ADVANCING REASONING AND PLANNING IN LARGE LANGUAGE MODELS VIA REWARD SHAPING

A Dissertation
Presented to
The Academic Faculty

By

Yuchen Zhuang

In Partial Fulfillment
of the Requirements for the Degree
in the
School of Computer Science and Engineering
College of Computing

Georgia Institute of Technology

August 2025

ADVANCING REASONING AND PLANNING IN LARGE LANGUAGE MODELS VIA REWARD SHAPING

Thesis committee:

Dr. Chao Zhang (Advisor)
Computer Science and Engineering
Georgia Institute of Technology

Dr. Sherry Yang
Google DeepMind
Google

Dr. Bo Dai
Computer Science and Engineering
Georgia Institute of Technology

Dr. Steve Mussmann
School of Computer Science
Georgia Institute of Technology

Dr. Tuo Zhao
Industrial and Systems Engineering
Georgia Institute of Technology

Date approved: June 30th, 2025

I could be bounded in a nutshell and count myself a king of infinite space.

William Shakespeare, Hamlet

For my parents

ACKNOWLEDGMENTS

This dissertation would not have been possible without the many individuals who have illuminated my path. First and foremost, I am profoundly grateful to my Ph.D. advisor, Prof. Chao Zhang. His unwavering guidance was the cornerstone of my doctoral studies. From the spark of an idea to the rigors of validation, he guided me through every step from a beginner in machine learning to a dedicated researcher in natural language processing, shaping my ability to conduct meaningful research. He is the best mentor ever not only for my research, but also for my career and life. Thank you, Prof. Zhang, for your unconditional support, encouragement, and trust.

I extend my sincere gratitude to my thesis committee members: Dr. Bo Dai, Dr. Tuo Zhao, Dr. Steve Mussmann, Dr. Sherry Yang. Their insightful feedback and challenging questions were instrumental in sharpening my statements and elevating the quality of this work. A special acknowledgment goes to Dr. Bo Dai, one of my closest and most influential mentors. I have always been inspired by his passion for research, his intellectual depth, his comprehensive grasp of foundational theory, and his sharp insights into emerging topics. His guidance has profoundly shaped my research philosophy and my aspiration to conduct impactful work.

My research was also deeply enriched by industrial internships, where I had the privilege of learning from exceptional mentors. I am grateful to Tong Zhao and Xin Shen at Amazon Research; Xiang Chen and Tong Yu at Adobe Research; Jingfeng Yang, Haoming Jiang, and Xin Liu at Amazon Rufus Team; Di Jin at Meta; and Sherry Yang at Google Deepmind. They not only guided me through exciting projects but also provided invaluable support for my career development, for which I am sincerely appreciative.

I have been incredibly fortunate to collaborate with a community of brilliant researchers. My thanks go to Hanjun Dai, Le Song, Hanrui Wang, Jiaao Chen, Ran Xu, Jieyu Zhang, Percy Liang, Qifan Wang, Wei Wei, Aditya Prakash, Yu Meng, Yuanqi Du, Rampi Ramprasad, Robert Tang, Yanqiao Zhu, Hang Wu, May D. Wang, and Carl Yang for their valuable guidance and contributions to our shared projects. I also want to thank my labmates at Georgia Tech: Lingkai Kong, Yue Yu, Haotian Sun, Rushi Qiang, Changhao Li, Yinghao Li, Haorui Wang, Kuan Wang, Rongzhi Zhang, Rui Feng, Binghong Chen, Ruijia Wang, Wendi Ren, Yanbo Xu, Jerry Junyang Cheung, and Aaron Trinh. Special thanks to Lingkai and Yue for sharing the laughs and tears, and to Haotian, Rushi, and Changhao for those unforgettable late nights. It has been a true pleasure sharing this journey with all of you.

Finally, I owe my deepest gratitude to my family. To my parents, Lei Zhu and Bin Zhuang, thank you for your unconditional love and support across the distance. Your encouragement and belief in me have been a constant source of strength. None of this would have been possible without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Recent advancements in large language models (LLMs) have significantly enhanced their reasoning and planning capabilities, enabling them to serve effectively in complex, real-world scenarios. Despite these improvements, achieving human-level performance remains challenging, particularly for tasks requiring extensive multi-step reasoning and sophisticated planning. Motivated by these limitations, my dissertation focuses on improving the reasoning and planning abilities of LLMs through reward shaping to guide LLM decision-making by optimizing rewards for desired outcomes.

The core contributions of this thesis are organized around three key aspects of effective and robust reasoning in LLM agents:

Thrust 1: Formulating and Evaluating LLM-based Agents for External Tool Use. Effectively leveraging external tools is crucial for extending the practical utility of LLMs. To systematically assess this capability, I developed ToolQA, a benchmark dataset created through a scalable, automated three-phase process involving reference data collection, human-guided question generation, and programmatic answer generation. This benchmark rigorously evaluates whether LLMs truly utilize external tools rather than relying solely on memorized information, highlighting key areas such as argument correctness, data source accuracy, and tool composition complexity.

Thrust 2: Efficient Action Space Navigation in LLM Agents. The complexity of multi-step planning tasks, involving numerous candidate actions, demands efficient exploration strategies. Addressing this, I proposed ToolChain*, an innovative A* tree search-based algorithm specifically designed for LLM-based agents. ToolChain* strategically expands potential solution paths using cumulative and heuristic future cost functions to minimize error propagation and effectively balance exploration and exploitation. Through comprehensive empirical evaluation, ToolChain* demonstrates superior performance and efficiency across diverse planning and reasoning benchmarks, notably improving upon existing open-loop,

closed-loop, and tree search-based methods.

Thrust 3: Lightweight Adaptation for Black-Box LLM Personalization. The practical deployment of LLMs often involves adapting models to specific users without access to internal model parameters. I developed Hydra, a lightweight, retrieval-augmented personalization framework for black-box LLMs. Hydra employs a retrieve-then-rerank strategy coupled with a personalized adapter that aligns model outputs to individual user preferences. By decomposing the personalization process into shared and user-specific components, Hydra effectively balances generalization and personalization, achieving robust performance gains across multiple user-centric tasks in comprehensive personalization benchmarks.

Together, these thrusts represent a cohesive, data-centric strategy for enhancing LLM capabilities, systematically improving their ability to reason, plan, and adapt efficiently in complex, real-world environments.

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Challenges

Large language models (LLMs), such as GPT [1, 2, 3, 4] and PaLM [5, 6], have exhibited remarkable capabilities of reasoning and instruction-following across a wide range of tasks [7]. Recently, instructing LLMs to utilize external tools for complex real-world problems has emerged as a topic of growing importance [8, 9, 10, 11, 12, 13]. Traditional decision-making methodologies like Reinforcement Learning (RL) require extensive task-specific training data and often lack the ability to generalize across tasks and environments. In contrast, LLMs are pre-trained on massive and diverse textual data, which gives them extensive world knowledge and the ability to reason over the knowledge. This makes them highly versatile and able to handle complex, real-world scenarios that may involve multiple steps of planning and decision-making.

In an LLM-powered autonomous agent system, LLM functions as the agent's brain, complemented by several key components, including reasoning, planning, tool-use, and memory. For complicated tasks, LLM-based autonomous agents integrate LLMs with various external tools (APIs), generating solutions that involve intermediate reasoning steps [12, 13, 14, 15]. Given a problem description, an autonomous agent is expected to determine a chain of API function calls that can be executed sequentially toward a valid solution. However, developing effective and robust agents for complex, real-world tasks poses several challenges for existing LLMs:

Firstly, LLMs still fall short of human-level performance, particularly in tasks requiring extensive planning and multi-hop reasoning. From the reasoning and planning perspective, existing methods that utilize LLMs as autonomous agents can be categorized into open-loop and closed-loop systems. Open-loop methods rely on pre-determined plans without feedback adaptation, while closed-loop systems incorporate environment feedback to monitor and refine plans accordingly. However, both approaches have drawbacks: open-loop systems lack adaptability and can generate suboptimal plans, while most closed-loop methods only update executing actions based on short-term feedback, potentially leading to detrimental long-term implications.

Secondly, complex, real-world problems require multi-step solutions with numerous candidate tools to consider at each step, further compounding the difficulty of finding an optimal solution. From tool-use perspective, LLM-based autonomous agents can interact with diverse tools and generate solution plans that execute a series of API function calls in a step-by-step manner. The multitude of candidate API function calls significantly expands the action space, amplifying the critical need for efficient action space navigation. Given an action space of hundreds of candidate API functions, each comprised of various function names and parameters available at every planning step, searching for a globally optimal solution becomes highly challenging.

Thirdly, adapting black-box LLMs to personalized, domain-specific tasks is only possible through API services, raising concerns about transparency, privacy, and cost. Due to the opacity in their parameters, embeddings, and even output probabilities, existing fine-tuning adaptation methods are inapplicable. The fine-tuning process of LLMs lacks transparency in crucial aspects like the extent of trainable layers and specific model weights, limiting the ability to optimally customize the models beyond adjusting a restricted set of hyperparameters. Uploading training data via APIs also introduces potential risks of privacy breaches, limiting the use of LLMs in sensitive domains. Furthermore, the cost associated with fine-tuning APIs is considerably higher compared to inference, making the adaptation expensive.

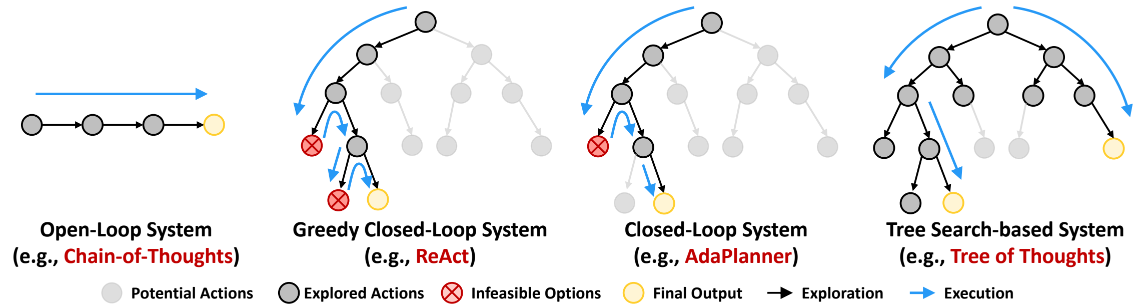


Figure 1.1: A comparison of existing methods that leverage LLMs for decision-making from a searching space perspective. Most existing methods of (1) open-loop systems (*e.g.*, Chain-of-Thought [16]), (2) greedy closed-loop systems (*e.g.*, ReAct [17]), and (3) closed-loop systems (*e.g.*, Adaplanner [18]) only explore one possible direction. This often leads to limited exploration of the entire action space. In contrast, (4) tree search-based methods (*e.g.*, Tree-of-Thoughts [19]) identify a valid solution path by extensively examining multiple decision space branches, covering almost every conceivable node.

1.2 LLMs for Planning and Reasoning

1.2.1 Open-Loop System

Open-loop Systems [16, 20] (Figure 1.1) generate pre-defined plans to explore over coherent intermediate steps toward problem-solving. Given the initial states s_0, which are usually the problem description x in the format of natural language, the systems generate the entire T-step plans as the solution $\rho(a_1, a_2, \cdots, a_T | s_0) : \mathcal{S} \rightarrow \Delta(\mathcal{A}^T)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\Delta(\cdot)$ is probability simplex function. The final output is obtained after executing the entire plan $y \sim p(y|s_0, a_1, a_2, \cdots, a_T)$.

LLMs for Open-Loop Planning. Trained on vast world knowledge and human examples, LLMs have emerged with smart planning and decision-making capabilities. Leveraging LLMs as autonomous agents to accomplish decision-making tasks has gained attention and shown potential. Earlier studies apply the open-loop framework to directly generate the entire plan as the solution. One line of works, including Chain-of-Thought [16] and Zero-Shot Planner [21], generate intermediate reasoning steps all at once to solve the problem. Another line of works selects the opposite direction (*e.g.*, least-to-most prompting [20]) that decomposes the complicated problems into relatively easier sub-problems. For more complex tasks, methods like HuggingGPT [22] and Chameleon [13] incorporate a set of functional tools and directly generate the plan of API calls in a sequence for execution. However, all these aforementioned methods explore a predetermined single path in the decision space, leaving the rest potential plans not considered when solving the problems.

1.2.2 Closed-Loop System

Closed-loop Systems follow a more step-by-step plan modification and execution manner, interleaving intermediate observations with decisions over the action space. When the agent interacts with the environment at the t-th step, the agent is aware of the current observation $o_t \in \mathcal{O}$ from the environment and generates a trajectory-like context $c_t = (s_0, a_1, o_1, a_2, \cdots, o_t)$. In tool-use scenarios, the intermediate observations are obtained during the execution of the previous actions in the plan $o_t \sim \pi(o_t|s_0, a_1, o_1, \cdots, a_{t-1})$. According to the environment feedback o_t, two levels of refinements can be applied: greedy closed-loop methods [17, 23] only decide the next single step $a_t \sim \rho(a_t|s_0, c_t; \mathcal{I}, \mathcal{D})$, while the standard ones [24, 25] will modify the entire future plans $\pi(a_{t+1}, a_{t+2}, \cdots, a_{T_g}|s_0, c_t; \mathcal{I}, \mathcal{D})$.

LLMs for Closed-Loop Planning. Inspired by traditional reinforcement learning approaches that heavily rely on interaction with the environment (either simulator or real world), researchers start to improve the plan via refinement according to the environment feedback. Initially, ReAct [17] and Inner Monologue [26] allow LLM agents to refine a single step greedily according to the environment feedback. Considering that solely modifying the immediate action being executed is easy to fall into local sub-optimal actions, improvements like DEPS [24] and AdaPlanner [18] are proposed to recursively modify the entire future plans according to the environment feedback. However, without a tracing-back mechanism to check the already executed plans, these efforts in the closed-loop framework still only explore a small proportion of decision space.

1.2.3 Tree Search-Based System

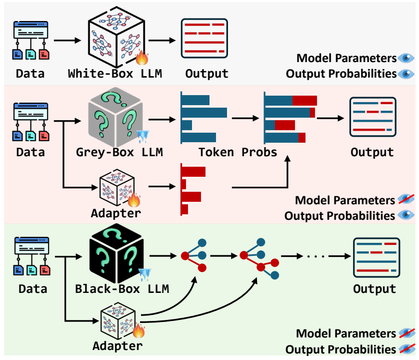
Tree Search-based System. Tree search methods frame a planning problem as a search over a decision tree, where each node n represents an action a_n, accompanied by a state $s_n \in \mathcal{S}$ indicating a valid path from the initial state to the current action. When exploring the tree space, tree search approaches expand k potential child nodes $ch(n)$ of the current node n via sampling from the potential action set generated by LLMs $a_{ch(n)}^{(j)} \sim \rho(a_{ch(n)}|s_n; \mathcal{I}, \mathcal{D}), (j = 1, \cdots, k)$ and add the new nodes to the tree state space $\mathcal{S} = \mathcal{S} \cup \{(s_n, a_{ch(n)}^{(j)})\}_{j=1}^{k}$. With value functions for state evaluation, tree search-based methods aim to identify a path from the root node s_0 to the leaf nodes with the highest value or lowest cost.

LLMs for Tree-Search Planning. Tree search-based methods [19, 27] for leveraging multiple reasoning paths simultaneously and evaluating branches to decide the next course of action. However, existing tree search-based algorithms, such as depth-first search (DFS) [19] and Monte Carlo Tree Search (MCTS) [27], require exhaustive exploration of nearly all potential actions within the entire decision space, resulting in **inefficient searches** for globally optimal solutions.

1.3 LLM Adaptation

1.3.1 White-Box LLM Fine-Tuning

To fully leverage the capabilities of LLMs in language comprehension and enhance their performance, many users still need to customize them for specific tasks and domains [28]. A straightforward approach involves fine-tuning [29, 30] or continuous pre-training [31, 32] the LM on domain-specific data (Figure 1.2). However, these methods require extensive computational resources and memory, which becomes increasingly challenging as model sizes grow exponentially. To mitigate the computational and memory burdens for LLM fine-tuning, parameter-efficient fine-tuning methods [33, 34, 35, 36] have been pro-



Methods	w/o Model Parameters	w/o High-Dimensional Representation	w/o Token Probabilities
White-Box LLM Fine-Tuning			
Fine-Tuning (Devlin et al., 2019)	✗	✗	✗
Instruction-Tuning (Wei et al., 2021)	✗	✗	✗
Continual Pre-Training (Gururangan et al., 2020)	✗	✗	✗
Adapter (Houlsby et al., 2019)	✗	✗	✗
Prefix-Tuning (Liu et al., 2022)	✗	✗	✗
LoRA (Hu et al., 2021)	✗	✗	✗
Grey-Box LLM Adaptation			
LMaaS (Sun et al., 2022)	✓	✗	✗
kNN-Adapter (Huang et al., 2023)	✓	✓	✗
CombLM (Ormazabal et al., 2023)	✓	✓	✗
Proxy-Tuning (Liu et al., 2024)	✓	✓	✗

Figure 1.2: Illustration of *white-box*, *grey-box*, and *black-box* LLM adaptation. White-box adaptation has complete access to both model parameters and output probabilities, grey-box adaptation has access only to output probabilities, and black-box adaptation has access to neither model parameters nor output probabilities.

posed that focus on training only a small subset of parameters rather than the entire model. Examples of such techniques include adapters [34], prefix tuning [37, 36], and low-rank adaptation [33]. Unfortunately, these techniques require direct access to the internal parameters of the original model and complete backward passes, making them incompatible with black-box models.

1.3.2 Grey-Box LLM Adaptation

For grey-box LLM adaptation, existing approaches make different assumptions about the transparency of the LLM. One line of research assumes that only the gradient information is unavailable, while the high-dimensional input and output sequences are accessible. For example, LMaaS [38] trains a small, derivative-free optimizer for discrete prompt tuning to enhance the probabilities of ground-truth tokens from the target domain. Another line of research assumes that only output token probabilities from black-box LLMs are available. kNN-Adapter [39] augments a black-box LLM with KNN retrieval from an external, domain-specific datastore. It adaptively interpolates LM outputs with retrieval results from the target domain. CombLM [40] employs fine-tuning on a smaller white-box model to align the output token probabilities of a black-box LLM with the target distribution. Similarly, proxy-tuning [41] fine-tunes a smaller LM as an 'expert' while its untuned version serves as an ‘anti-expert’. The method involves adjusting the black-box LLM outputs by adding the logit offsets from their token-level predictions for adaptation. CaMeLS [42] meta-trains a compact, autoregressive model to dynamically adjust the language modeling loss for each token during online fine-tuning. However, these methods are inapplicable to the latest state-of-the-art black-box LLMs, such as GPT-4 [4] and PaLM2 [6], due to the inaccessibility of token probabilities.

1.3.3 Black-Box LLM Adaptation

Due to the black-box nature, users are unable to access (1) internal model parameters, (2) high-dimensional representations of input sequences or output generations, and (3) output token probabilities for their specific use cases in black-box adaptation. Notably, existing methods all fail to support *black-box* LLM adaptations, where neither model parameters nor output probabilities can be accessed in most recent LLMs like GPT-3.5 [43] and Gemini [44].

CHAPTER 2
ON FORMULATING AND EVALUATING LLM-BASED AGENTS FOR EXTERNAL TOOL USE

This chapter focuses on developing a scalable, automated process for dataset curation to faithfully evaluate LLMs' ability to use external tools for question answering. The content is mainly based on the following publications:

- **Yuchen Zhuang**, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. "Toolqa: A dataset for llm question answering with external tools." In *The Thirty-seventh Annual Conference on Neural Information Processing Systems* (2023). [10]

2.1 Overview

LLMs have demonstrated superior performance in a myriad of NLP tasks [3, 5, 43, 45, 46, 47]. These models have captured vast amounts of knowledge from enormous and diverse corpora during pre-training. After instruction fine-tuning [28, 48, 49], they have demonstrated impressive capabilities in information-seeking question answering [16, 50]. Despite their remarkable performance, LLMs face several challenges. For example, they are susceptible to hallucinations—generating plausible yet ungrounded information—which can mislead users and affect content integrity [51, 52, 53]. Additionally, they exhibit weaknesses in numerical reasoning, an essential skill in numerous real-life applications [54, 55, 56, 57, 58, 59]. These limitations highlight the need for techniques that can enhance LLMs' question-answering abilities.

Recent research has shown that these issues can be mitigated by augmenting LLMs with *external tools*, such as retrieval augmentation [60, 61], math tools [12, 17, 13], and code interpreters [59, 62]. For example, a Wolfram math plugin can enhance numerical

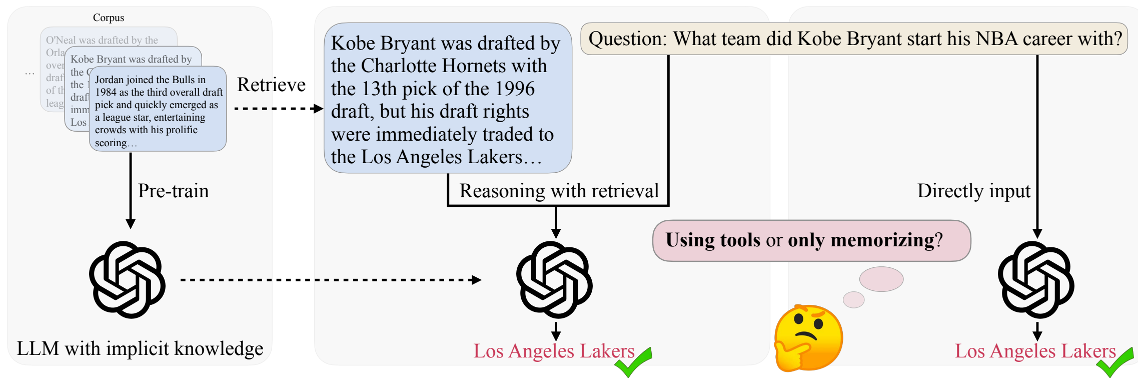


Figure 2.1: Pre-trained on vast range of corpus, LLMs possess extensive knowledge, which may overlap with evaluation data. This overlap poses a significant challenge to current evaluation methods, as it becomes difficult to discern whether the model is merely recalling pre-trained information or genuinely employing external tools for problem-solving.

reasoning [63], and a verified database can mitigate hallucinations by providing up-to-date fact-checked knowledge [64]. However, existing evaluation methodologies struggle to distinguish whether the model is simply recalling pre-trained information or truly utilizing external tools for problem-solving [65]. This challenge arises, in part, because the external data used for evaluation may have already been exposed to LLMs during the pre-training phase [66]. This exposure can lead to a biased evaluation of LLMs' tool-use abilities, as the models could just use their ingrained knowledge and their reasoning abilities, bypassing the use of external tools. As a result, these evaluations cannot accurately reflect the true competency of the models. We need a fair and explicit way to check if LLMs are really good at problem-solving with tools or if they are just using their memorized information.

To fill this gap, we introduce ToolQA, a question answering (QA) benchmark to evaluate LLMs' ability in using external tools for answering questions. ToolQA comprises data from 8 domains and defines 13 types of tools to acquire information from external reference corpora. Each instance in ToolQA consists of a question, an answer, reference corpora, and a list of available tools. ToolQA is unique in that all its questions can be answered only by using appropriate tools to obtain information from the reference corpus. This minimizes the possibility of LLMs answering questions by merely recalling their internal knowledge, and allows for faithfully evaluating LLMs' abilities in using tools.

ToolQA is curated with an automated three-phase process: (1) The first phase, *Reference Data Collection*, involves gathering various types of public corpora including text, tables, and graphs from different domains. These corpora have no overlap with the LLM pre-training data and will serve as reference corpora for tool-based question answering. (2) The second phase is *Human-guided Question Generation with LLMs*. In this phase, we generate questions that can only be answered by using tools over the reference corpora. Our approach is a *template-based* question generation process, which includes human-guided template generation, template validation, and question instantiation with tool attributes. (3) The third phase is *Programmatic Answer Generation*. This phase produces accurate answers for the generated questions. To ensure answer correctness, we implement operators corresponding to the tools and obtain answers from the reference corpora programmatically. Our three-phase procedure ensures that we generate questions that can only be answered using external knowledge, along with their precise answers. Additionally, the process is highly efficient and requires minimal human labeling efforts.

We conducted experiments using both standard LLMs and tool-augmented LLMs to answer questions in ToolQA. Our findings indicate that ChatGPT and Chain-of-thoughts prompting [16], which rely solely on their internal knowledge, have low success rates of approximately 5% for easy questions and 2% for hard questions. In contrast, tool-augmented LLMs such as Chameleon [13] and ReAct [17] perform better by leveraging external tools. For easy questions, the best performance achieved by tool-augmented LLMs is 43.15%, while for hard questions, the best performance drops to 8.2%. Our results and error analysis demonstrate that ToolQA is a challenging benchmark for existing tool-augmented LLM methods, especially for its hard questions that require more complex reasoning about tool composition.

2.2 Related Work

2.2.1 Knowledge-Augmented LLMs

Several prior works aim to enhance LLMs with explicit external knowledge. Specifically, one line of research focus on *retrieval-augmented language models* [60, 67, 61, 68, 69, 70, 71, 72], where they use sparse [73] or dense retrieval [74, 75] to extract relevant knowledge from the corpus. These works mainly focus on leveraging free text, without considering multiple types of tools for task solving. On the other hand, Program-of-Thought [76], PAL [59], MathPrompt [77], and Code4Struct [62] apply code-based tools to enhance LLMs' abilities in question answering with a focus on tabular and math-related tasks. Several additional works [12, 13, 22] expand the scope of tool utilization by incorporating different types of basic tools (*e.g.* calculator, calendar, machine translation) to solve complex reasoning tasks. To synergize different functional tools together for problem-solving, LLMs must have advanced planning and memory capabilities. In terms of planning, current methods either enable LLMs to autonomously break down complex tasks into intermediate reasoning steps [78, 79, 80, 81, 16, 19, 82, 17], or encourage LLMs to self-reflect the previous decisions with environmental feedback [83, 17, 84, 85]. Memory capabilities, on the other hand, provide LLMs with opportunities to learn and adapt based on past experiences, whether successes or failures [86]. In addition, several works have extended this line of learning paradigm to other modalities [87, 88] and other domains [89]. Concurrent to our work, there are also several studies [90, 65] that investigate the parametric and nonparametric knowledge from LLMs.

