Encoder of Thoughts: Enhancing Planning Ability in Language Agents through Structural Embeddings

Yuxiang Zhang[1,], **Jitao Sang**[1,2*]

[1]Beijing Key Lab of Traffic Data Analysis and Mining, Beijing Jiaotong University
[2]Peng Cheng Lab
yuxiangzhang@bjtu.edu.cn, jtsang@bjtu.edu.cn

Abstract

Large Language Models (LLMs), when combined with agent mechanisms, show great promise in applications requiring robust planning ability, such as financial analysis and medical diagnostics. However, the increasingly complex reasoning structures designed to enhance the planning ability of language agents often exceed the processing and comprehension capabilities of LLMs, thereby limiting their effectiveness. To address these challenges, we introduce the Encoder of Thoughts (EoT), a novel reasoning structure modeling method based on graph neural networks. EoT processes the reasoning structures of planning methods through a plug-and-play structural encoder and aligns these structural information with the input space of LLMs, enabling seamless integration with existing language agents. Experiments on multi-step reasoning and plan generation demonstrate that EoT significantly improves the performance of language agents. Moreover, EoT demonstrated stable results when combined with different LLMs and planning algorithms, further underscoring its potential for broader application.

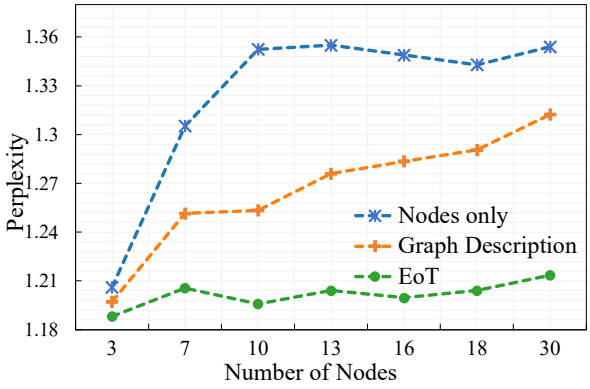

Figure 1: Perplexity comparison of reasoning path reconstruction on GSM8K. The figure compares the perplexity of reasoning paths across action graphs of different sizes. The blue curve represents node-only reconstruction, the yellow curve includes graph descriptions, and the green curve uses EoT.

Introduction

Large Language Models (LLMs) combined with agent mechanisms, known as language agents, have achieved significant success in various fields, such as financial analysis (Li et al. 2023) and intelligent recommendation systems (Huang et al. 2023). The core of this approach lies in integrating planning methods with LLMs, providing reasoning capabilities that surpass those of standalone LLMs, thereby broadening their application scope. However, a fundamental limitation of built-in planning methods is their difficulty in effectively understanding and managing the current planning state when faced with complex reasoning structures. Despite enhancements from existing work, such as tree search algorithms (Yao et al. 2024; Hao et al. 2023), language agents still face significant challenges in complex planning and reasoning tasks.

Some studies approach language agent planning as the task of finding the optimal solution path within an action graph, sometimes referred to as a task graph (Wang et al. 2024) or thought topology (Besta et al. 2024). The effectiveness of the planning module primarily depends on the structure of the prompt engineering, as demonstrated in frameworks like CoT-SC (Wang et al. 2022), ReAct (Yao et al. 2023), Tree of Thoughts (ToT) (Yao et al. 2024), Reasoning via Planning (Hao et al. 2023), and Graph of Thoughts (GoT) (Besta et al. 2024). These various reasoning structures can be viewed as special forms of graph (Besta et al. 2024). Based on these reasoning structures, language agents can effectively choose the optimal solution path using search algorithms such as BFS (Yao et al. 2024), MCTS (Hao et al. 2023) and A* (Hazra, Dos Martires, and De Raedt 2024), thus improving performance during the inference process.

The performance of planning methods hinges on the comprehensive modeling of reasoning structures. However, existing planning-based prompt methods are devoted to designing increasingly complex reasoning structures, which is evidenced by the evolution from CoT to GoT. This makes it harder for language agents to grasp the current planning state and increasingly challenging to reflect on and learn from past explorations. To address these challenges, re-

*Corresponding author.

searchers have developed various strategies to simplify the understanding and processing of global structures. For instance, some methods preprocess reasoning structures using manually designed algorithms, enabling the LLM to selectively handle certain parts (Hao et al. 2023). Others adopt a bottom-up approach, first generating strategies for local reasoning structures and then aggregating them into a global strategy (Chen, Li, and Niu 2024; Zhou et al. 2023). Nonetheless, the direct enhancement of language models' understanding of reasoning structures has been largely overlooked. This oversight is likely due to the inherent inefficiencies of using natural language to describe graph structures (Chen et al. 2024; Guo et al. 2023), which significantly increases token consumption and processing complexity. Previous work in knowledge graph has shown that integrating embedded graph information enables language models to perform tasks like attribute annotation and link prediction on the graph (Tang et al. 2024; Jin et al. 2023). Building on this insight, we propose to embed the reasoning structures into the input space of language models. This not only alleviates the limitations posed by natural language descriptions but also enhances the ability of language models to process and comprehend complex reasoning frameworks, ultimately improving their performance in intricate planning tasks.

