ij})	Edge representation obtained after concatenation of encoded edge and node features
q_j^k, k_j^k, v_{ij}^k	Query, Key and Value projections of k-th GNN node
α_{ij}^k	Normalized node attention based on out degree of k-th GNN node
$N(i)$	Out degree of i-th node
m_i^k	Head Specific attention of k-th GNN node after message passing to neighbors
$h_i^{'}$	Edge representation of GNN node after message passing
I_c	Intermediate LLM+GNN interaction layer
$\theta_{key}, \theta_{query}, \theta_{value}$	Key, Query and Value obtained from LLM decoder
$\phi_{key}, \phi_{query}, \phi_{value}$	Key, Query and Value obtained from GNN decoder

Table 4: Notation table in Graph-O-Planner

ule, significantly improving efficiency and performance over conventional large language model (LLM)-only approaches by storing precomputed task graph embeddings and lower context length requirement.

- **Integrated Fusion Strategy.** Our fusion strategy facilitates concurrent information propagation from tool graphs (hidden embeddings) and (task-specific output vectors) to the query input. This enables structured knowledge injection and task-specific adaptation, making our LLM + Graph Network (GNN) paradigm superior to LLM-only models, which often struggle with structural reasoning and compositional generalization. By leveraging graph representations, our approach effectively captures relational dependencies, improving both adaptability and interpretability across tasks.

A.4 Necessity of GNN integration

Large Language Model (LLM)-only planners require all tool information—including tool descriptions and capabilities to be included within the input prompt at both training and inference time. This quickly becomes infeasible due to the limited context window of current LLMs **(Context Overflow Problem)**. When the tool-related content exceeds the model's context length, critical parts of the input are truncated, leading to incomplete information available for planning.

Even though models with longer context windows are emerging, they come with practical trade-offs:

- **Latency and Memory Bottlenecks:** The latency increases nearly quadratically as attention has $(O(n^2))$ time complexity. So, when the prompt size increases linearly, the latency increases quadratically.

- **Windowed Attention Limitations:** Many long-context LLMs rely on windowed attention mechanisms, which suffer from "window sink" issues that prevent the model from capturing long-range dependencies across windows, as demonstrated in earlier works (Xu et al., 2021; Xiao et al., 2024).

To address these limitations, Graph Neural Networks (GNNs) encodes and pools tool information into a fixed-size vector representation, independent of the number of tools. GNNs are well-suited to model complex and structured data, capturing long-range and interdependent relationships between tools efficiently.

A.5 How Graph-O-Planner overcomes context overflow and latency problem

The integration of Large Language Models (LLMs) with Graph Neural Networks (GNNs) presents a compelling advancement over LLM-only approaches for task planning, particularly in scenarios involving an extensive set of tools with complex specifications. Traditional LLM-based methods rely on tokenization to encode tool-related information, which inherently limits scalability due to increasing sequence lengths and associated computational costs. In our experiments, we observed that models such as Qwen struggle when provided with a large number of tools and their descriptions in Ultratool dataset. The excessive tokenization required to process tool details not only constrains the model's ability to handle more elaborate queries but also results in increased latency and memory overhead, making real-time task planning inefficient.

Context Overflow mitigation. Our proposed Graph-O-Planner framework mitigates these limitations by encoding tool information as embeddings within a graph structure, rather than representing them as lengthy text sequences. By leveraging GNNs to store and propagate tool-specific embeddings, we significantly reduce tokenization overhead, enabling the LLM to allocate more of its token budget toward processing complex queries rather than repetitive tool descriptions. This structured approach enhances efficiency by shifting the burden of tool representation from token-based encoding to a graph-based framework, leading to more scalable and interpretable reasoning over available tools. Furthermore, the graph structure inherently captures relational dependencies between tools, facilitating a more structured understanding of tool applicability and interoperability.