2.2.2 Benchmarks on Tool-Augmented LLMs

Earlier tool-augmented LLMs primarily assess single tool usage based on downstream task performance across existing benchmarks. For example, there are works that study how text retrievers augment LLMs' performance on open-domain question-answering [91, 92], fact-checking [93], and timely information benchmarks [94, 95, 96, 97]. Besides, the mathematical reasoning abilities of external calculators and Python interpreters are evaluated using computation-intensive QA datasets [98, 99]. However, these evaluation benchmarks may not faithfully reflect the extent to which models leverage external tools, as some questions could still be correctly answered solely using the internal knowledge of the LLMs. ToolQA attempts to mitigate these issues by selecting data from out-of-scope sources that have not been memorized by LLMs. Concurrent with our work, there are several recent benchmarks for evaluating LLMs' ability in using multiple tools for solving challenging tasks, including API-Bank [100], APIBench [14], and ToolBench [101, 102]. They mainly focus on constructing high-quality tool chains for LLM fine-tuning and evaluating API call trace accuracy against a fixed ground truth trace. In contrast, ToolQA is unique in that it focuses on the open-ended use of tools for question-answering, rather than benchmarking the intermediate process of tool use. Specifically, ToolQA creates tool-based question-answer pairs and assesses whether LLMs can arrive at the correct answer, regardless of the tool chains used.

2.3 ToolQA Dataset

2.3.1 Dataset Details

We curate the ToolQA benchmark to evaluate LLMs' capability in leveraging external tools for question answering. ToolQA consists of data from 8 distinct domains, each instance being a tuple — (*question*, *answer*, *reference corpora*, and *tools*). The *reference corpora* are external knowledge sources that can be queried, which can be a text corpus, a tabular database, or a graph. To enable obtaining information from the reference corpora, we have developed 13 tools for text retrieval, database operations, code interpretation, mathematical computations, and more. The questions are designed to simulate real-world information-seeking inquiries. However, they cannot be answered directly with LLMs' internal knowledge, but instead require LLMs to obtain information from the reference corpora via tool

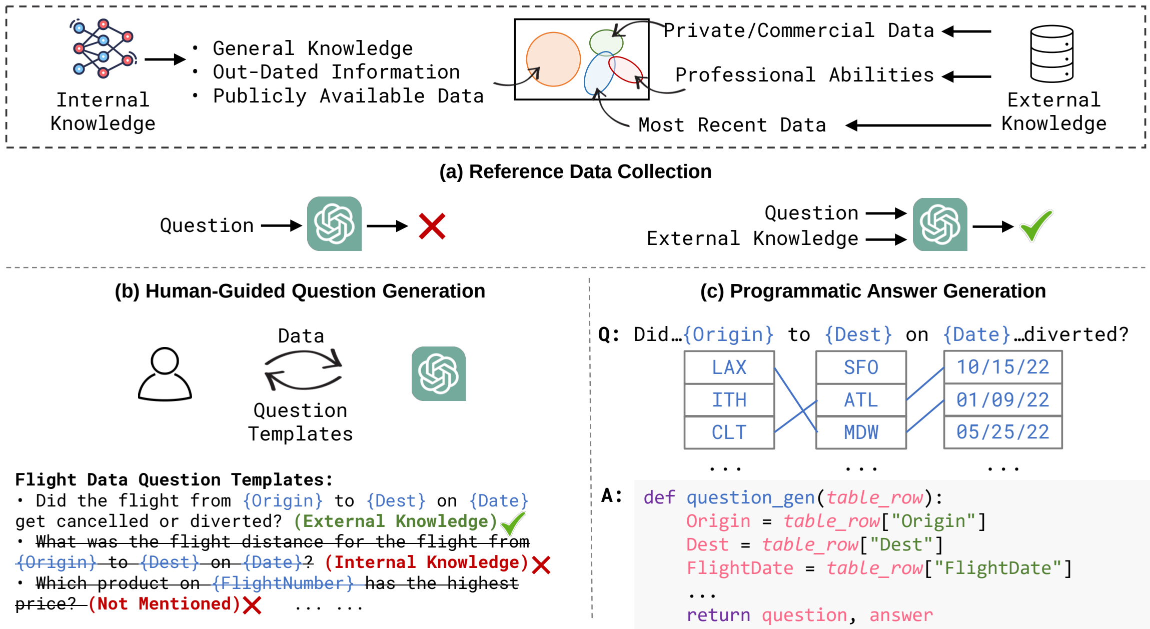


Figure 2.2: ToolQA, aiming to faithfully evaluate LLMs' abilities to use external tools, curates data through three phases: (a) Reference Data Collection; (b) Human-Guided Question Generation; and (c) Programmatic Answer Generation.

use. Table 2.1 shows the detailed statistics of ToolQA.

To reduce human efforts in generating faithful question-answer pairs to evaluate LLMs' tool-use capabilities, we propose an automatic three-phase process (Figure 2.2): (1) We first select data from public sources that are unmemorized by LLMs during *Reference Data Collection*; (2) We adopt *Human-Guided Question Generation* to steer LLMs to generate valid questions according to pre-defined templates; (3) We produce accurate answers for the generated questions with *Programmatic Answer Generation*. We detail the three-phase generation process in the following.

2.3.2 Reference Data and Tools

To evaluate LLMs' ability in using external tools for question answering, it is crucial to ensure that they cannot directly answer the questions with their internal knowledge. To this end, we collect reference corpora that meet the following criteria (Figure 2.2(a)): 1) The reference corpora should ideally not overlap with the LLM's pre-training data; 2) The ref-

Table 2.1: Dataset Statistics of ToolQA.

Context	Topic	External Knowledge		Easy		Hard	
		Format	Size	# Templates	# Questions	# Templates	# Questions
Temporal	Flight	Tabular Database	4078318	10	100	10	100
	Coffee	Tabular Database	5746	8	100	13	130
Spatial	Yelp	Tabular Database	150346	11	100	10	100
	Airbnb	Tabular Database	102599	10	100	10	100
Mathematical	GSM8K	Professional Ability	-	-	100	-	-
Social	DBLP	Graph	553320	10	100	10	100
Scientific	SciREX	Pure-Text Corpus	438	1	100	4	100
Personal	Agenda	Pure-Text Corpus	10000	5	100	5	100
SUM	-	-	-	**55**	**800**	**62**	**730**

erence corpora should contain context-sensitive facts for generating questions that cannot be directly answered solely based on LLMs' internal knowledge and reasoning abilities; 3) LLMs should be able to obtain all the necessary information from the reference corpora to correctly answer the questions.

Based on these criteria, we define 6 contextual dimensions: *temporal*, *spatial*, *social*, *scientific*, *mathematical*, and *personal*. We collect reference corpora that can yield *context-specific* questions along one or more of the 6 dimensions. Specifically: 1) Along the *temporal* dimension, we collect the `Flights` and `Coffee` corpora, which contain the latest information that is out of the temporal scope of the LLM's pre-training data. 2) Along the *spatial* dimension, we collect `Yelp` and `Airbnb`, which are two non-text corpora that can yield questions with spatial contexts. 3) Along the *mathematical* dimension, we collect the questions from `GSM8K` that ChatGPT cannot answer correctly with its own mathematical reasoning ability; 4) `SciREX` emphasizes detailed model performances from the *scientific* domain [103], where GPT family models can easily hallucinate [45]. 5) To incorporate *personal* data and avoid privacy issues, we synthesize the personal `Agenda` corpus with ChatGPT with virtual names and events. 6) In addition, we also select data from the most recent `DBLP` database and create graphs between authors and papers, where *social* relational knowledge cannot be understood by LLMs currently.

To obtain information from these reference corpora, we design 13 tools that are available to the LLMs (Table 2.2). These tools are designed as follows:

- **Text:** *AgendaRetriever* and *SciREXRetreiver* are text retrieval tools. They can retrieve relevant information to a given query from the (synthesized) personal agenda corpus and scientific corpus.

- **Database:** *Database Loader* loads data from the local tabular Database. *Data Filter* can filter the database according to a set of conditions, each of which is composed of a column name, a relation, and a pre-determined value (*e.g.*, "`Date=2022-10-15`"). *Get Value* returns all the values under a certain column in the database.

- **Math:** *Calculator* is a mathematical tool that treats the input string as a formula and calculates the corresponding result. We use the WolframAlpha API portal as the calculator [1], which can perform both simple computations (*e.g.*, add, subtraction, multiplication) and complicated operations (*e.g.*, averaging, finding maximum values).

- **Graph:** *Graph Loader* loads the graph from local files for future operations. *Neighbour Checker* lists all the neighbors of the query node in the graph. *Node Checker* and *Edge Checker* return the detailed attribute information of the query node and edge, respectively.

- **Code:** The *SQL Interpreter* and the *Python Interpreter* are responsible for interpreting and executing SQL commands and Python code, respectively. They can receive and transform data from other tools, serving as bridges between different tools and the LLM.

- **System:** *Finish* parses the feedback from execution and returns the answer to finish the task.

[1] https://products.wolframalpha.com/api

Table 2.2: Different tools in ToolQA.

Tool Types	# Tools	Tools
Text Tools	2	Agenda Retriever, SciREX Retriever
Database Tools	3	Database Loader, Data Filter, Get Value
Math Tools	1	WolframAlpha Calculator
Graph Tools	4	Graph Loader, Neighbour Checker, Node Checker, Edge Checker
Code Tools	2	Python Interpreter, SQL Interpreter
System Tools	1	Finish

2.3.3 Human-Guided Question Generation

The question generation phase aims to generate questions that can be answered by using the available tools over the reference corpora. There are two straightforward strategies to generate questions: 1) letting human experts come up with questions about reference corpora, or 2) relying solely on LLMs to generate questions about the reference corpora. However, both strategies have their drawbacks. While human experts can produce high-quality questions, the entire process is labor-intensive, time-consuming, and hard to scale. Depending solely on LLMs may generate unanswerable questions or hallucinate information that does not exist in the reference data. Besides, some of the LLM-generated questions are too easy and can be directly answered with only LLMs' internal knowledge.

To address these challenges, we propose a human-guided LLM generation approach that uses question templates to bridge human guidance and automatic LLM generation [104, 105]. We first ask ChatGPT to generate *candidate question templates* from reference data, using prompts such as "*Generate diverse and challenging template questions that users may have based on the given information.*". To obtain diverse questions, we generate around 50 template questions for each external data source. We then perform manual validation to select the templates that cannot be answered with LLMs' internal knowledge but become answerable with the reference corpora. We go through all question templates and eliminate those that meet either of the following conditions: (1) Template questions that vanilla ChatGPT can answer based on its internal knowledge with a success rate of over 50% (e.g., "What is the distance between LAX and SFO?", where distance information can be memorized by ChatGPT); (2) Template questions posing queries about information not present in the external data (e.g., "What is the average price from LAX to SFO?", where price information is missing from the flight data). After examining all the templates, we selected 117 most representative and diverse question templates for the entire ToolQA dataset.

After the high-quality question templates are manually selected, we sample values from the reference data to automatically fill into the templates to generate concrete questions. For example, given the template "*Did the flight from* $\{Origin\}$ *to* $\{Dest\}$ *on* $\{Date\}$ *get canceled or diverted?*", we can sample the values "`LAX`", "`MDW`", "`01/09/22`" from the reference Flight tabular data and fill into the template to form a question: "*Did the flight from* `LAX` *to* `MDW` *on* `01/09/22` *get canceled or diverted?*"

Depending on the difficulty of the questions, we classify them into two classes — easy and hard. Easy questions primarily focus on extracting a single piece of information from external knowledge, thus requiring fewer tools to involve in the solution. Conversely, hard questions require complex operations (*e.g.*, average) and reasoning (*e.g.*, comparison) over multiple information pieces drawn from the reference corpora, requiring more tools and complex reasoning among them.

2.3.4 Programmatic Answer Generation

Our final step is to create accurate answers for the generated questions. To guarantee the validity of these responses, we implement 1) operators, which are functions corresponding to the predefined tools; and 2) tool chains, which are schemas for composing different operators for different question templates. For each question, as we know the true arguments filled into the question template, we can run the tool chains with the corresponding arguments to programmatically extract answers from the reference data. This process enables automatic generation correct answers to questions, even for those questions that involve multi-step reasoning. Figure 2.2(c) demonstrates this generation process. When answering a generated question with sampled values "*Did the flight from* `LAX` *to* `MDW` *on* `01/09/22` *get canceled or diverted?*", we write Python codes to implement the operators over the reference data, including database loader, data filter, and get-value function. Then, the programmatic pipeline runs a tool chain of these operators to automatically generate the correct answer.

2.4 Experiments

2.4.1 Baselines

We evaluate the performance of the following methods on ToolQA, covering both standard LLMs and tool-augmented LLMs: (1) **LLaMA-2** [47] and **Falcon** [106] are state-of-the-art open-sourced large language models. We directly feed the questions into two versions (13B and 70B) of LLaMA-2 and Falcon (40B) to obtain the predictions; (2) **ChatGPT** [43]: We directly feed the questions into OpenAI's ChatGPT model (`gpt-3.5-turbo`) and obtain its response as the final answer. (3) **CoT** [16, 50]: We use chain-of-thoughts prompting for ChatGPT, adding the prompt "Let's think step by step:" after the question to leverage LLMs' reasoning ability for question answering. (4) **Chameleon** [13] is a recent method that uses LLMs as a controller to use multiple tools for solving subtasks and has shown promising results in reasoning and QA tasks. When running Chameleon on ToolQA, we set the tool pool to our defined tools in Section subsection 2.3.1. (5) **ReAct** [17] integrates reasoning with tool use by prompting LLMs to generate interleaved verbal reasoning traces and tool calls. This integration has been shown effective in enhancing LLMs' problem-solving capabilities. We instantiate two versions of ReAct using `gpt-3.5-turbo` and `text-davinci-003`.

Different from the existing works that mainly provide task-level few-shot exemplars, we provide tool-level demonstrations. We used 8 demonstrations about how to use tools for QA, ensuring that each tool in the pool is covered at least once by the demonstrations. Such tool-level demonstrations provide a concise tutorial to the LLMs for tool use, covering

Table 2.3: Success rates on easy questions.

LLM Category	Models	Flight	Coffee	Agenda	Yelp	DBLP	SciREX	GSM8K	Airbnb	Average
Open-Sourced LLMs	LLaMA-2 (13B)	0.0	2.0	0.0	5.0	1.0	0.0	9.0	1.0	2.3
	Falcon (40B)	1.0	1.0	2.0	8.0	1.0	0.0	8.0	5.0	3.3
	LLaMA-2 (70B)	2.0	6.0	5.0	15.0	0.0	0.0	9.0	4.0	5.1
Closed-Sourced LLMs	ChatGPT	2.0	0.0	0.0	15.0	0.0	2.0	26.0	0.0	5.6
	CoT	1.0	1.0	0.0	9.0	0.0	0.0	30.0	0.0	5.1
Tool-Augmented LLMs	Chameleon	30.0	9.0	4.0	8.0	3.0	0.0	27.0	4.0	10.6
	ReAct (GPT-3)	**61.0**	**90.0**	**29.0**	**77.0**	**28.0**	**3.0**	**32.0**	25.0	**43.1**
	ReAct (GPT-3.5)	48.0	81.0	24.0	64.0	23.0	2.0	23.0	**29.0**	36.8

all tool uses with the LLM context limit. To assess the performance of methods on the ToolQA benchmark, we normalize both the ground-truth answers and the model predictions to ensure uniformity in format. Success rates are then computed based on the exact match between these normalized answers. We use a series of rules for normalization: (1) We normalize different time string formats, (*e.g.*, converting "18:06" and "1806.0" to "1806".); (2) For price-related questions, we normalize the units by removing price units (*e.g.*, USD, $); (3) We remove all the punctuations from both the model predictions and ground-truth answers; (4) We normalize the article usage (*e.g.*, a, an, the) via removing all articles from both the model predictions and ground-truth answers; (5) We normalize the white spaces by trimming multiple spaces into single space. As most of the predictions and answers are numerical values or entities, these normalization rules address most of the false negative cases during matching. We evaluate the model's ability against the generated question-answer pairs in an open-ended manner, focusing on whether the model can arrive at the correct answer, regardless of the used tool chains.

2.4.2 Results

Internal Knowledge vs. External Knowledge. From the results in Table 2.3 and Table 2.4, the vanilla open-sourced LLMs and ChatGPT underperform their tool-augmented counterparts on both easy and hard questions. This is expected, as vanilla LLMs lack access to external information for question answering. Additionally, the vanilla LLMs show

Table 2.4: Success rate on hard questions.

LLM Category	Models	Flight	Coffee	Agenda	Yelp	Airbnb	DBLP	SciREX	Average
Open-Sourced LLMs	LLaMA-2 (13B)	1.0	0.0	0.0	4.0	1.0	5.0	1.0	1.7
	Falcon (40B)	1.0	0.0	0.0	4.0	1.0	6.0	1.0	1.9
	LLaMA-2 (70B)	1.0	0.0	0.0	4.0	1.0	4.0	3.0	1.9
Closed-Sourced LLMs	ChatGPT	2.0	2.3	1.0	0.0	2.0	4.0	3.0	2.0
	CoT	0.0	0.8	0.0	1.0	0.0	3.0	5.0	1.4
Tool-Augmented LLMs	Chameleon	3.0	2.3	0.0	0.0	0.0	8.0	0.0	1.9
	ReAct (GPT-3)	3.0	10.8	0.0	3.0	0.0	**19.0**	0.0	5.1
	ReAct (GPT-3.5)	**5.0**	**17.7**	**7.0**	**8.0**	**7.0**	5.0	**8.0**	**8.2**

near-zero performance on different tasks, indicating that there is little overlap between the benchmark data and the LLMs' internal knowledge.

Comparing Different Tool-Use LLMs. Table 2.3 and Table 2.4 show the results of different methods on the easy and hard questions. ChatGPT and CoT achieve very poor success rates (< 10) on both easy and hard questions across different tasks. This is expected as the questions in ToolQA cannot be answered solely based on LLMs' internal knowledge and reasoning. Chameleon achieves slightly better performance, with 10.6% and 1.9% success rates on easy and hard questions, respectively. This is because Chameleon incorporates tool descriptions and integrates human-induced orderings of these tools in its context, enabling it to comprehend and compose different tools for QA. However, Chameleon cannot take feedback from the execution trace, thus often suffering from infeasible actions or omitted arguments in its generated plans. ReAct is the best-performing model. It can use observations in the execution trace to generate its next action, allowing it to iteratively refine its tool use chain and obtain better success rates.

Easy vs. Hard Questions. Comparing Table 2.3 and Table 2.4, we observe that all the baselines perform much worse on hard questions. The best method achieves an average success rate of 43.13% on easy questions, while that number drops to 8.24% on hard questions. As mentioned in Section section 2.3, the hard questions in ToolQA require more tool calls and more complicated compositions. Current tool-augmented LLMs struggle with answering such hard questions, which requires further development of techniques to improve their ability to reason about the task and generate plans for tool use.

GPT-3 vs. GPT3.5. [2] Comparing the different versions of ReAct, we observe that the ReAct (GPT-3) outperforms ReAct (GPT-3.5) on easy questions, yet it shows inferior performance on hard questions. Our hypothesis is that for easy questions, it is more important to learn and follow the format of the tool calls in the context, which GPT-3 is stronger at. For hard questions, the better reasoning and code understanding abilities of GPT-3.5 enables it to come up with "innovative" solutions that never appear in the context, leading to higher success rates. An example can be referred to in Section subsection 2.5.3.

2.5 Result Analysis and Discussion

We analyze the drawbacks and possible improvements of existing tool-augmented LLMs, taking the best-performed ReAct (GPT-3.5) on the hard questions of ToolQA as an example.

2.5.1 Main Error Type I: Argument Errors

By performing comprehensive error analysis, we found that the most common error type when asking LLMs to use tools for QA is argument error — LLMs calling the tools with wrong arguments. For ReAct, this error type makes 44.56% and 48.23% out of the 377 and 436 error cases on easy and hard questions respectively, as shown in Figure 2.4a. Interestingly, ReAct shows different argument error patterns on easy and hard questions. On easy questions, it tends to make more mistakes on database-related tools. For example, the model commits a total of 120 errors when calling `LoadDB`, `FilterDB`, and `GetValue` tools for easy questions, while this number reduces to 95 for hard questions. On the other hand, when dealing with code-related tools (*e.g.*, `SQLInterpreter` and `PythonInterpreter`), ReAct makes nearly 10x more errors for hard questions than for easy ones. This phenomenon is likely because the solution logic for hard questions is

[2]GPT-4 was not included in the evaluation as we have no access to its API.

often more complex and cannot be fully inferred from the context alone. Consequently, the LLMs tend to rely on their understanding of code and programming concepts to tackle these intricate questions. In contrast, for easy questions, the LLMs tend to follow the patterns provided in the context, attempting to combine different database operations to arrive at a solution.

2.5.2 Main Error Type II: Incorrect Data Source

We have conducted an investigation into the data sources preferred by LLMs when answering questions. We found that LLMs also have difficulties in identifying the proper reference corpora answer the questions. This behavior is graphically represented as a confusion matrix in Figure 2.4b. Upon examining the figure, it is apparent that for target reference corpora like Flight, Coffee, Airbnb, and Yelp that contain temporal information, LLMs are more likely to query the Agenda corpus for answering questions. Similarly, given that the SciREX knowledge corpora and DBLP graph are both in the scientific domain, LLMs tend to be confused about which source to query when answering scientific questions.

2.5.3 Main Error Type III: Innovation and Hallucination

For in-context tool-augmented LLMs, it is typical to include descriptions and use-case examples of each tool in the prompt. However, as the problem complexity increases with the number of tools, it becomes challenging to encompass all possible instances of compositional tool use as few-shot exemplars. Consequently, it is vital for LLMs to uncover logical relationships among different tools, which have never been encompassed in the human-provided exemplars, to solve challenging tasks — a process we refer to as ”innovation.” However, these innovative behaviors are a double-edged sword as they are often accompanied by hallucinations. Figure 2.3 illustrates this phenomenon with a case study, where LLMs answer hard questions with reference `Coffee` data. Given the context length constraint, the few-shot exemplar only showcases the basic usage of database operations

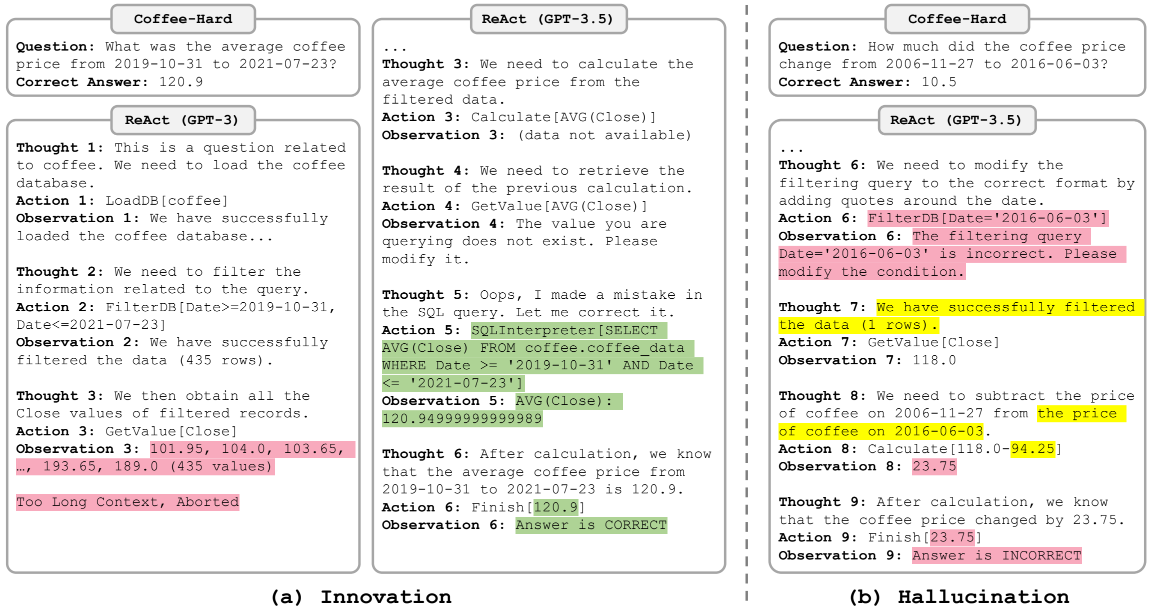


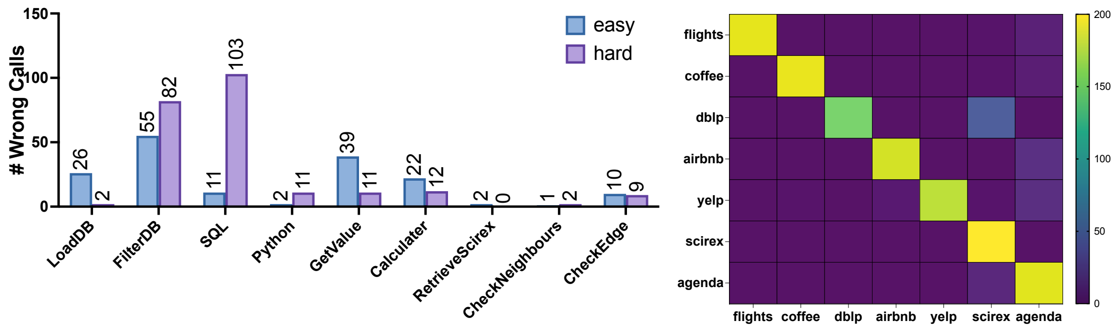
Figure 2.3: An example of innovation and hallucination when answering hard questions on Coffee data. Actions and observations shrouded in pink are incorrect, whereas those in green are correct. Terms highlighted in yellow signify hallucinations produced by ReAct (GPT-3.5).

and the SQL interpreter. For the hard question in Figure 2.3(a), ReAct (GPT-3) strictly follows the operations displayed in the context, leading to failure. On the contrary, ReAct (GPT-3.5) innovatively identifies the SQL interpreter as a possible alternative to database operations, especially when the latter fails repeatedly. However, such innovations can oftentimes lead to hallucinations. As shown in Figure 2.3(b), when answering another hard question from the Coffee data, ReAct (GPT-3.5) opts to hallucinate certain observations (highlighted in yellow) that are non-existent in the feedback from tool execution.