Specifically, we propose Encoder of Thoughts (EoT), a model designed to capture the structures of action graphs and reasoning states during the planning process. By leveraging Graph Neural Networks (GNNs), EoT encodes the structures of action graphs during the planning phase, transforming this information into special token embeddings within the input context of LLMs. As shown in Fig. 1, we conducted an experiment to reconstruct reasoning paths from action graphs to validate the effectiveness of EoT in enhancing language models' understanding of reasoning structures. It shows that EoT significantly outperforms current text-based methods by reducing perplexity in reasoning path reconstruction as the graph size increases. This integration allows language models to more effectively navigate and comprehend structured information, demonstrating EoT's potential in managing larger and more complex reasoning frameworks.

Evaluation results in the domains of mathematical reasoning and plan generation demonstrate three key contributions of EoT: **a) Effectiveness:** EoT demonstrated significant improvements in both mathematical reasoning and plan generation tasks. Specifically, it achieved a 5-10% performance increase on the GSM8K datase and 5-20% boost on the Blockworlds dataset. Additionally, despite being primarily trained on GSM8K, EoT consistently outperformed on the more challenging MATH dataset, even with minimal training data from Blockworlds and none from MATH. **b) Adaptivity:** EoT proves to be highly adaptable, showing stable and robust performance across different LLMs and planning algorithms. This adaptability ensures that EoT can be seamlessly integrated into existing language agent framework. **c) Novelty:** By integrating structural embeddings into prompts, our approach reduces the reliance on fine-tuning LLMs or introducing complex prompt engineering, thereby maintaining the original model's performance without significant computational overhead. These contributions underscore EoT's potential as a powerful and efficient tool for advancing the reasoning capabilities of language models.

Related Work

Planning in Language Agents The performance of language agents largely depends on their ability to plan future actions. To address the limitations of the input-output prompt approach, planning-based prompt methods have been introduced, enabling exploration and backtracking. These methods decompose complex reasoning tasks into manageable components, using structured methods like ToT (Yao et al. 2024), RAP (Hao et al. 2023), SayCanPay (Hazra, Dos Martires, and De Raedt 2024), and GoT (Besta et al. 2024) to find optimal reasoning paths. To improve LLM performance LLM performance, search algorithms such as BFS, MCTS, and A* are employed, conceptualizing planning as the search for optimal paths within graphs. Although promising, these methods often face reliability challenges in LLM action prediction and state estimation, leading to inefficiencies and inaccuracies in the planning process (Chen et al. 2024; Guo et al. 2023).

To overcome these obstacles, reflection methods have been introduced, enhancing model outputs through iterative feedback and adjustments. Techniques like (Madaan et al. 2024; Du et al. 2023) emphasize feedback-based enhancements, but they often have limited integration of historical reflections, which may result in a less comprehensive understanding of the tasks. Some methods such as ProTeGi (Pryzant et al. 2023), PromptAgent (Wang et al. 2023), and TRAN (Yang, Li, and Liu 2023) optimize prompts using task-level reflections to guide future generations. Unlike these methods, our approach directly improves language models' understanding of reasoning structures through an external plugin, thereby addressing the limitations of previous approaches.

Graph Understanding in LLMs Despite the significant success of LLMs in various domains, its autoregressive architecture still faces limitations in efficiency and efficacy in understanding and solving graph-related issues. Some studies, such as GPT4Graph (Guo et al. 2023), have attempted to improve LLMs' ability to process graph data via prompt techniques or by constructing fine-tuning datasets with graph reasoning paths. However, these studies show that the inherent graph processing capabilities of LLMs, the effectiveness of these capabilities, is greatly affected by the choice of prompts and input formats.

In contrast, Graph Neural Networks (GNNs) (Zhou et al. 2020) excel in processing graph data, offering better solutions to graph-related tasks. To bridge the gap in LLMs' capabilities, some research has integrated GNNs into LLMs. For example, GraphLLM (Chai et al. 2023) and GraphGPT (Tang et al. 2024) have significantly improved LLMs' performance in tasks such as graph classification, node classification, and link prediction by introducing GNN-based encoders. Building on the successful integration of GNNs with LLMs for processing graph data, this work proposes utilizing GNNs to model reasoning structures, enhancing the rea-

soning capabilities of LLMs in complex planning tasks.

