Journal of Computer Technology and Software

ISSN:2998-2383

Vol. 4, No. 3, 2025

Optimizing LLM-Agents with History-Driven Task Planning

Ewan Ritchford

School of Computer Science, University of Windsor, Windsor, Canada

ewan.ritchford@uwindsor.ca

Abstract: The proliferation of intelligent agents based on large language models (LLMs) has shifted the focus towards efficient task planning and tool invocation for complex tasks. Traditionally, agents from scratch for each task, leading to inefficiencies in task planning and tool usage. This study introduces a history-driven task planning method to leverage execution history and enhance task planning efficiency. The method addresses two key issues: the repetitive exploration of task planning paths and the regeneration of intermediate code artifacts in similar tasks. By transforming historical task planning trajectories into actionable experience and automating the generation of reusable modules, the approach significantly reduces LLM token consumption and increases task completion rates. The study also presents the History-based Agent Optimization Kit (HAOK), which integrates seamlessly with existing agents to optimize prompts and enhance performance.

Keywords: Intelligent Agents, Large Language Models, Task Planning, History-Driven Optimization.

1. Introduction

Building intelligent agents (agents) based on large language models has become a trend, with the core process involving constructing prompts to request large language models in order to achieve task planning and tool invocation to complete complex tasks.

Currently, research on agents mainly focuses on the execution of single tasks, which starts from scratch each time: first, task planning is conducted to decompose the task, followed by invoking tools to solve each subtask. However, there is actually transferability between similar tasks, and existing agents are unable to mine experience from execution history to provide references for subsequent tasks, which is specifically manifested in two issues: (1) During the task planning phase, similar tasks still require repeated exploration of task planning paths and cannot leverage previous experience to avoid erroneous exploration. (2) During the tool invocation phase, similar tasks result in the repeated generation of intermediate code artifacts, and cannot accumulate and reuse them. In summary, this leads to frequent calls to large language models and high token consumption. This study addresses the aforementioned issues by analyzing the execution history of agents and carries out the following research work:

(1) To address the issue of repeated exploration of task planning paths for similar tasks, a history-driven task planning method is proposed. The task planning trajectory from execution history is transformed into experience, and related experience is retrieved through semantic similarity in subsequent tasks to preemptively avoid ineffective exploration. Specifically, the following steps are taken: extracting subtask decomposition and task execution plan experience from successful cases to guide the agent in breaking down complex tasks and selecting tools to complete subtasks; at the same time, extracting error-prone point reminders from failed cases. To mitigate the impact of task description diversity on similarity retrieval, a data augmentation scheme is designed in this paper to automatically generate task descriptions from multiple perspectives.

(2) To address the issue of repeatedly generating the same intermediate code artifacts for similar tasks, a method for intelligently generating reusable modules is proposed. The intermediate code from execution history is transformed into general-purpose modules, and when executing similar tasks in the future, related modules are automatically retrieved to directly invoke modules to solve subtasks, rather than regenerating intermediate code. Modules are ensured to be correct through automatic testing and can be further refactored and merged to shorten the invocation chain of large language models. With continued use by users, the agent will gradually accumulate modules rich in domain and scenario knowledge.

(3) Based on the above research findings, a History-based Agent Optimization Kit (HAOK) is constructed. HAOK asynchronously converts execution history into experience (including related module recommendations and related task planning experience) in the background through the aforementioned two methods and provides two interfaces for the agent to upload execution history and obtain experience. HAOK can be seamlessly integrated with existing agents without changing the core architecture of the agent; it automatically optimizes prompts by calling interfaces.

Finally, this study applies HAOK to practice by developing a personal assistant agent. Experiments show that HAOK can effectively reduce the number of ineffective

planning explorations by the agent and prevent the agent from repeatedly generating intermediate code artifacts, thereby reducing the consumption of large language model tokens and improving the task completion rate.

3. Background

Large Language Models (LLMs) refer to language models with a vast number of parameters, typically on the scale of billions or even trillions [1]. These models are trained on extensive textual data and fine-tuned with human feedback to align their responses [2], ultimately achieving powerful natural language understanding and reasoning capabilities.

In November 2022, OpenAI released ChatGPT, which is based on GPT-3.5, marking a milestone in the development of large language models. ChatGPT is capable of operating according to human instructions and engaging in natural, coherent conversations. It contains a wealth of knowledge and possesses strong contextual understanding capabilities. Its emergence has demonstrated that a significant increase in the scale of parameters can lead to a leap in model capabilities, sparking a surge of interest in large language models.