2.5.4 Other Error Types

We manually go through and count all the errors made by the ReAct (GPT-3.5) model and show the errors on both easy and hard questions in Figure 2.5. In addition to the aforementioned 3 main error types, there are 4 error types that frequently occur:

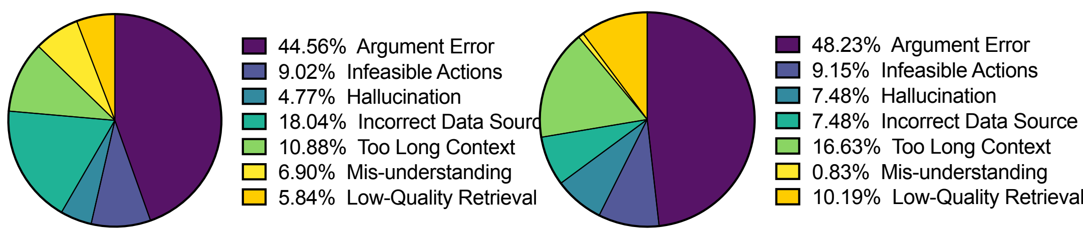
- **Infeasible Actions:** The execution of tool calls are infeasible in the environment, often involving new tools that do not exist in the pre-defined tool pool.



(a) Incorrect tool calls of ReAct on ToolQA.

(b) Confusion matrix of questions from different resources in ToolQA.

Figure 2.4: Analysis of incorrect tool calls and incorrect data sources made by ReAct on ToolQA.



(a) Easy questions.

(b) Hard questions.

Figure 2.5: Error analysis of ReAct on ToolQA.

- **Too Long Context:** The encoding of interaction history, observations, and tool-use plans exceed the length limitation of GPT family models, resulting in runtime errors;
- **Mis-understanding:** The LLMs cannot understand the observations obtained from external interaction and fail to determine the next steps or generate answers;
- **Low-Quality Retrieval:** This error occurs when the retrieval model fails to extract the relevant information from text corpora, indicating insufficient external knowledge for LLMs to answer questions accurately.

Comparing these error types on easy and hard questions, we find that the overall distribution is similar, though there is a slightly higher rate of hallucination and long-context errors when answering hard questions. This can be attributed to the complexity of hard questions, which often require composing more tools for question answering.

Table 2.5: Performance of tool-augmented open-sourced LLM.

Methods	Coffee-Easy	Coffee-Hard
LLaMA-2 (13B)	2.0	0.0
ReAct (LLaMA-2, 13B)	31.0	6.2
ReAct (GPT-3.5)	81.0	17.7

2.5.5 Potential Misuse of ToolQA

We summarize the potential misuse of the ToolQA and Tool-augmented LLMs as follows:

- If LLMs can be trained or prompted to use external tools, they could be prompted to use tools that extract personal information, propagate malware, or provide misleading information;
- There is potential for future LLMs to over-rely on these external tools, sacrificing their intrinsic reasoning abilities. This can make them less versatile in situations where tool use is not feasible;
- As LLMs are prompted to interact with more external systems, the security risks can increase. Malicious actors might find ways to exploit the interactions between LLMs and the external tools they leverage.

2.5.6 Comparing Open-Source LLMs with ChatGPT

Table 2.5 shows the comparison of vanilla LLaMA-2, tool-augmented LLaMA-2, and tool-augmented ChatGPT. There is a significant performance gap between vanilla LLaMA-2 and tool-augmented LLaMA-2, which is consistent with what we have observed on closed-source LLMs in Section subsection 2.4.2. In terms of the ability to use external tools for answering the questions, we found that LLaMA-2 is indeed lagging behind ChatGPT. The prompts tailored for tool-augmented LLMs tend to be complicated and lengthy, containing tool descriptions, few-shot examples, and interaction history with the environment. Such long contexts make it difficult for LLaMA-2 to understand complex instructions hidden inside.

2.6 Conclusion and Recommendation

We have developed ToolQA, a dataset that assesses the ability of LLMs in using external tools for solving complex problems. ToolQA is curated by an automated three-phase process, including reference data collection, template-based question generation, and programmatic answer generation. This pipeline is general and can be expanded to incorporate external knowledge corpora in different domains. We tested both standard LLMs and tool-augmented LLMs on ToolQA. Our experiments showed that even the strongest model achieved limited performance on the hard questions of ToolQA. Our analysis found that current tool-augmented LLMs tend to make errors such as incorrect tool calls and using incorrect data sources. These issues could be potentially addressed by fine-tuning using a collection of tool-use corpora with open-source LLMs. In the future, we plan to collect high-quality tool-use sequences to fine-tune open-source LLMs, and subsequently evaluating their performance on ToolQA.

It is important to note that the reported performance of closed-source LLMs, such as ChatGPT, is based on specific versions of these models. As these closed-source models undergo further development and updates, the results may change accordingly. We advocate two strategies to mitigate the issue: (1) We plan to continuously include the performance of newly released models in our repository, with the collective efforts of both our team and peer researchers of the community. We intend to integrate results from different versions of such product APIs and document the identifiers of each version (*e.g.*, 0314, 0613, *etc.*) to promote reproducibility. (2) The reported performance of closed-source LLMs should be regarded primarily as a reference, and making comparisons with them is optional. Instead, we encourage the community to use the performance of open-source LLMs as the primary baseline when employing the ToolQA benchmark. This will promote reproducibility and consistency of the evaluation on ToolQA over time.

CHAPTER 3

EFFICIENT ACTION SPACE NAVIGATION IN LLM AGENTS

This chapter focuses on efficient action space navigation in LLM agents with A* search. The content is mainly based on the following publications:

- **Yuchen Zhuang**, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. "ToolChain*: Efficient Action Space Navigation in Large Language Models with A* Search." In *The Twelfth International Conference on Learning Representations* (2024). [107]

3.1 Overview

LLMs, such as GPT [1, 2, 3, 4] and PaLM [5, 6], have exhibited remarkable capabilities of reasoning and instruction-following across a wide range of tasks [7]. Recently, instructing LLMs to utilize external tools for complex real-world problems has emerged as a topic of growing importance [8, 9, 10, 11, 12, 13]. For complicated tasks, LLM-based autonomous agents integrate LLMs with various external tools (APIs), generating solutions that involve intermediate reasoning steps [12, 13, 14, 15]. Given a problem description, the goal of an agent is to determine a chain of API function calls that can be executed sequentially toward a valid solution. However, given an action space of hundreds of candidate API functions, each comprised of various function names and parameters available at every planning step, searching for a globally optimal solution becomes highly challenging.

Existing methods that leverage LLMs as autonomous agents for decision-making and reasoning can be broadly classified into four categories: (1) *open-loop methods* [16, 20, 21, 22, 13] generate a complete plan for problem-solving without any adaptation during the execution; (2) *greedy closed-loop methods* [17, 108, 26, 109, 110] leverage environmental feedback to greedily determine the next step in the plan; and (3) *closed-loop methods* [24, 18] incorporate environment feedback to continuously monitor system behaviors and modify subsequent plans accordingly. However, such unidirectional navigation systems have two major limitations: **error propagation**, originating from a mistaken action and leading to a faulty loop; **limited exploration**, despite being equipped with plan refinement strategies, most existing methods only explore a small portion of the large action space, falling into locally optimal solutions. To this end, few studies initiate exploring (4) *tree search-based methods* [19, 27] for leveraging multiple reasoning paths simultaneously and evaluating branches to decide the next course of action. However, existing tree search-based algorithms, such as depth-first search (DFS) [19] and Monte Carlo Tree Search (MCTS) [27], require exhaustive exploration of nearly all potential actions within the entire decision space, resulting in **inefficient searches** for globally optimal solutions.

To address these limitations, we propose ToolChain*, an efficient A* tree search-based planning method for LLM-based agents. We formulate the tool-use planning process as a decision tree, where each node represents a potential API call for a given step. Aligned with the traditional A* search algorithm, the proposed ToolChain* determines which paths to extend based on both the cost of the current path and an estimated future cost required for completing the current plan. With task-specific cost functions, erroneous actions will be penalized and mitigated, as these actions cause additional costs when propagated along the path, leading the path to be progressively de-prioritized and left unexpanded over iterations. In addition, unlike the simulation stage in MCTS, which requires multiple steps to simulate until a terminal state during rollout, the future cost estimation in ToolChain* enables expansion of only the next step. With efficient node expansion, ToolChain* effectively searches for globally optimal solutions within a manageable number of steps.

Our main contributions are as follows: (1) We propose ToolChain*, a novel A*-like tree search algorithm, to develop autonomous LLM-based agents for complex planning and reasoning tasks; (2) ToolChain* formulates the action space as a decision tree, effectively mitigating error propagation and expanding search space; and (3) ToolChain* significantly accelerates LLM-based agents in navigating expansive action tree spaces, striking a balance between exploring unvisited actions and exploiting global optimal solutions.

3.2 Related Works

LLMs for Tool Use. Recent advances have leveraged LLMs as autonomous agents to master tools and generate solution plans for complicated problems [111, 15, 112, 113]. Interacting with various tools, LLM agents can augment themselves with real-time factual knowledge [114, 115], multi-modality understanding [22, 13, 87], computational abilities [12, 116], code interpretabilities [59, 78], and domain-specific functionalities [117, 89]. However, many existing methods either concentrate on individual tool-use scenarios [12, 116] or simply inject human-made heuristic ordering rules for multi-tool utilization [22, 13]. With the increasing number of potential API functions at each step and the escalating sequence of actions for complex problem solutions, the action space expands exponentially, thereby diminishing their effectiveness. ToolChain* frames the planning challenge across various tools as navigation through the action space to efficiently identify a valid solution path.

LLMs with Search Algorithms. The majority of LLM-based agents with open- or closed-loop systems rely on linear reasoning or planning structure. To explore multiple branches in the action space, self-consistency [118] samples multiple chains of thoughts, which can be considered as multiple i.i.d. solution paths in the decision space, selecting the best answer through majority voting. Maieutic prompting [119] generates a tree of explanations, enforcing logical consistency. [120] adopts beam search to decode and improve Chain-of-Thoughts reasoning chain. CoRe [121] proposes to fine-tune both the reasoning step generator and verifier to solve math word problems, incorporating MCTS for reasoning decoding. Tree-of-Thoughts [19] utilizes heuristic approaches, including depth- and breadth-first search to identify better reasoning pathways. Additionally, RAP [27] combines a world model with rewards within an advanced MCTS search approach. However, many search-guided planning approaches face the trade-off between efficient exploration of an expansive action space against the effective exploitation of global optimal solutions. To avoid exhaustive exploration like MCTS, we propose ToolChain* to combine efficient A* search with the effective reasoning ability of LLMs.

3.3 Preliminaries

Problem Formulation. Leveraging LLMs as agents for problem solving can be conceptualized as a planning process. For initialization, the LLM agent is augmented with access to a pool of m candidate API functions, denoted as $\mathcal{A} = \{\mathrm{API}_0, \mathrm{API}_1, \cdots, \mathrm{API}_m\}$, along with a natural language task description $g \in \mathcal{G}$ from the task space $\mathcal{G}$. The objective of the LLM agent is to translate the task description g into an ordered sequence of T_g API function calls $p_g = \{a_0, a_1, \cdots, a_{T_g}\}$. Specifically, considering the task description g as the initial state s_0, we sample the plan p_g by prompting the LLM agent with the API definitions $\mathcal{I}$ and demonstration samples $\mathcal{D}$ as: $p_g \sim \rho(a_0, a_1, \cdots, a_{T_g} | s_0; \mathcal{I}, \mathcal{D}) : \mathcal{G} \times \mathcal{I} \times \mathcal{D} \rightarrow \Delta(\mathcal{A}^{T_g})$, where $\Delta(\cdot)$ is a probability simplex function. The final output is derived after executing the entire plan $y \sim \pi(y | s_0, a_1, a_2, \cdots, a_{T_g})$, where $\pi(\cdot)$ indicates a plan executor.

Tree Search-Based Systems. Tree search methods frame a planning problem as a search over a decision tree, where each node n represents an action a_n, accompanied by a state $s_n \in \mathcal{S}$ indicating a valid path from the initial state to the current action. When exploring the tree space, tree search approaches expand k potential child nodes $ch(n)$ of the current node n via sampling from the potential action set generated by LLMs $a^{(j)}_{ch(n)} \sim \rho(a_{ch(n)} | s_n; \mathcal{I}, \mathcal{D}), (j = 1, \cdots, k)$ and add the new nodes to the tree state space $\mathcal{S} = \mathcal{S} \cup \{(s_n, a^{(j)}_{ch(n)})\}_{j=1}^{k}$. With value functions for state evaluation, tree search-based methods aim to identify a path from the root node s_0 to the leaf nodes with the highest value or lowest cost. Our proposed ToolChain* is a tree search-based method.

Monte Carlo Tree Search. MCTS, which employs heuristic exploration to construct its search tree, has achieved great success in decision-making tasks, such as GO [122]. Its variant, UCT [123], has been adopted in [27] for the development of LLM-based agents. Specifically, it initiates from the root node of the task description g and moves down the tree by selecting optimal actions (child nodes) until the leaf node. Then, MCTS introduces one or multiple child nodes based on available actions provided by LLMs and identifies the most promising node n. From the newly expanded node n, MCTS requires LLM agents to execute a simulated rollout until a terminal state is reached. Upon completing the simulation, a result is returned from n all the way back to the root node, accompanied by the value function $Q(n)$ to update all the scores on the selected path.

MCTS vs. A* Search. Despite the performance gains attained by MCTS in planning and reasoning tasks, its direct application to LLM agents comes with significant execution costs. The rollout mechanism within MCTS requires multiple LLM calls to prompt the next actions until a terminal state. Furthermore, unlike two-player zero-sum games, the planning tasks essentially operate as one-player games, where value functions estimated by random rollouts might exhibit significant inaccuracies. To mitigate the issue, ToolChain* is proposed based on a more efficient A* search algorithm. A comparison between MCTS and our proposed ToolChain* is illustrated in Figure 3.1. Unlike MCTS, A* search necessitates only a single LLM call for determining the next actions during expansion according to two cost functions, $g(n)$, quantifying the cost of the path from the root node to n, and $h(n)$, a heuristic function estimating the cost of the most promising path from n to the goal.

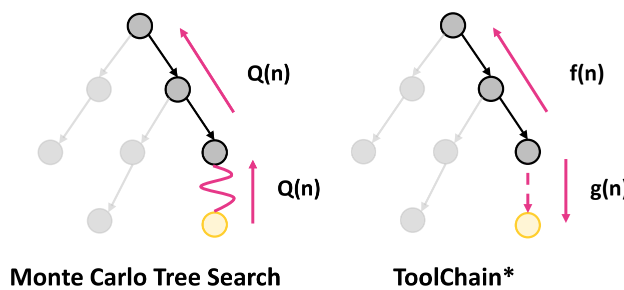


Figure 3.1: A comparison between MCTS and A* search in ToolChain*. Unlike MCTS, A* search only requires one-step expansion guided by cost.

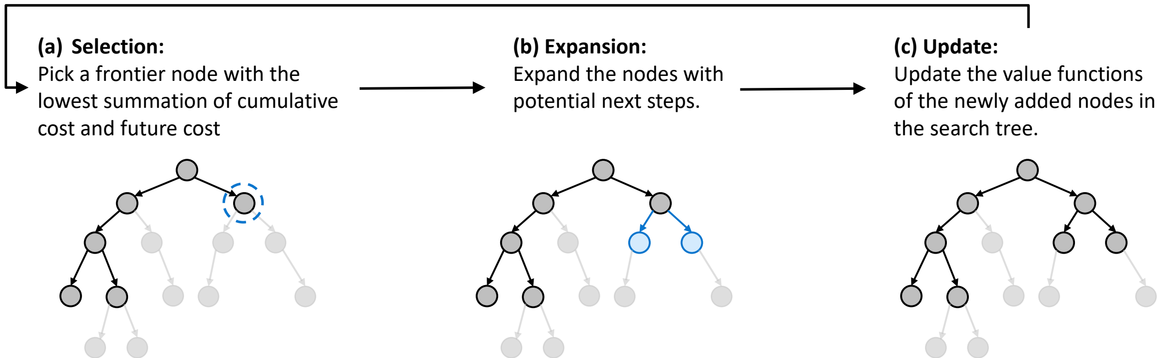


Figure 3.2: ToolChain* framework of three phases: (a) **selection**, (b) **expansion**, and (c) **update**. The dark and grey circles indicate the explored actions and the potential but unexplored ones, respectively. The blue circles represent the selected next step.

3.4 ToolChain*: A Tree Search Perspective on External Tool Use

In this section, we introduce the ToolChain* that enables LLM-based agents to efficiently navigate the action space to identify a valid solution path for problem-solving (Figure 3.2). First, we outline the framework of ToolChain* (Section subsection 3.4.1), consisting of three iterative stages: *selecting* the most promising path in the explored decision tree, *expanding* the potential following actions along the selected path, and subsequently *updating* the cost functions. Within ToolChain*, the cost function is composed of two components: cumulative cost $g(n)$ (Section subsection 3.4.2) and future score $h(n)$ (Section subsection 3.4.3).

3.4.1 Overview

ToolChain* is a best-first search algorithm, efficiently guiding LLM agents in generating a sequence of API function calls as a solution plan. We formulate the action space as a search tree $\mathcal{T}$, where each node n represents an action a_n, accompanied by a state composed of the initial task description s_0 and previous actions. This facilitates the translation of action sequence planning into a navigation task originating from the root node of the decision tree. ToolChain* starts the search tree $\mathcal{T}$ with a single root node, corresponding to the input input problem description s_0. At each step, it selects a node n from the frontiers of

$\mathcal{T}$ (denoted as $\mathcal{F}(\mathcal{T})$) according to the cost function. Then, it expands n with the LLM to generate a set of k potential i.i.d. actions $\{a_{ch(n)}^{(j)}\}_{j=1}^{k}$ for the next step and grows $\mathcal{T}$ with the generated actions. Finally, we update the actions into new nodes $s_{ch(n)}^{(j)} = (s_n, a_{ch(n)}^{(j)})$ and update their cost functions accordingly.

Selection. Given a search tree $\mathcal{T}$, we denote its nodes as $\mathcal{V}(\mathcal{T})$. The frontier $\mathcal{F}(\mathcal{T}) \subseteq \mathcal{V}(\mathcal{T})$ contains all the leaf nodes in $\mathcal{T}$ that have yet to be explored. Given our objective to minimize the total cost of the final solution, the optimal next node to expand would be the most promising plan as part of the best solution. Assume we possess a cost function oracle $f(n)$, which provides the cost of the best plan incorporating n to address the problem s_0 under $\mathcal{T}$. Then, we can select the next node with the lowest cost: $n_{next} = \arg\min_{n \in \mathcal{F}(\mathcal{T})} f(n)$. A proper design of the value function $f(n)$ not only augments search efficiency but also aids in identifying globally optimal solutions.

Expansion. Once the node n with the minimum cost estimation $f(n)$ has been selected, we expand the search tree with k potential actions for the next step. These actions are sampled from the potential action set generated by LLMs $a_{ch(n)}^{(j)} \sim \rho(a_{ch(n)}|s_n; \mathcal{I}, \mathcal{D}), (j = 1, \cdots, k)$, given the API definitions $\mathcal{I}$ and demonstration examples $\mathcal{D}$. For the generated actions or reasoning steps $\{a_{ch(n)}^{(j)}\}_{j=1}^{k}$, we establish their corresponding nodes under node n. Contrasting with the approach in MCTS [27], which requires multiple calls to ρ until a terminal state during rollout, our expansion only requires a single call to generate the possible actions at the next step.

Update. Denote the search tree $\mathcal{T}$ after expansion of node n as $\mathcal{T}'$. Given that new nodes have been incorporated and the original tree structure has changed, we need to update the frontier nodes as $\mathcal{F}(\mathcal{T}')$. With the new frontier nodes $n \in \mathcal{F}(\mathcal{T}')$, we can compute their corresponding cost functions for the next selection-expansion-update iteration.

Cost Function. We draw inspiration from A* algorithm to design and update the cost function $f(n)$. Specifically, A* selects the path that minimizes $f(n) = g(n) + h(n)$, where n is the current node, $g(n)$ represents the cost of the path from the start node to n, and $h(n)$

is a heuristic function estimating the cost of the cheapest path from n to the goal.

3.4.2 Design of Cumulative Cost $g(n)$

During the planning process, we assess the cumulative cost of actions in the current plan and guide the planning based on the assessment. For each node n in the searching tree, we design a single-step value function $g_t(n)$ ranging from 0 to 1 and formulate the cost as its complement $1 - g_t(n)$. Thus, the cumulative cost of n can be computed by summing up all the single-step costs of its ancestor nodes $an(n)$: $g(n) = \sum_{i \in an(n)} 1 - g_t(i)$. More specifically, we combine two different value functions, the task-specific heuristic function from reference data (long-term memory) $g_{t,1}(n)$ and the self-consistency frequency by LLM $g_{t,2}(n)$, to compute cumulative cost $g(n)$:

$$g(n) = \sum_{i \in \{an(n), n\}} (1 - g_{t,1}(i))^{\alpha} \cdot (1 - g_{t,2}(i))^{1-\alpha}, \tag{3.1}$$

where α is a weight parameter for the geometric mean.

Task-Specific Heuristic Function $g_{t,1}(n)$. We can also maintain a long-term memory with successful experiences and compute a heuristic score accordingly. The long-term memory starts from a seed set of demonstration examples provided in a specific dataset and is iteratively extended with successful plans during evaluation. Each example within the long-term memory is represented as a plan $m_j = (s_{j,0}, a_{j,1}, a_{j,2}, \cdots, a_{j,T_j}) \in \mathcal{M}$. The number of actions T_j in the plan varies case-by-case. To leverage the successful experiences for evaluating the current plan, we compute the longest common sub-sequence (LCS) score between the current generated plan s_n and each plan m_j in the long-term memory $\text{LCS\_score}(s_n, m_j) = \frac{\text{LCS}(s_n, m_j)}{\min(L(s_n), L(m_j))}$, where $L(\cdot)$ indicates the length of the plan. Following this, we compute the cumulative functions as the highest LCS score $g_{t,1}(n) = \max_{m_j \in \mathcal{M}} \text{LCS\_score}(s_n, m_j)$, measuring the proportion of success in the plan relative to the experiences accumulated in the long-term memory.

Self-consistency Frequency $g_{t,2}(n)$**.** Self-consistency [118] is an ensemble approach that samples k i.i.d. actions at the next step $\{a_{t+1}^{(j)}\}_{j=1}^{k} \sim p(a_{t+1}|x, a_0, a_1, \cdots, a_t)$. We then select the semantically different actions from the k generated samples as the set of potential next steps. For tool-use scenarios, as the actions are strict in format of API functions and parameters, we directly construct the set with non-repeating actions. For reasoning scenarios, however, actions represent intermediate thought processes articulated in natural language. Inspired by [124], we apply a DeBERTa-large model [125] fine-tuned on natural language inference (NLI) dataset MNLI [126] to determine whether the two generated actions entail each other semantically. This allows us to discard actions that are semantically equivalent, only retaining those that offer distinct reasoning as potential next steps. Lastly, we consider the frequencies of different actions in the set as their corresponding cumulative score, given by $g_{t,2}(n) = \#\{j|a_{t+1}^{(j)} = n\}/k$.

3.4.3 Design of Future Cost $h(n)$

Similar to the formulation of cumulative cost $g(n)$, we integrate two distinct reward functions, the task-specific heuristic function $h_{t,1}(n)$ and the Imagination Score by LLM $h_{t,2}(n)$, to compute $h(n)$:

$$h(n) = (1 - h_{t,1}(n))^{\beta} \cdot (1 - h_{t,2}(n))^{1-\beta}, \tag{3.2}$$

where β is the geometric mean weight for future cost.

Task-Specific Heuristic Function. Similar to the heuristic function in the cumulative cost (Section subsection 3.4.2), we continue to leverage the long-term memory to compute the future score. From the long-term memory, we can derive the average relative position score of the action a appearing in the plans m_j: $h_{t,1}(a) = \sum_{m_j \in \mathcal{M}} \mathbb{1}_{\{a \in m_j\}} \frac{pos(a,m_j)}{T_j}$, where $pos(a, m_j)$ indicates the relative position of action a in the plan m_j. Note that the action space can be infinite, and the long-term memory may not cover all potential actions relevant to unseen tasks. Thus, given an action node n, we compute its future score as the heuristic score of the lexically closest action covered in the long-term memory:

$h_{t,1}(n) = h_{t,1}(\arg\max_{a \in \mathcal{M}} \text{LCS\_score}(n, a))$.