Methodology

Problem Definition

We conceptualize the planning process of language agents as a path exploration task on a directed acyclic graph (DAG), where each node represents an action, and the edges indicate the order of execution. The sequence of nodes from the source to any node defines a state, and all action and transitions form the action graph $\mathcal{G}$. A state s is defined as an ordered sequence of actions $\{a_1, a_2, \ldots, a_n\}$ within the graph $\mathcal{G}$. The challenge is to find the optimal path in the graph that forms the desired state.

EoT Workflow

We introduce the Thought Encoder to enhance the reasoning capabilities of language agents, especially in complex planning tasks. This module enables agents to process and understand structured information in action graphs more effectively. The core components of this integration and their roles in enhancing reasoning are described below.

Core Components To enhance the planning and reasoning abilities of language agents, we introduce the **Thought Encoder** module g, which integrates into existing language models to provide structural embeddings. This integration involves the following key components:

- **Thought Encoder** $g(\mathcal{G})$: The Thought Encoder g projects the graph of all actions $\mathcal{G}$ into thought embeddings e_{thought}. These embeddings capture the features of the action graph and are aligned with the input space of the LLM, enabling it to understand the planning context.
- **Generator** $\pi : (e_{\textbf{thought}}, s) \rightarrow a$: The Generator π, implemented by the language model, uses the thought embedding and the current state to generate possible next action a. This process drives the reasoning forward by determining the most suitable subsequent actions.
- **Evaluator** $\rho : (e_{\textbf{thought}}, s) \rightarrow r$: The Evaluator ρ evaluates the effectiveness of the current state s by analyzing the thought embedding e_{thought} and considering the feedback from the environment. This evaluation assigns a quality score to the actions and provides the results to the search algorithm for optimization.

Integration into Language Agent As shown in fig. 2, the Thought Encoder is integrated into language agents through the following steps:

1. **Graph Encoding**: The action graph is encoded into thought embeddings by the Thought Encoder, which captures the structural information. These embeddings can serve as special tokens corresponding to the content of each node, indicating the relationships between nodes.
2. **Embedding Integration**: These embeddings are combined with the content of action, representing both node content and dependencies. This method of representing graphs can reduce the number of context tokens, as nodes are not repeatedly placed, unlike in path-based or triplet-based methods where common nodes are duplicated.
3. **Reasoning and Execution**: The thought embeddings, together with task-specific instructions, guide the Generator in selecting the next actions and the Evaluator in assessing their quality, thereby actively contributing to the agent's decision-making and execution process. As the language agent executes each step, the action graph is continuously updated, reflecting the sequence of actions taken and the evolving decision-making process.

This integration complements existing language agent frameworks and enhances their planning capabilities while preserving established workflows.

Search Algorithms With this integration, we now turn to the implementation of search algorithms. The structural embeddings from the Thought Encoder are incorporated into these algorithms to enhance their effectiveness in navigating the state space. In this study, we explore two widely adopted search algorithms: Breadth-First Search (BFS) and Monte Carlo Tree Search (MCTS).

Algorithm 1: EoT-BFS(s_0, g, π, ρ, b, T)

Require: Initial state s_0, thought encoder g, generator π, evaluator ρ, breadth limit b, depth limit T

$\mathcal{G} \leftarrow \{s_0\}$
$S_0 \leftarrow \{s_0\}$
for $t \leftarrow 1, .., T$ **do**
  $e_{\text{thought}} \leftarrow g(\mathcal{G})$
  $S'_t \leftarrow \{(s, a) \mid s \in S_{t-1}, a \in \pi(e_{\text{thought}}, s)\}$
  $V_t \leftarrow \rho(e_{\text{thought}}, S'_t)$
  $S_t \leftarrow \arg\max_{S \subseteq S'_t, |S|=b} \sum_{s \in S} V_t(s)$
  $\mathcal{G} \leftarrow \mathcal{G} \cup S_t$
end for
return $\arg\max_{s \in S_T} V_T(s)$

Algorithm 2: EoT-MCTS(s_0, g, π, ρ, N, T)

Require: Initial state s_0, thought encoder g, generator π, evaluator ρ, number of iterations N, maximum depth T

Initialize search tree $\mathcal{T}$ with root node s_0
for $n \leftarrow 1, .., N$ **do**
  // Selection
  $\mathcal{G} \leftarrow \{s_0\}, t \leftarrow 0$
  while $N(s_t) > 0$ **do**
    $s_t \leftarrow \text{Select}(\mathcal{T}, g, \rho)$
    $t \leftarrow t + 1, \mathcal{G} \leftarrow \mathcal{G} \cup \{s_t\}$
  end while
  // Expansion and Simulation
  while not terminal(s_t) and depth(s_t) $< T$ **do**
    $e_{\text{thought}} \leftarrow g(\mathcal{G})$
    $\mathcal{T}, s_{t+1} \leftarrow \text{Expand}(\mathcal{T}, s_t, \pi, e_{\text{thought}})$
    $r \leftarrow \text{Simulate}(\mathcal{T}, s_{t+1}, \rho, e_{\text{thought}})$
    Update $t \leftarrow t + 1, \mathcal{G} \leftarrow \mathcal{G} \cup \{s_t\}$
  end while
  // Back propagation
  $\text{Backpropagate}(\mathcal{T}, r)$
end for
return $\text{BestRollout}(\mathcal{T})$