Wei et al. formally defined the emergent abilities of large language models: "abilities that do not exist in small models but emerge in large models"[3]. Emergent abilities typically manifest in three ways: In-Context Learning (ICL), instruction following, and step-by-step reasoning.

In-Context Learning was officially introduced by GPT-3, allowing the model to produce expected outputs without further training by providing it with a natural language prompt or several task demonstrations. This concept was exemplified by Brown et al. with few-shot demonstrations and later extended to zero-shot by Kojima et al. [4]. Instruction following refers to the fine-tuning of large language models using multi-task datasets described in natural language (i.e., instruction tuning), enabling the models to understand new task instructions without explicit examples [5].

Step-by-step reasoning is exemplified by the Chain-of-Thought (CoT) prompting strategy [6], where large language models can perform complex reasoning by introducing multiple intermediate steps, particularly showing significant improvements in solving mathematical problems. Zhang et al. further proposed Auto-CoT, which can automatically generate demonstrations for the chain of thought without manual creation by humans [7-8] .Yao et al. introduced the concept of thought trees, extending the chain of thought to a tree structure and combining it with search algorithms like DFS [9]. Besta et al. further proposed thought graphs, introducing integration, refinement, and generation actions on graph nodes. Sel et al. proposed thought algorithms, combining human intuitive abilities with algorithmic rationality, attempting to mimic the human thought process when solving complex problems by exploring ideas in layers and filtering out infeasible options [10]. Chen et al. proposed thought procedures, separating reasoning for computational and numerical reasoning tasks [11]. Wang et al. introduced table chains, effectively utilizing tabular data in reasoning chains [12].

In-context learning, instruction following, and thought chains endow intelligent agents based on large language models with boundless potential.

4. Method

Task planning is essential for intelligent agents to handle complex scenarios efficiently, particularly when powered by large language models (LLMs). As demonstrated in the work of Wang et al. [13], optimizing model efficiency through techniques like knowledge distillation directly impacts how agents manage planning-related computations. Similarly, Du et al. [14] emphasized the importance of addressing semantic complexity in language understanding, a core component in enabling agents to interpret goals and constraints during task execution. Fang et al. [15] further highlighted the significance of contextual and semantic modeling, which supports more accurate prediction and selection of optimal actions in a task plan. Together, these contributions inform our understanding of how LLM-based agents perform task decomposition, predict potential actions, and solve complex tasks. As a result, numerous agent task planning strategies have been developed to enhance these capabilities.

The concept of history-driven task planning, as proposed in this study, aligns with the memory-enhanced planning approach mentioned in the literature. This method leverages historical cases to improve the agent's ability to transfer learning across tasks. It not only capitalizes on successful

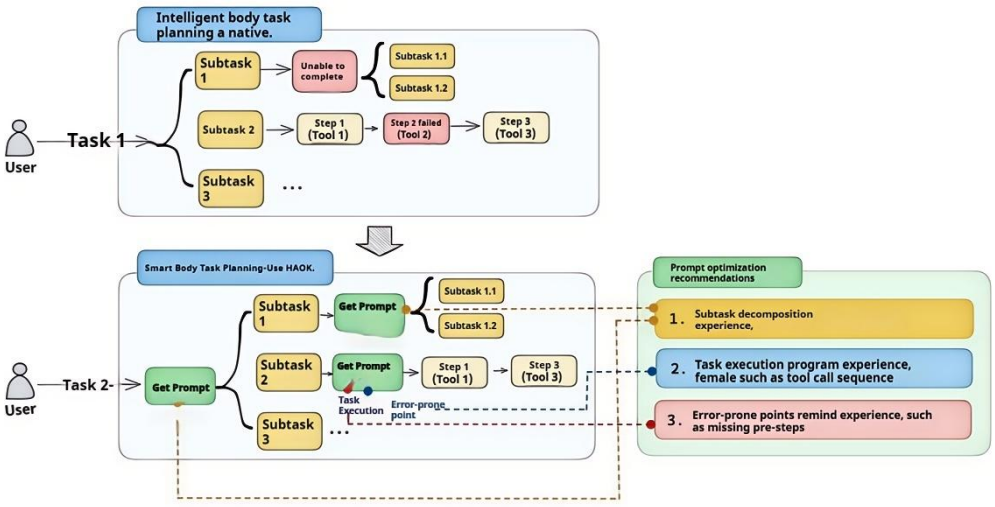

Figure 1. Optimization of Task Planning through "History-Driven Task Planning"

cases but also extracts lessons from failures, which can serve as reminders of common pitfalls. This approach is particularly innovative as it addresses the gap in leveraging past experiences to guide current and future task planning, enhancing the agent's ability to learn from history and apply that knowledge to new, yet similar, tasks. By doing so, it equips agents with a "battle manual" of sorts, allowing them to navigate through tasks more efficiently by building upon previous experiences and learned strategies.