Reduced Model Latency. Beyond tokenization efficiency, the incorporation of GNNs also enhances inference speed by caching for repetitive Task info embeddings on first fly. In LLM-only approaches, each query requires reprocessing tool descriptions, leading to redundant computation. In contrast, our GNN-enhanced model precomputes and stores tool embeddings, allowing for direct retrieval and propagation of relevant tool information without unnecessary recomputation. This not only accelerates inference but also ensures that the model retains a more contextually enriched and persistent representation of tools across different task planning queries. By leveraging message-passing mechanisms within the GNN, our approach ensures efficient information flow, reducing the reliance on autoregressive decoding for tool-related reasoning.

By encoding tool knowledge in a structured graph representation, we achieve a dual advantage: reducing tokenization demands while improving inference efficiency. This allows for handling more complex and multi-step task planning scenarios, where an LLM alone would struggle due to token constraints and redundant processing. Our findings demonstrate that integrating structured graph-based reasoning with LLMs enables more effective tool selection, faster response times, and improved scalability, making it a superior approach for real-world Agentic planning applications.

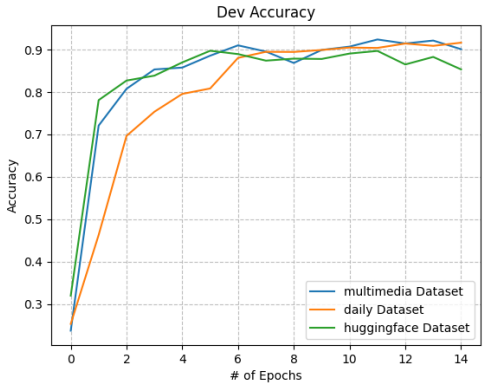


Figure 6: Dev edge-f1 of Graph-O-Planner

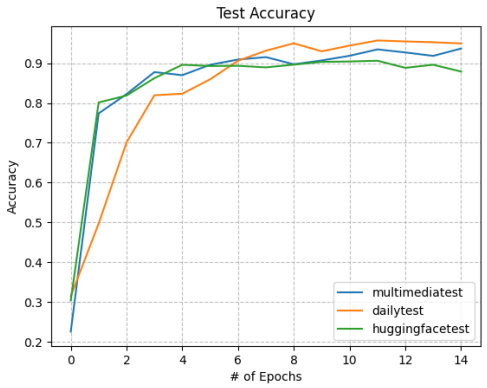


Figure 7: Test edge-f1 of Graph-O-Planner

A.6 Dataset

In this section we will deep dive into dataset mentioned in section 5.1.

Ultratool. It consist of 260 tools with 3527 task and steps samples. On average each sample's

