Graph-O-Planner: Graph-Based Task Execution for Multi Agent Planning

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 upto **68% higher and 60% higher performance** with our approach, compared to state-of-the-art graph based planners and LLM-finetuned planners respectively, while significantly reducing any hallucination in LLM generation. The method's tool knowledge compression further reduces inference latency by **50%**, validating its effectiveness in resource-constrained environments and making it more compatible for real-life practical deployment. We release our code here.

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) demonstrates problem-solving flexibility in controlled benchmarks, and agents such as Voyager (Wang et al., 2023a) showcase emergent strategic reasoning in gaming environments.

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) pioneered chain-of-thought 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).

All these methods however are purely prompt based, and hence are encumbered by some core concerns due to the base LLM performance - demonstrated in the recent work by (Wu et al., 2024). They specifically note the following:

1. **Hallucination** significantly drives down task-sequence determination, especially with a bigger set of tools/sub-tasks (CodeLlama13B and GPT3.5T see 60% hallucination in edge prediction with a set of 260 sub-tasks).
2. Next-token autoregression tends to prioritize **frequent patterns** over optimal solutions, due to its reliance on statistical distributions learned from the training data. These limitations persist even in tool-specialized frameworks (Schick et al., 2023), suggesting inherent challenges in reconciling stochastic text generation with structured system requirements.

In addition to above, there are other practical constraints when considering real-life industry applications. In a connected tech ecosystem, the set of tools and functions can range in thousands, with multiple tools often executing the same sub-task. Not only does providing long prompts **increase LLM generation latency** significantly, it also makes it more **prone to erroneous choices**. These concerns have led to exploring ways in which

the planning can be made better, instead of just focusing on effective prompting techniques.

Recent works have explored the use of task graphs to effectively model the interlinked subtasks and provide the relevant information to the LLM for tool alignment. Graph Neural Networks (GNNs) have shown a lot of proficiency in handling complex decision making problems (Khalil et al., 2017; Xu et al., 2019; Dudzik and Veličković, 2022). Graph based question answering has also seen a lot of advancements - wherein, LLMs are tasked to answer factual questions using retrieved relevant knowledge from an external knowledge graph (KG). While initial work focused primarily on retrieving the relevant subgraph from KG as foundation for LLM reasoning (Luo et al., 2023; Zhang et al., 2023; Sun et al., 2023), recent works have explored building a deeper information injection bridge to reduce errors propagated due to fault sub-graph determination (Zhang et al., 2024; Yao et al., 2022; Liu et al., 2024).

We draw upon these past insights to propose a module that effectively uses GNN for tool information injection while task-planning. Building upon the initial work done by Wu et al. (2024), we propose **Graph-O-Planner** that closely integrates the interaction between LLM and GNN layers. We craft an attention enhanced GNN based multi-layer interaction that allows for a layered learning of graph representations. This ensures that subgraphs with distracting hard positive cases do not cause the LLM to focus on irrelevant information *(challenge 1)*. We also propose a retrieval free graph mechanism, indicating that the model can be applied to a diverse range and size of planning tasks and datasets without being encumbered by increased latency with humongous prompt sizes *(challenge 2)*. To validate our hypothesis, we convert various open-source planning datasets into tool graphs, where tool name, description and its input-output format are used to create graph embedding. We conduct extensive experimentation to show that proposed interaction methodology is independent of the type of LLM used, and can be integrated with any existing architecture to note significant planning alignment improvements with reduced hallucinations.

Our main contributions are summarized as follows:

- To the best of our knowledge, Graph-O-Planner is the first attempt to integrate multi-level interaction of a custom 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 comprehensively assess Graph-O-Planner across multiple datasets, evaluating accuracy, hallucination as well as latency. The experimental results confirms the effectiveness of our approach, showing remarkable improvements over state-of-the-art baselines in both the scenarios by huge margin.
- We prove the efficacy of adding GNN based interaction by comparing against base LLM-only models for the same task prediction. Graph-O-Planner improves upon the finetuned LLM performance by nearly 50% while beating most graph-based benchmarks by 35%.

