Query Routing for Homogeneous Tools: An Instantiation in the RAG Scenario

Feiteng Mu[1*], **Yong Jiang**[2†], **Liwen Zhang**[2], **Chu Liu**[2], **Wenjie Li**[1], **Pengjun Xie**[2], **Fei Huang**[2],
[1]The Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[2]Institute for Intelligent Computing, Alibaba Group
{csfmu,cswjli}@comp.polyu.edu.hk,{yongjiang.jy,zlw439616}@alibaba-inc.com

Abstract

Current research on tool learning primarily focuses on selecting the most effective tool from a wide array of options, often overlooking cost-effectiveness, a crucial factor in human problem-solving. In this paper, we address query routing for homogeneous tools by predicting both their performance and the associated cost required to accomplish a given task. We then assign queries to the optimal tools in a cost-effective manner. Our experimental results demonstrate that our method achieves higher performance at a lower cost compared to strong baseline approaches.

"*Don't use a sledgehammer to crack a nut.*"
—— *Proverb*

1 Introduction

Tool learning (Qin et al., 2023a; Qu et al., 2024b) aims to arm large language models (LLMs) (Touvron et al., 2023; Bai et al., 2023; Achiam et al., 2023) with real world tools, to alleviate hallucinations (Ji et al., 2023; Zhang et al., 2023) of LLMs.

Existing tool learning methods focus on routing a query to the most effective tool from a large number of options (Qin et al., 2023b; Tang et al., 2023), while overlooking the crucial aspect of cost-effectiveness, which is a significant criterion in human problem-solving. To bridge this gap, query routing for homogeneous tools has become an important issue. Taking the retrieval-augmented generation (RAG) (Lewis et al., 2020; Gao et al., 2023) scenario as an example. Given a user query, a RAG system first uses a search tool to retrieve the background documents. Then, a LLM reads the query and documents to give the response. Given the availability of various web search tools, such as Google and Bing, the choice of which tool to use

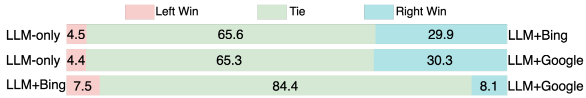


Figure 1: The win/tie/lose rates on our PrivateTimeQA test set when the LLM uses two compared tools. Besides Bing and Google, we consider a non-retrieval baseline, and denote the method as "LLM-only".

for a given query becomes an important consideration. First, a query that cannot be solved by one tool might be resolved by another. As shown in Figure 1, Our dataset indicates that (1) in 4.5% of cases, directly utilizing the LLM to answer queries surpasses the performance of the Bing-based RAG. And (2) there are 8.1% of instances where the Bing-based RAG fails to deliver a solution, whereas the Google-based RAG successfully provides one. Furthermore, different tools come with varying usage costs. From a cost-effectiveness perspective, it is preferable to select the least expensive tool that can effectively solve the query. If we can adaptively allocate queries to the optimal tool, we can achieve a balance between cost and performance. Additionally, leveraging the complementarity between tools offers the potential for enhanced overall performance. The key challenge in achieving this goal is estimating the performance of candidate tools in handling user queries, instead of making decisions after calling all tools for user queries (Chen et al., 2023; Jiang et al., 2023). To overcome this challenge, we automatically construct training data including queries submitted to all tools, along with the returned scores. The training data allows us to learn neural models to predict the performance that LLM calls each tool to solve each query.

In this work, we explore query routing for homogeneous tools and instantiate it in the RAG scenario. We propose a router that assigns input queries to the most appropriate search tool without directly accessing any of the tools. To achieve this, we de-

*Work done during internship at Alibaba Inc.
†Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 10225–10230

velop an automated pipeline to construct training data without manual effort. Finally, we design various assignment strategies to allocate user queries to the optimal tool as needed.

In summary, we make the following contributions. **(1)** We propose a general approach that dynamically assigns an input query to the optimal tool from a set of homogeneous candidates, only based on the input query. **(2)** As far as we know, we are the first to consider the selection of homogeneous tools. Our framework is generalizable to any type of homogeneous tools, extending beyond the RAG scenario. **(3)** We evaluate our approach on several QA tasks with multiple LLMs. Experiments show that our method outperforms the use of a single, fixed search tool across different datasets.