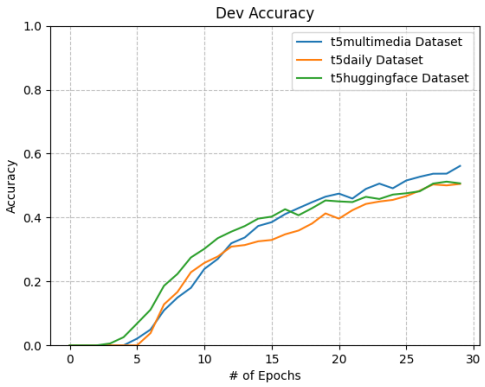


Figure 8: Dev edge-f1 of Flan T5 XL

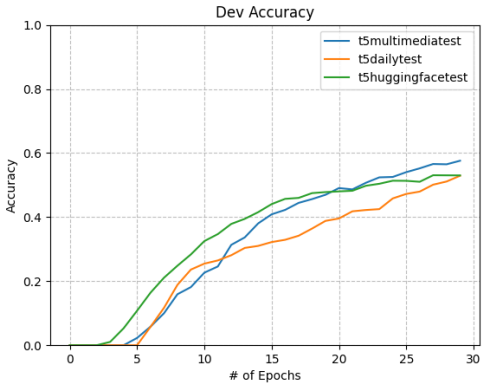


Figure 9: Test edge-f1 of Flan T5 XL

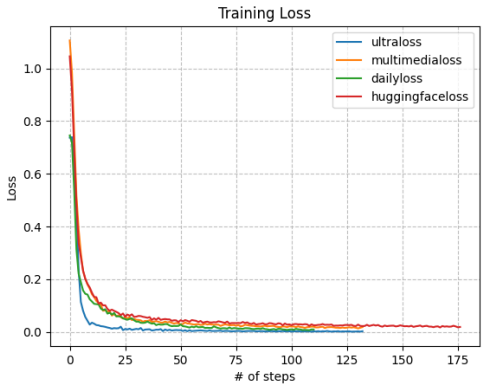


Figure 10: Loss graph of Graph-O-Planner after training for 15 epochs

plan include 2.42 tool callings. All samples within ultratool have at least one tool calling. In particular 64.24% of samples consist of two tool calling and rest consist of multiple tool calling. Each sample contains at least two tool calls.

Huggingface. It consist of 40 tools with 7546 training samples. The tools comprise of hugging face hosted models fine-tuned to perform various

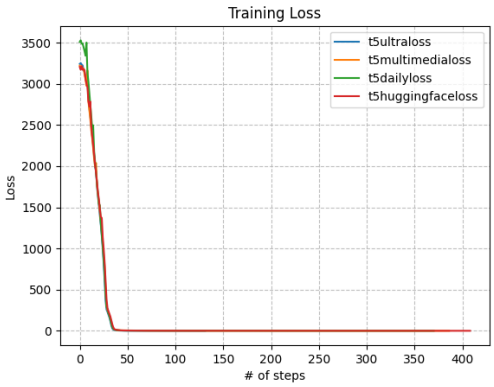


Figure 11: Loss of Flan T5 XL while training for 30 Epochs

downstream tasks. The overall dataset requires 20177 tool callings with an average of 3.28 arguments per tool call. The dataset consists of 40.64% of samples with single tool calling.

Multimedia. This dataset also consist of 40 unique tools with 5584 training samples. The tools comprises of generic multimedia tools like 'Video-to-Audio', 'Audio-Splicer' etc. It consist of 15860 distinct tool calls with 3.49 arguments per tool call. Out of 5584 samples 36.48% of samples requires only single tool calling.

Dailylife. The dataset contains 40 distinct tools with 4320 samples out of which 1258 samples contains single tool calls. In the dataset it requires on average of 3.09 tool calls per sample with average of 4.95 arguments per tool call. The tool consist of general tool present in most virtual assistants like 'book_hotel' , 'book_flight' etc.

Next we show sample input data fed from these dataset.

Huggingface

```
text{'id': '57993067',
 'seed': 513420,
 'n_tools': 1,
 'sampled_nodes':
 [{'task': 'Object Detection',
   'input-type': ['image'],
   'output-type': ['text']}],
 'sampled_links': [],
 'user_request': "I need
 to identifyand label objects
 in the provided image
 'example.jpg'.",
 'task_steps': [
    'Step 1: Use Object
    Detection to identify
```

```
    objects in the image
    and label them.'
 ],
 'task_nodes': [{
    'task': 'Object Detection',
    'arguments':['example.jpg']
    }
 ],
 'task_links': [],
 'type': 'single'}
```

Multimedia

```
{'id': '16097613',
 'seed': 154967,
 'n_tools': 3,
 'sampled_nodes':
   [{'input-type': ['audio',
   'text'],
   'output-type': ['audio'],
   'task': 'Audio Effects'},
  {'input-type': ['audio'],
   'output-type': ['audio'],
   'task': 'Audio Noise
   Reduction'},
  {'input-type': ['video'],
   'output-type': ['audio'],
   'task': 'Video-to-Audio'}],
 'sampled_links': [
    {'source': 'Audio Noise
    Reduction',
     'target': 'Audio Effects'},
    {'source': 'Video-to-Audio',
     'target': 'Audio Noise
     Reduction'}],
 'user_request': 'I have a video
 file example.mp4,  and I want to
 extract its audio track, reduce
 background
 noise, and then add a reverb
 effect.
 Please provide the
 processed audio file.',
 'task_steps': [
    'Extract audio from the given
     video file',
    'Reduce noise from the
    extracted audio',
     'Apply audio effects to the
     noise-reduced
      audio according to user
      instructions'],
 'task_nodes': [
   {'task': 'Audio Effects',
    'arguments':
        ['<node-1>', 'reverb']},
   {'task': 'Audio Noise
   Reduction',
    'arguments': ['<node-2>']},
   {'task': 'Video-to-Audio',
  'arguments':['example.mp4']}],
 'task_links': [
   {'source': 'Audio
   Noise Reduction',
    'target': 'Audio Effects'},
   {'source': 'Video-to-Audio',
      'target': 'Audio Noise
      Reduction'}],
 'type': 'chain'}
```