2 Related Work

2.1 GNN-based Learning

Graph Neural Networks (GNNs) contribute significantly to enhancing the performance of Large language models by enabling the modeling of complex relational structures in textual data. While transformer-based models (Guo et al., 2025; Chung et al., 2022; Dubey et al., 2024; Yang et al., 2024) excel at capturing sequential dependencies, they struggle with long-range syntactic or semantic dependencies like tool dependency info and relation based knowledge. GNNs address this by representing text as graphs, where nodes represent linguistic elements, and edges capture their relationships(Huang et al., 2019; Pham et al., 2023; Zhu et al., 2021). Through message-passing, GNNs enable the propagation of contextual information, enriching LLMs' understanding of both local and global dependencies (Wu et al., 2024, 2021b). This integration improves performance on various NLP tasks, including machine translation, task planning.

Recent research (Wu et al., 2021a; Hu et al., 2020; Yang et al., 2021) has also shown that GNNs enhance multi-hop reasoning knowledge graph handling and can scale more efficiently. The synergy between GNNs and transformers had led to breakthrough in tasks requiring deep semantic understanding and reasoning. Moreover, GNNs help LLMs handle noisy data and improve generalization, making them a promising approach for advanced NLP models (Yasunaga et al., 2021; Mavromatis and Karypis, 2024; Chen et al., 2024; Peng et al., 2024).

2.2 Tool Graph-based Planning

Taking inspiration from Knowledge graph 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 graph have become a powerful tool for task planning, enabling the modeling of complex dependencies between tasks, resources, and constraints through graph structure. 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 where they excel in dynamically adjusting to changing conditions. In multi-agent system (Wu et al., 2023; Chan et al., 2023; Talebirad and Nadiri, 2023; Nascimento et al., 2023), they facilitate decentralized decision making, GNNs have been used to enhance scheduling efficiency in combinatorial optimization problems. Leveraging the success of graph knowledge, we propose to infuse graph based knowledge using trained GNNs to support LLMs for more efficient planning.

3 Preliminaries

In this section, we focus to describe tool graph. For clarity, we propose tool graphs as dynamically changing graphs. Subsequent paragraphs defines detailed description of tool graph, including the use of tool description and input output format utilized by proposed Graph-O-Planner.

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: Given a task query Q, it 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 a unique tool v_i. The objective is to construct a Directed Acyclic Graph (DAG) that represents the sequence of processes to solve the query Q.

For the query Q and tool graph G, the DAG can be formally represented as:

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

This equation represents the sequence of tools that need to be executed in order to solve the query Q, where each tool v_i is connected to the next tool v_{i+1} through a dependency edge.

4 Methodology

In this section, we introduce Graph-O-Planner, a novel infused-graph 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 [similar to the one proposed by Q-KGR (Zhang et al., 2024)] 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 pass the dataset's tool informarion through a pre-trained embedding model (ModernBERT Large(Warner et al., 2024)) for its embeddings 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