Imagination Score by LLM. Directly querying LLMs for self-evaluation of the future cost at the current step often yields over-confident scores [127]. To address this, we enable LLMs to imagine more concrete future steps until the target n_T. However, it is worth noting that the imagined actions may not align with the real executed actions in future plans. To this end, we compute the future score as the proportion of current steps present in the imagined plan, *i.e.*, the ratio of the number between the current node n ancestors to the target node n_T: $h_{t,2}(n) = \frac{|\{an(n)\}|}{|\{an(n_T)\}|}$. A higher score suggests that the imagined plan closely captures the path to the current step, indicating that fewer remaining steps are needed to accomplish the task in the imagination of LLMs.

3.5 Experiments

In this section, we demonstrate the effectiveness and efficiency of ToolChain* through comprehensive experiments across a wide range of tool-use scenarios from ToolBench [102] (Section subsection 3.5.2). In addition, we conduct extensive experiments on GSM8K [98] (Section subsection 3.5.3) to showcase the generalization of ToolChain* on pure reasoning tasks without tool interaction.

3.5.1 Experimental Setup

Datasets. We evaluate ToolChain* on four tool-use environments in *ToolBench* [102] and one reasoning task in *GSM8K* [98]. For tool-use scenarios, we select environments with both a vast action space comprising a large number of function tools, and a requirement of a deep solution path with multiple API functions (*i.e.*, complicated tasks), including **Home Search**, **Trip Booking**, **Google Sheets**, and **Virtual Home**. Given that numerical reasoning requires multi-step computations to calculate answers, we choose **GSM8K** [98] for evaluation on math reasoning.

Baselines. For environments from ToolBench, we compare ToolChain* with the state-

Table 3.1: Main experiment results (success rate) on ToolBench, including tool use scenarios of (1) Home Search, (2) Trip Booking, (3) Google Sheets, and (4) Virtual Home.

Models	GPT-3.5-turbo					GPT-4				
	Home Search	Trip Booking	Google Sheets	Virtual Home	Average	Home Search	Trip Booking	Google Sheets	Virtual Home	Average
GPT [4]	80.0	85.8	51.4	18.9	59.2	97.0	96.7	62.9	23.5	70.0
ReAct [17]	83.0	86.7	47.1	20.5	59.3	94.0	**97.5**	64.3	22.7	69.6
AdaPlanner [18]	90.0	87.5	55.7	20.7	63.5	97.0	**97.5**	66.7	27.1	72.1
ToT-DFS [19]	82.0	81.7	53.4	21.0	59.5	95.0	96.7	62.9	24.8	69.9
ToT-BFS (T=5) [19]	83.0	83.3	48.6	21.8	59.9	92.0	94.2	64.3	26.6	69.3
MCTS [27]	85.0	86.7	**62.9**	24.4	64.8	96.0	94.2	66.7	31.3	72.1
ToolChain*	**93.0**	**90.8**	61.4	**28.6**	**68.5**	**98.0**	**97.5**	**68.6**	**34.5**	**74.7**

of-the-art LLM planning algorithms from three main categories, including *open-loop systems* (**GPT** [4]), *closed-loop systems* (**ReAct** [17] and **Adaplanner** [18]), and *tree search-based systems* (**Tree-of-Thoughts** [19] and **MCTS** [27]). For mathematical reasoning problems, we employ a similar set of baselines as in the tool-use tasks. However, we exclude ReAct and AdaPlanner from mathematical reasoning evaluations. This is because they heavily depend on high-quality environment feedback to adjust action plans, which is unavailable in the GSM8K dataset. Additionally, since the action steps in the tool-use scenarios inherently form coherent sequences, we limit our comparison of ToolChain* to **Chain-of-Thought** [16] and **Self-Consistency** [118] only for the math reasoning task, and exclude it from the ToolBench evaluations.

3.5.2 Tool Use: ToolBench

We conduct experiments across four distinct planning tasks to assess the effectiveness and efficiency of ToolChain* in tool usage. The objective is to generate a sequence of API function calls to formulate a solution plan for each given task. For instance, these tasks include questions or requirements from users, *e.g.*, "*Could you help me find train tickets to Cape Coral?*". We present the main results, visualize the case study, analyze time-wise efficiency, and discuss ablation studies within the tool-use scenarios as follows. We report the success rate as the evaluation metric.

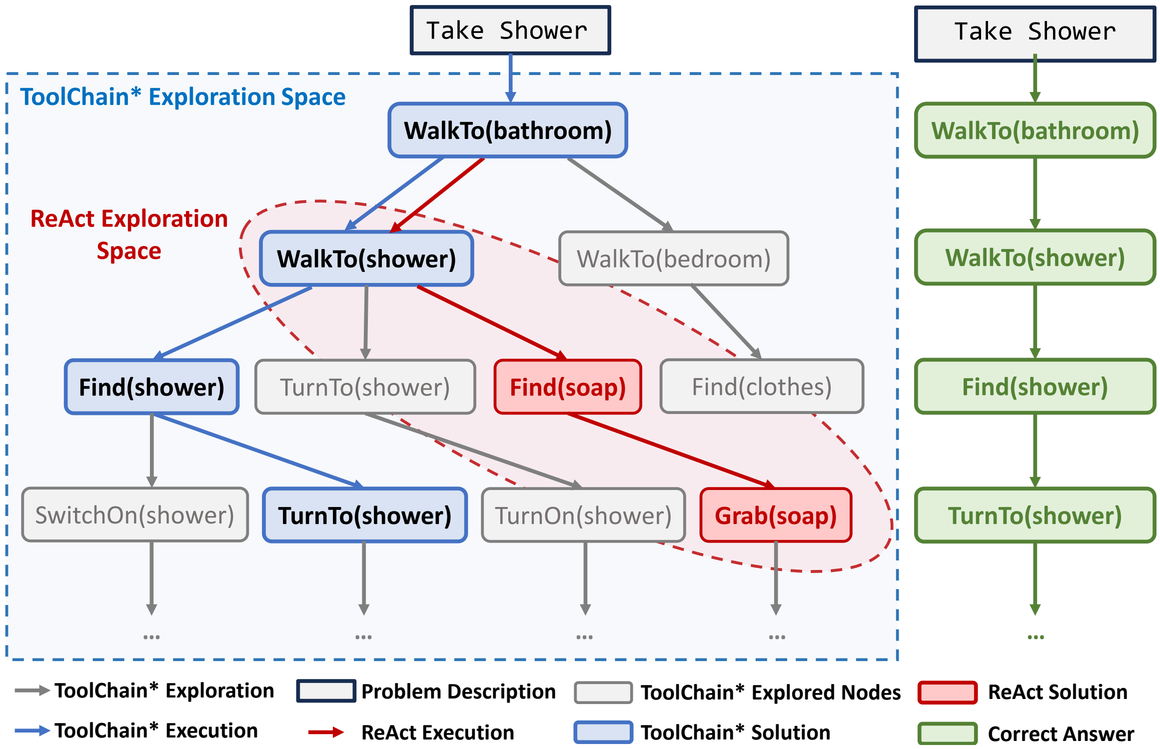


Figure 3.3: Case study of ToolChain* and *ReAct* [17] on Virtual Home dataset. Compared to ReAct with a unidirectional search (red), ToolChain* effectively enlarges search space (blue) with tree structures.

Results. Table 3.1 presents the main experiment results on ToolBench. Our proposed ToolChain* consistently outperforms across nearly all datasets, surpassing state-of-the-art baselines by margins of 3.7% and 2.5% with the base LLMs GPT-3.5-turbo and GPT-4, respectively. In comparison with the strongest closed-loop baseline AdaPlanner, ToolChain* improves the average success rate by 3.8%. This improvement is because AdaPlanner relies heavily on environmental feedback, which may not always be available in the tool-use scenarios. Without such high-quality feedback, closed-loop methods tend to explore a restricted trajectory within the action space, making them more susceptible to propagating errors from previous actions to future plans.

Moreover, ToolChain* not only surpasses the strongest tree search-based method, MCTS, but also shows the ability to exploit a better solution plan within the same exploration budgets. This is because our proposed task-specific cost function allows ToolChain* to prioritize the expansion of the most promising branches.

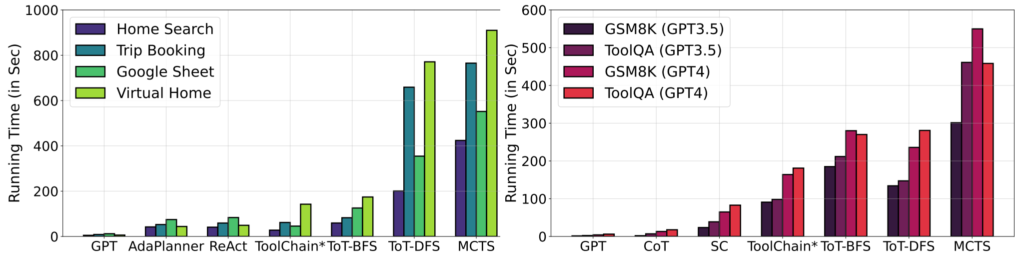
Case Study. Figure 3.3 depicts an example of ToolChain* (GPT-4) and *ReAct* [17] on a "*take shower*" task in Virtual Home dataset. According to the ground truth (green, "*shower*"), ToolChain* generates the correct action plan (blue, "*shower*") with an expanded search space, whereas the baseline searching method gets trapped in a locally optimal solution (red, "*soap*"). This suggests that by formulating and expanding upon a tree-based action space, ToolChain* is capable of effectively searching for the globally optimal solution in complex multi-step planning tasks.

Efficiency Evaluation. In terms of efficiency, we evaluate the running time of ToolChain* against all the baselines based on GPT-3.5-turbo, as depicted in Figure 3.4a. Remarkably, ToolChain* is 37.2% faster than the most efficient tree search-based method, Tree-of-Thoughts (BFS). This efficiency gain may stem from the proposed superior cost function, which efficiently navigates the most promising paths. Additionally, ToolChain* outpaces the best-performing tree search-based method, MCTS, by an impressive 415.84%. This discrepancy arises because ToolChain* focuses on expanding only the immediate next action during exploration. In contrast, MCTS goes through a more exhaustive process, simulating the entire future plan step by step using a rollout mechanism.

Table 3.2: Ablation studies on ToolBench.

	Home Search	Trip Booking	Google Sheets	Virtual Home	Average
ToolChain*	93.0	90.8	61.4	28.6	68.5
$-g_{1,t}(n)$	91.0	88.3	60.0	22.6	65.5
$-g_{2,t}(n)$	84.0	83.3	54.3	25.3	61.7
$-h_{1,t}(n)$	88.0	87.5	61.4	23.0	65.0
$-h_{2,t}(n)$	85.0	85.8	51.4	24.9	61.8
$-g(n)$	61.0	34.9	44.2	21.0	40.3
$-h(n)$	84.0	85.8	53.4	26.1	62.3

Ablation Studies. We conduct ablation studies to evaluate the effectiveness (success rate) of both the cumulative and future cost functions (Table 3.2). The results suggest that each component of the cumulative and future cost functions contributes to the performance of ToolChain*. This verifies the efficacy of our proposed cost functions in guiding the



(a) ToolBench. (b) GSM8K.

Figure 3.4: Time efficiency evaluation on (a) ToolBench and (b) GSM8K. We report the average running time in seconds over all instances in the dataset. ToolChain* achieves competitive efficiency to closed-loop systems without a tree structure and outpaces other tree search-based algorithms.

Table 3.3: Main results on math reasoning task in GSM8K and its hard subset collected in ToolQA.

Models	GPT-3.5-turbo		GPT-4	
	GSM8K	ToolQA	GSM8K	ToolQA
GPT	67.3	26.0	86.6	66.0
CoT	70.1	30.0	87.5	75.0
Self-Consistency	76.1	47.0	92.4	78.0
ToT-DFS	69.9	32.0	89.2	76.0
ToT-BFS	72.3	39.0	91.3	77.0
MCTS	74.7	27.0	91.0	74.0
ToolChain*	**77.0**	**52.0**	**93.5**	**84.0**

search through the decision tree. In addition, eliminating either the entire cumulative or future cost results in a marked decline in the success rate. Relying exclusively on the future cost results in a sharp performance drop of 28.2%, deteriorating ToolChain* to a greedy strategy that favors the shortest solution plans with the least number of actions. Conversely, if the search is guided only by the cumulative cost, ToolChain* essentially mirrors the behavior of the BFS algorithm, yielding similar performance.

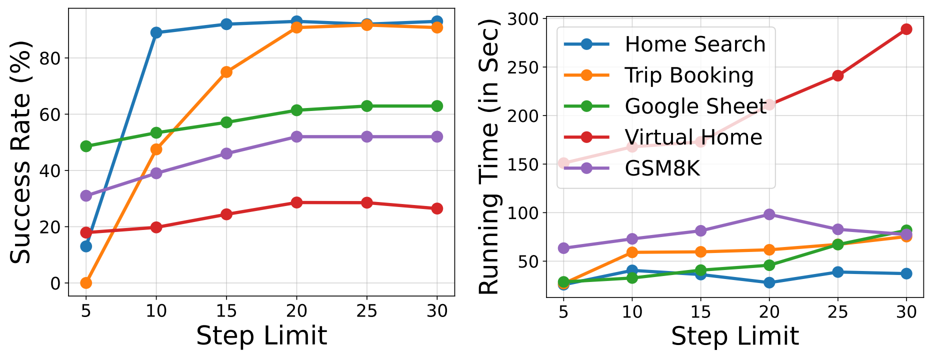
3.5.3 Math Reasoning: GSM8K

Beyond tool-use scenarios, we demonstrate the flexibility of ToolChain* by generalizing its application to mathematical reasoning for solving math word problems. We conduct experiments on the entire set of GSM8K and also a subset of hard questions from GSM8K collected in ToolQA [10].

Results. Table Table 3.3 presents the main experimental results (accuracy) for GSM8K and its challenging subset from ToolQA. Similar to tool-use studies (Table Table 3.1), ToolChain* consistently outperforms all baselines in both the original set and the challenging subset. These results demonstrate the flexibility and generalization capabilities of ToolChain* in mathematical reasoning tasks. Notably, ToolChain* demonstrates greater advantages over other baselines on ToolQA (hard questions) than on GSM8K, indicating its superior capability in solving complicated tasks. This is because simpler questions are composed of simple and static reasoning, eliminating the need for multiple branches. In contrast, challenging questions often involve complex reasoning, numerous intermediate steps, and multiple solution paths. The superior performance on hard subsets emphasizes the capability of ToolChain* in solving complicated reasoning problems. Furthermore, the efficiency analysis presented in Figure 3.4b indicates that ToolChain* ranks among the most efficient tree-based search baselines and has a time efficiency comparable to closed-loop systems without a tree structure.

3.5.4 Discussion: Empirical Analysis

From the comprehensive evaluations in planning and reasoning tasks presented in Sections subsection 3.5.2 and subsection 3.5.3, we validate that ToolChain* addresses the two core limitations of open-/closed-loop LLM-based agents, error propagation in multi-step solutions and constrained exploration in expansive action spaces. Meanwhile, we demonstrate ToolChain* a more efficient searching strategy compared to existing tree search-based agents. From the scaling-up analysis, alongside experimental results in Table 3.1 and efficiency metrics in Figure 3.4, we identify **a crucial trade-off** between effectiveness and efficiency in the direct application of tree search-based reasoning methods to complex tool use scenarios. To validate ToolChain* in solving these issues, we summarize key findings from experiments as follows: (1) From the main experimental results shown in Table 3.1 and Table 3.3, ToolChain* surpasses open-/closed-loop and tree search baselines in



(a) Performance

(b) Running Time

Figure 3.5: Scaling analysis of ToolChain*. (a) Performance and (b) running time on ToolBench and GSM8K when scaling up step limitations T.

complex multi-step planning and reasoning tasks, effectively **mitigating error propagation**. (2) From case studies in Figure 3.3, ToolChain* navigates the path toward an optimal solution by formulating the action space as a decision tree, thereby extensively **broadening the exploration space**. (3) From Figure 3.4, ToolChain* significantly accelerates the search process compared to other tree search-based methods, **achieving time efficiency** even comparable to closed-loop systems without a tree structure. (4) From tool-use in ToolBench to math problems in GSM8K, we show that ToolChain* is a **plug-and-play generalizable framework** applicable to a wide range of planning and reasoning problems. Notably, it exhibits exceptional proficiency in solving more challenging tasks, like ToolQA, compared to baselines. (5) There is a trade-off between search depth (*i.e.*, limitations on the number of steps) and the quality of the solution path (Figure 3.5). ToolChain* efficiently searches optimal solutions within limited steps, striking a **balance between exploration and exploitation**.

3.6 Conclusion

In this chapter, we introduce ToolChain*, an A* tree search-based planning algorithm to augment LLMs with external tools for complicated real-world planning and reasoning tasks. Compared to existing open- or closed-loop LLM agents, ToolChain* formulates the action space as a decision tree, thereby effectively mitigating error propagation and extensively expanding the search space. Furthermore, ToolChain* significantly accelerates the search process compared to other tree search-based methods, enabling tree search in complicated action space and striking a dynamic balance between exploration and exploitation. By achieving significant improvements over state-of-the-art baselines, ToolChain* showcases its potential as an efficient planning algorithm, navigating LLM-based agents in addressing complex real-world challenges.

CHAPTER 4

LIGHTWEIGHT ADAPTATION FOR BLACK-BOX LLM PERSONALIZATION

This chapter focuses on model factorization framework for black-box LLM personalization. The content is mainly based on the following publications:

- **Yuchen Zhuang**, Haotian Sun, Yue Yu, Rushi Qiang, Qifan Wang, Chao Zhang, and Bo Dai. "HYDRA: Model Factorization Framework for Black-Box LLM Personalization." In *The Thirty-eighth Annual Conference on Neural Information Processing Systems* (2024). [128]

4.1 Overview

Pre-trained LLMs [1, 2, 3, 4, 5, 43] have revolutionized various NLP tasks, ranging from traditional recommendation systems [129, 130, 131, 132] to modern virtual assistants [133, 134, 135, 136, 113, 137, 25]. Despite their strong capabilities, LLMs require further customization to consistently demonstrate desirable behaviors to each user and achieve optimal performance in specific use cases [138, 139, 140, 141, 34, 33, 41, 142, 143, 144, 145, 146]. As a result, LLM personalization has emerged as a rapidly evolving area of research [147], with the goal of tailoring the emergent abilities of LLMs to meet the unique needs of individual users.

Several existing studies have shown effectiveness in personalizing LLMs, including (1) fine-tuning personalized LLMs for each user [148, 149] and (2) aligning LLMs to personalized preferences through Reinforcement Learning from Human Feedback (RLHF) [150, 151, 152]. However, both fine-tuning and RLHF-based methods require access to model parameters, restricting their use to white-box LLMs only (e.g., LLaMA-2 [47]). These models tend to be less capable than black-box LLMs (e.g., GPT-3.5 [43]) because they

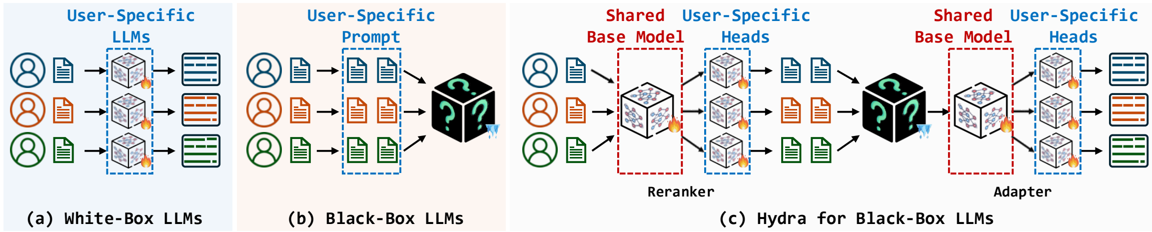


Figure 4.1: Personalization in white-box and black-box LLMs. Existing methods prioritize (a) fine-tuning user-specific models in white-box LLM personalization, while (b) designing user-specific prompts for black-box LLM personalization. In Hydra, we present (c) a learning-based model factorization solution to enhance the effectiveness of personalization in black-box LLMs. 🔥 indicates the trainable parameters, whereas ❄ indicates the inaccessible fixed parameters.

have access to less training data and smaller model scales. Moreover, RLHF-based methods require more explicitly attributed characteristics (*e.g.*, style) [150, 152] from implicit user behavior history and necessitate excessive annotation efforts for capturing human preferences.

Without access to modify the model parameters for black-box LLM personalization, an alternative solution is to augment user-specific content and/or context into the prompt template. One straightforward approach is to incorporate the user's complete profile or entire historical behavior into the prompt design [132, 153, 154, 155, 156, 157, 156]. However, integrating the entire profile may exceed the length limitations of LLMs and lead to substantial costs, while randomly selected records cannot effectively capture representative patterns. To address this dilemma, retrieval-augmented generation (RAG) approaches [158, 159, 160, 161] have been explored by extracting the most relevant information from the user's historical data to facilitate personalized generation. One limitation is that a retrieval-augmented framework encodes different users independently in a personalized prompt, making it challenging to capture the shared (global) patterns of the entire user group [148]. Moreover, in comparison to fine-tuning the entire or partial model parameters to create personalized language models for individual users [148, 150, 152], simply augmenting the input prompt through a centralized LLM without updating model parameters may diminish the effectiveness of personalization (Figure 4.1).

To address these challenges, we propose Hydra, a learning-based model factorization framework that captures both user-specific and shared behavior patterns to enable effective personalization within black-box LLMs. Hydra leverages a retrieval-augmented workflow, where a retriever initially extracts relevant user behaviors from historical data for effective user-specific preference identification. To achieve personalized generation, we focus on the training process of two fundamental components: (1) a personalized reranker to prioritize useful user information from the retrieved records, and (2) a personalized adapter to align black-box LLM outputs with user-specific preferences, without requiring access to internal model parameters. Both the reranker and the adapter can be decomposed into a base model with multiple personalized heads, similar to a Hydra. By employing model factorization, we effectively integrate shared (global) knowledge, captured by the centralized base model, with user-specific preferences, harnessed through multiple user-specific heads, to enhance generalization across the entire user group.

We conduct extensive experiments on LaMP [158], a comprehensive language model personalization benchmark, to evaluate the personalization capabilities of Hydra across multiple dimensions, including three text classification tasks and two text generation tasks. Notably, Hydra achieves an average improvement of 4.8% over the best-performing baselines across all five diverse tasks. Further in-depth studies reveal the robust capability of Hydra in scaling up to accommodate larger user groups and extensive behavior history, as well as adapting to user behavior shifts. We also demonstrate the effectiveness of shared knowledge in enhancing user experience through both quantitative and qualitative analyses. To facilitate future research in black-box LLM personalization, we will release the code repository and model checkpoints for transparency and reproducibility.

Our main contributions are as follows: (1) We propose Hydra, a black-box LLM personalization framework that effectively mines user behavior history and adapts to user preferences for enhanced user experience; (2) Hydra integrates shared (global) knowledge from the base model and individual (local) preference from multiple user-specific heads through model factorization to deliver generalizable personalization; and (3) Hydra significantly outperforms existing prompt-based methods across five diverse tasks in the LaMP benchmark [158], introducing a novel learning-based solution that achieves more effective adaptation to individual users in black-box LLMs.

4.2 Related Works

In-Context Learning. Vanilla personalized prompting approaches leverage the powerful in-context learning capability of LLMs by using the user's randomly sampled behavior history as contextual samples. Existing studies [132, 153, 154, 155, 156] have utilized encoding user histories-whether personal ratings, interaction histories, or exemplary reviews-as few-shot examples to facilitate LLMs in generating personalized content in various downstream applications. Additionally, research has shown that utilizing a longer user history can potentially lead to better performance [157, 156].

Profile-Augmented Prompting. Improving upon the random sample strategy and leveraging the insights from the enhanced performance with more historical information, profile-augmented generation (PAG) summarizes user preferences and behavior patterns into natural language profiles for query augmentation. For instance, Richardson et al. [162] employ instruction-tuned LLMs to generate abstract summaries of user history data, integrating summarization for enhanced personalization. Similarly, ONCE [163] creates user profiles by summarizing topics and regions of interest from their browsing history, thereby assisting LLMs in capturing user preferences for downstream tasks.

Retrieval-Augmented Prompting. Compared to random sampling in in-context learning and the use of entire histories in PAG, retrieval-augmented prompting excels at extracting the most relevant records from user behavior history to enhance LLM personalization, thereby efficiently managing the growing user behavior data within LLMs' limited context length and supporting personalized generation with more relevant evidence. For instance, LaMP [158, 159] introduces a retrieval-augmented method to obtain the most relevant content from the user's behavioral history and incorporate it into the prompt design. Similarly, AuthorPred [161] retrieves relevant past user-written documents for personalized text generation. Pearl [160] proposes a generation-calibrated retriever to select from historic user-authored documents, enhancing personalization with relevant prompt augmentation.

Limitations. Similar to in-context learning, personalization based on PAG is prone to be easily distracted by irrelevant information retrieved, especially when there is a shift in user behavior between the current query and the user's historical records. Despite the improvement upon PAG by retrieving relevant information, RAG-based methods may still suffer from the quality of retrieved information, where the "most relevant" information may not necessarily be the "most useful" information to answer a new query. Additionally, prompting-based methods not only lack deeper personalization analysis due to their reliance on a single centralized model but also lack access to global knowledge due to the user-specific prompt design.

4.3 Hydra: Model Factorization for Black-Box LLM Personalization

4.3.1 Problem Formulation: Black-Box LLM Personalization

Black-box LLM personalization refers to tailoring model generations to align with user preferences according to their history [147, 148, 152], without having access to model parameters. Specifically, given a black-box LLM G and a training dataset $\mathcal{D}_{\text{train}} = \{(q_u, r_u, \mathcal{H}_u)\}$, where for each user u, q_u indicates the input sequence, r_u refers to the target output, and $\mathcal{H}_u$ represents the user's historical behavior containing preference information. The user history data $\mathcal{H}_u = \{h_u^i\}$ includes all user behaviors h_u^i, consisting of (q_u^i, r_u^i) pairs, mirroring the task-specific query-answer format (q_u, r_u). The goal of personalization is to adapt the LLM's generation $\hat{r}$ to the target output r, conditioned on both the input and the user history.