Dailylife

```
{'id': '13590101',
 'seed': 283717,
 'n_tools': 1,
 'sampled_nodes': [
 {'task': 'play_movie_by_title',
   'arguments':[{'name':'title',
     'type': 'string',
     'desc': 'The title of the
     movie to play'}]}],
 'sampled_links': [],
 'user_request': "I want to
 watch the movie titled
 'Example Movie'",
 'task_steps': ["Step 1: Call
 play_movie_by_title API
 with title: 'Example Movie'"],
 'task_nodes': [
   {'arguments': [
    {'name': 'title',
    'value': 'Example Movie'}],
'task':'play_movie_by_title'}],
 'task_links': [],
 'type': 'single'}
```

Ultratool

```
{'id': '3186',
 'user_request': 'I need to
 cancel the single alarm set
 for 8:00 AM today, and change
 the daily alarm from 7:00 AM
 to 6:30 AM every day.\n',
 'task_steps': [
'Step 1 Call clock_alarm_cancel
to cancel the alarm set for
```

```
8:00 AM today',
'Step 2 Call clock_alarm_change
to change the daily
alarm from 7:00 AM to 6:30 AM
every day'],
 'task_nodes': [
    {'task':
    'clock_alarm_cancel'},
    {'task':
    'clock_alarm_change'}],
 'task_links': [
 {'source':
 'clock_alarm_cancel',
   'target':
   'clock_alarm_change'}],
 'n_tools': 2,
 'type': 'chain'}
```

A.7 Baselines

In this appendix section, we present the details of baselines shown in Table 1.

- **Graph Token.** A method that introduces a global virtual token to GNNs allowing improved global information aggregation and better graph-level representations.
- **GraphSAGE.** A GNN that learns node embeddings by sampling and aggregating information from a nodes' neighborhood, enabling scalable learning on large graphs.
- **GCN(Graph Convolutional Network).** A fundamental GNN model that extends convolutional operations to graph structure by propagating and aggregating node features using adjacency-based weight metrics
- **GAT(Graph Attention Network).** A GNN model that incorporates attention mechanism to assign different importance weights to neighboring nodes, improving feature aggregation adaptively.
- **GIN(Graph Isomophism Network).** A powerful GNN variant designed to be as expressive as the Weisfeiler-Lehman graph isomorphism test, using MLP-based neighborhood aggregation.
- **Deepseek R1.** DeepSeek-R1 is a reasoning model that achieves performance comparable to OpenAI-o1 across math, code, and reasoning tasks, and is open-sourced along with its distilled dense models to support the research community. DeepSeek-R1 is developed through a pipeline that incorporates reinforcement learning and supervised fine-tuning, and its reasoning patterns can be distilled into smaller models, resulting in better performance on benchmarks. like Multi head Latent attention.
- **Qwen 2.5 Coder.** Qwen2.5-Coder is a large language model series with six mainstream model sizes, offering improved code generation, reasoning, and fixing capabilities. It has become the state-of-the-art open-source codeLLM, matching the coding abilities of GPT-4o with enhanced coding capabilities and long-context support up to 128K tokens.