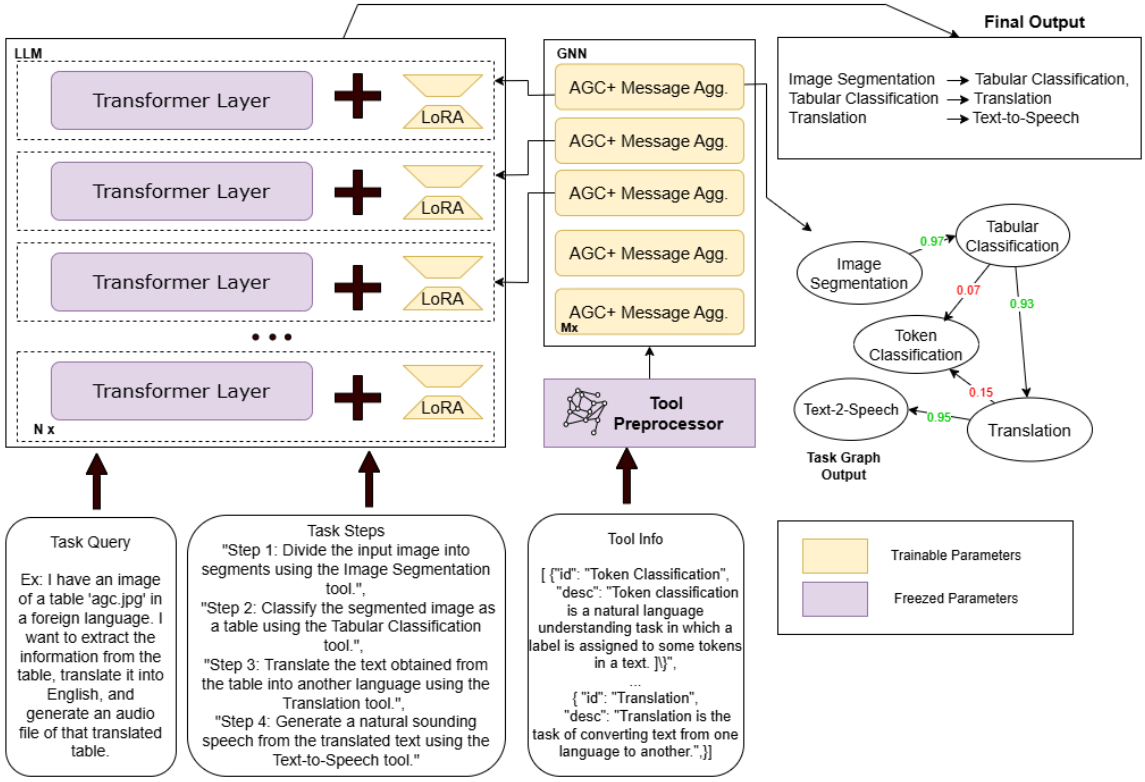


Figure 1: The overall framework of Graph-O-Planner. The figure illustrates the data flow, key components and interactions between different layers and modules. The visualization provides a comprehensive understanding of the model's training pipeline. Yellow layers consist of trainable parameters while purple represents frozen parameters

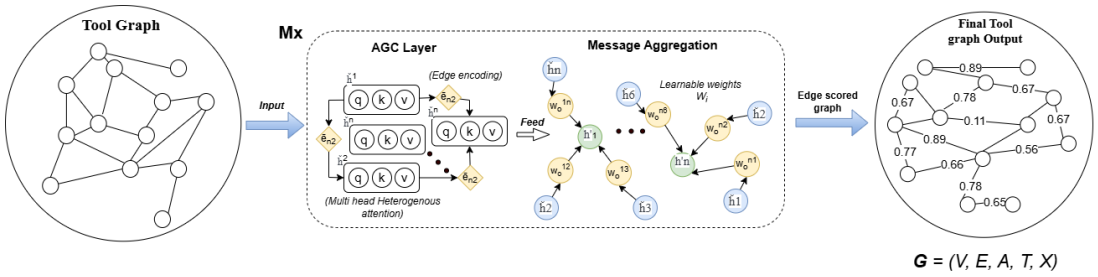


Figure 2: Representation of the Attention-Enhanced Graph Convolution

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 (mathematically defined in Section 3).