2 Related Work

Tool Learning Existing tool-learning methods focus on developing different tools to solve various problems. (Schick et al., 2024) teaches LLMs to call tools like Calculator, Calendar, etc. (Qin et al., 2023b) facilitates LLMs to master 16000+ real-world APIs. (Zheng et al., 2024; Qu et al., 2024a; Du et al., 2024) select the most suitable tool for a given query from a vast tool set, according to the similarity between the query and tool descriptions. Differently, we investigate the problem of selecting homogeneous tools, which are less considered.

LLMs Selection Earlier, Different LLMs charged differently due to their uneven performance. Practitioners focus on effectively choosing LLM to save costs. (Chen et al., 2023) gradually employs more expensive LMs until a satisfactory performance is obtained. (Lu et al., 2023; Šakota et al., 2024; Shnitzer et al., 2023) propose the neural routing function that can precisely distribute each query to the expert LLMs. However, recently, the cost of using LLMs has continued to decrease. The usage cost of tools has become a major obstacle to LLMs-based applications. Thus, we explore the selection of homogeneous tools to achieve a balance between performance and cost.

3 Method

3.1 Problem Formulation

We demonstrate our work within the RAG framework to provide a realistic illustration. It should be noted that our approach is universal and applicable to various scenarios, not limited to just RAG. Suppose that a LLM needs to solve a set of user queries. The LLM is permitted to call a search tool to generate responses. We consider a situation where there are M homogeneous tools $\{T_m\}_{m=1}^{M}$, e.g., Bing and Google, each with different costs. Our goal is to develop an adaptive assignment method to achieve a cost-performance trade-off.

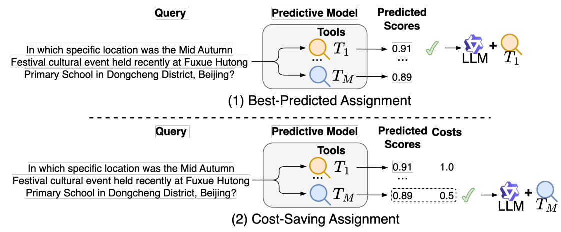


Figure 2: The framework of our method. We first predict the scores where the LLM calls each tool to solve each query. Then, we design different strategies to assign each query to the optimal tool on demand.

We decompose this process into two steps. In the first step, we learn a predictive model $\mathcal{M}$ that predicts the scores where the LLM calls each tool to solve each query. We restrict $\mathcal{M}$ to only depend on the input query q and the used tool T_m:

$$p_m = \mathcal{M}(q, T_m), \forall m \in \{1, \cdots, M\}. \tag{1}$$

In the second step, we devise diverse strategies to assign a query to the optimal tool on demand. The framework of our method is shown in Figure 2.

Next, we introduce the training of $\mathcal{M}$ and the implementation of the assignment strategies.

3.2 Training Predicative Model

Data Preparation We develop an automated pipeline to construct training data. Given the observed data $\mathcal{D}$ consisting of query-answer pairs $\{(q_n, r_n^{gold})\}_{n=1}^{|\mathcal{D}|}$, for each q_n, the LLM calls each search tool to obtain the query-related documents, and generates retrieval-augmented response r_n^m:

$$\begin{aligned} \mathrm{doc}_n^m &= T_m(q_n), \\ r_n^m &= LLM(q_n, \mathrm{doc}_n^m). \end{aligned} \tag{2}$$

Then, we use the Text-Matching score between the generated answer r_n^m and the ground-truth answer r_n^{gold} to automatically calculate the labeled score when the LLM calls T_m to handle q_n:

$$l_n^m = \text{Text-Matching}(r_n^m, r_n^{gold}). \tag{3}$$

Training Given collected $\{(q_n, \{l_n^m\}_{m=1}^{M})\}_{n=1}^{|\mathcal{D}|}$, we train $\mathcal{M}$ through the following objective:

$$\begin{aligned} p_n^m &= \mathcal{M}(q_n, T_m), \\ \min_{\mathcal{M}} \frac{1}{|\mathcal{D}|} &\sum_{n=1}^{|\mathcal{D}|} \sum_{m=1}^{M} \mathcal{L}(p_n^m, l_n^m), \end{aligned} \tag{4}$$

where we set $\mathcal{L}$ as Mean Square Error and initialize $\mathcal{M}$ as RoBERTa (Liu et al., 2019). The objective directly simulates the score of the LLM calling each tool to solve each query, enabling it to handle unseen cases where only the query is available.