EoT-BFS (Algorithm 1): This approach employs Breadth-First Search, starting from the root node and ex-

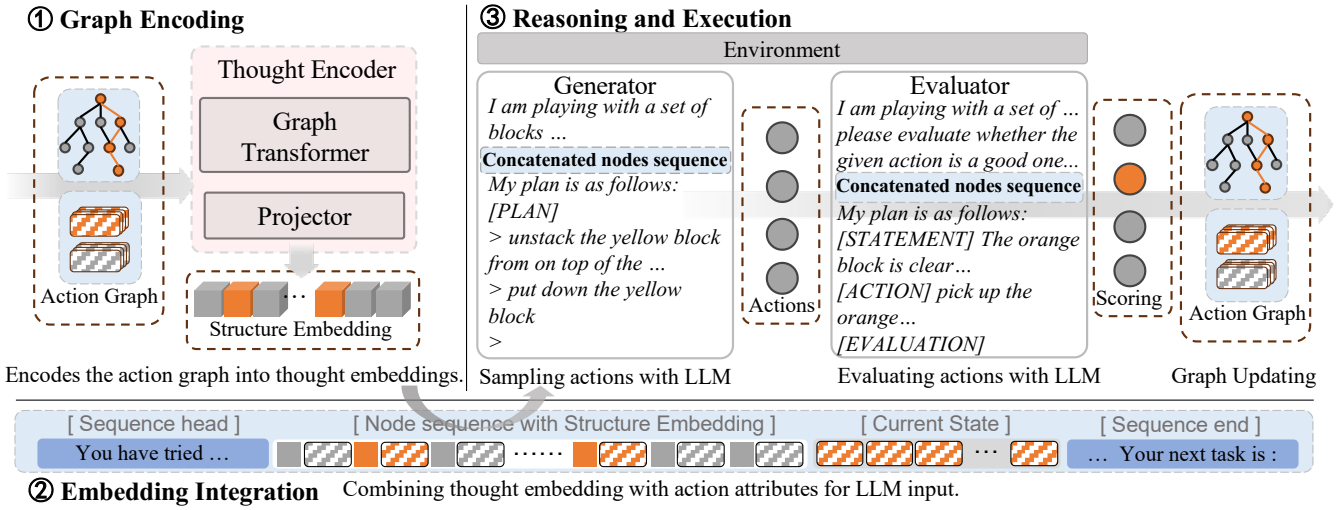

Figure 2: Integrate EoT with language agent. The orange nodes and rectangles represent the actions and action attributes in the current reasoning path.

ploring neighboring nodes layer by layer. At each layer, the b most promising states are maintained to ensure efficient exploration of the state space. The algorithm offers control over the depth and breadth of the search tree, allowing it to adapt to the specific requirements of the task.

EoT-MCTS (Algorithm 2): This approach uses Monte Carlo Tree Search, which involves four phases: selection, expansion, simulation, and backpropagation. The algorithm evaluates potential states by conducting random simulations, subsequently optimizing decision paths based on the outcomes of these simulations.

Encoder Design

The Thought Encoder consists of two key components: a graph encoder and a projection layer. The graph encoder, adaptable to various GNN backbones, uses a message-passing approach to derive feature vectors. The projection layer aligns these vectors with the LLM's input space, ensuring seamless integration with the generator and evaluator.

Graph Encoder We implement the graph encoder using a graph transformer (Shi et al. 2020). The updated feature vector for node i is computed as:

$$\mathbf{x}_i' = \mathbf{W}_1\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{x}_j,$$

where $\alpha_{i,j}$ represents the attention coefficient calculated using a multi-head dot-product attention mechanism:

$$\alpha_{i,j} = \text{softmax}\left(\frac{(\mathbf{W}_2\mathbf{x}_i)^\top(\mathbf{W}_3\mathbf{x}_j)}{\sqrt{d}}\right).$$

Here, $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$ are trainable weight matrices, and d is the dimension used in the dot product. The attention mechanism captures the contributions of neighboring nodes by evaluating the contributions of neighboring nodes.Node features are initialized by averaging the word embeddings of the node's text, as provided by the embedding layer of LLM.