4.2 Attention-Enhanced Graph Convolution

The scored tool graph is then passed through a convolutional graph network, to attain its graphical embedding representation at every layer. This is pictorially shown in Figure 2. We use Attention-Enhanced Graph convolution (AGC) 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 3.1), 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:

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

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 node features and its features of its neighboring edges $N(i)$, to better capture neighborhood patterns every iteration:

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

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

$$Q_i^k = W_Q^k \tilde{h}_i \tag{4}$$

$$K_i^k = W_K^k \tilde{h}_i \tag{5}$$

$$Q_i^k = W_V^k \tilde{h}_i \tag{6}$$

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

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

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

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

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, the AGC layer uses a three phase computation over the edge-attention M to obtain the embeddings for every time-stamp:

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

4.3 Graph-O-Planner

In this section, we discuss the process by which we enabled the injection of 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 accumulated hidden embeddings from graph H of nodes are fused with LLM decoder module to obtain an intermediate representation I_c formulated as follows:

$$I_c = \{\theta_{key} \oplus \psi_{key}, \theta_{query} \oplus \psi_{query}, \theta_{value} \oplus \psi_{value}\} \tag{10}$$

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),

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 graph based approaches and that the numbers are sourced from Wu et al. (2024) (NA when scores were unavailable). ‡ indicates LLM-only models finetuned for each dataset. The information about the baseline models are described in Appendix A.6.

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
GraphToken*	20.08	32.55	62.15	NA	NA	NA	35.06	74.57	63.71	69.42	73.57	92.50
GraphSAGE*	33.88	52.62	78.49	37.22	47.68	70.75	62.37	70.25	88.86	86.57	85.80	97.42
GCN*	NA	40.74	66.54	NA	NA	NA	61.25	50.76	73.34	75.49	65.49	86.39
GIN*	NA	53.07	78.45	NA	NA	NA	NA	69.84	88.74	NA	85.80	97.42
Graph Transformer*	NA	52.27	78.30	NA	NA	NA	NA	70.24	88.90	NA	85.80	97.42
Qwen 2.5 Coder 3B‡	81.32	68.23	87.77	NA	NA	NA	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
Graph-O-Planner(Ours)	**82.6**	**88.25**	**96.21**	**97.39**	**97.56**	**98.88**	**92.0**	**93.55**	**97.11**	**96.81**	**95.06**	97.51

HuggingFace Tool, 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.5.

5.2 Results & Analysis

We present our model performance across various metrics, detailed in the section below. In all experiments, our base pipeline uses FlanT5-XL as LLM and a convolutional 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. 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.8.

For F1 scores, the set of predicted nodes/edges and set of ground truth nodes/edges is used for each sample i among N total samples.

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

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

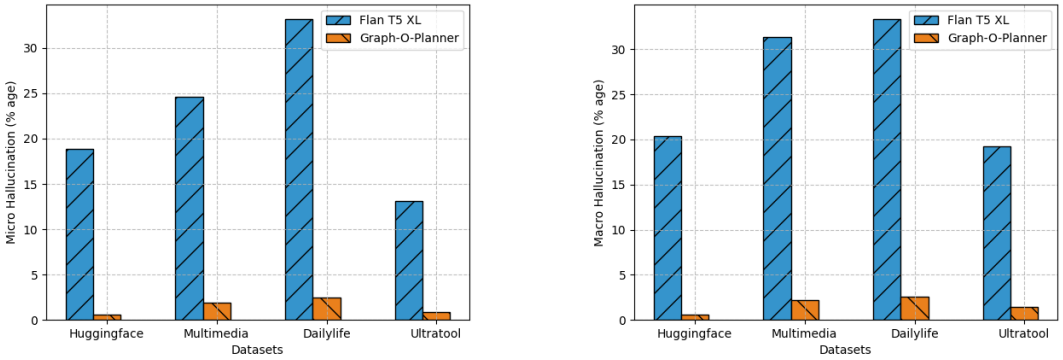
$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

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 **daily life dataset** (with 260+ tools), graph planner shows a **33% improvement** over older graph interaction methods and 8% improvement against deepseek, even when the pipeline using a lesser competent llm (flant5). This improvement is even more significant when considering the more convoluted **huggingface dataset**, with graph-o-planner seeing nearly **68% improvement** over previous models.