3.3 Assignment Strategies

Given search tools $\{T_m\}_{m=1}^{M}$, we denote their usage costs as $\{C_m\}_{m=1}^{M}$. Assuming the user has a collection of queries $\{q_n\}_{n=1}^{\mathcal{N}}$ that need to be processed, we first use $\mathcal{M}$ to obtain the predicted scores $\{p_n^m\}_{m=1}^{M}$ for the query q_n. Then, inspired by (Lu et al., 2023; Šakota et al., 2024; Shnitzer et al., 2023), we consider the following strategies to assign queries to corresponding tools.

(1) **Fixed-Tool**: LLM always calls a fixed tool to handle queries. (2) **Best-Performance**: for q_n and the predicted $\{p_n^m\}_{m=1}^{M}$, the LLM selects $T_{m^\star}$ ($m^\star = \arg\max_m p_n^m$) with the maximal predicted score to handle q_n. (3) **Cost-Saving**: This strategy aims to ensure that the average predicted score, i.e., p_n^m, exceeds a user-defined threshold while minimizing the average cost. This problem can be solved by the integer linear programming (ILP):

$$\begin{aligned} \text{minimize} \quad & \sum w_n^m C_m \\ \text{s.t.} \quad & \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} \sum_{m=1}^{M} w_n^m p_n^m \geq P_{\text{min}} \\ & \sum_{m=1}^{M} w_n^m = 1, \quad \forall n \in \{1, \cdots, \mathcal{N}\}, \end{aligned} \tag{5}$$

where the variable $w_n^m \in \{0, 1\}$ indicates whether to assign q_n to T_m, P_{min} denotes the threshold.

4 Experiment

4.1 Experimental Setup

Datasets and Training details We focus on the QA scenario. The adopted datasets include public datasets CDQA (Xu et al., 2024) and WebQA (Li et al., 2016), as well as a private dataset PrivateTimeQA[1] that we construct ourselves. Data processing details, dataset statistics, and training details are shown in Appendix A.

Baselines We use multiple LLMs, including Qwen-Max[2] (Bai et al., 2023), ChatGPT, and GPT4 in all of our experiments, due to their strong reasoning ability. The used search tools include **Quark**, **Bing**, and **Google**. Considering that the LLM can answer easy queries independently, we set a **non-retrieval** baseline, i.e., the LLM directly responds to the query. Consequently, its usage cost is set to 0. In addition, since existing methods (Shnitzer et al., 2023; Šakota et al., 2024) assign queries to the most suitable LLM, we devise "LLMs-UB", Specifically, For a query and corresponding scores obtained from several non-retrieval methods, such as Qwen-only, ChatGPT-only, and GPT4-only, we always assign the query to the LLM with the highest score. Therefore, this can be seen as an **u**pper **b**ound of (Shnitzer et al., 2023; Šakota et al., 2024).

4.2 Evaluation the Predictive Model

We first examine the ability of $\mathcal{M}$ to simulate the score that LLM calls each tool to solve test queries.

4.2.1 Metrics

For the testset $\{(q_n, \{l_n^m\}_{m=1}^{M})\}_{n=1}^{\mathcal{N}}$, we report the final QA *accuracy* by selecting the tool with the maximal predicted p_n^m for RAG:

$$\begin{aligned} m^\star &= \arg\max_m p_n^m, \forall n \in \{1, \cdots, \mathcal{N}\}, \\ \text{accuracy} &= \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} l_n^{m^\star}. \end{aligned} \tag{6}$$

We also report the average cost of each method. For simplicity, we set the average costs of Quark, Bing, and Google to 0.33, 2, and 1, respectively.

4.2.2 Results

The result is shown in Table 1. Whether using Qwen-max, ChatGPT, or GPT4 for response generation, LLM+Google usually outperforms LLM+Bing, LLM+Quark, and LLM-only, which coincides with human intuition that Google is currently the best search tool in the world. RAG outperforms LLM-only by a large margin, proving the necessity of utilizing tools for response generation. In addition, we have the following observations.