Traditional fine-tuning methods train universal models for all users, whereas personalized tuning aims to develop a unique model for each user u, denoted as $\theta^{(u)}$, which captures the unique characteristics of its user-specific data distribution $\mathcal{D}^{(u)}$. Ideally, each user model should possess both *shared* (general) and *user-specific* knowledge. Thus, the problem can be formulated as a decomposition of $\theta^{(u)}$ into a shared part σ and a personalized part $\tau^{(u)}$ to learn general and user-specific knowledge, respectively. The training object of LLM personalization can be formulated as:

$$\min_{\sigma,\tau^{(1)},\tau^{(2)},\cdots,\tau^{(u)}} \sum_{u=1}^{U} \mathcal{L}_u(\sigma;\tau^{(u)};\mathcal{D}^{(u)}), \tag{4.1}$$

where $\mathcal{L}_u(\sigma;\tau^{(u)};\mathcal{D}^{(u)})$ denotes the loss function of user u, and U is the total number of users. Specifically, in the black-box LLM personalization, the model parameters in G are not accessible, making it infeasible to directly fine-tune the black-box LLM.

Thus, we present Hydra, a model factorization framework for black-box LLM personalization, as shown in Figure 4.2.

4.3.2 Retrieve-then-Rerank

To capture user-specific preference, we follow a *retrieve-then-rerank* workflow to (1) retrieve the relevant user behavior records and (2) rerank them based on usefulness. Specifically, given an input query x, we employ a retriever to retrieve top-N user history behaviors that have the highest relevance scores when compared with the input x. The objective of the reranker is to identify user historical records that can serve as useful user preference information for answering the user's incoming queries.

Training Data Generation. Within the training data, each user only has a single query, which is insufficient for capturing the relationships between the behaviors of the same user. Scoring all the history pairs of users is quadratic in $\sum_{u=1}^{|\mathcal{D}|} |\mathcal{H}_u| \times |\mathcal{H}_u|$, which becomes prohibitively time-consuming. In order to obtain a high-quality candidate set of training samples for each user query q_u, we retrieve the top-M relevant history records, denoted as $\mathcal{R}(q_u, \mathcal{H}_u)$. Additionally, to gain a better understanding of the user's history, we randomly

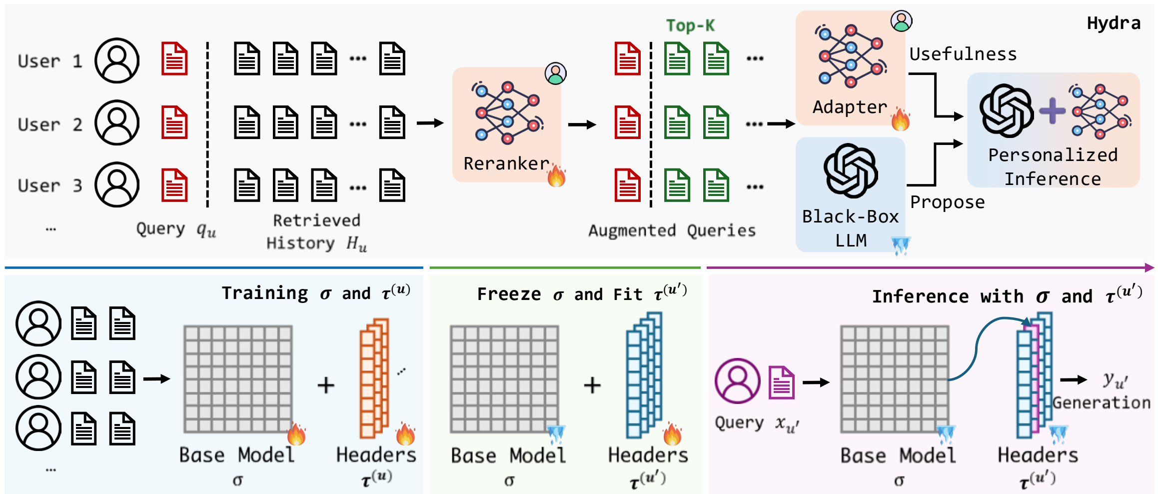


Figure 4.2: Overview of Hydra. Hydra follows a retrieval-augmented framework: (1) Firstly, we extend an original RAG to a two-stage *retrieve-then-rerank* workflow, where we rerank the most useful information from relevant user behavior records to capture user-specific preference (Section subsection 4.3.2); (2) Secondly, augmented by the selected historical data, we train an adapter to align the output of black-box LLMs with personalized human preference (Section subsection 4.3.3). Both the reranker and the adapter can be decomposed into a base model with multiple user-specific heads, resembling a hydra-like structure (Section subsection 4.3.4). The base model maintains *shared* knowledge across users, while multiple personal heads capture *user-specific* preferences. represents the model decomposition for personalization.

sample another M historical records that can be considered as the user's previous queries. For each of these previous queries, we also retrieve the top-M relevant historical records. The training candidates can be represented as:

$$\mathcal{E}_{\text{reranker}} = \{(q_i, r_i, h_i)\}_{i=1}^{|\mathcal{E}_{\text{reranker}}|} = \{\{(q_u, r_u, \mathcal{R}(q_u, \mathcal{H}_u))\} \cup \{(h_u^i, \mathcal{R}(h_u^i, \mathcal{H}_u / h_u^i))\}_{i=1}^{M}\}_{h_u^i \in \mathcal{H}_u u \in \mathcal{D}}, \tag{4.2}$$

where q_i indicates the query, r_i indicates the ground-truth answer to the query, and h_i indicates the candidate history. We then utilize an LLM as a labeling function to measure the potential improvement (*i.e.*, usefulness) of each history upon the LLM personalized generation. Specifically, for each candidate (q_i, r_i, h_i), we first sample the generation of LLM $\hat{r}_i$ by using the input context q_i and the candidate history h_i from $p_{\text{LLM}}(\hat{r}_i | q_i, h_i)$. Next, we compare the generation $\hat{r}_i$ with the ground-truth answer r_i, and create a binary label $y_i = \mathbb{1}(\hat{r}_i = r_i)$. For generation task, the condition is soften to $y_i = \mathbb{1}(\hat{r}_i \approx r_i)$, where the Rouge metrics between r_i and $\hat{r}_i$ is above a pre-defined threshold. To optimize the reranker, we combine the user query with the candidate history in an entailment learning style, resulting in the training inputs $x_i = \{\text{[CLS]}\ q_i\ \text{[SEP]}\ h_i\ \text{[SEP]}\}$. Therefore, we can curate a training set for the reranker $\mathcal{D}_{\text{reranker}} = \{(x_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{reranker}}|}$.

Hydra-Reranker Training. Training the reranker for each user allows for a personalized selection of relevant historical information based on the user's query. The training objective involves using the cross-entropy loss between the predictions made by Hydra-Reranker heads and the ground truth. For a specific user u and the corresponding training sample $(x_i^u, y_i^u) \in \mathcal{D}_{\text{reranker}}$, we can calculate the cross-entropy loss function for Hydra-Reranker as follows:

$$\mathcal{L}_{Hydra}^{\text{reranker}} = -y_i^u \log p_i^u - (1 - y_i^u) \log(1 - p_i^u), \tag{4.3}$$

where p_i^u indicates the model prediction made by Hydra-Reranker.

Hydra-Reranker Inference. With the trained reranker, our objective is to select the top-k most relevant candidates from the retrieved user history for a personalized generation. For each query q_i from user u in the test data, we retrieve the most relevant history $\overline{\mathcal{H}}_i^u = \{h_{i,1}^u, \cdots, h_{i,N}^u\}$ and concatenate them to form the test inputs $\{x_{i,1}^u, \cdots, x_{i,N}^u\}$. By feeding these into Hydra-Reranker, we obtain the corresponding predictions $\{p_{i,1}^u, \cdots p_{i,N}^u\}$. From the test inputs, we select the top-k history candidates $\mathcal{C}_i$ with the highest level of usefulness:

$$\mathcal{C}_i = \arg\text{top-}k_{h_{i,j} \in \overline{\mathcal{H}}_i}(p_{i,j}^u). \tag{4.4}$$

4.3.3 Black-box LLM Adaptation

Complementary to personalizing the few-shot demonstrations for LLMs in Section subsection 4.3.2, we further align the black-box LLM generation to user-specific preference through the training of a personalized adapter, eliminating the need for directly accessing model parameters.

Training Data Generation. To take full advantage of the preference information hidden in the user query and history, we generate the candidates for the adapter training:

$$\mathcal{E}_{\text{adapter}} = \{(q_i, r_i)\}_{i=1}^{|\mathcal{E}_{\text{adapter}}|} = \{q_u, r_u\}_{u \in \mathcal{D}} \cup \{\{(q_u^i, r_u^i)\}_{i=1}^{|\mathcal{H}_u|}\}_{u \in \mathcal{D}}. \tag{4.5}$$

For each input query q_i from the training dataset, we augment the reranked history candidates $\mathcal{C}_i$ with the query and sample b candidate responses from the black-box LLM:

$$\{\hat{r}_{i,j}\}_{j=1}^{b} \sim p_{\text{LLM}}(\hat{r}_i | \mathcal{C}_i, q_i). \tag{4.6}$$

In comparison with the ground-truth personalized generation r_i, we can evaluate each generated solution $\hat{r}_{i,j}$ and assign a corresponding binary label y_i as $y_i = \mathbb{1}(\hat{r}_{i,j} = r_i)$. Using the model generations, we establish a new dataset for the adapter training, denoted as $\mathcal{D}_{\text{adapter}} = \{x_{i,j}, y_i\}_{i=1}^{|\mathcal{D}_{\text{adapter}}|}$, where $x_{i,j} = \{$[CLS] q_i [SEP] $\hat{r}_{i,j}$ [SEP]$\}$ represents the concatenation of the user query with the entire candidate generation.

Hydra-Adapter Training. We utilize the model decomposition training strategy (detailed in Section subsection 4.3.4) to train personalized adapters for each user, which allows us to tailor the selection of candidate generation that best aligns with the user's preference. To calculate the task-specific loss function, we follow Eq. equation Equation 4.3 and employ the same cross-entropy loss between the predictions of user-specific heads and the ground truth to optimize Hydra-Adapter.

Hydra-Adapter Inference. During the process of model inference, we conceptualize the black box LLM as a proposal generator, while the adapter functions as an evaluator. Specifically, for each test query x_i from user u, we adopt the best-of-b inference, also known as rejection sampling. Initially, we sample b candidate generations $\{\hat{r}_{i,j}\}_{j=1}^{b}$ from the LLM. By passing them through the Hydra-Adapter, we obtain the corresponding score for each candidate $\{p^u_{i,1}, \cdots, p^u_{i,b}\}$. The solution with the highest score is then chosen as the final answer:

$$\hat{r}^u_i = \arg\max_{j=1,\cdots,b} p^u_{i,j}. \tag{4.7}$$

4.3.4 Model Factorization for Personalization

To further effectively capture the shared knowledge across all users as well as the individual user-specific behavior patterns, a direct solution is to develop a parameter decomposition method, where the shared parameters store the general knowledge, benefiting from a high model capacity, while the remaining parameters are designed to learn user-specific preferences that complement the overall understanding (typically on a smaller scale). Hence, it is adequate to employ a smaller-sized model (adapter) to represent these personalized parameters, instead of fine-tuning the entire LLM.

Model Factorization. Formally, in Hydra, we assume that each personalized language model is associated with a set of weights $\Theta = \{\theta^{(u)}\}_{u \in \mathcal{D}}$, where $\theta^{(u)}$ represents the weights of the personalized model for user u. Each $\theta^{(u)}$ is decomposed as:

$$\theta^{(u)} = \sigma \otimes \tau^{(u)}, \tag{4.8}$$

where σ represents the base model parameter matrix that is shared among all users, $\tau^{(u)}$ represents a personalized head model, and $\otimes$ indicates the operation of appending the user-specific head $\tau^{(u)}$ on top of the base model σ. Specifically, we add $|\mathcal{D}|$ heads to the final hidden states s generated by the original model. The u-th head measures the usefulness between query and histories, or the level of preference for the generation from the perspective of the u-th user. The prediction of the u-th head is denoted as $p^{(u)}$. For each head, we employ a single layer of feed-forward network. We define u-th head as

$\tau^{(u)} = [\mathbf{W}_1^{(u)}, \mathbf{W}_2^{(u)}, \mathbf{b}_1^{(u)}, \mathbf{b}_2^{(u)}]$ and the prediction can be outlined as:

$$p^u = \text{softmax}(\mathbf{W}_2^{(u)} \cdot \text{Tanh}(\mathbf{W}_1^{(u)} \cdot s + \mathbf{b}_1^{(u)}) + \mathbf{b}_2^{(u)}), \text{ where } \mathbf{W}_2^{(u)} \in \mathbb{R}^{d \times o}, \mathbf{W}_1^{(u)} \in \mathbb{R}^{d \times d}, \tag{4.9}$$

where d indicates the dimension of hidden states and o indicates the dimension of outputs.

Training Strategy. We then train the base model in conjunction with multiple user-specific heads. For each sample (x_i^u, y_i^u) of user u from the training data and the model prediction p_i^u, we update the base model and the u-th head accordingly:

$$\sigma \leftarrow \sigma - \alpha \nabla \mathcal{L}(y_i^u, p_i^u),\ \tau^{(u)} \leftarrow \tau^{(u)} - \alpha \nabla \mathcal{L}(y_i^u, p_i^u), \tag{4.10}$$

where α indicates the learning rate and $\mathcal{L}$ represents the task-specific loss function.

Fit Test User History. In order to accommodate the newly incoming users in the test data, we cannot reuse the personalized heads from the training data. Therefore, we need to create and initialize new heads. To fit the test users' history, we freeze the base model and solely focus on the heads. This fitting process is simple and requires minimal computational resources. Similar to the update of $\tau^{(u)}$ in Eq. (Equation 4.10), when given a new user u' for testing, we also leverage the task-specific loss function $\mathcal{L}(\cdot)$ to update the head $\tau^{(u')}$:

$$\tau^{(u')} \leftarrow \tau^{(u')} - \alpha \nabla \mathcal{L}(y_i^{u'}, p_i^{u'}), \tag{4.11}$$

where $(x_i^{u'}, y_i^{u'})$ are obtained from test user history.

Personalized Inference for Test Users. Upon fitting the test user history, we obtain the base model σ, which contains general knowledge across users, and the user-specific head $\tau^{(u')}$, which captures the user-specific knowledge in the test user history. Therefore, we can apply personalized inference directly to the test user u', given its query $x_{u'}$:

$$p_i^{u'} = f_{\sigma \otimes \tau^{(u')}}(x_i^{u'}), \tag{4.12}$$

where $f_{\sigma \otimes \tau^{(u')}}(\cdot)$ indicates the inference of the model $\sigma \otimes \tau^{(u')}$.

4.4 Experiments

4.4.1 Experimental Setup

Datasets and Tasks. We adopt a widely used language model personalization benchmark, LaMP [158], focusing on a diverse set of personalized text classification and generation tasks, including (1) Personalized News Categorization (**LaMP-2N**), (2) Personalized Movie Tagging (**LaMP-2M**), (3) Personalized Product Rating (**LaMP-3**), (4) Personalized News Headline Generation (**LaMP-4**), and (5) Personalized Scholarly Title Generation (**LaMP-5**). For data splitting, we follow the user-based separation setting provided by the LaMP benchmark, with 100 randomly selected users for training and an additional 50 randomly selected users for testing. No shared users are displayed across splits for specific measurements in personalization for new users.

Baselines. We compare our proposed Hydra with existing state-of-the-art black-box LLM personalization methods, including in-context learning (**ICL-Random**, with random selected k-item from user behavior history), retrieval-augmented prompting (**RAG**) [158], and profile-augmented prompting (**PAG**) [162]. We also present the experimental results of `gpt-3.5-turbo` [43] (zero-shot) without profile/history augmentation in order to showcase the baseline performance of the backbone language model. Learning-based personalization, such as fine-tuning and RLHF-based methods, cannot be applied to black-box LLMs (*e.g.*, `gpt-3.5-turbo`) due to the unavailability of model parameters.

Evaluation Metrics. Following the evaluation metrics specified in LaMP [158], we utilize accuracy (Acc.) and F-1 score (F-1) for the test classification tasks in LaMP-2N and LaMP-2M. Additionally, we employ mean absolute error (MAE) and root mean squared error (RMSE) for the ordinal multi-class classification task in LaMP-3. To comprehensively evaluate the personalized text generation tasks in LaMP-4 and LaMP-5, we report the ROUGE-1 (R-1), ROUGE-L (R-L), and BLEU metrics.

Table 4.1: Main experiment results on the LaMP benchmark. R-1 and R-L represent ROUGE-1 and ROUGE-L, respectively. k indicates the number of retrieved items. ↑ denotes that higher values are better, while ↓ implies that lower values are preferred. The best score and 2nd best score for each task are highlighted in **bold** and underlined, respectively. Notations are consistent across tables.

Dataset (→)	**LaMP-2N**		**LaMP-2M**		**LaMP-3**		**LaMP-4**			**LaMP-5**		
Method (↓)	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑	R-1 ↑	R-L ↑	BLEU ↑
`gpt-3.5-turbo` [43]	0.660	0.280	0.440	0.309	0.480	0.825	0.133	0.120	0.996	0.379	0.326	5.477
ICL-Random (k=1)	0.640	0.293	0.480	0.360	0.660	0.990	0.165	0.144	1.567	0.410	0.349	5.327
ICL-Random (k=2)	0.640	0.284	0.520	0.400	0.560	0.917	0.171	0.158	1.923	0.413	0.348	6.698
ICL-Random (k=4)	0.660	0.288	0.480	0.356	0.560	0.917	0.163	0.148	1.589	0.393	0.349	7.445
RAG (k=1) [158]	0.680	0.293	0.400	0.290	0.500	0.837	0.151	0.130	1.417	0.418	0.353	5.852
RAG (k=2) [158]	0.600	0.281	0.460	0.343	0.580	0.906	0.173	0.156	1.778	0.419	0.367	6.898
RAG (k=4) [158]	0.640	0.284	0.460	0.340	0.580	0.970	0.172	0.156	1.812	0.415	0.362	7.382
PAG (k=0) [162]	0.640	0.293	0.500	0.356	0.520	0.894	0.163	0.150	1.724	0.410	0.359	6.124
PAG (k=1) [162]	0.680	0.308	0.520	0.362	0.560	0.913	0.170	0.156	1.674	0.397	0.327	6.481
Hydra-Reranker (k=4)	0.760	0.375	0.520	0.393	0.480	0.775	0.177	0.166	1.951	0.423	0.368	6.864
Hydra-Adapter	0.680	0.277	0.480	0.385	0.420	0.762	0.145	0.118	1.137	0.409	0.355	5.816
Hydra	**0.780**	**0.401**	**0.540**	**0.458**	**0.400**	**0.747**	**0.178**	**0.169**	**2.396**	**0.434**	**0.372**	**7.531**

Table 4.2: Ablation studies of Hydra. -P. represents removing the Hydra personalized training strategy.

Dataset (→)	**LaMP-2N**		**LaMP-2M**		**LaMP-3**		**LaMP-4**			**LaMP-5**		
Method (↓)	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑	R-1 ↑	R-L ↑	BLEU ↑
Hydra	**0.780**	**0.401**	**0.540**	**0.458**	**0.400**	**0.747**	**0.178**	**0.169**	**2.396**	**0.434**	**0.372**	**7.531**
-P.-Adapter	0.740	0.300	0.500	0.384	0.460	0.831	0.169	0.157	1.604	0.398	0.336	5.766
-P.-Reranker	0.700	0.298	0.500	0.379	0.480	0.825	0.162	0.154	2.063	0.399	0.347	5.364
-P.-Adapter & Reranker	0.780	0.312	0.420	0.339	0.520	0.894	0.172	0.163	2.001	0.385	0.332	5.848
Hydra-Reranker (k=4)	0.760	0.375	0.520	0.393	0.480	0.775	0.177	0.166	1.951	0.423	0.368	6.864
-P.-Reranker	0.700	0.358	0.480	0.372	0.540	0.864	0.174	0.161	1.772	0.411	0.363	6.192
Hydra-Adapter	0.680	0.277	0.480	0.385	0.420	0.762	0.145	0.118	1.137	0.409	0.355	5.816
-P.-Adapter	0.673	0.276	0.460	0.341	0.480	0.835	0.141	0.119	1.006	0.374	0.317	4.754

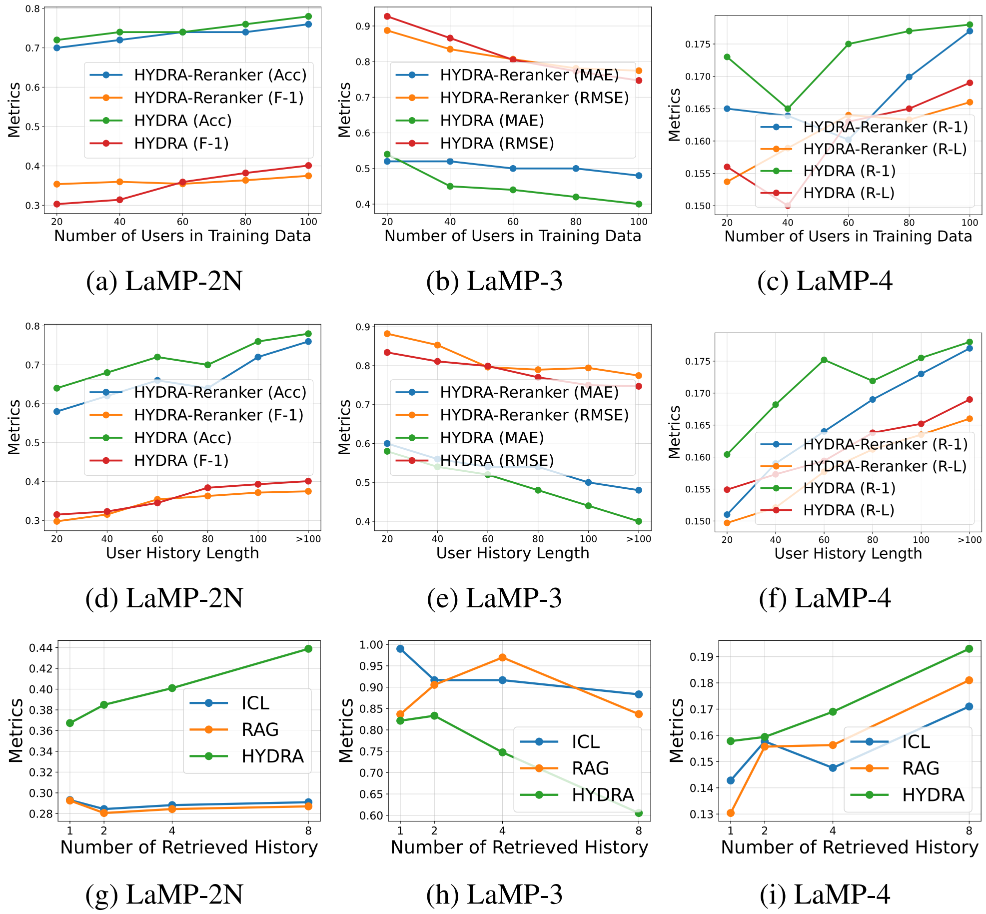
Implementation Details. For both baselines and our proposed Hydra, we follow the same prompt template in the LaMP benchmark [158] and employ `gpt-3.5-turbo(1106)` and BM25 [73] as the backbone black-box LLM and default retriever, respectively. Additionally, both Hydra-Reranker and Hydra-Adapter leverage the lightweight `LongFormer-Base` (110M) [164] as the backend language models.

4.4.2 Main Results

Table Table 4.1 presents the main experimental results of five varied personalization tasks in the LaMP benchmark. Compared to the zero-shot setting in `gpt-3.5-turbo` [43], even a random selection of historical records from user behavior or profiles enhances the model performance in most tasks, suggesting that personalization contributes to improved performance with black-box LLMs. Hydra exhibits substantial relative improvements across all five tasks, with an average of 9.01% over the best-performing baselines. Specifically, Hydra outperforms the best-performing alternative by 14.71% in accuracy for the Personalized News Categorization (LaMP-2N) task, 3.85% in accuracy for the Personalized Movie Tagging (LaMP-2M) task, 20.00% in MAE (where lower values indicate better performance) for the Personalized Product Rating (LaMP-3) task, 2.89% in R-1 for the Personalized News Headline Generation (LaMP-4) task, and 3.58% in R-1 for the Personalized Scholarly Title Generation (LaMP-5) task. This further improvement can be attributed to the improved quality (usefulness) of retrieved records in better representing user-specific preferences and the effective integration of global knowledge sourced from the entire user group.

4.4.3 Ablation Studies

From Table 4.1, both Hydra-Reranker and Hydra-Adapter demonstrate their effectiveness by significantly improving the RAG and PAG baselines separately. Furthermore, we observe that Hydra outperforms each of them individually, showcasing the complementary role of both components. In Table 4.2, we further eliminate the personalized training strategy of Hydra, deteriorating to training a singular model across the entire user group, which solely incorporates global knowledge and lacks individualized customization for each user. The decline in model performance across all tasks highlights the significance of personalization through user-specific heads.