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

Figure 3: Hallucination in Flan T5 XL vs Graph-O-Planner

5.2.2 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:

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

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:

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

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 3a and 3b, our proposed GNN-based approach achieves a substantial reduction in hallucination, with a **13% decrease in incorrect edge predictions**.

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.4. In the current literature, training Large Language Model (LLM) planners typically involves passing all the tool information, including name, description, input/output format, directly to the prompt. This approach can become cumbersome and even pose significant challenges when dealing with a large number of tools, as seen in the Ultratool dataset. Even with a generous context length of 8192, we observed a spill-over of input tokens, highlighting the limitations of this method. Our approach, on the other hand, addresses this issue by injecting 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.

Graph-O-Planner significantly reduces input context size by bypassing extensive tool information, alleviating information overflow, reducing computational requirements, and enabling the LLM to focus on essential task-related information - resulting in more accurate and relevant generations. This makes it a more scalable and reliable solution for handling complex task sequences and a large number of tools.

Table 2: Computational analysis of the model across different datasets.

Model	Parameters	Time (min/epoch)			
		Huggingface	Multimedia	Dailylife	Ultratool
GraphSAGE	337,240,064	120.3	114.3	98.4	134.7
GCN	336,191,488	45.6	43.4	39.8	44.3
GIN	337,241,088	52.3	45.8	48.1	56.7
Graph Transformer	339,340,288	47.8	51.3	44.2	53.6
Flan-T5 XL	2,858,014,720	40.0	47.0	44.3	42.1
DeepSeek	1,840,564,264	65.2	64.0	61.9	72.2
Qwen 2.5	3,263,276,464	71.2	69.3	73.5	83.3
Graph-O-Planner	2,864,335,140	23.1	27.3	29.6	32.2

This also significantly reduces inference time latency, as shown in Figure 4, making it more suitable for real-time applications due to the reduced input size, computational requirements, and focused input context. The faster generation times and compact input length contribute to a more robust generation process, enabling the LLM to produce high-quality outputs more efficiently while being less prone to generating false or irrelevant information.

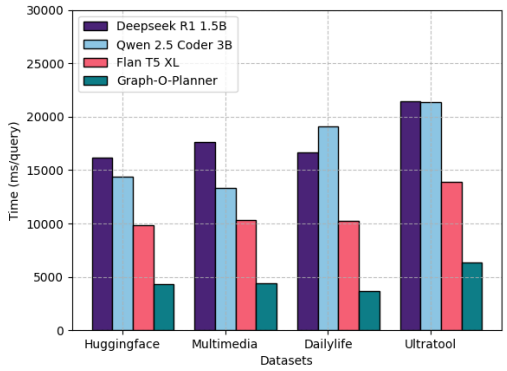


Figure 4: Inference time comparison of different models. The bar plot illustrates the average inference time (in milliseconds) for each model, highlighting performance variations. On average, across all datasets Graph-O-Planner generates under 7500 ms.

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 2x increased inference speed. To the best of our knowledge, proposed work is the first of its kind, exploring a deeply integrated GNN-LMM framework for effective task planning.

Limitations

Despite encouraging performance, this work is only the beginning of exploring the GNN-LMM interaction in-depth. We also want to extend the pipeline to make the planning truly generalizable across any unseen tool-graph and task type as well. 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.

Zihao Chen, Ying Wang, Fuyuan Ma, Hao Yuan, and Xin Wang. 2024. Gpl-gnn: Graph prompt learning for graph neural network. *Knowledge-Based Systems*, 286:111391.

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.

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*.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1857–1867.

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.

Junnan Liu, Qianren Mao, Weifeng Jiang, and Jianxin Li. 2024. Knowformer: Revisiting transformers for knowledge graph reasoning. *Preprint*, arXiv:2409.12865.

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*.

Costas Mavromatis and George Karypis. 2024. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*.

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.

Jinxue Peng, Yong Wang, Jingfeng Xue, and Zhenyan Liu. 2024. Fast cross-platform binary code similarity detection framework based on cfgs taking advantage of nlp and inductive gnn. *Chinese Journal of Electronics*, 33(1):128–138.