(1) On PrivateTimeQA and CDQA, our method significantly outperforms the best baseline, i.e., "LLM+Google". This trend is observed across multiple LLMs. This is because despite "LLM+Google" achieving the highest overall accuracy, it still fails to solve some cases that "LLM+Quark" or "LLM+Bing" can handle. Specifically, the proportion of such samples is 6.6% in the CDQA testset. Our method can adaptively assign these queries to the optimal tools, leading to better overall results. **(2)** Compared with "LLM+Bing", our method achieves a better result with a lower cost. Compared with "LLM+Google", our method

[1]We leave the detail in Appendix A.

[2]https://dashscope.aliyuncs.com/compatible-mode/v1

Methods	PrivateTimeQA		CDQA		WebQA	
	Acc.	Cost	Acc.	Cost	Acc.	Cost
ChatGPT						
LLM-only	30.47	0	29.91	0	59.25	0
LLMs-UB	55.64	0	48.32	0	93.00	0
LLM+Quark	67.81	0.33	55.91	0.33	93.47	0.33
LLM+Bing	66.63	2	60.08	2	91.18	2
LLM+Google	68.63	1	61.13	1	**94.07**	1
Ours (Roberta-large)	**68.89**	1.24	**63.28**	0.68	93.55	0.52
GPT4						
LLM-only	29.9	0	28.46	0	71.77	0
LLMs-UB	55.64	0	48.32	0	93.00	0
LLM+Quark	71.87	0.33	57.22	0.33	94.42	0.33
LLM+Bing	70.66	2	61.22	2	93.36	2
LLM+Google	73.35	1	64.42	1	**95.5**	1
Ours (Roberta-large)	**73.6**	1.01	**65.74**	0.58	94.54	0.79
Qwen-max						
LLM-only	50.30	0	45.90	0	91.14	0
LLMs-UB	55.64	0	48.32	0	93.00	0
LLM+Quark	77.54	0.33	59.84	0.33	**96.33**	0.33
LLM+Bing	76.91	2	63.86	2	93.48	2
LLM+Google	77.95	1	66.75	1	94.97	1
Ours (Roberta-large)	**78.41**	1.23	**69.56**	1.20	94.68	1.30

Table 1: The result of our method when we use ChatGPT, GPT4 and Qwen-max for response generation. "LLM-only" means the non-retrieval baseline. "LLM+Quark", "LLM+Bing" and "LLM+Google" means the LLM always calls Quark, Bing, or Google to solve queries. “Acc. (%)" denotes *accuracy*. Scores with **bold** denote the best values among baselines.

Methods	Model Size	TimeQA		CDQA	
		Acc.	Cost	Acc.	Cost
ChatGPT					
Ours (Roberta-base)	125M	68.68	1.24	62.3	0.68
Ours (Roberta-large)	355M	**68.89**	1.24	**63.28**	0.68
Ours (Qwen-0.5B)	0.5B	67.99	1.25	62.14	1.38
Ours (Qwen-1.8B)	1.8B	68.73	1.02	61.37	1.03
GPT4					
Ours (Roberta-base)	125M	72.48	1.45	65.57	0.83
Ours (Roberta-large)	355M	**73.6**	1.01	**65.74**	0.58
Ours (Qwen-0.5B)	0.5B	72.11	1.32	63.16	1.42
Ours (Qwen-1.8B)	1.8B	73.06	1.10	64.76	1.07

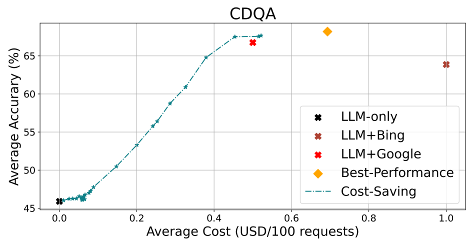
Table 2: The result of different backbones.

has a better performance with a approaching cost. This demonstrates the generality of our method for different search engines and LLMs.