A.8 More Detailed study

1. Effect of multilayer interaction between LLM and GNN. We investigate the contribution of multilayer interaction, experiments using single-layer interaction of SGC, GCN, GAT, and GraphSAGE without multi-layer feature injection showed that this component significantly boosts performance (up to 30%) by aggregating richer contextual information across layers as observed in Table 5. Our experiments demonstrate the critical role of hierarchical feature aggregation through direct comparison with shallow graph convolution baselines. Single-layer variants (SGC, GCN, GAT, GraphSAGE) exhibit substantially inferior performance across all datasets—Huggingface edge-F1 reaches just 43.09% (GraphSAGE) versus our 89.73%, while Multimedia node-F1 plateaus at 75.51% (GraphSAGE) versus our 99.13%. The performance differential is most pronounced in edge prediction tasks, where our method achieves relative improvements of 108.3% (Huggingface edge-F1: 89.73 vs. 43.09) and 44.6% (Dailylife edge-F1: 95.59 vs. 66.57) over the strongest shallow baselines. Even node classification, traditionally less depth-sensitive, shows absolute gains of 29.85% (Huggingface) and 23.62% (Multimedia) compared to single-layer counterparts, empirically validating the necessity of cross-layer information fusion.

2. Architectural implication of Multilevel fusion. The stark performance margins (Δedge-F1 > 46% across all datasets) reveal fundamental limitations of shallow interaction paradigms. While shallow methods like SGC achieve computational efficiency through layer truncation, they

Method/Dataset	Huggingface		Multimedia		Dailylife	
	Node-F1	Edge-F1	Node-F1	Edge-F1	Node-F1	Edge-F1
SGC	67.43	42.08	74.07	49.90	87.13	66.49
GCN	66.54	40.74	73.34	50.76	86.39	65.49
GAT	66.77	40.74	73.36	50.20	86.39	65.49
GraphSAGE	68.12	43.09	75.51	52.94	87.51	66.57
Graph-O-Planner (ours)	**97.36**	**89.73**	**99.13**	**91.49**	**99.82**	**95.59**

Table 5: Comparison of shallow interaction between LLM and various GNN settings vs. ours

forfeit the ability to capture hierarchical dependencies—evidenced by Huggingface's edge prediction collapse to 42.08% (SGC) versus our 89.73%. Our multi-layer architecture addresses this through deliberate feature injection across depths, enabling progressive refinement of both local and global graph patterns. This is particularly crucial for complex edge prediction tasks, where shallow models lack the representational capacity to resolve indirect relationships (e.g., Multimedia edge-F1: 52.94% vs. 91.49%). The consistent outperformance (minimum Δnode-F1: 12.31% in Dailylife) across diverse graph types further substantiates multi-layer interaction as a generalizable design principle rather than a dataset-specific optimization.

3. Proof of effectiveness of GNN.

In this section, we will theoretically prove that using Graph-O-Planner can significantly improve the LLM generation performance. Assume X_l as input tokens to the LLM and G as input tool graph features to the GNN, Y represents target output tool labels. We introduce a dependency function $DF(.)$ that quantifies the dependency between input labels X and Y, which reflects the performance of LLM. By introducing tool-graph knowledge into GNN, we can impactfully improve model performance in predicting labels Y as $DF(X_l, G, Y) \geq DF(X_l, Y)$. The following outlines the derivation:

$$
\begin{aligned}
&DF(X_l, G, Y) - DF(X_l, Y) \\
&= \sum_{X_l,G,Y} p(X_l, G, Y) \log\left(\frac{p(X_l, G, Y)}{p(X, G)p(Y)}\right) \\
&\quad - \sum_{X_l,Y} p(X_l, Y) \log\left(\frac{p(X_l, Y)}{p(X)p(Y)}\right) \\
&= \sum_{X_l,G,Y} p(X_l, G, Y) \log\left(\frac{p(X_l, G, Y)}{p(X, G)p(Y)}\right) \\
&\quad - \sum_{X_l,G,Y} p(X_l, G, Y) \log\left(\frac{p(X_l, Y)}{p(X)p(Y)}\right) \\
&= \sum_{X_l,G,Y} p(X_l, G, Y) \log\left(\frac{p(X_l, G, Y)}{p(X, G)p(Y)} \cdot \frac{p(X_l)p(Y)}{p(X_l, Y)}\right) \\
&= \sum_{X_l,G,Y} p(X_l, G, Y) \log\left(\frac{p(X_l, G, Y)}{p(G|X)p(Y)p(X_l, Y)}\right) \\
&= \sum_{X_l,G,Y} p(Y, G|X)p(X) \log\left(\frac{p(Y, G|X)}{p(G|X)p(Y)p(Y|X)}\right)
\end{aligned}
$$