4.1 Agent Architecture

Our history-driven task planning method allows agents to extract actionable experience from previous executions, grouped into three main categories: subtask decomposition, task execution plans, and error-prone point reminders. The first two categories are derived from successful execution cases, helping agents reuse effective breakdowns and planning paths. The third category captures recurring mistakes from failed attempts, offering reminders to help avoid similar errors in future planning.

To improve consistency across differently phrased task descriptions, we introduce a data augmentation strategy aimed at enhancing semantic retrieval. This helps reduce the influence of linguistic variation on how relevant historical examples are matched. This component is informed by prior work on adaptive attention mechanisms and enhanced embedding techniques [16], model efficiency optimization in large-scale training [17], and retrieval-augmented generation for complex reasoning using graph-based structures [18]. These contributions collectively support more robust representation learning and improved semantic alignment within our planning framework.

Subtask decomposition experience helps the agent determine how to break down a task based on its complexity. For straightforward tasks, a linear sequence of one or several steps may be sufficient. However, more complex tasks require deeper planning, where the agent must first divide the task into multiple subtasks, solve them step by step, and potentially further decompose subtasks into smaller components. This results in a hierarchical, tree-like structure of planning and execution. This approach is influenced by recent advancements in multi-task learning [20], hierarchical modeling [21], and semantic guidance strategies [22], which highlight how complex objectives can be effectively handled through structured decomposition and layered reasoning [23]. These insights support the design of agents that can adapt their planning depth to match task difficulty, improving overall flexibility and accuracy.

Task Execution Plan Experience: This experience guides the agent on how to efficiently complete the current task. For example, to solve Task A, the agent might sequentially invoke Tool 1, Tool 2, and Tool 3.

Error-Prone Point Reminder Experience: This experience guides the agent to minimize errors. For instance, certain tools may require extra attention to their parameters, or specific execution sequences may have particular prerequisites.

Figure 1 illustrates how this method optimizes the original agent task planning. In the example, Task 1 and Task 2 are similar tasks, and the role of HAOK in the entire task planning process is as follows.

Suggest the agent not to directly complete Subtask 1, but to break it down into Subtasks 1.2 and 1.3. This is because when the agent first executed the task, it found that Subtask 1 was too large to be completed directly.

Advise the agent to sequentially invoke Tools 1 and 3 when completing Subtask 2, skipping the erroneous and ineffective attempts on Tool 2.

4.2 Implementation Method

The extraction of task planning experience from the agent's execution history is primarily achieved through the following three sub-strategies:

Analysis of Successful Cases: Analyze the task planning trajectories in successful cases, which can be structured (tree-based) or unstructured (text-based), to extract multiple subtask decomposition experiences and task execution plan experiences.

Error-Prone Point Reminders from Failed Cases: For failed cases or execution flows containing failed sub-steps, leverage large language models to analyze and identify error-prone points, such as missing parameters, omitted necessary steps, etc.

Data Augmentation: Use data augmentation to mitigate the interference caused by the diversity of task descriptions on retrieval.

Figure 2 illustrates how the three sub-strategies of HAOK (History-Driven Task Planning) work together to extract relevant experiences from task planning trajectories.