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*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. *arXiv preprint arXiv:2307.07697*.

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, Heng Ji, and Bang Liu. 2021a. Deep learning on graphs for natural language processing. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2651–2653.

Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. 2021b. 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*.

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*.

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*.

Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2021. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Advances in Neural Information Processing Systems*, 34:28798–28810.

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.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. *arXiv preprint arXiv:2104.06378*.

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.

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 3.

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.

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 5, 6, 7, 8, 9, 10 shown Dev, test and loss for only LLM approach in comparison with Graph-O-Planner approach. From figure 5 and 6, 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 7 and 10.

We also observed a much reliable training with Graph-O-Planner approach. As seen from Figure 9, 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 10 we can observe that the loss drops drastically till 40th epoch, but soon reaches a stagnant curve, however as can be observed from 8, 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" module, 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.

Table 3: Notation table in Graph-O-Planner

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

A.4 Graph-O-Planner overcomes context overflow 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.

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.

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.

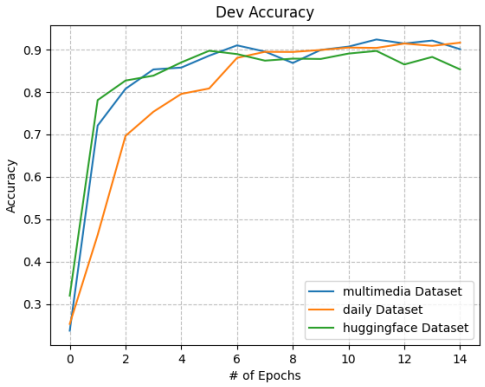


Figure 5: Dev edge-f1 of Graph-O-Planner

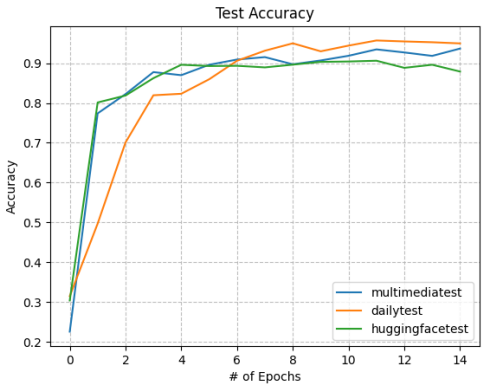


Figure 6: Test edge-f1 of Graph-O-Planner

A.5 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 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.