4. Training time Computation analysis In this section we provide the computation comparison of LLM only v/s Graph-O-Planner approach. During the experiments, the number of trainable parameters remains constant across all the dataset. We observed that for Graph-O-Planner approach the time required for each epoch ranges between **23-32 minutes**, while for LLM only approach the time taken to complete one epoch ranges between **150-180 minutes**. From these results we infer that our approach is much faster and promising than other SOTA methods.

5. Rationale for usage of ModernBERT as text-embedder for encoding tool info. ModernBERT-Large has been used to generate initial embeddings for each node/tool description, which are then passed as input features to the GNN. The recent work Warner et al. (2024) shows ModernBERT-Large performing better than its predecessors on NLU tasks. It is also finetuned on using triplet networks (i.e. NLI tasks), thus a suitable choice to work with similarity, clustering and retrieval tasks. Morever, it improves upon BERT

and RoBERTa by combining masked and permuted language modelling, capturing global dependencies better other approaches like MiniLM or MPNet.

A.9 Algorithms

In this section we provide detailed description of all the major algorithms explained in section 5.2.1.

Algorithm 1: Node F1. The algorithm takes two list as input. Lines 1-2 contains ground truth tools L and predicted tools R which is given to the function in Line 3-19 to calculate node f1. In more detail Lines 4-5 computes the length of lists L and R and stores them in gt_len and $pred\_len$ respectively. Lines 8-10 stores unique tool names from ground truth in set gt_tools and respectively for predicted tools in $pred\_tools$ in Lines 11-13.

Finally, the node f1 is calculated in Line 14-17 by taking precision and recall and storing in variables p and r and then computing $node\_f1$.

Algorithm 2: Edge F1. The algorithm takes two list. Lines 1-2 contains ground truth tools L and predicted tools R which is given to the function in Line 3-19 to calculate edge f1. Lines 4-5 computes the length of lists L and R and stores them in gt_len and $pred\_len$ respectively. Lines 7-13 takes every tool link present in predicted links and checks if the tool is present in ground truth links. If the tools is present, the counter of common links c_links is increased by 1. Finally in Line 14, 15 precision and recall for links are computed and then $edge\_f1$ is calculated in Line 17.

Finally, the node accuracy is calculated in Line 14-15 as length of intersection set between predicted tool names and ground truth tool names set over length of gt_tools.

Algorithm 1: Node-F1

1: $L \leftarrow [l_1, l_2, \ldots, l_n]$ ▷ List of ground truth tool pairs. $r_i : (tool_{i1}, tool_{i2})$
2: $R \leftarrow [r_1, r_2, ..., r_n]$ ▷ List of predicted tool pairs. $r_i : (tool_{j1}, tool_{j2})$
3: **procedure** NODE F1(L, R)
4: $gt\_len \leftarrow length\ of L$
5: $pred\_len \leftarrow length\ of R$
6: $gt\_tools \leftarrow \{\}$
7: $pred\_tools \leftarrow \{\}$
8: **for** $i = 1$ to gt_len **do**
9: $gt\_tools = gt\_nodes \cup L[i][0] \cup L[i][1]$
10: **end for**
11: **for** $i = 1$ to $pred\_len$ **do**
12: $pred\_tools = gt\_nodes \cup R[i][0] \cup R[i][1]$
13: **end for**
14: $c\_tools \leftarrow pred\_tools \cap gt\_tools$
15: $p \leftarrow \frac{length(c\_tools)}{length(pred\_tools)}$
16: $r \leftarrow \frac{length(c\_tools)}{length(gt\_tools)}$
17: $node\_f1 = \frac{2*p*r}{p+r+\alpha}$
18: **return** $node\_f1$
19: **end procedure**