Projection Layer The features derived for each node i are projected into the input space of the LLM through a linear layer with a bias term. Mathematically, the projected embedding for node i, denoted as $e_{\text{thought},i}$, is defined as:

$$e_{\text{thought},i} = \mathbf{W}_p\mathbf{x}_i' + \mathbf{b}_p,$$

where $\mathbf{W}_p \in \mathbb{R}^{d\times d_m}$ is a weight matrix and $\mathbf{b}_p \in \mathbb{R}^{d_m}$ is a bias vector, where d_m denotes the dimension of the LLM.

Encoder Training

The goal of training is to enable the LLM to effectively utilize the outputs from the Thought Encoder to generate appropriate responses. The training data are automatically generated by Llama3-8b, without relying on manual annotations or more advanced models such as ChatGPT.

Data Construction The data construction process leverages the capabilities of LLM to autonomously generate training instances $\langle G_t, s, a_{t+1}\rangle$ required for training the Thought Encoder. This approach, inspired by the ToT method, extracts reasoning steps and action graphs to ensure alignment between the generated text and the reasoning states. Starting with an initial query, the BFS algorithm identifies and maintains promising states. For each state, possible subsequent actions are sampled and evaluated, leading to the generation of new actions. The algorithm selects the most promising paths through successive layers, thereby maintaining consistent reasoning quality. Each data instance consists of an action graph G and a set of paths that achieve the target. The Evaluator then assesses these states and scores them based on the quality of reasoning. The terminal paths with the highest scores, $\mathcal{P}_{\text{high}} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m\}$, are selected, and each path $\mathcal{P}_j$ represents the reasoning process from the initial query to the final state.

Training Objective The training objective is to enable EoT to guide the LLM in selecting actions. Specifically,

the aim is to maximize the conditional probability of high-scoring actions given the action graph and the thought embeddings, $P(a_{t+1} \mid G_t, e_{\text{thought}}, s)$. Here, e_{thought} denotes the thought embedding produced by EoT. The loss function for training is defined as:

$$\mathcal{L} = -\sum_{t} \sum_{\mathcal{P}_j \in \mathcal{P}_{\text{high}}} \log P(a_{t+1} \mid G_t, e_{\text{thought}}, s)$$

Experiment Setup

Datasets

We evaluated the performance of the EoT framework using three datasets: GSM8K, MATH, and Blockworlds.

GSM8K The GSM8K dataset (Cobbe et al. 2021) consists of mathematical problems from elementary school exams, focusing on basic arithmetic operations. To address the multi-step reasoning required by these problems, we decomposed the solutions into a sequence of smaller question-answer pairs during the reasoning phase.

MATH The MATH dataset (Hendrycks et al. 2021) includes more challenging competition-style problems spanning a variety of mathematical domains. We constructed the MATH test set by selecting 30 questions from each of the seven subjects. Each problem was decomposed into a series of subquestions, similar to the approach used for GSM8K, to facilitate multi-step reasoning.

Blockworlds The Blockworlds dataset (Valmeekam et al. 2022) was used to evaluate model performance in plan generation tasks. We adopted the environment configuration from RAP (Hao et al. 2023), where the tasks involve manipulating stacks of blocks in a dynamic environment. The goal is to organize the blocks into specific arrangements.

Baseline Methods

To evaluate how effective EoT is, we conducted comparisons against several baseline methods:

- **Chain-of-Thought (CoT):** CoT (Wei et al. 2022) utilizes intermediate reasoning steps to guide language models towards improved final outputs.
- **Self-Refine:** Self-Refine enhances a model's outputs through iterative self-generated feedback and environmental feedback in the Blockworlds dataset.
- **ToT-BFS:** ToT-BFS expands and evaluates the Tree of Thoughts layer by layer, systematically exploring intermediate steps before advancing in the search space.
- **RAP-MCTS:** RAP (Hao et al. 2023) employs Monte Carlo Tree Search (MCTS) to simulate future states and make strategic decisions. This method explores various reasoning paths, anticipates future outcomes, and iteratively refines the reasoning process.

Alternative Approaches for EoT embedding To further demonstrate the superiority of EoT, we replaced the thought encoding using two alternative approaches.

- **Textual Graph:** In this approach, we replaced the EoT embeddings with a textual description of an action graph. Here, all reasoning paths are directly provided to the language model as sequences of actions. This comparison allows us to assess the performance of EoT against the approach of explicitly presenting all reasoning paths.
- **Prompt-Tuning:** Prompt-tuning (Lester, Al-Rfou, and Constant 2021) was included to assess whether EoT's embeddings outperform standard prompt-tuning embeddings.

Training Details

Data Construction During the training process, we did not use data annotated by more powerful models (e.g., ChatGPT) or human annotators. Instead, we initially employed the Llama3 8B model to generate the foundational training data. Building on this, we applied the Tree of Thoughts approach to refine and optimize the data, tailoring it to the specific requirements of our tasks.