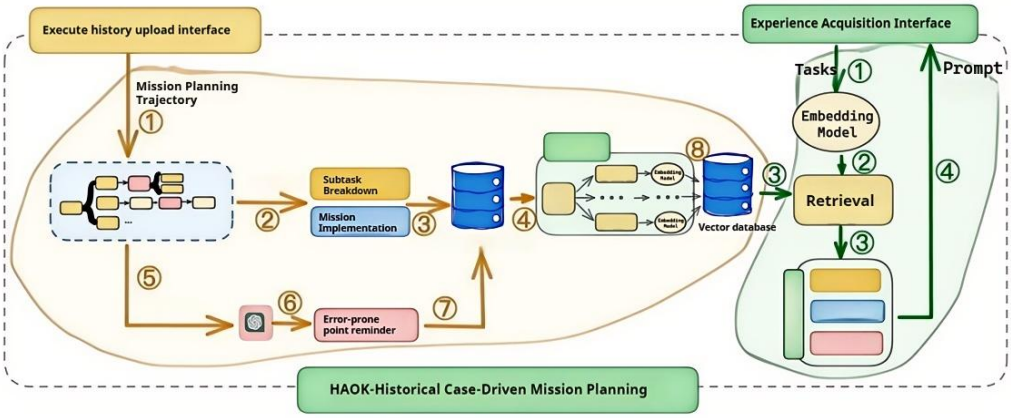

Figure 2. History-Driven Task Planning - Internal Sub-Strategies

Extraction of Subtask Decomposition Experience

This sub-strategy has the following requirements for the format and content of task planning trajectories:

Supports two types of task planning trajectories: structured (tree-based) and unstructured (text-based).

Task planning trajectories must include the following information: task description, status (success or failure), tools invoked, tool execution results, and parent-child task relationships (only required for tree-based structures).

For structured (tree-based) task planning trajectories, the experience extraction method is as follows:

Employ a purely programmatic strategy to analyze structured task planning trajectories, extracting as much task planning experience as possible from the tree structure to provide suggestions for subsequent similar tasks or subtasks.

Tree node classification: Divide the nodes of the tree into two categories:

Tool nodes: Represent tool invocations.Task nodes:

If its child nodes are several tool nodes, then all its child nodes represent the list of tools that have been invoked in sequence to complete that task.

If its child nodes are several task nodes, then all its child nodes represent the multiple subtasks that the task was decomposed into when completed.

Fine-grained case reuse: To extract as much experience as possible from execution history, this study adopts a fine-grained, multi-level extraction approach. The entire task planning is a complete tree, but this method does not directly save the whole tree; instead, it traverses each subtree and generates experience from each. By fine-grained dissection of cases, not only can it provide path planning references for overall similar tasks, but it can also provide references for small, similar parts within complex tasks. This enables the agent's task planning learning to be transferable, allowing it to reference experiences from similar subtasks even when encountering entirely new tasks.

HAOK can learn not only from successful cases but also from failed cases to draw lessons from mistakes. Agents sometimes fail for various unexpected reasons, which may stem from systematic issues in task planning, complex external environments, implicit domain knowledge, missing prerequisites, and more. Since these errors contain a wealth of semantic knowledge, HAOK uses large language models with natural language understanding capabilities to summarize and induce error-prone point reminders. By employing this anticipatory and strategic learning method, it can reduce the number of erroneous attempts, adapt to more complex environments, and accumulate more domain knowledge. Figure 3 illustrates the key steps in error-prone point analysis. Similar to the analysis of successful cases, multiple "error-prone point reminders" can be extracted from the same execution history. For tree-based trajectories, extract all abnormal subtasks (for text-based, select all logs), fill them into the "Error-Prone Point Analysis Prompt," request the large language model, and parse the output. Finally, store them in the experience repository in the form of "Task - Error-Prone Point Reminder" key-value pairs.

5. Experiment

5.1 Settings

To validate the performance of the history-driven task planning method in improving the efficiency of agent task planning, the experiment utilized the ALFWorld dataset, proposed by Mohit Shridhar et al. in 2020. The ALFWorld dataset encompasses a variety of tasks that require the agent to navigate and operate within a simulated home environment. Agents first interact with the environment through text actions in TextWorld, which are then directly mapped to specific, vision-based environments. This design allows agents to explore, interact, and learn in an abstract linguistic environment before encountering complex embodied environments.