Algorithm 2: Edge-F1

1: $L \leftarrow [l_1, l_2, \ldots, l_n]$ ▷ List of ground truth tool pairs. $r_i : (tool_{i1}, tool_{i2})$
2: $R \leftarrow [r_1, r_2, ..., r_n]$ ▷ List of predicted tool pairs. $r_i : (tool_{j1}, tool_{j2})$
3: **procedure** EDGE F1(L, R)
4: $gt\_len \leftarrow length\ of L$
5: $pred\_len \leftarrow length\ of R$
6: $c\_links \leftarrow 0$
7: **for** $i = 1$ to $pred\_len$ **do**
8: **for** $j = 1$ to gt_len **do**
9: **if** $R[i] == L[j]$ **then**
10: $c\_links \leftarrow c\_links + 1$
11: **end if**
12: **end for**
13: **end for**
14: $c\_tools \leftarrow pred\_tools \cap gt\_tools$
15: $p \leftarrow \frac{length(c\_tools)}{length(pred\_len)}$
16: $r \leftarrow \frac{length(c\_tools)}{length(gt\_len)}$
17: $edge\_f1 \leftarrow \frac{c\_links}{gt\_len}$
18: **return** $edge\_f1$
19: **end procedure**

Algorithm 3: Pseudocode for Graph-O-Planner

1: **Input:** Task query Q, tool metadata $T = \{(n_i, d_i, i_i, o_i)\}_{i=1}^{V}$
2: **Output:** Tool execution DAG $DAG = (v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_P)$
3: **procedure** GRAPHOPLANNER(Q, T)
4: **// Step 1: Tool Graph Construction**
5: **for** each tool $t_i \in T$ **do**
6: Compute embedding: $x_i = \text{Emb}(n_i, d_i, i_i, o_i)$ (Eq. 1)
7: **end for**

Form tool graph $G = (V, E, A, T, X)$ with nodes V, edges E, and embeddings $X = \{x_i\}$

// Step 2: Edge Encoding

for each edge $(v_i, v_j) \in E$ **do**

Compute edge representation:

$$\epsilon_{ij} = f_{\text{edge}}(\phi_{e_{ij}} \oplus \tau_{k_i} \oplus \tau_{k_j}) \quad \text{(Eq. 2)}$$

end for

// Step 3: Attention-Enhanced Graph Convolution

for each GNN layer l **do**

for each node v_i **do**

$h_i^{(l)} = x_i$ if $l = 0$

Aggregate neighboring edge features:

$$\tilde{h}_i = h_i^{(l)} \oplus \bigoplus_{j \in N(i)} \epsilon_{ij} \quad \text{(Eq. 3)}$$

Compute attention heads:

$$Q_i^k = W_Q^k \tilde{h}_i, \quad K_i^k = W_K^k \tilde{h}_i, \quad V_i^k = W_V^k \tilde{h}_i \quad \text{(Eqs. 4–6)}$$

Normalize attention:

$$\alpha_{ij}^k = \frac{d_j}{Z_i} \cdot \exp\left(\frac{\langle Q_j^k, K_i^k \rangle}{\sqrt{d}}\right) \quad \text{(Eq. 7)}$$

Aggregate messages:

$$m_i^k = \sum_{j \in N(i)} \alpha_{ij}^k V_{ij}^k \quad \text{(Eq. 8)}$$

end for

Fuse heads and update node embeddings:

$$H^{(l+1)} = \text{MLP}(\text{Agg}(M(H^{(l)}, E, \epsilon))) \quad \text{(Eq. 9)}$$

end for

// Step 4: Inject GNN Embeddings into LLM

For selected LLM layers:

$$I_c = \{\theta_{key} \oplus \psi_{key}, \theta_{query} \oplus \psi_{query}, \theta_{value} \oplus \psi_{value}\}$$

(Eq. 10)

where $\psi_\cdot$ are GNN projections passed through FFNs

// Step 5: Decode Tool DAG

Use LLM (augmented with injected GNN knowledge) to decode:

$$DAG = (v_1 \to v_2 \to \ldots \to v_P)$$

end procedure