For the GSM8K dataset, we set the BFS depth to 5, sampled up to 3 instances per child node, and limited each layer to 4 nodes. For Blockworlds, the depth matched the ground truth reasoning sequences (4 to 12), with the same sampling strategy and a layer width of up to 5 nodes. Each instance was constructed three times to ensure data generation success. Table 1 provides detailed statistics. The model was trained on data from both datasets simultaneously.

	GSM8K	Blockwords	Total
Successful instances	1750	100	1850
Avg. number of nodes	9.65	27.38	10.61
Avg. correct paths	2.57	3.81	2.64

Table 1: Statistics of EoT training dataset.

Model Setup In our study, we employed two large language model backbones: Llama3-3b-Instruct and Qwen1.5-7B. The different encoders were trained independently for each model. Each encoder was configured with a Graph Transformer featuring 2 layers, a hidden size of 512 dimensions, and 8 attention heads.

The encoder training was conducted with a batch size of 32 on NVIDIA L20 GPUs. We used the Adam optimizer with a cosine-decaying learning rate, starting at 1×10^{-4}. The training process lasted for 2000 steps.

Evaluation Result

Mathematical Reasoning

Task Setup During the evaluation of the BFS algorithm on the GSM8K and MATH datasets, we set the breadth limits b to 3 and 5, and the depth limits T to 5 and 7, respectively. Each action was sampled four times. For the MCTS configuration, we adopted the hyperparameters and reward settings proposed in RAP (Hao et al. 2023).

Model	CoT	Self-Refine	ToT-BFS		RAP-MCTS$^{(5)}$		Prompt-tuning		EoT	
			Vanilla	Textul graph	Vanilla	Textul graph	BFS	MCTS$^{(5)}$	BFS	MCTS$^{(5)}$
Llama-3 8B	71.5	76.3	80.3	77.0	82.2	79.7	83.0	85.5	**88.3**	**86.3**
Qwen-1.5 7B	59.4	64.2	70.8	68.4	69.7	71.6	71.1	**73.8**	**73.4**	72.9

Table 2: Results on GSM8k. 4-shot random example demonstrations were used across all methods. Superscripts denote the number of iterations. The term *textual graph* refers to adding the textual form of action graphs into the prompts.

steps	CoT	Self-Refine	ToT-BFS		RAP-MCTS$^{(8)}$		EoT-BFS		EoT-MCTS$^{(8)}$	
	Llama	Llama	Llama	Qwen	Llama	Qwen	Llama	Qwen	Llama	Qwen
4-step	15.8	25.0	84.2	81.6	78.9	80.3	90.8(+7.8%)	84.2(+3.2%)	89.5(+13.4%)	78.9(-1.7%)
6-step	2.6	4.8	47.6	49.0	57.2	49.7	58.6(+23.1%)	53.1(+8.4%)	60.6(+5.9%)	53.2(+8.4%)
8-step	0.0	1.4	28.7	23.8	34.3	28.0	32.9(+14.6%)	25.2(+5.8%)	35.0(+2.0%)	31.5(+12.4%)
10-step	0.0	0.0	19.4	18.4	17.5	15.4	24.3(+25.3%)	16.5(-10.3%)	22.3(+27.3%)	19.6(+27.3%)
12-step	0.0	0	8.7	6.5	5.6	4.3	10.9(+33.3%)	13.0(+100%)	13.0(+150%)	10.9(+150%)

Table 3: Results on Blockworlds. 4-shot random example demonstrations were used across all methods. Superscripts denote the number of iterations.

Method	Accuracy(%)
COT	28.2
BFS	34.6
MCTS$^{(8)}$	32.1
EoT-BFS	36.7
EoT-MCTS$^{(8)}$	37.4

Table 4: Results on MATH with Llama-3 8B. 4-shot example demonstrations were used across all methods. Superscripts denote the number of samples or iterations.

GSM8K The results of comparing various reasoning methods and language models on the GSM8K dataset are shown in Table 2. These results highlight the notable performance improvements achieved with EoT in all experiments. Whether combined with BFS or MCTS, EoT consistently achieved substantial gains in solve rates. For instance, with the Llama-3 8B model, EoT improved the solve rate by 0.4% in BFS and 5.0% in MCTS. Similarly, with the Qwen-1.5 7B model, EoT improved the solve rate by 3.7% in BFS and 4.6% in MCTS. These results not only demonstrate the applicability of EoT across different reasoning methods but also highlight its ability to consistently improve performance across various base models.