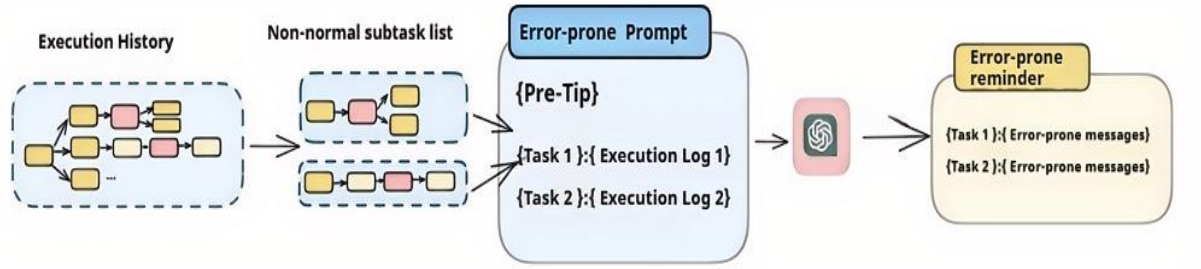

Figure 3. Error-Prone Point Analysis Process

Selection of Large Language Models: The experiment employed OpenAI's GPT-3.5-turbo model API. The temperature coefficient for the large language model was set to 0. The embedding model used was OpenAI's text-embedding-3-small. For case reference, the top 10 most relevant task planning experiences were retrieved for each task as references. The maximum number of calls to the large language model for the same task was capped at 30; if the task was not resolved after 30 calls, the attempt was terminated, and the task was marked as failed.

5.2 Experiment

Table 1: Performance of Two Agents on ALFWorld Tasks

Agents	SuccessRate	Average LLM Invocations	Task Completion Rate
ReAact	68.60 %	20.3	52.2%
ReAct-HAOK	68.60 %	18.6	81.6%

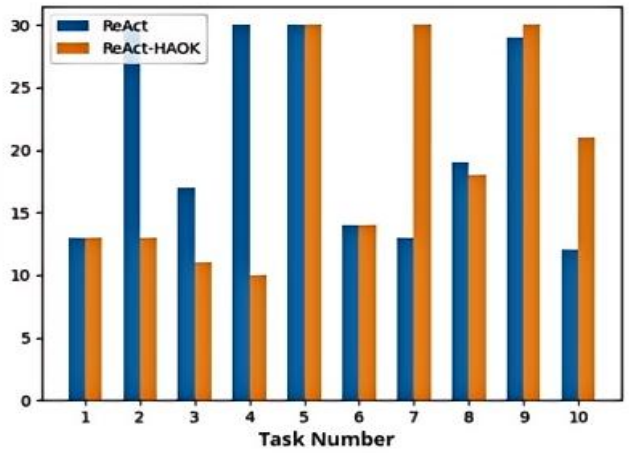

Figure 4. Comparison of LLM Call Counts (Total Task Number 10)

When the task number is 30 and 60, there is a significant increase in task completion rates and a sharp decrease in LLM call counts: when the task number is 30, the completion rate increased from 53.33% to 83.33%, and the LLM call count decreased by about 7; when the task number is 60, the completion rate increased from 45.0% to 75.0%, and the LLM call count decreased by about 5. This demonstrates that "history-driven task planning" can effectively improve task planning efficiency. When the task number is 10, the two agents perform almost identically: this is due to the insufficient sample size and lack of experience accumulation.

Querying the experimental logs to analyze HAOK's specific performance:

HAOK can accumulate more task-related knowledge; for example, ReAct often searches for a pan in places like drawers and refrigerators, which leads to an increase in the number of call rounds and ultimately task failure. The steps generated by HAOK for finding a pan are: ["go to stoveburner 2" and "take pan 1 from stoveburner 2"]. Although "stoveburner 2" may not exist in future tasks, this allows the agent to first try to find the pan from the stove, reducing incorrect attempts.

HAOK can detect some errors: for example, agents often forget to perform the pick-up action after finding an item and then rush directly to the next location, leading to failure.

The experiment shows that the history-driven task planning method (1) can optimize the task planning path, thereby reducing the number of calls to large language models, which can improve agent efficiency and save costs. (2) can effectively increase the task completion rate. This is due to two reasons: on the one hand, HAOK can introduce more task-related experience; on the other hand, because this experiment set an upper limit on the number of calls to large language models, when the useless steps in front are shortened, the probability of completing the task within a limited number of steps will also increase.

6. Conclusion

This paper offers a comprehensive overview of the history-driven task planning method, detailing its foundational principles, implementation techniques, and empirical validation. This method adeptly extracts nuanced insights from successful cases, including fine-grained subtask decomposition and task execution strategies, while also distilling error-prone point reminders from unsuccessful ones. The experimental results highlight the method's key benefits: it significantly cuts down on futile task planning explorations, thereby solidifying successful practices and steering clear of repeated mistakes. Moreover, it reduces the reliance on large language model invocations and concurrently boosts task completion rates, showcasing its efficacy in enhancing task planning efficiency and success.

References

[1] Zhao W X, Zhou K, Li J, et al. A survey of large language models[J]. arXiv preprintarXiv:2303.18223, 2023.