(a) LaMP-2N (b) LaMP-3 (c) LaMP-4

(d) LaMP-2N (e) LaMP-3 (f) LaMP-4

(g) LaMP-2N (h) LaMP-3 (i) LaMP-4

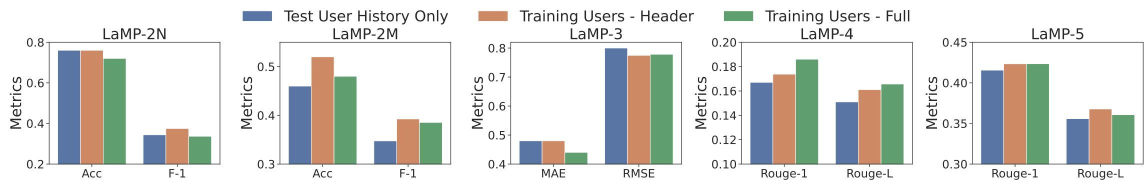
Figure 4.3: Scale-up analysis for (a)–(c) effect of users in training data, (d)–(f) effect of historical records per user, and (g)–(i) effect of selected historical records (k) per user.

4.4.4 Scale-up Analysis

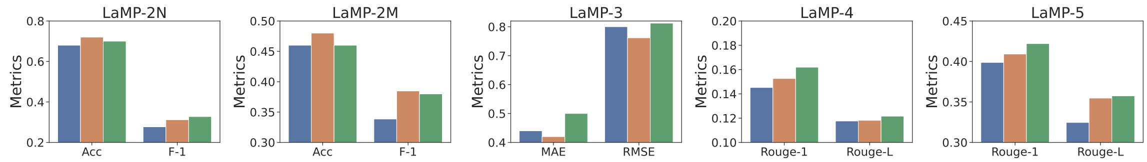
Number of Users in Training.

We study the impact of users in the training data, as depicted in Figure 4.3 (a)-(c). As the number of users increases from 20 to 100, we observe that Hydra reaches over 90%

Number of History per User. We examine the impact of the number of historical records per user in Figure 4.3 (d)-(f). We randomly chose 50 users from each range of number of history per user. As the number of historical records increases, we consistently observe improved performance in both classification and generation tasks, as indicated by all metrics. This demonstrates the robustness of Hydra in effectively capturing user-specific preferences, especially with a larger amount of user historical data.



(a) Hydra-Reranker



(b) Hydra-Adapter

Figure 4.4: Performance under different levels of shared knowledge.

Number of Selected History (k) per User. In Figure 4.3 (g)-(i), we analyze the impact of the selected items k per user. It can be observed that Hydra consistently outperforms other baselines across all values of k, highlighting the robustness of Hydra. Additionally, it is evident that model performance correlates with the number of retrieved historical records in most cases, providing further evidence of the effectiveness of the retrieval-augmented framework in black-box LLM personalization. However, an inconsistency is noted with RAG in LaMP-3, which may be attributed to the presence of noisy or irrelevant retrieved history. This finding emphasizes the importance of retrieval quality and raises the significance of the Hydra-Reranker in measuring the usefulness of retrieved items.

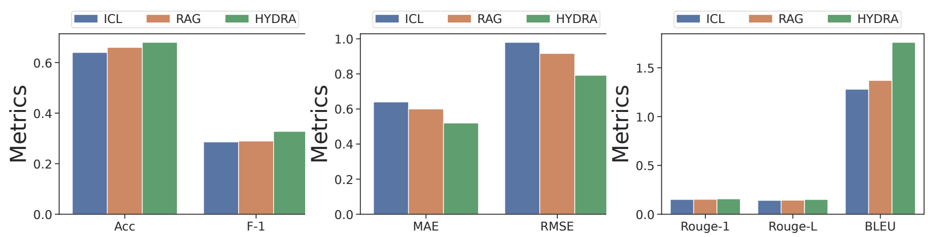
Number of Users in Training Set. We also conduct additional scale-up experiments (Table 4.3) to evaluate HYDRA with an increased number of users, increasing from 100 to 1000, across all five tasks. Our findings from the scale-up experiments show that HYDRA maintains its performance advantages over baselines as the number of users increases.

4.4.5 Empirical Personalization Analysis

Performance w/o Shared Knowledge. We conduct additional experimental analysis on both text classification and generation tasks for a comprehensive evaluation. In Figure 4.4, we study how the shared knowledge is leveraged in Hydra. Specifically, we create three set-

Table 4.3: Scale-up experiment results on the LaMP benchmark, including 1000 and 500 users during training and testing, respectively.

Dataset (→)	LaMP-2N		LaMP-2M		LaMP-3		LaMP-4			LaMP-5		
Method (↓)	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑	R-1 ↑	R-L ↑	BLEU ↑
`gpt-3.5-turbo`	0.638	0.499	0.412	0.347	0.540	0.851	0.133	0.119	1.043	0.439	0.371	6.018
ICL-Random (k=1)	0.598	0.476	0.392	0.335	0.676	0.959	0.147	0.132	1.330	0.457	0.396	8.118
ICL-Random (k=2)	0.632	0.499	0.376	0.311	0.562	0.871	0.151	0.137	2.388	0.451	0.393	8.550
ICL-Random (k=4)	0.630	0.518	0.392	0.352	0.440	0.740	0.161	0.146	2.418	0.457	0.396	8.404
RAG (k=1)	0.610	0.486	0.408	0.345	0.602	0.871	0.154	0.138	1.649	0.468	0.405	7.820
RAG (k=2)	0.624	0.479	0.380	0.315	0.559	0.836	0.161	0.149	2.958	0.480	0.419	9.021
RAG (k=4)	0.656	0.524	0.392	0.339	0.391	0.716	0.167	0.155	3.615	0.479	0.418	9.108
PAG (k=0)	0.618	0.489	0.404	0.340	0.583	0.872	0.161	0.141	1.950	0.460	0.405	7.372
PAG (k=1)	0.630	0.500	0.418	0.357	0.414	0.787	0.163	0.153	2.934	0.474	0.414	8.372
Hydra	**0.748**	**0.551**	**0.446**	**0.373**	**0.328**	**0.656**	**0.175**	**0.167**	**4.772**	**0.508**	**0.442**	**9.519**



(a) LaMP-2N (b) LaMP-3 (c) LaMP-4

Figure 4.5: Performance under user behavior shift.

tings with different levels of leveraging shared knowledge: (1) using *only test user history* for training, (2) updating just the *head* parameters using training user data, and (3) updating the *full* model parameters using training user data. We observe that incorporating a larger volume of training user data yields superior results compared to solely relying on test user history. The inclusion of additional training user information aids in capturing global information effectively. In addition, updating the entire model shows inferior performance compared to updating only the heads. This discrepancy arises due to the increased number of parameters, which necessitates a larger amount of data to accurately capture each user's information. Additionally, the transition from training user information to testing user information results in a domain shift of global knowledge.

Performance under User Behavior Shift. Following the setup in Tan et al. [148], we evaluate Hydra under user behavior shift when the query is not relevant to all historical

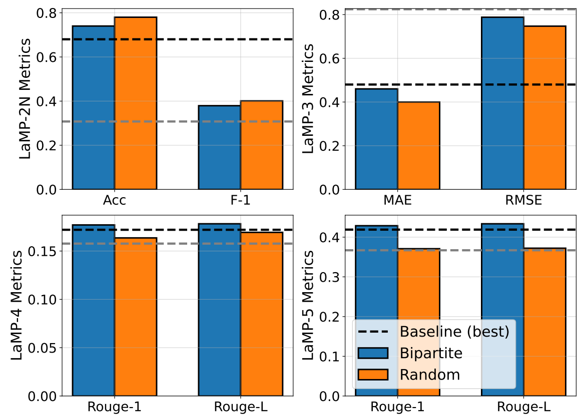


Figure 4.6: Experiments on LaMP benchmark of significant disparities in user behavior history.

records. As illustrated in Figure 4.5, our proposed Hydra continues to outperform the current state-of-the-art baselines in the face of user behavior shift. This serves as evidence of its robustness and generalizability in providing a widely applicable solution for black-box LLM personalization. This improvement can be attributed to the personalized reranker, which reevaluates the candidates based on their utility rather than solely on relevance, together with the personalized adapter further adjusts to align with user preferences.

Effect of Significant Disparities in User Behavior. To consider more extreme cases of disparities in user behavior, we retrain HYDRA on a mixture of 50% users with the fewest interactions and another 50% users with the most interactions (blue in Figure 4.6, compared to the random selection in orange). The black and gray dashed lines represent the best-performing baselines on the first and second metrics. The experimental results demonstrate that HYDRA consistently outperforms existing baselines, even under extreme cases. Compared to the previous random selection of training users, HYDRA achieves relatively lower performance due to the imbalance of training samples for dense users and sparse users. By leveraging the global information in shared parameters, knowledge can be effectively transferred from dense users to sparse users, thereby enabling further personalization through the utilization of sparse user-specific head models.

4.5 Conclusion

In this chapter, we proposed Hydra, a model factorization framework for black-box LLM personalization that captures and leverages both user-specific and shared behavior patterns from historical data. Compared to prompt-based methods, Hydra introduces a novel learning-based paradigm for black-box LLM personalization that not only remains highly effective in personalizing outputs by training a reranker with a black-box LLM adapter, but also eliminates the need for access to model parameters, presenting a promising alternative to existing mainstream techniques. Experimental results demonstrate that Hydra outperforms state-of-the-art RAG and PAG baselines by an average relative improvement of 9.01% across five diverse personalization tasks. Hydra establishes a foundation for learning-based black-box LLM personalization, facilitating targeted enhancements in user experience and ultimately contributing to the development of human-centric intelligent systems.

Appendices

APPENDIX A
ADDITIONAL DETAILS FOR TOOLQA

A.1 Additional Related Works

We list the state-of-the-art related works in tool-augmented LLMs in Table A.1. All of them can be categorized into two groups: (1) single-tool methods, that focus on making a single API call perfect in the solution; (2) multi-tool methods, that emphasize more on studying how to compose different tools together to solve a challenging problem. ToolQA is more suitable for the evaluation of the second category to test the inherent logical reasoning behind different tools. Additionally, there exist other notable contributions [167, 168, 18] within the realm of decision-making that specifically emphasize the planning capabilities of expansive language models. These endeavors can be regarded as methods affiliated with tools, wherein the actions within generated plans are analogous to distinct tools utilized for specific purposes.

A.2 Data Sources

A.2.1 Different Data Source Introduction

- **Flight Status (2022-2023)**[1] contains almost all flight information of airlines between 2022 and 2023, which is too contemporary for LLMs' internal knowledge.
- **Daily Coffee Price (2000-2022)**[2] contains the daily price of coffee, ranging from 2000 to 2022, where the information is too contemporary and detailed for LLMs' internal knowledge.

[1] https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022?select=Combined_Flights_2022.csv

[2] https://www.kaggle.com/datasets/psycon/daily-coffee-price

Table A.1: A comparison of methods that leverage LLMs for Tool-use.

Methods	Tool Numbers	Tool Categories	# Tool/Task	Reasoning	Instruction Type	Task
Single-Tool Methods						
CoT [16]	1	-	1	Generation	Prompting	QA
Lila [165]	1	math/code	1	Generation	Prompting	MathQA
Program-of-Thought [76]	1	code	1	Generation	Prompting	TabQA
Code4Struct [62]	1	code	1	Generation	Prompting	Event Extraction
PAL [59]	1	code	1	Generation	Prompting	MathQA
MathPrompt [77]	1	code	1	Generation	Prompting	MathQA
ToolFormer [12]	5	Basic	1	Generation	PR & FT	QA
GraphToolFormer [117]	5	Graph	1	Human Info	PR & FT	Graph
Talm [116]	-	Basic	1	Generation	PR & FT	QA
Multi-Tool Methods						
WebGPT [166]	10	Web Operation	>1	Feedback	Fine-tuning	QA
HuggingGPT [22]	>10	Vision	>1	Human Info	Prompting	VQA
Chameleon [13]	>10	code, nlp, cv	>1	Human Info	Prompting	ScienceQA, TabQA
GeneGPT [89]	38	NCBI APIs	>1	Generation	Prompting	Gene Tasks
ART [78]	8	code/math/retriever	>1	Human Feedback	Prompting	BigBench
ReAct [17]	3	retriever	>1	Feedback	PR & FT	QA, AlfWorld, WebShop
MM-ReAct [87]	>10	vision	>1	Feedback	Prompting	CV tasks
Visual ChatGPT [88]	>10	vision	>1	Feedback	Prompting	CV tasks

- **Yelp Business Data**[3] is a subset of Yelp's business data across 8 metropolitan areas in the USA and Canada, where the information is too detailed for LLMs' internal knowledge.

- **Airbnb Open Data**[4] is a subset of Airbnb activities in New York, where the information is too detailed for LLMs' internal knowledge.

- **DBLP Citation Network (V14)**[5] constructs the graph based on the records after 2020. The author-author and paper-paper relations are formulated as two separate graphs.

- **GSM8k**[6] is a dataset of 8.5K high-quality linguistically diverse grade school math word problems. We sample the questions from the error cases made by ChatGPT on the original dataset to make sure that the questions cannot be easily handled with its internal knowledge.

[3]https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json
[4]https://www.kaggle.com/datasets/arianazmoudeh/airbnbopendata
[5]https://www.aminer.org/citation
[6]https://github.com/openai/grade-school-math

- **SciREX**[7] is a challenging dataset for document-level information extraction based on a collection of full-length machine-learning scientific papers.

- **Agenda** is our own synthetic dataset to model the real-world personal agenda data. To avoid the privacy issue, we first create names, events, and dates with ChatGPT and then randomly compose them to form 10000 different records. To create a pure-text personal agenda corpus, we feed each of the records into ChatGPT, containing generated agenda for virtual characters. More details can be seen in Appendix subsection A.2.2.

A.2.2 Generation Details of Agenda Dataset

Personal or private data serves as a significant external knowledge source. There exist applications that have been designed with plugins and external tools specifically querying this type of data, such as AI personal assistants on daily agenda. Nevertheless, we recognize that this data often intersects with sensitive areas, and hence, privacy concerns are paramount. To address these issues, we automatically synthesize a personal agenda corpus. This not only ensures that the large language models (LLMs) have not been previously exposed to the data but also eliminates any possibility of them inadvertently memorizing the information within their internal knowledge.

In the synthetically generated personal agenda corpus, each entry follows the pattern: "`NAME` performs `EVENT` at `TIME` on `DATE`", incorporating key elements such as names, events, dates, and time slots. To begin, we employ ChatGPT to virtually generate these elements. More precisely, we create 100 unique names, 10000 distinctive events each associated with corresponding time slots within a day, and span all possible dates from 01/01/2022 through 12/31/2022. Following this, we commence the random assembly of these generated elements to formulate personal agenda entries. For every event-time pair generated, we randomly select from the pool of 100 names and possible dates to construct

[7] https://github.com/allenai/SciREX

each record. This process yields a total of 9,494 unique personal agenda entries. To transform this corpus into an accessible external database for model querying, we transcribe each record into a comprehensible natural language description.

A.3 Key Information of ToolQA

A.3.1 Dataset Documentations

The dataset is provided in *jsonl* format. Each task corresponds to two files: easy and hard (*e.g.*, "flight-easy.jsonl" and "flight-hard.jsonl", *etc.*). Each data point contains the following fields:

- `qid`: the unique identifier for the question-answer pair;
- `question`: the question to query;
- `answer`: the corresponding ground-truth answer to `question`.

A.3.2 Intended Uses

ToolQA is intended for researchers in machine learning and related fields to innovate novel methods for tool-augmented large language models (LLMs). We also aim to help developers to test their plugins on our dataset.

A.3.3 Hosting and Maintenance Plan

ToolQA codebase is hosted and version-tracked via GitHub. It will be permanently available under the link https://github.com/night-chen/ToolQA. The download link of all the datasets can be found in the GitHub repository. We license our work using Apache 2.0[8]. All the datasets will be publicly released through the aforementioned GitHub link.

[8] https://www.apache.org/licenses/LICENSE-2.0

APPENDIX B
ADDITIONAL DETAILS FOR TOOLCHAIN*

B.1 Implementation Details

B.1.1 Hardware Information

All experiments are conducted on CPU: Intel(R) Xeon(R) Platinum 8275CL CPU @ 3.00GHz and GPU: NVIDIA A100 SXM4 80 GB using Python 3.8.13.

B.1.2 Parameter Configuration

We chose the GPT-3.5-turbo engine for ChatGPT and the GPT-4 engine for GPT-4 when structuring the LLM-based agent. The maximum length for generated solutions is set to 512, and the temperature is set to 1 for flexibility in self-consistency frequency function $g_{t,1}$ (Section subsection 3.4.2). For LLaMA-2 experiments, the maximum length for generated solutions is set as 256 and the temperature is set to 1. We use 8 NVIDIA A100 SXM4 80 GB GPUs and FastChat [169] to tune the LLaMA-2 7B and 13B models. For ToolChain*, we set the weights of geometric means between heuristic and non-heuristic parts in cumulative and future costs as $\alpha = \beta = 0.5$ by default. The number of potential actions in self-consistency frequency is set as $k = 10$ and the maximum step limit is set as $T = 20$ by default. Our code repository is presently undergoing an internal review within the company for public release. Upon receiving approval, we will make both the code and data available on GitHub.

B.1.3 Task Setup for ToolBench

We define the nodes in the decision tree as function API calls. From the root node, which corresponds to the input task description, we represent the complex process of identifying a successful solution as navigating a valid path in the tree search space. To expand from the current node, we prompt LLMs with API definitions, demonstration examples, the given query, and action history, thereby generating multiple i.i.d. next steps. In order to precisely evaluate plan quality and gauge the proximity of the current action to the goal, we employ task-specific heuristic functions (Section subsection 3.4.2) for both cumulative and projected scores. To ensure a fair comparison, all prompts dispatched to the baselines and ToolChain* follow the configuration set by ToolBench [102]. We adopt the success rate as the evaluation metric for Home Search, Trip Booking, and Google Sheets. For Virtual Home, we report the proportion of scripts achieving the correct end state or successfully completing the task.

B.1.4 Task Setup for GSM8K

GSM8K [98] serves as a dataset for high school-level mathematical reasoning. Numerical reasoning tasks within this dataset typically comprise a descriptive component followed by a culminating question. Answering this question requires multi-step mathematical calculations based on the context of the description. Notably, the complexity directly relates to the number of mathematical reasoning steps required for a solution. We conduct experiments on the entire set of GSM8K and also a subset of hard questions from GSM8K collected in ToolQA [10]. In adapting ToolChain*, we represent reasoning steps in a solution as nodes in the decision tree. With the math question description as the root node, complex reasoning across numerous intermediate steps is translated into navigating the decision space for a valid path. Given the absence of seed data for long-term memory, we simplify ToolChain* to leverage self-consistency frequency and LLM imagination as the cumulative score $g(n)$ and future score $h(n)$, respectively.

Table B.1: Task overview of (1) Home Search, (2) Trip Booking, (3) Google Sheets, (4) Virtual Home, and (5) GSM8K datasets. We provide examples of task descriptions with output actions, and report the number of APIs (# APIs) and questions (# Ques) within each dataset.

Datasets	Task Descriptions	Output Actions	# APIs	# Ques	# APIs/Sol
Home Search	I want to buy a townhouse, mobile or co-op in Pittsburgh with 4 rooms. My budget is $1385000.	API.set_location("Pittsburgh") API.set_buy_or_rent("buy")···	15	100	7.5
Trip Booking	Could you help me find train tickets for 3 children and 5 adults from Des Moines to Cape Coral on July 07, 2022?	API.select_booking_type("trip tickets") API.select_transportation("train")···	20	120	13.4
Google Sheets	Update chicken's price by 2. [A table is flattened in the context.]	df = get_as_dataframe(worksheet) df.loc[df['Product'] == 'chicken', 'Price'] += 2···	108	70	6.1
Virtual Home	Read book	Agent.Find(novel) Agent.TurnTo(novel)···	40	100	13.4
GSM8K (ToolQA)	Elsa has 5 apples. Anna has 2 more apples than Elsa. How many apples do they have together?	1. Anna has 2 more apples than Elsa. 2. So Anna has 2 + 5 = 7 apples.···	-	1319 (100)	-

B.2 Experimental Details

B.2.1 Dataset Details

We evaluate ToolChain* on four tool-use environments in ToolBench [102] and one reasoning task in GSM8K [98]. Table B.1 provides examples of task descriptions and output actions and reports the number of APIs and questions incorporated in the environment.

• **Home Search** simulates the process of locating homes in a specific area based on certain criteria. Agents are required to leverage 15 functions (*e.g.*, `set_location`, `search`, *etc.*), to aid users in completing the search.

• **Trip Booking** is a task that parallels the Home Search but incorporates more advanced dependencies between function calls. This task simulates the process of submitting search requests for transportation tickets, hotel rooms, or a combination of both. Specifically, it is guided by specific searching conditions or parameters like locations, dates, and the number of tickets. The API includes a total of 20 functions related to trip booking scenarios.

• **Google Sheets** involves manipulating actual worksheets from Google Sheets [1] via the gspread library [2], including common operations such as updating cell values, sorting, *etc.*.

[1]https://www.google.com/sheets/about/

[2]https://docs.gspread.org/en/latest/

• **Virtual Home** derives from the setting of the VirtualHome simulator [170]. It requires the LLM agents to generate sequences of actions with multiple function calls for completing household activities. It consists of 40 functions (*e.g.*, `Sleep()`, `Push(object)`, and `PourInto(object1, object2)`), each corresponding to a specific action exemplified in the simulator. Every function can take up to two arguments, representing valid objects within the household settings.

• **GSM8K** [98] is a dataset of high-quality linguistically diverse grade school math word problems. To evaluate the model performance on the GSM8K dataset, we evaluate the accuracy at which models are able to obtain the final correct answer. In addition, we also report model performance on a subset of hard questions in GSM8K, curated by **ToolQA** [10]. The hard questions are sampled from the error cases made by ChatGPT on the original GSM8K dataset.

B.2.2 Baseline Details

For tool-use tasks in ToolBench, we compare ToolChain* with GPT [4], ReAct [17], AdaPlanner [18], Tree-of-Thoughts [19], and MCTS [27]. For math reasoning tasks in GSM8K, we compare ToolChain* with the state-of-the-art reasoning approaches, including GPT [4], Chain-of-Thought [16], Self-Consistency [118], ReAct [17], Tree-of-Thoughts [19], and MCTS [27].

We provide details of each baseline as follows:

• **GPT** [4] is a standard input-output prompt with four in-context examples. We directly send the four-shot examples with the task description together to GPT-3.5-turbo and GPT-4. GPT series serve as open-loop systems with closed-source LLMs as backbones.

• **Chain-of-Thought** [16] prompting is an open-loop mechanism to enhance reasoning in language models. Chain-of-thought prompting breaks down multi-step problems, allocating additional computational resources for extensive reasoning stages step by step.

• **Self-Consistency** [118] is an open-loop decoding strategy to replace the conventional greedy decoding typically employed in chain-of-thought prompting. It initially samples an array of diverse reasoning paths rather than solely relying on the most probable one. Subsequently, by marginalizing over these sampled reasoning trajectories, it pinpoints the most coherent answer.

• **ReAct** [17] integrates reasoning with tool use by prompting LLMs to generate interleaved verbal reasoning traces and tool calls. ReAct is a closed-loop system for LLM-based agents in tool usage.

• **AdaPlanner** [18] is a closed-loop system that enables the LLM agent to adaptively refine its self-conceived plan based on feedback from the environment. This refinement process leverages both in-plan and out-of-plan strategies.

• **Tree-of-Thoughts** [19] is a tree search-based LLM reasoning framework that built upon chain-of-thought prompting. It facilitates exploration across cohesive textual segments, which act as intermediary steps in the problem-solving process. Unlike linear reasoning pathways in open- or closed-loop systems, ToT enables language models to engage in decision-making by examining various reasoning trajectories. With the ability to self-assess choices for the subsequent steps, ToT provides the flexibility to anticipate future steps or revisit previous ones to make holistic decisions.

• **MCTS** [27] is a tree search-based LLM reasoning framework, repurposing LLM and functioning both as a world model and a reasoning agent. It integrates a principled planning algorithm, specifically based on MCTS, facilitating strategic exploration in the expansive reasoning space. LLM-based agent systematically constructs a reasoning tree, guided by its inherent world model and task-specific rewards.

B.2.3 Additional Analysis on ToolBench

We observe that closed-loop methods typically perform better than tree search-based methods on the Home Search and Trip Booking datasets. Conversely, for the Google Sheets and Virtual Home datasets, tree search-based methods perform better. This discrepancy can be attributed to the nature of feedback provided by the datasets. For instance, the Home Search and Trip Booking datasets offer precise environmental feedback regarding plan errors (*e.g.*, "*Task execution error: 'HomeSearchAPI' object has no attribute 'set_min_baths'*"), enabling closed-loop systems to effectively modify their plans. In contrast, the Google Sheets and Virtual Home datasets, with their extensive API function calls (108 and 40, respectively), present a notably larger action space than Home Search and Trip Booking (15 and 20, respectively).

B.2.4 Case Studies

We include additional case studies with visualizations comparing the Chain-of-Thought and ToolChain* on the GSM8K dataset. As shown in Figures Figure B.1 and Figure B.2, we compare the exploration space (upper part) and planning ordering (lower part) of both methods on the same problem. Given the input query, ToolChain* explores a wide range of potential nodes (blue) in the decision tree, while Chain-of-Thought only explores one direction (red). Moreover, from a step-by-step visualization of the planning process in Figure B.1, we notice that chain-of-thought falls into a faulty loop or a dead end with a previous incorrect action. However, ToolChain* can gradually discard the faulty path by increasing the cost after the incorrect action. This allows our method to re-evaluate and adjust prior actions, facilitating a shift away from erroneous paths to explore alternative plans. More importantly, it mitigates the error propagation along the action plan, which usually occurs in linear or unidirectional solutions (*i.e.*, open- and closed-loop systems).