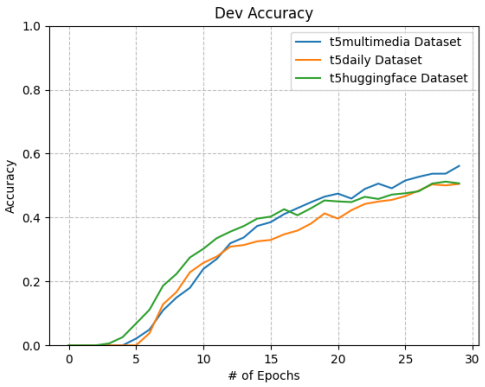


Figure 7: Dev edge-f1 of Flan T5 XL

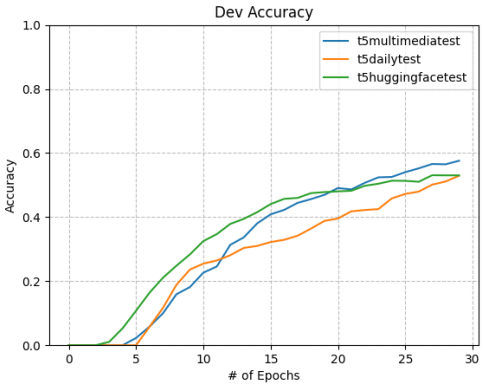


Figure 8: Test edge-f1 of Flan T5 XL

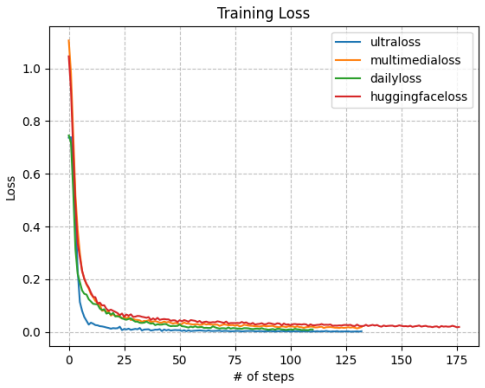


Figure 9: Loss graph of Graph-O-Planner after training for 15 epochs

Huggingface. It consist of 40 tools with 7546 training samples. The tools comprises of hugging face hosted models fine-tuned to perform various downstream tasks. The overall dataset requires 20177 tool callings with an average of 3.28 arguments per tool call. The dataset consist of 40.64% of samples with single tool calling.

Multimedia. This dataset also consist of 40

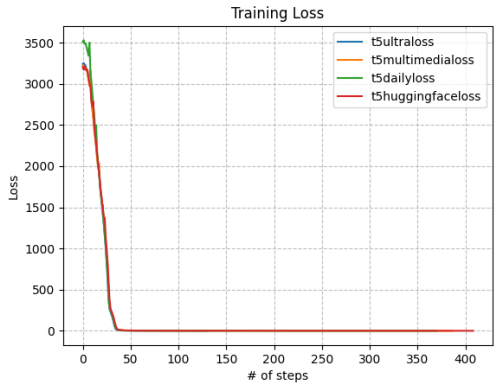


Figure 10: Loss of Flan T5 XL while training for 30 Epochs

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.6 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.7 More Detailed Study

1. Proof of effectiveness of GNN. In this section, we will theoretically prove that using <our-approach> 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:

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

2. Computational cost analysis In this section we provide the computation comparison of LLM only v/s Graph-O-Planner approach. Results are tabulated in Table 2.

Training Efficiency. During the experiments, the number of trainable parameters remains constant across all the dataset. We observed that for Graph-O-Planner approach the time 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.

A.8 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**

$gt\_tools = gt\_nodes \cup L[i][0] \cup L[i][1]$
end for
for $i = 1$ to $pred\_len$ **do**
$pred\_tools = gt\_nodes \cup R[i][0] \cup R[i][1]$
end for
$c\_tools \leftarrow pred\_tools \cap gt\_tools$
$p \leftarrow \frac{length(c\_tools)}{length(pred\_tools)}$
$r \leftarrow \frac{length(c\_tools)}{length(gt\_tools)}$
$node\_f1 = \frac{2*p*r}{p+r+\alpha}$
return $node\_f1$
end procedure

Algorithm 2: Edge-F1

$L \leftarrow [l_1, l_2, \ldots, l_n]$ ▷ List of ground truth tool pairs. $r_i : (tool_{i1}, tool_{i2})$
$R \leftarrow [r_1, r_2, ..., r_n]$ ▷ List of predicted tool pairs. $r_i : (tool_{j1}, tool_{j2})$
procedure EDGE F1(L, R)
$gt\_len \leftarrow length\ of L$
$pred\_len \leftarrow length\ of R$
$c\_links \leftarrow 0$
for $i = 1$ to $pred\_len$ **do**
for $j = 1$ to gt_len **do**
if $R[i] == L[j]$ **then**
$c\_links \leftarrow c\_links + 1$
end if
end for
end for
$c\_tools \leftarrow pred\_tools \cap gt\_tools$
$p \leftarrow \frac{length(c\_tools)}{length(pred\_len)}$
$r \leftarrow \frac{length(c\_tools)}{length(gt\_len)}$
$edge\_f1 \leftarrow \frac{c\_links}{gt\_len}$
return $edge\_f1$
end procedure