[2] Ouyang L, Wu J, Jiang X, et al. Training language models to follow instructions with human feedback[J]. Advances in Neural Information Processing Systems, 2022, 35: 27730-27744.

[3] Wei J, Tay Y, Bommasani R, et al. Emergent abilities of large language models[J]. arXiv preprint arXiv:2206.07682, 2022.

[4] Kojima T, Gu S S, Reid M, et al. Large language models are zero-shot reasoners[J]. Advances in neural information processing systems, 2022, 35: 22199-22213.

[5] Sanh V, Webson A, Raffel C, et al. Multitask Prompted Training Enables Zero-Shot Task Generalization[C]//ICLR 2022-Tenth International Conference on Learning Representations. 2022.

[6] Zhang Z, Zhang A, Li M, et al. Automatic chain of thought prompting in large language models[J]. arXiv preprint arXiv:2210.03493, 2022.

[7] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in Neural Information Processing Systems, 2022, 35: 24824-24837.

[8] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.

[9] Yao S, Yu D, Zhao J, et al. Tree of thoughts: Deliberate problem solving with large language models[J]. Advances in Neural Information Processing Systems, 2024, 36.

[10] Sel B, Al-Tawaha A, Khattar V, et al. Algorithm of thoughts: Enhancing exploration of ideas in large language models[J]. arXiv preprint arXiv:2308.10379, 2023.

[11] Chen W, Ma X, Wang X, et al. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks[J]. arXiv preprint arXiv:2211.12588, 2022.

[12] Wang Z, Zhang H, Li C L, et al. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding[C]//The Twelfth International Conference on Learning Representations. 2023.

[13] Wang, S., Wang, C., Gao, J., Qi, Z., Zheng, H., & Liao, X. (2024). Feature Alignment-Based Knowledge Distillation for Efficient Compression of Large Language Models. arXiv preprint arXiv:2412.19449.

[14] Du, J., Jiang, Y., & Liang, Y. (2024). Transformers in Opinion Mining: Addressing Semantic Complexity and Model Challenges in NLP. Transactions on Computational and Scientific Methods, 4(10).

[15] Fang, Z., Zhang, H., He, J., Qi, Z., & Zheng, H. (2025). Semantic and Contextual Modeling for Malicious Comment Detection with BERT-BiLSTM. arXiv preprint arXiv:2503.11084.

[16] Wu, L., Gao, J., Liao, X., Zheng, H., Hu, J., & Bao, R. (2024, December). Adaptive attention and feature embedding for enhanced entity extraction using an improved BERT model. In 2024 4th International Conference on Communication Technology and Information Technology (ICCTIT) (pp. 702-705). IEEE.

[17] Chen, J., Liu, B., Liao, X., Gao, J., Zheng, H., & Li, Y. (2024). Adaptive Optimization for Enhanced Efficiency in Large-Scale Language Model Training. arXiv preprint arXiv:2412.04718.

[18] Dong, Y., Wang, S., Zheng, H., Chen, J., Zhang, Z., & Wang, C. (2024, November). Advanced RAG Models with Graph Structures: Optimizing Complex Knowledge Reasoning and Text Generation. In 2024 5th International Symposium on Computer Engineering and Intelligent Communications (ISCEIC) (pp. 626-630). IEEE.

[19] Qi, Z., Chen, J., Wang, S., Liu, B., Zheng, H., & Wang, C. (2024). Optimizing Multi-Task Learning for Enhanced Performance in Large Language Models. arXiv preprint arXiv:2412.06249.

[20] Liao, X., Zhu, B., He, J., Liu, G., Zheng, H., & Gao, J. (2025). A Fine-Tuning Approach for T5 Using Knowledge Graphs to Address Complex Tasks. arXiv preprint arXiv:2502.16484.

[21] Sun, D., He, J., Zhang, H., Qi, Z., Zheng, H., & Wang, X. (2025). A LongFormer-Based Framework for Accurate and Efficient Medical Text Summarization. arXiv preprint arXiv:2503.06888.

[22] Hao, R., Xiang, Y., Du, J., He, Q., Hu, J., & Xu, T. (2025). A Hybrid CNN-Transformer Model for Heart Disease Prediction Using Life History Data. arXiv preprint arXiv:2503.02124.

[23] Gao, J., Lyu, S., Liu, G., Zhu, B., Zheng, H., & Liao, X. (2025). A Hybrid Model for Few-Shot Text Classification Using Transfer and Meta-Learning. arXiv preprint arXiv:2502.09086.