EoT vs Textual Graph: When we substituted EoT embeddings with textual representations of the action graph, we observed no improvement in the performance of the existing reasoning methods. In fact, it considerably decreased the model's efficiency, particularly in the BFS algorithm with larger action graphs. Repeating the same actions across different reasoning paths leads to excessively long contexts, which can pose challenges to the model's understanding. Furthermore, mistakes in earlier reasoning paths can further mislead the model. EoT reduces redundancy by ensuring that each action node is distinct and non-repetitive. Additionally, EoT maximizes the conditional probability of high-scoring paths during optimization, enabling the model to focus on the most relevant information and effectively reducing the impact of erroneous histories on performance.

EoT vs Prompt Tuning: The results show that EoT outperforms Prompt-Tuning, particularly when used alongside BFS. Alternatively, static Prompt-Tuning tends to be more effective when paired with MCTS, benefiting from MCTS's more consistent graph structures. In fact, if the weights of the projection layer are set to zero, then EoT essentially functions just like Prompt-Tuning, with the main difference being whether input from the graph encoder is included.

MATH We intentionally excluded MATH dataset examples from EoT training to better assess its generalizability on MATH. The results presented in Table 4, EoT successfully achieved the highest performance on the more challenging MATH dataset. This suggests that EoT is not only effective with specific datasets but also capable of generalizing to new, unseen data, further highlighting its adaptability to a broader range of applications.

Plan Generation

Task Setup. Except for the 100 samples used during the construction of the training data, the remaining data was split into six different difficulty levels according to the minimum steps required to complete the tasks. For BFS, a breadth limit of 5 was established, and the depth limit was adjusted to align with the minimum step count for each level. For MCTS, we adhered to the configuration detailed in RAP (Hao et al. 2023).

Blockworlds As shown in Table 3, incorporating EoT into BFS and MCTS resulted in significant performance improvements across most scenarios in the Blockworlds dataset. The experimental results showed that even with a limited amount of Blockworlds data used during training,

EoT consistently boosted performance by 10% to 20% in most scenarios. Notably, as the number of reasoning steps increased, the effectiveness of EoT in improving MCTS became even more pronounced, demonstrating its particular strength in more complex reasoning scenarios. These results underline EoT's robustness and adaptability in various complex reasoning tasks, making it a valuable asset across different algorithmic frameworks.

BFS vs MCTS: It can be observed from the results that the performance improvement of EoT combined with MCTS remains more stable as the reasoning difficulty increases. This can be explained by the distinct ways in which action graph expansion occurs in each search algorithm; BFS generally experiences a quicker enlargement of the graph as reasoning steps advance, while MCTS shows a more gradual growth. As a result, the action graphs generated by the BFS algorithm in the Blockworlds scenario are significantly larger than the majority of graph sizes in the training data.

Discussion

A common criticism of using language agents is the significant computational resources consumed during algorithm iterations. We evaluated the accuracy on GSM8k for EoT-MCTS, MACS, and CoT at different iteration counts. As shown in fig. 3, the introduction of EoT allows MCTS to achieve acceptable accuracy with fewer iterations. Despite increasing the input context length, EoT remains highly efficient because recent optimizations in transformer inference have reduced the computational cost of input tokens. This effectively balances the trade-off between input context length and iteration speed.

Encoder Architecture The graph encoder can be conveniently adapted to various GNN architectures. For instance, we employed a graph convolutional network (GCN) in place of the graph transformer. As shown in Figure 3, the EoT-MCTS(GCN) configuration still demonstrated acceptable performance. The decision to use two layers in the graph neural network is based on empirical evidence. Our experiments revealed that increasing the number of layers resulted in a tendency to overfit the training data.

Ablation Study The ablation study was conducted on both the projection layer and the GNN. The results from EoT-MCTS (w/o Projector) and EoT-MCTS (w/o GNN) revealed significant differences in performance. Omitting the projection layer impacted the model's performance, whereas omitting the GNN led to a greater drop, emphasizing the importance of both components in the architecture.

Impact of structural Embeddings on CoT We explore whether EoT can enhance the LLM's ability to generate reasoning paths by enabling a deeper understanding of thought structures. The underlying hypothesis is that if the LLM can effectively grasp the reasoning structure through EoT, it may better replicate the reasoning process during generation.

We integrated EoT examples into CoT reasoning. Specifically, we converted GSM8K examples into thought graphs and extracted their thought embeddings to construct EoT-format examples. These EoT examples were then concatenated with standard context examples to guide the LLM in generating output in the CoT format. As shown in Table 5, this prompt enhanced the performance of CoT, particularly when using examples containing correct subgraphs. Conversely, using examples with random or incorrect subgraphs did not significantly impact performance. These findings suggest that EoT can help the LLM gain a more comprehensive understanding of reasoning structure.