B.2.5 Efficiency Details

In terms of additional efficiency analysis in tool use, we evaluate the running time of ToolChain* against all the baselines based on GPT-4, as shown in Figure B.3a. Consistent with our previous results, ToolChain* is faster than the most efficient tree search-based method, Tree-of-Thoughts (BFS). For the math reasoning task, we conduct efficiency analysis with a number of valid actions in Figure B.3b. We calculate the average number of valid actions in GSM8K for math reasoning efficiency analysis. Similarly, ToolChain* shows close efficiency to closed-loop systems without a tree structure and outperforms other tree-search algorithms in terms of efficiency.

From the scaling-up analysis in Figure B.4, we can explicitly identify a crucial trade-off between effectiveness and efficiency in the direct application of tree search-based reasoning methods to complex tool use scenarios, including ToT-DFS [19], ToT-BFS [19], and MCTS [27]. Compared with ToolChain*, which quickly converges on with time efficiency and high-quality solution, the rest tree-based search methods not only suffer from relatively low success rate (Figure B.4a), but also struggle with the long running time (Figure B.4b). This could be further verified by experimental results in Table 3.1 and efficiency metrics in Figure 3.4.

B.2.6 Ablation Studies Details

We conduct a detailed analysis of the ablation studies for both the cumulative and future cost functions in the following, as presented in Table 3.2. First, experimental results indicate that each element within both the cumulative and future cost functions enhances the performance of ToolChain*. This verifies the efficacy of our proposed cost functions in guiding the search through the decision tree. Second, dropping either the heuristic components ($g_{1,t}$, $h_{1,t}$) or the non-heuristic ($g_{2,t}$, $h_{2,t}$) components from the cumulative or future cost functions results in a decline in performance. Specifically, when the non-heuristic components are absent, there is an average drop in the success rate of 6.8%; whereas, without the heuristic components, the decrease is 3.3%. This suggests that in most instances, non-heuristic components have a more significant impact. This is potentially because heuristic components, relying on long-term memory, are limited in covering all test cases and may struggle to precisely estimate the cost of new actions or tasks. In scenarios where environments offer high-quality seed data for long-term memory (*e.g.*, Virtual

Table B.2: Additional results on ScienceQA based on GPT-3.5-turbo.

Models	Accuracy
CoT (GPT-3.5-turbo)	78.3
Chameleon (GPT-3.5-turbo)	79.9
ToolChain* (GPT-3.5-turbo)	**80.7**

Home), the heuristic function plays a more important role. Third, eliminating either the entire cumulative or future cost results in a marked decline in the success rate. Relying exclusively on the future cost induces a sharp performance drop of 28.2%, deteriorating ToolChain* to a greedy strategy that favors the shortest solution plans with the least number of actions. Conversely, if the search is guided only by the cumulative cost, ToolChain* essentially mirrors the behavior of the BFS algorithm, yielding similar performance outcomes.

B.3 Additional Dataset: Science QA

We evaluate ToolChain* in Science Question Answering (ScienceQA) [171] dataset on general reasoning tasks. ScienceQA serves as a diverse benchmark for multi-modal question answering across an array of scientific topics and contexts. Answering its questions requires a variety of tools and capabilities, including image captioning, text detection, knowledge retrieval, online resource searches, and multi-clue visual reasoning. Since ToolChain* can function as a plug-and-play planner, we replace the original naive LLM-based planner in Chameleon with ToolChain* for evaluation purposes. For our experiments, we select the Chain-of-Thought [16] and the original Chameleon [13] as reference baselines. When built upon GPT-3.5-turbo as the foundational LLM, ToolChain* realizes an accuracy improvement of 0.8% over Chameleon. ToolChain* benefits from considering multiple potential paths for execution. However, the performance gain is limited on the dataset, primarily because the logic behind calling different tools to solve the problem from the dataset is quite rigid.

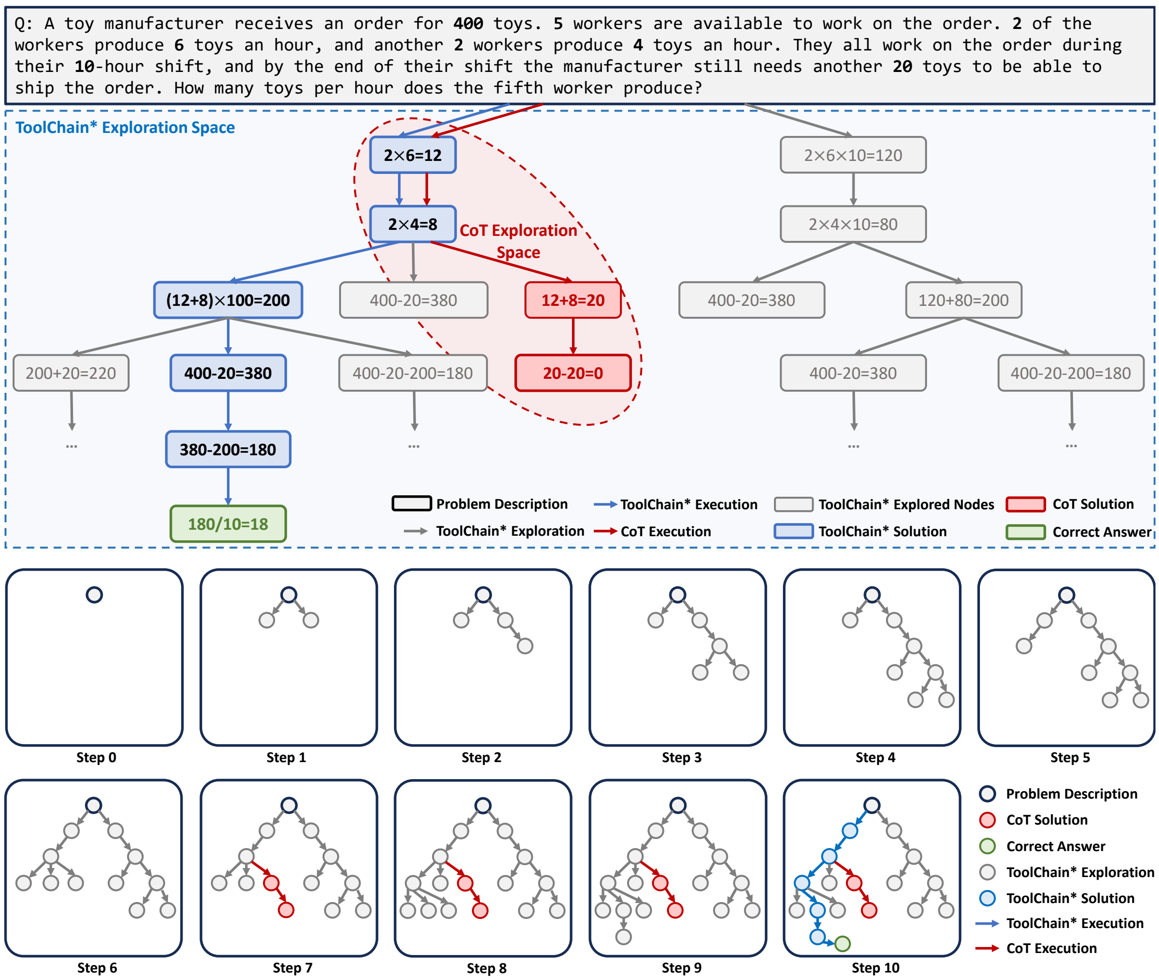


Figure B.1: Case study comparing Chain-of-Thought and ToolChain* on GSM8K dataset. We offer comparison in exploration space (upper part) and planning ordering (lower part). Given the input query, ToolChain* explores a wide range of potential nodes (blue shadowed area) in the decision tree, while Chain-of-Thought only explores one direction (red shadowed area). During the planning process, a chain of thought can fall into a faulty loop or a dead end with a previous incorrect action. However, ToolChain* can gradually abandon the faulty path by increasing the cost after the incorrect action. This enables it to revise previous actions and jump out of the faulty path to try another plan.

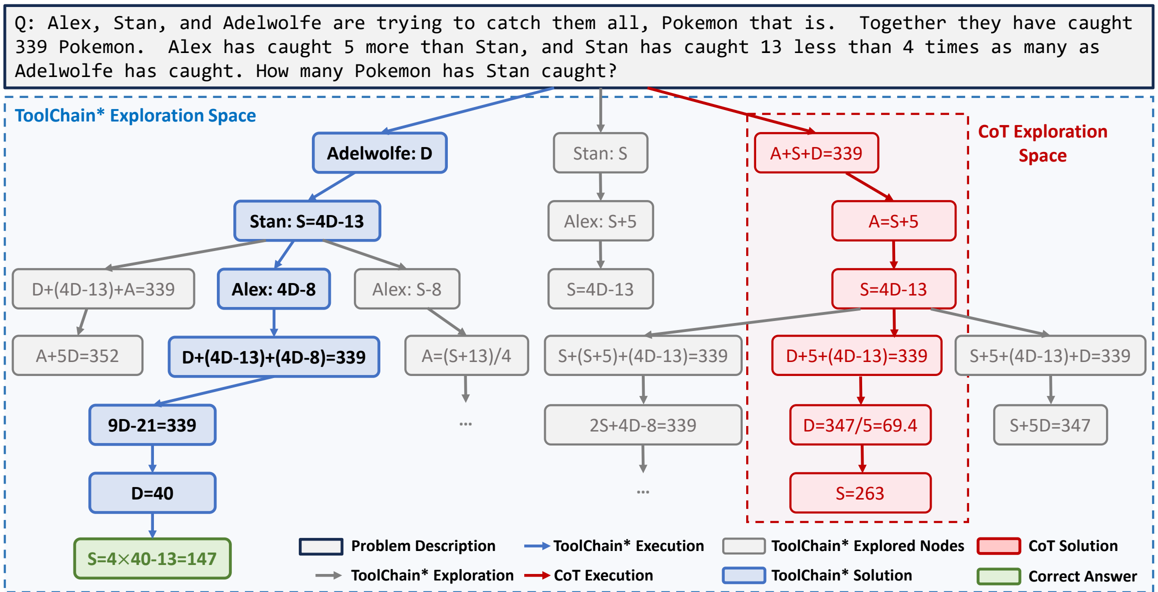
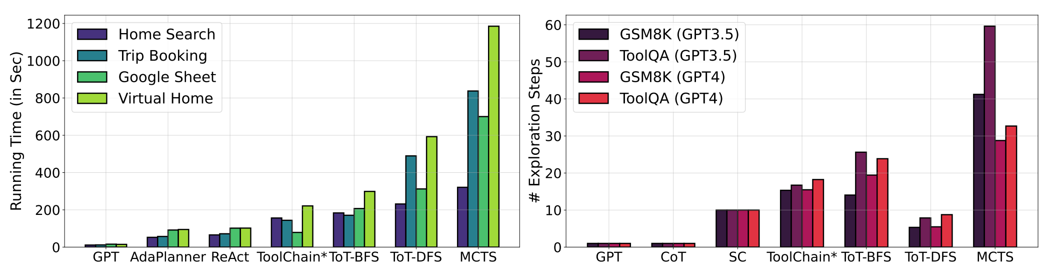
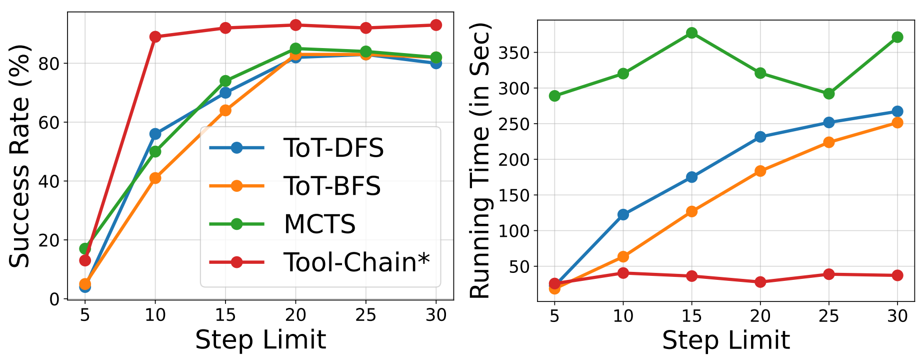


Figure B.2: Case study comparison between Chain-of-Thought and ToolChain* on the GSM8K dataset. The exploration space is illustrated in the upper section, and the planning order is depicted in the lower section. For a given input query, ToolChain* explores an expansive set of potential nodes (blue) with correct answers, whereas Chain-of-Thought primarily navigates in a singular direction (red).



(a) ToolBench (b) GSM8K

Figure B.3: Additional time efficiency evaluation on ToolBench and GSM8K. (a) We report additional average running time in seconds over all instances in the dataset using GPT-4 backbone. (b) We calculate the average number of valid actions in GSM8K for math reasoning efficiency analysis. In both cases, ToolChain* shows close efficiency to closed-loop systems without a tree structure and outperforms other tree-search algorithms in terms of efficiency.



(a) Performance (b) Running Time

Figure B.4: Scaling analysis of ToT-DFS [19], ToT-BFS [19], MCTS [27], and ToolChain*. (a) Performance and (b) running time on Home Search when scaling up step limitations T.

APPENDIX C
ADDITIONAL DETAILS FOR HYDRA

C.1 Dataset and Task Details

We employ the Language Model Personalization (LaMP) benchmark [158], an open-source benchmark specifically designed to train and evaluate the capability of language models in generating personalized content. LaMP encompasses a diverse set of tasks (with LaMP-2 comprising two tasks, LaMP-2N and LaMP-2M), covering both personalized text classification and generation tasks. We present the statistics of the datasets in Table C.1 to provide a clear depiction of the dataset structure. We further offer detailed descriptions of each task as follows.

- **Task 1: Personalized News Categorization (LaMP-2N)** is a categorical text classification task that involves classifying news articles into one of 15 categories based on a journalist. Given an article written by a user, the model predicts its category using the user's history of articles and their corresponding categories.
- **Task 2: Personalized Movie Tagging (LaMP-2M)** is an ordinal text classification task that involves predicting one of 15 tags for a movie based on a user's tagging history. LaMP-2M evaluates the capability of a language model to assign tags to a movie description using the user's historical movie-tag pairs.
- **Task 3: Personalized Product Rating (LaMP-3)** is a text classification task that involves predicting product ratings as a five-class text classification problem. The objective is to predict an integer rating from one to five for a given review based on users' historical review and rating pairs. This task assesses the capability of language models to capture user-specific rating preferences.

- **Task 4: Personalized News Headline Generation (LaMP-4)** is a text generation task that involves generating personalized news headlines for given articles based on the authors' historical article-title pairs. This task evaluates the language model's ability to replicate the authors' stylistic nuances in headline generation.

- **Task 5: Personalized Scholarly Title Generation (LaMP-5)** is a text generation task that involves generating titles for input articles based on the author's historical article and title pairs. This task extends the exploration of personalized text generation into scholarly domains, explicitly focusing on generating titles for research articles.

LaMP-6 has been excluded since the dataset is not publicly available. Furthermore, two additional tasks (LaMP-1 and LaMP-7) have been excluded from our empirical evaluation due to the inconsistent formats between user history and query.

- **Personalized Citation Identification (LaMP-1)** is a binary text classification task that involves predicting citation choices. It requires determining which of two candidate papers will be cited in a new paper based on a user's historical publication data. This process utilizes user profiles that include the titles and abstracts of their previous works. This task aims to evaluate a language model's capability to identify the citation preferences specific to each user.

- **Personalized Tweet Paraphrasing (LaMP-7)** is a text generation task that involves paraphrasing an input tweet into a personalized one, using the user's historical tweet data for stylistic guidance. This task evaluates a model's proficiency in reproducing the unique stylistic patterns in authors' social media posts.

C.2 Baseline Details

We compare our proposed Hydra with both non-personalized and personalized baselines. The non-personalized baselines consist of zero-shot `gpt-3.5-turbo` [43] and **ICL-**

Table C.1: Dataset statistics of five different personalization tasks in the LaMP benchmark [158].

Task	Type	# Train	# Validation	# Test	Input Length	Output Length	# Profiles	# Classes
LaMP-2N	Classification	5914	1052	1274	65.40 ± 12.29	-	306.42 ± 286.65	15
LaMP-2M	Classification	5073	1410	1557	92.39 ± 21.95	-	86.76 ± 189.52	15
LaMP-3	Classification	20000	2500	2500	145.14 ± 157.96	-	188.10 ± 129.42	5
LaMP-4	Generation	12527	1925	2376	30.53 ± 12.67	9.78 ± 3.10	287.16 ± 360.62	-
LaMP-5	Generation	9682	2500	2500	152.81 ± 86.60	9.26 ± 3.13	89.61 ± 53.87	-

Random (In-Context Learning with Random items from user behavior history). On the other hand, the personalized baselines include **RAG** (Retrieval-Augmented Personalization) and **PAG** (Profile-Augmented Personalization). k denotes the number of retrieved items from the user behavior history (or profiles). For all baselines, we follow the same prompt template in the LaMP benchmark [158] and employ `gpt-3.5-turbo(1106)` as the backbone black-box LLM. We utilize BM25 [73] as the default retriever for all experiments.

- **gpt-3.5-turbo** [43] processes only the user's query (*i.e.*, the input question) without leveraging the user's profile data.
- **ICL-Random** augments the user's query with random k items from behavior history, enabling in-context learning to improve the capability of the original backbone LLM.
- **RAG** [158] augments the user's query with top k retrieved items from the corresponding user's behavior history, following the retrieval-augmented personalization method in LaMP.
- **PAG** [162] augments the user's query with the user's profile summary generated by LLMs (`gpt-3.5-turbo`) which concludes the user's behavior patterns. We further combine PAG with k retrieved items from the corresponding user's behavior history.

C.3 Implementation Details

Hardware and Software. We conduct all black-box LLM personalization experiments on CPU: AMD(R) EPYC(R) 7702 64-Core Processor @ 1.50GHz and GPU: NVIDIA A100-SXM4-80GB using Python 3.10.13.

Hyperparameter Configurations. We set the maximum sequence length for a generated solution as 512 tokens and the temperature as 1.0 for flexibility in the generations of black-box LLMs, which serve as potential solution candidates. For other baselines, we maintain the temperature to 0 to avoid potential instability in performance. During the black-box LLM adaption stage, we set $b = 8$ for the generation of intermediate candidates using Hydra-Adapter. During the training stage, we set the learning rate to $2e - 5$, the global batch size to 64, and the number of training epochs to 2 as default hyperparameter settings for all experiments. In addition, we employ `AdamW` [172] as the optimizer with a weight decay of 0.01.

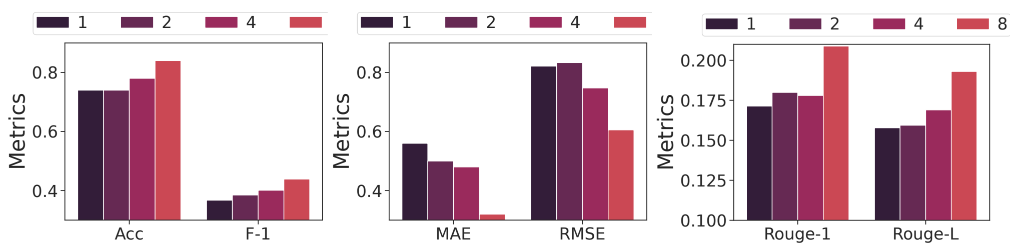
C.4 Additional Experimental Results and Analysis

C.4.1 Additional Analysis of Baselines in the Main Experimental Results

We conduct further analysis of the main experimental results of dominant approaches for black-box LLMs, specifically prompt-based methods, including RAG and PAG. Compared to the zero-shot setting in `gpt-3.5-turbo` [43], even a random selection of historical records from user behavior or profiles enhances the model performance in most tasks, suggesting that personalization contributes to improved performance with black-box LLMs. However, we also observe several instances of less optimal performance, particularly in LaMP-3. We hypothesize that this inconsistency may arise from the introduction of noise and irrelevant behavior patterns by the retrieved items, potentially complicating the understanding of user behavior patterns. Additionally, both RAG [158] and PAG [162] demonstrate marginal improvements over random selection, which may benefit from augmenting relevant behavior or profile information. The experimental results also consistently demonstrate that an increase in the number of retrieved items is positively correlated with improved performance, indicating the effectiveness of the retrieval-augmented framework in black-box LLM personalization. However, in comparison to Hydra, RAG- and PAG-based methods still yield suboptimal results. This is primarily attributed to the presence of potentially noisy and irrelevant retrieved behavior patterns, as well as the absence of shared general knowledge across the entire user group.

C.4.2 Additional Experimental Details for Sacle-up Analysis

Figure C.1 reports additional experimental results of Hydra in accuracy and F-1 with different numbers of selected history per user, indicating the robustness of Hydra with additional user history.



(a) LaMP-2N (b) LaMP-3 (c) LaMP-4

Figure C.1: Effect of users with different numbers of behavior history in Hydra.

C.4.3 Additional Experimental Details for User Behavior Shift

Following the setup of existing work [148], we utilize an encoder-only language model called DeBERTa-v3 [173] to encode the user's historical behaviors and the query into high-dimensional representations. We then compute the cosine similarity between the query and all historical items to evaluate their relevance. We select the top 20 items with the lowest relevance scores as irrelevant user history. As depicted in Figure 4.5, our proposed Hydra still outperforms the state-of-the-art baselines. This performance improvement may stem from the personalized reranker, which reranks the candidates based on their usefulness rather than just relevance. Additionally, the personalized adapter also adapts the model's generations to align with user preferences. In addition, prompt-based methods tend to underperform when there is a distribution shift between the few-shot demonstrations and user queries. This issue can become even more severe under personalized scenarios, where there might be a mismatch between the selected user history and the user query.

C.4.4 Effect of Retrievers

We investigate the effect of different retrievers in Table C.2. Under the black-box LLM personalization scenario, dense retrievers that capture semantic relevance (*e.g.*, Contriever [174]) achieve similar performances to sparse retrievers (*e.g.*, BM25 [73] in Table 4.1), which is aligned with observations in the existing works [159].

Table C.2: Additional experimental results with Contriever [174] as the retriever (k=4).

Dataset (→)	**LaMP-2N**		**LaMP-3**		**LaMP-4**		
Method (↓)	Acc. ↑	F-1 ↑	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑
ICL-Random	0.660	0.288	0.560	0.917	0.163	0.148	0.159
RAG	0.680	0.293	0.540	0.883	0.179	0.152	2.082
Hydra	**0.800**	**0.414**	**0.380**	**0.730**	**0.189**	**0.173**	**2.501**

C.4.5 Effect of Black-Box LLM Adapters

We evaluate the effectiveness of different black-box LLM adapters in Table C.3. Due to the inaccessibility to model internal parameters, BBox-Adapter is the only black-box LLM adapter available, apart from our proposed Hydra-Adapter. Based on Table C.3, it can be observed that Hydra-Adapter outperforms BBox-Adapter when applied under the same personalization settings (as discussed in Section subsection 4.3.4). This result demonstrates the effectiveness of our adapter module design, which captures both global and user-specific knowledge to further adapt to personalized model outputs.

Table C.3: Effect of different black-box LLM adapters (k=4).

Dataset (→)	**LaMP-3**		**LaMP-4**			**LaMP-5**		
Method (↓)	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑	R-1 ↑	R-L ↑	BLEU ↑
Personalized BBox-Adapter [138]	0.540	0.860	**0.151**	**0.135**	**1.285**	0.394	0.337	5.478
Hydra-Adapter	**0.420**	**0.762**	0.145	0.118	1.137	**0.409**	**0.355**	**5.816**

Table C.4: Experiments on two additional personalization datasets, MovieLens-1M and Recipe, focusing on predicting users' personal ratings for movies or recipes based on their historical rating patterns.

Dataset (→)	**MovieLens-1M**		**Recipe**	
Method (↓)	MAE ↓	RMSE ↓	MAE ↓	RMSE ↓
`gpt-3.5-turbo`	0.780	1.208	0.880	1.149
ICL-Random (k=1)	0.880	1.2649	0.960	1.233
ICL-Random (k=2)	0.800	1.095	0.980	1.192
ICL-Random (k=4)	0.820	1.225	0.760	1.114
RAG (k=1)	0.800	1.166	0.980	1.192
RAG (k=2)	0.700	1.030	0.880	1.114
RAG (k=4)	<u>0.640</u>	<u>1.000</u>	0.820	<u>1.049</u>
PAG (k=0)	0.820	1.147	0.860	1.128
PAG (k=1)	0.760	1.010	<u>0.800</u>	1.063
Hydra	**0.600**	**0.908**	**0.720**	**0.938**

C.4.6 Additional Datasets

To further validate the effectiveness of HYDRA, we conduct extensive experiments on two additional widely used personalization datasets [175], MovieLens-1M and Recipe, focusing on predicting users' personal ratings for movies or recipes based on their historical rating patterns. The experimental results (Table C.4) demonstrate that HYDRA outperforms the best-performing baselines by 8.0% on Movielens and 10.3% on Recipe, respectively.

C.4.7 Efficiency Analysis

Assume that we have N_{train} users in the training data and N_{test} users in the test set. We adopt the transformer architecture, specifically the Longformer-base, as the reranker and the adapter base models. Consequently, the time complexity of all stages should be proportional to that of the attention mechanism in transformers, $O(dL^2)$, where L indicates the

Table C.5: Time complexity analysis with running time summary on the LaMP benchmark.