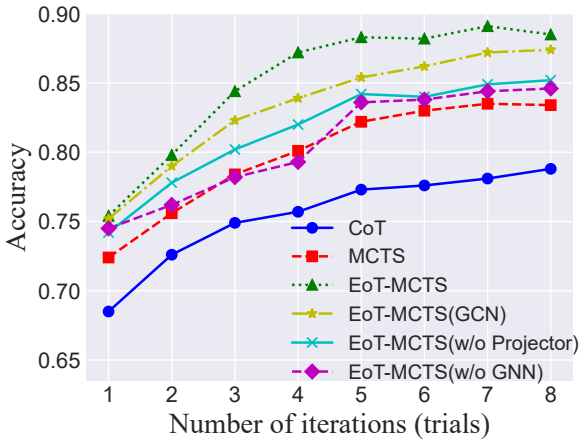

Figure 3: Results on GSM-8K, with different numbers of iterations or trials.

Method	Accuracy(%)
COT	68.5
+ all paths	71.6
+ random paths	70.3
+ incorrect paths	69.2
+ correct paths	73.2

Table 5: Results on GSM8K with Llama-3 8B. Leveraging EoT to build in-context demonstrations, the information following the '+' symbol denotes distinct graph configurations.

Conclusion

Building an effective language agent capable of handling complex reasoning requires an LLM to deeply understand reasoning states and dynamically select optimal actions. In this paper, we present the Encoder of Thoughts (EoT), an adaptable framework based on graph transformer designed to enhance an LLM's understanding of its planning states. By encoding action graphs into structured representations that align with the LLM's input space, EoT not only improves the model's reasoning capabilities but also extends its applicability to a variety of language agent planning methods. Our experimental results show that EoT significantly improves the performance of language agents in mathematical reasoning and plan generation tasks. By integrating structural embeddings into prompts, EoT reduces the need for fine-tuning or complex prompt engineering, making it a powerful and efficient tool for enhancing LLMs' reasoning and planning in complex tasks.

Acknowledgments

This work is supported by the National Key R&D Program of China (No. 2023YFC3310700) and the National Natural Science Foundation of China (No. 62172094).

References

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17682–17690.

Chai, Z.; Zhang, T.; Wu, L.; Han, K.; Hu, X.; Huang, X.; and Yang, Y. 2023. Graphllm: Boosting graph reasoning ability of large language model. *arXiv preprint arXiv:2310.05845*.

Chen, S.; Li, B.; and Niu, D. 2024. Boosting of thoughts: Trial-and-error problem solving with large language models. *arXiv preprint arXiv:2402.11140*.

Chen, Z.; White, M.; Mooney, R.; Payani, A.; Su, Y.; and Sun, H. 2024. When is tree search useful for llm planning? it depends on the discriminator. *arXiv preprint arXiv:2402.10890*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Du, Y.; Li, S.; Torralba, A.; Tenenbaum, J. B.; and Mordatch, I. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Guo, J.; Du, L.; Liu, H.; Zhou, M.; He, X.; and Han, S. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.

Hao, S.; Gu, Y.; Ma, H.; Hong, J. J.; Wang, Z.; Wang, D. Z.; and Hu, Z. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Hazra, R.; Dos Martires, P. Z.; and De Raedt, L. 2024. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20123–20133.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Huang, X.; Lian, J.; Lei, Y.; Yao, J.; Lian, D.; and Xie, X. 2023. Recommender ai agent: Integrating large language models for interactive recommendations. *arXiv preprint arXiv:2308.16505*.

Jin, B.; Liu, G.; Han, C.; Jiang, M.; Ji, H.; and Han, J. 2023. Large language models on graphs: A comprehensive survey.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Li, N.; Gao, C.; Li, Y.; and Liao, Q. 2023. Large language model-empowered agents for simulating macroeconomic activities. *arXiv preprint arXiv:2310.10436*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Pryzant, R.; Iter, D.; Li, J.; Lee, Y. T.; Zhu, C.; and Zeng, M. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Shi, Y.; Huang, Z.; Feng, S.; Zhong, H.; Wang, W.; and Sun, Y. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*.

Tang, J.; Yang, Y.; Wei, W.; Shi, L.; Su, L.; Cheng, S.; Yin, D.; and Huang, C. 2024. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 491–500.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*.

Wang, S.; Shen, Y.; Feng, S.; Sun, H.; Teng, S.-H.; and Chen, W. 2024. ALPINE: Unveiling the Planning Capability of Autoregressive Learning in Language Models. *arXiv preprint arXiv:2405.09220*.

Wang, X.; Li, C.; Wang, Z.; Bai, F.; Luo, H.; Zhang, J.; Jojic, N.; Xing, E. P.; and Hu, Z. 2023. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Yang, Z.; Li, P.; and Liu, Y. 2023. Failures pave the way: Enhancing large language models through tuning-free rule accumulation. *arXiv preprint arXiv:2310.15746*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

Zhou, A.; Yan, K.; Shlapentokh-Rothman, M.; Wang, H.; and Wang, Y.-X. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI open*, 1: 57–81.