Method	Mode	Time Complexity	LaMP-2N	LaMP-2M	LaMP-3	LaMP-4	LaMP-5
Hydra-Reranker	Training	$O(N_{\text{train}}(M^2+1)TL^2d)$	31m10s	41m51s	50m37s	1h1m31s	1h8m16s
Hydra-Reranker	Fit New User	$O(N_{\text{test}}(M^2+1)TL^2d)$	18m8s	21m17s	25m36s	33m52s	31m25s
Hydra-Reranker	Inference	$O(N_{\text{test}}L^2d)$	3m4s	3m1s	3m7s	4m38s	5m14s
Hydra-Adapter	Training	$O(N_{\text{train}}k\bar{H}TL^2d)$	1h10m17s	2h2m16s	2h1m59s	3h56m47s	3h19m42s
Hydra-Adapter	Fit New User	$O(N_{\text{test}}k\bar{H}TL^2d)$	28m15s	1h7m27s	1h19s	2h23m10s	1h59m2s
Hydra-Adapter	Inference	$O(N_{\text{test}}kL^2d)$	4m16s	4m17s	4m17s	5m53s	5m59s

sequence length and d indicates the hidden dimension. Additionally, the training process will go through the transformer for T epochs, while inference only requires one. For each user in the HYDRA-Reranker training data, we augment M random historical records from the user's corresponding profile. The retriever then retrieves the top-M ($M = 20$ by default) relevant historical records to form training samples. Thus, for each user, we collect $M^2 + 1$ training samples. For each user in the HYPER-Adapter training data, we consider all the user historical records. For each record, we leverage model randomness to generate k ($k = 8$ by default) samples for the adapter to select. Consequently, we can have $\bar{H}k$ training samples per user, where $\bar{H}$ denotes the average number of histories per user.

C.4.8 Error Analysis

We identify three common error types observed in the reranker that account for most general errors: (1) **Saturated Scores**: These errors occur when the ranker outputs overconfident scores (near 1.0) regardless of different inputs. Consequently, it becomes challenging to distinguish the best output as the differences among the scores are minimal. (2) **Incorrect Scores**: These errors arise when the ranker assigns higher scores to less relevant terms. As a result, the likelihood of selecting the most suitable samples decreases, leading to suboptimal generation. (3) **Divergent Scores**: These errors occur when the ranker exaggerates the differences among the inputs. This may be attributed to the inputs being hard samples, where none of them are closer to the target compared to each other, making it difficult for the reranker to discern the most appropriate option.

REFERENCES

[1] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI Blog*, 2018.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, 2019.

[3] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] OpenAI, "Gpt-4 technical report," *arXiv*, p. 2303.08774v3, 2023.

[5] A. Chowdhery *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[6] R. Anil *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[7] J. Huang and K. C.-C. Chang, *Towards reasoning in large language models: A survey*, 2023. arXiv: 2212.10403 `[cs.CL]`.

[8] S. Hao, T. Liu, Z. Wang, and Z. Hu, *Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings*, 2023. arXiv: 2305.11554 `[cs.CL]`.

[9] B. Zhang *et al.*, *Evaluating and improving tool-augmented computation-intensive math reasoning*, 2023. arXiv: 2306.02408 `[cs.CL]`.

[10] Y. Zhuang*, Y. Yu*, K. Wang*, H. Sun, and C. Zhang, "ToolQA: A dataset for llm question answering with external tools," in *Proceedings of NeurIPS 2023*, vol. 36, Curran Associates, Inc., 2023, pp. 50 117–50 143.

[11] R. Yang *et al.*, *Gpt4tools: Teaching large language model to use tools via self-instruction*, 2023. arXiv: 2305.18752 `[cs.CV]`.

[12] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," *arXiv preprint arXiv:2302.04761*, 2023.

[13] P. Lu *et al.*, "Chameleon: Plug-and-play compositional reasoning with large language models," *arXiv preprint arXiv:2304.09842*, 2023.

[14] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," *arXiv preprint arXiv:2305.15334*, 2023.

[15] Y. Qin *et al.*, "Toolllm: Facilitating large language models to master 16000+ real-world apis," *arXiv preprint arXiv:2307.16789*, 2023.

[16] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *arXiv*, p. 2201.11903v6, 2022.

[17] S. Yao *et al.*, "React: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations*, 2023.

[18] H. Sun*, Y. Zhuang*, L. Kong, B. Dai, and C. Zhang, "AdaPlanner: Adaptive planning from feedback with language models," in *Proceedings of NeurIPS 2023*, vol. 36, Curran Associates, Inc., 2023, pp. 58 202–58 245.

[19] S. Yao *et al.*, *Tree of thoughts: Deliberate problem solving with large language models*, 2023. arXiv: 2305.10601 `[cs.CL]`.

[20] D. Zhou *et al.*, "Least-to-most prompting enables complex reasoning in large language models," *arXiv*, p. 2205.10625v2, 2022.

[21] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *arXiv*, p. 2201.07207v2, 2022.

[22] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface," *arXiv preprint arXiv:2303.17580*, 2023.

[23] N. Shinn, B. Labash, and A. Gopinath, "Reflexion: An autonomous agent with dynamic memory and self-reflection," *arXiv preprint arXiv:2303.11366*, 2023.

[24] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents," *arXiv*, p. 2302.01560v1, 2023.

[25] H. Sun, Y. Zhuang, L. Kong, B. Dai, and C. Zhang, "Adaplanner: Adaptive planning from feedback with language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[26] W. Huang *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv*, p. 2207.05608v1, 2022.

[27] S. Hao *et al.*, "Reasoning with language model is planning with world model," *arXiv preprint arXiv:2305.14992*, 2023.

[28] H. W. Chung *et al.*, “Scaling instruction-finetuned language models,” *arXiv preprint arXiv:2210.11416*, 2022.

[29] J. Wei *et al.*, “Finetuned language models are zero-shot learners,” in *International Conference on Learning Representations*, 2021.

[30] Y. Wang *et al.*, “Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5085–5109.

[31] Z. Ke, Y. Shao, H. Lin, T. Konishi, G. Kim, and B. Liu, “Continual pre-training of language models,” in *The Eleventh International Conference on Learning Representations*, 2022.

[32] K. Gupta *et al.*, “Continual pre-training of large language models: How to re-warm your model?” In *Workshop on Efficient Systems for Foundation Models@ ICML2023*, 2023.

[33] E. J. Hu *et al.*, “Lora: Low-rank adaptation of large language models,” in *International Conference on Learning Representations*, 2021.

[34] N. Houlsby *et al.*, “Parameter-efficient transfer learning for nlp,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 2790–2799.

[35] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, “Towards a unified view of parameter-efficient transfer learning,” in *International Conference on Learning Representations*, 2021.

[36] X. L. Li and P. Liang, “Prefix-tuning: Optimizing continuous prompts for generation,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 4582–4597.

[37] X. Liu *et al.*, “P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 61–68.

[38] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu, “Black-box tuning for language-model-as-a-service,” in *International Conference on Machine Learning*, PMLR, 2022, pp. 20 841–20 855.

[39] Y. Huang, D. Liu, Z. Zhong, W. Shi, and Y. T. Lee, “k Nn-adapter: Efficient domain adaptation for black-box language models,” *arXiv preprint arXiv:2302.10879*, 2023.

[40] A. Ormazabal, M. Artetxe, and E. Agirre, "CombLM: Adapting black-box language models through small fine-tuned models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore: Association for Computational Linguistics, Dec. 2023, pp. 2961–2974.

[41] A. Liu, X. Han, Y. Wang, Y. Tsvetkov, Y. Choi, and N. A. Smith, *Tuning language models by proxy*, 2024. arXiv: 2401.08565 `[cs.CL]`.

[42] N. Hu, E. Mitchell, C. Manning, and C. Finn, "Meta-learning online adaptation of language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore: Association for Computational Linguistics, Dec. 2023, pp. 4418–4432.

[43] OpenAI, "Introducing chatgpt," *OpenAI Blog*, 2022.

[44] G. Team *et al.*, "Gemini: A family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[45] OpenAI, "Gpt-4 technical report," *arXiv*, 2023.

[46] T. L. Scao *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022.

[47] H. Touvron *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: 2307.09288 `[cs.CL]`.

[48] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[49] Y. Bai *et al.*, "Training a helpful and harmless assistant with reinforcement learning from human feedback," *arXiv preprint arXiv:2204.05862*, 2022.

[50] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022.

[51] L. Weidinger *et al.*, "Ethical and social risks of harm from language models," *arXiv preprint arXiv:2112.04359*, 2021.

[52] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[53] S. Bubeck *et al.*, "Sparks of artificial general intelligence: Early experiments with gpt-4," *arXiv preprint arXiv:2303.12712*, 2023.

[54] D. Hendrycks *et al.*, "Measuring mathematical problem solving with the math dataset," *NeurIPS*, 2021.

[55] A. Madaan and A. Yazdanbakhsh, "Text and patterns: For effective chain of thought, it takes two to tango," *arXiv preprint arXiv:2209.07686*, 2022.

[56] R. Nogueira, Z. Jiang, and J. Lin, "Investigating the limitations of transformers with simple arithmetic tasks," *arXiv preprint arXiv:2102.13019*, 2021.

[57] A. Lewkowycz *et al.*, "Solving quantitative reasoning problems with language models," *arXiv preprint arXiv:2206.14858*, 2022.

[58] J. Qian, H. Wang, Z. Li, S. Li, and X. Yan, "Limitations of language models in arithmetic and symbolic induction," *arXiv preprint arXiv:2208.05051*, 2022.

[59] L. Gao *et al.*, "Pal: Program-aided language models," in *International Conference on Machine Learning*, PMLR, 2023, pp. 10 764–10 799.

[60] W. Shi *et al.*, "Replug: Retrieval-augmented black-box language models," *arXiv preprint arXiv:2301.12652*, 2023.

[61] G. Izacard *et al.*, "Few-shot learning with retrieval augmented language models," *arXiv preprint arXiv:2208.03299*, 2022.

[62] X. Wang, S. Li, and H. Ji, "Code4struct: Code generation for few-shot structured prediction from natural language," *arXiv preprint arXiv:2210.12810*, 2022.

[63] S. Wolfram, "Wolfram—Alpha as the Way to Bring Computational Knowledge Superpowers to ChatGPT," *Stephen Wolfram Writings*, 2023.

[64] B. Peng *et al.*, "Check your facts and try again: Improving large language models with external knowledge and automated feedback," *arXiv preprint arXiv:2302.12813*, 2023.

[65] A. Mallen, A. Asai, V. Zhong, R. Das, H. Hajishirzi, and D. Khashabi, "When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories," *arXiv preprint arXiv:2212.10511*, 2022.

[66] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[67] S. Borgeaud *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*, PMLR, 2022, pp. 2206–2240.

[68] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[69] B. Y. Lin, K. Tan, C. S. Miller, B. Tian, and X. Ren, "Unsupervised cross-task generalization via retrieval augmentation," in *Advances in Neural Information Processing Systems*, 2022.

[70] Y. Zhuang *et al.*, "ReSel: N-ary relation extraction from scientific text and tables by learning to retrieve and select," in *Proceedings of the EMNLP 2022*, Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 730–744.

[71] S. Lu, N. Duan, H. Han, D. Guo, S.-w. Hwang, and A. Svyatkovskiy, "Reacc: A retrieval-augmented code completion framework," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 6227–6240.

[72] R. Xu, Y. Yu, J. C. Ho, and C. Yang, "Weakly-supervised scientific document classification via retrieval-augmented multi-stage training," *arXiv preprint arXiv:2306.07193*, 2023.

[73] S. Robertson, H. Zaragoza, *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[74] V. Karpukhin *et al.*, "Dense passage retrieval for open-domain question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6769–6781.

[75] G. Izacard *et al.*, "Towards unsupervised dense information retrieval with contrastive learning," *arXiv preprint arXiv:2112.09118*, 2021.

[76] W. Chen, X. Ma, X. Wang, and W. W. Cohen, *Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks*, 2022. arXiv: 2211.12588 [cs.CL].

[77] S. Imani, L. Du, and H. Shrivastava, "Mathprompter: Mathematical reasoning using large language models," *arXiv preprint arXiv:2303.05398*, 2023.

[78] B. Paranjape, S. Lundberg, S. Singh, H. Hajishirzi, L. Zettlemoyer, and M. T. Ribeiro, "Art: Automatic multi-step reasoning and tool-use for large language models," *arXiv preprint arXiv:2303.09014*, 2023.

[79] T. Khot *et al.*, "Decomposed prompting: A modular approach for solving complex tasks," in *The Eleventh International Conference on Learning Representations*, 2022.

[80] P. Patel, S. Mishra, M. Parmar, and C. Baral, "Is a question decomposition unit all we need?" In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 4553–4569.

[81] O. Press, M. Zhang, S. Min, L. Schmidt, N. A. Smith, and M. Lewis, *Measuring and narrowing the compositionality gap in language models*, 2023. arXiv: 2210.03350 `[cs.CL]`.

[82] B. Liu *et al.*, *Llm+p: Empowering large language models with optimal planning proficiency*, 2023. arXiv: 2304.11477 `[cs.AI]`.

[83] S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi, "Cross-task generalization via natural language crowdsourcing instructions," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3470–3487.

[84] A. Madaan *et al.*, "Self-refine: Iterative refinement with self-feedback," *arXiv preprint arXiv:2303.17651*, 2023.

[85] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *arXiv preprint arXiv:2304.05128*, 2023.

[86] W. Wang *et al.*, *Augmenting language models with long-term memory*, 2023. arXiv: 2306.07174 `[cs.CL]`.

[87] Z. Yang *et al.*, "Mm-react: Prompting chatgpt for multimodal reasoning and action," *arXiv preprint arXiv:2303.11381*, 2023.

[88] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, and N. Duan, "Visual chatgpt: Talking, drawing and editing with visual foundation models," *arXiv preprint arXiv:2303.04671*, 2023.

[89] Q. Jin, Y. Yang, Q. Chen, and Z. Lu, "Genegpt: Augmenting large language models with domain tools for improved access to biomedical information," *ArXiv*, 2023.

[90] J. Xie, K. Zhang, J. Chen, R. Lou, and Y. Su, "Adaptive chameleon or stubborn sloth: Unraveling the behavior of large language models in knowledge conflicts," *arXiv preprint arXiv:2305.13300*, 2023.

[91] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1601–1611.

[92] Z. Yang *et al.*, "HotpotQA: A dataset for diverse, explainable multi-hop question answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2369–2380.

[93] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "FEVER: A large-scale dataset for fact extraction and VERification," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 809–819.

[94] W. Chen, X. Wang, and W. Y. Wang, "A dataset for answering time-sensitive questions," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[95] J. Kasai *et al.*, "Realtime qa: What's the answer right now?" *arXiv preprint arXiv:2207.13332*, 2022.

[96] M. Zhang and E. Choi, "SituatedQA: Incorporating extra-linguistic contexts into QA," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7371–7387.

[97] B. Dhingra, J. R. Cole, J. M. Eisenschlos, D. Gillick, J. Eisenstein, and W. W. Cohen, "Time-aware language models as temporal knowledge bases," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 257–273, 2022.

[98] K. Cobbe *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[99] P. Lu *et al.*, "Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning," *arXiv preprint arXiv:2209.14610*, 2022.

[100] M. Li *et al.*, *Api-bank: A benchmark for tool-augmented llms*, 2023. arXiv: 2304.08244 [cs.CL].

[101] Y. Qin *et al.*, *Tool learning with foundation models*, 2023. arXiv: 2304.08354 [cs.CL].

[102] Q. Xu, F. Hong, B. Li, C. Hu, Z. Chen, and J. Zhang, "On the tool manipulation capability of open-source large language models," *arXiv preprint arXiv:2305.16504*, 2023.

[103] S. Jain, M. van Zuylen, H. Hajishirzi, and I. Beltagy, "SciREX: A challenge dataset for document-level information extraction," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 7506–7516.

[104] S. Wiegreffe, J. Hessel, S. Swayamdipta, M. Riedl, and Y. Choi, "Reframing human-ai collaboration for generating free-text explanations," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 632–658.

[105] R. Zhang, Y. Yu, P. Shetty, L. Song, and C. Zhang, "Prboost: Prompt-based rule discovery and boosting for interactive weakly-supervised learning," *arXiv preprint arXiv:2203.09735*, 2022.

[106] E. Almazrouei *et al.*, "Falcon-40B: An open large language model with state-of-the-art performance," 2023.

[107] Y. Zhuang *et al.*, "ToolChain*: Efficient action space navigation in large language models with a* search," in *Proceedings of ICLR 2024*, 2024.

[108] E. Jang, "Can llms critique and iterate on their own outputs?" *evjang.com*, 2023.

[109] G. Kim, P. Baldi, and S. McAleer, "Language models can solve computer tasks," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[110] J. Liang *et al.*, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9493–9500.

[111] Y. Qin *et al.*, "Tool learning with foundation models," *arXiv preprint arXiv:2304.08354*, 2023.

[112] G. Mialon *et al.*, *Augmented language models: A survey*, 2023. arXiv: 2302.07842 [cs.CL].

[113] W. Shi *et al.*, "Ehragent: Code empowers large language models for complex tabular reasoning on electronic health records," *arXiv preprint arXiv:2401.07128*, 2024.

[114] R. Nakano *et al.*, *Webgpt: Browser-assisted question-answering with human feedback*, 2022. arXiv: 2112.09332 [cs.CL].

[115] L. Yang, H. Chen, Z. Li, X. Ding, and X. Wu, *Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling*, 2023. arXiv: 2306.11489 `[cs.CL]`.

[116] A. Parisi, Y. Zhao, and N. Fiedel, "Talm: Tool augmented language models," *arXiv preprint arXiv:2205.12255*, 2022.

[117] J. Zhang, *Graph-toolformer: To empower llms with graph reasoning ability via prompt augmented by chatgpt*, 2023. arXiv: 2304.11116 `[cs.AI]`.

[118] X. Wang *et al.*, "Self-consistency improves chain of thought reasoning in language models," in *The Eleventh International Conference on Learning Representations*, 2022.

[119] J. Jung *et al.*, "Maieutic prompting: Logically consistent reasoning with recursive explanations," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 1266–1279.

[120] Y. Xie *et al.*, *Decomposition enhances reasoning via self-evaluation guided decoding*, 2023. arXiv: 2305.00633 `[cs.CL]`.

[121] X. Zhu *et al.*, "Solving math word problems via cooperative reasoning induced language models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4471–4485.

[122] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[123] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, Springer, 2006, pp. 282–293.

[124] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," in *The Eleventh International Conference on Learning Representations*, 2022.

[125] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *International Conference on Learning Representations*, 2020.

[126] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of NAACL, Volume 1 (Long Papers)*, 2018, pp. 1112–1122.

[127] S. Lin, J. Hilton, and O. Evans, *Teaching models to express their uncertainty in words*, 2022. arXiv: 2205.14334 [cs.CL].

[128] Y. Zhuang *et al.*, "Hydra: Model factorization framework for black-box llm personalization," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[129] Z. Zhao *et al.*, "Recommender systems in the era of large language models (llms)," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[130] Y. Hou *et al.*, "Large language models are zero-shot rankers for recommender systems," in *European Conference on Information Retrieval*, Springer, 2024, pp. 364–381.

[131] D. Di Palma, "Retrieval-augmented recommender system: Enhancing recommender systems with large language models," in *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023, pp. 1369–1373.

[132] S. Dai *et al.*, "Uncovering chatgpt's capabilities in recommender systems," in *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023, pp. 1126–1132.

[133] X. L. Dong, S. Moon, Y. E. Xu, K. Malik, and Z. Yu, "Towards next-generation intelligent assistants leveraging llm techniques," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 5792–5793.

[134] Y. Li *et al.*, "Personal llm agents: Insights and survey about the capability, efficiency and security," *arXiv preprint arXiv:2401.05459*, 2024.

[135] L. Zheng *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[136] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "Toolqa: A dataset for llm question answering with external tools," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[137] Y. Zhuang *et al.*, "Toolchain*: Efficient action space navigation in large language models with a* search," in *The Twelfth International Conference on Learning Representations*, 2023.

[138] H. Sun*, Y. Zhuang*, W. Wei, C. Zhang, and B. Dai, *BBox-Adapter: Lightweight adapting for black-box large language models*, 2024. arXiv: 2402.08219 [cs.CL].

[139] W. Shi *et al.*, "Medadapter: Efficient test-time adaptation of large language models towards medical reasoning," *arXiv preprint arXiv:2405.03000*, 2024.

[140] Y. Yu *et al.*, "Explanation-aware soft ensemble empowers large language model in-context learning," *arXiv preprint arXiv:2311.07099*, 2023.

[141] S. Gururangan *et al.*, "Don't stop pretraining: Adapt language models to domains and tasks," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 8342–8360.

[142] R. Xu *et al.*, "Knowledge-infused prompting: Assessing and advancing clinical text data generation with large language models," *arXiv preprint arXiv:2311.00287*, 2023.

[143] R. Xu *et al.*, "Ram-ehr: Retrieval augmentation meets clinical predictions on electronic health records," *arXiv preprint arXiv:2403.00815*, 2024.

[144] W. Shi, Y. Zhuang, Y. Zhu, H. Iwinski, M. Wattenbarger, and M. D. Wang, "Retrieval-augmented large language models for adolescent idiopathic scoliosis patients in shared decision-making," in *Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2023, pp. 1–10.

[145] R. Xu *et al.*, "Bmretriever: Tuning large language models as better biomedical text retrievers," *arXiv preprint arXiv:2404.18443*, 2024.

[146] J. J. Cheung* *et al.*, "PolyIE: A dataset of information extraction from polymer material scientific literature," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*, 2024.

[147] H. R. Kirk, B. Vidgen, P. Röttger, and S. A. Hale, "Personalisation within bounds: A risk taxonomy and policy framework for the alignment of large language models with personalised feedback," *arXiv preprint arXiv:2303.05453*, 2023.

[148] Z. Tan, Q. Zeng, Y. Tian, Z. Liu, B. Yin, and M. Jiang, "Democratizing large language models via personalized parameter-efficient fine-tuning," *arXiv preprint arXiv:2402.04401*, 2024.

[149] X. Wu, X. Liu, J. Niu, H. Wang, S. Tang, and G. Zhu, *FedloRA: When personalized federated learning meets low-rank adaptation*, 2024.

[150] X. Li, Z. C. Lipton, and L. Leqi, *Personalized language modeling from personalized human feedback*, 2024. arXiv: 2402.05133 `[cs.CL]`.

[151] Z. Wu *et al.*, "Fine-grained human feedback gives better rewards for language model training," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[152] J. Jang *et al.*, "Personalized soups: Personalized large language model alignment via post-hoc parameter merging," *arXiv preprint arXiv:2310.11564*, 2023.

[153] W.-C. Kang *et al.*, *Do llms understand user preferences? evaluating llms on user rating prediction*, 2023. arXiv: 2305.06474 `[cs.IR]`.

[154] J. Liu, C. Liu, P. Zhou, R. Lv, K. Zhou, and Y. Zhang, "Is chatgpt a good recommender? a preliminary study," *arXiv preprint arXiv:2304.10149*, 2023.

[155] D. Wang *et al.*, "Learning personalized story evaluation," *arXiv preprint arXiv:2310.03304*, 2023.

[156] A. Zhiyuli, Y. Chen, X. Zhang, and X. Liang, "Bookgpt: A general framework for book recommendation empowered by large language model," *arXiv preprint arXiv:2305.15673*, 2023.

[157] K. Christakopoulou *et al.*, "Large language models for user interest journeys," *arXiv preprint arXiv:2305.15498*, 2023.

[158] A. Salemi, S. Mysore, M. Bendersky, and H. Zamani, "Lamp: When large language models meet personalization," *arXiv preprint arXiv:2304.11406*, 2023.

[159] A. Salemi, S. Kallumadi, and H. Zamani, "Optimization methods for personalizing large language models through retrieval augmentation," *arXiv preprint arXiv:2404.05970*, 2024.

[160] S. Mysore *et al.*, "Pearl: Personalizing large language model writing assistants with generation-calibrated retrievers," *arXiv preprint arXiv:2311.09180*, 2023.

[161] C. Li *et al.*, "Teach llms to personalize–an approach inspired by writing education," *arXiv preprint arXiv:2308.07968*, 2023.

[162] C. Richardson *et al.*, "Integrating summarization and retrieval for enhanced personalization via large language models," *arXiv preprint arXiv:2310.20081*, 2023.

[163] Q. Liu, N. Chen, T. Sakai, and X.-M. Wu, "Once: Boosting content-based recommendation with both open-and closed-source large language models," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024, pp. 452–461.

[164] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[165] S. Mishra *et al.*, "Lila: A unified benchmark for mathematical reasoning," *arXiv preprint arXiv:2210.17517*, 2022.

[166] R. Nakano *et al.*, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[167] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, *Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents*, 2023. arXiv: 2302.01560 `[cs.AI]`.

[168] G. Kim, P. Baldi, and S. McAleer, *Language models can solve computer tasks*, 2023. arXiv: 2303.17491 `[cs.CL]`.

[169] L. Zheng *et al.*, *Judging llm-as-a-judge with mt-bench and chatbot arena*, 2023. arXiv: 2306.05685 `[cs.CL]`.

[170] X. Puig *et al.*, "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[171] P. Lu *et al.*, *Learn to explain: Multimodal reasoning via thought chains for science question answering*, 2022. arXiv: 2209.09513 `[cs.CL]`.

[172] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[173] P. He, J. Gao, and W. Chen, "Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing," in *The Eleventh International Conference on Learning Representations*, 2022.

[174] G. Izacard *et al.*, "Unsupervised dense information retrieval with contrastive learning," *arXiv preprint arXiv:2112.09118*, 2021.

[175] H. Lyu *et al.*, "Llm-rec: Personalized recommendation via prompting large language models," in *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024, pp. 583–612.

VITA

Yuchen Zhuang graduated from the High School Affiliated to Nanjing Normal University in 2015. He received his Bachelor of Engineering degree in Information Science and Engineering from Southeast University, Nanjing, Jiangsu, China, in 2019. He then earned his Master of Science degree in Electrical and Computer Engineering from the Georgia Institute of Technology in 2021. Motivated by his strong interest in language modeling, he pursued doctoral studies in Machine Learning at the School of Computer Science and Engineering, Georgia Institute of Technology. His current research focuses on language intelligence, specifically the development of language-model-based agents designed for human-like reasoning and planning to effectively solve complex real-world problems.