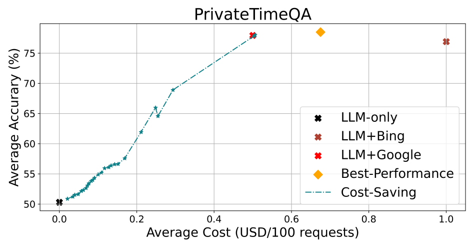
4.3 Evaluation of the Adaptively Assignment

Settings According to different strategies in Section 3.3, we assign test queries to the corresponding tools and calculate the average accuracy and usage cost[3]. Cost–accuracy curves are plotted, which present the relationship between the average accuracy and the average cost per query needed.

Result We plot cost-accuracy curves on the test sets of PrivateTimeQA and CDQA in Figure 3. "LLM+Google" has a higher accuracy and a lower cost compared to "LLM+Bing". We additionally have the following observations. **(1)** Based on our $\mathcal{M}$, "Best-Performance" achieves a higher accuracy than "LLM+Google", albeit at a higher cost. The accuracy increment comes from the supplement from "LLM+Bing", with a side-effect of higher cost. **(2)** Compared to both "LLM+Google" and "Best-Performance", the "Cost-Saving" strategy is able to achieve higher accuracy with a lower cost. This shows that "Cost-Saving" is more flexible and can effectively save costs.



(a) The curve on the CDQA testset.



(b) The curve on the PrivateTimeQA testset.

Figure 3: The cost-accuracy curves on PrivateTimeQA and CDQA when using Qwen-max.

4.4 Further Discussion

Impact of Different Backbones We investigate the impact of different backbones for training our router. We consider Roberta-base(125M), Roberta-large(355M), Qwen-0.5B (0.5B), Qwen-1.8B(1.8B) for experiments. When using Roberta-large as the backbone, the result is better than that based on Roberta-base. But when using the larger Qwen-0.5B (0.5B), Qwen-1.8B (1.8B), even if we have tried various training techniques, the results do not increase significantly. We speculate that the decoder-only models are not suitable for this routing task which requires accurate numerical predictions. In addition, the larger model leads to higher

[3]Bing Search charges $10 for 1000 requests, and Google Search charges $5 for 1000 requests.

latency. Therefore, we finally chose Roberta-large as our backbone. Due to limited computing resources, we are unable to attempt larger models such as Qwen-7b, Llama7b, etc.

5 Conclusion

In this work, we investigate the problem of selecting homogeneous tools for the sake of cost-effective trade-offs. We provide a clear definition of the problem and propose a general framework that can be adapted to any type of homogeneous tools. The experiment shows that our method is effective and can achieve higher performance while at a lower cost compared to strong baselines. This indicates the potential of our framework where multiple homogeneous tools are available.

Limitations

Due to the high cost of search tools, our training data only contains approximately 16000 examples,. In the future, with a sufficient budget, we will conduct testing on more datasets. We use small encoding models to train our predictive model instead of LLMs, mainly for the following two reasons: (1) Tool selection has a high requirement for latency, while the speed of large models is relatively slow and does not meet the requirement. (2) Tool selection requires the model to predict the score of each tool solving each problem, which is actually a numerical prediction problem. However, the auto-regressive LLMs have significant issues in numerical prediction. In our experiment, the performance of LLMs was inferior to that of small encoding models.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Peng Li, Wei Li, Zhen He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. 2016. Dataset and neural recurrent sequence labeling model for open-domain factoid question answering. *ArXiv*, abs/1607.06275.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024a. Colt: Towards completeness-oriented tool retrieval for large language models. *arXiv preprint arXiv:2405.16089*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024b. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*.

Marija Šakota, Maxime Peyrard, and Robert West. 2024. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 606–615.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. 2023. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Zhikun Xu, Yinghui Li, Ruixue Ding, Xinyu Wang, Boli Chen, Yong Jiang, Xiaodong Deng, Jianxin Ma, Hai-Tao Zheng, Wenlian Lu, Pengjun Xie, Chang Zhou, and Fei Huang. 2024. Let llms take on the latest challenges! a chinese dynamic question answering benchmark. *ArXiv*, abs/2402.19248.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's song in the ai ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. Toolrerank: Adaptive and hierarchy-aware reranking for tool retrieval. *arXiv preprint arXiv:2403.06551*.

A Data Processing Details and Statistics

Construction Details about PrivateTimeQA PrivateTimeQA primarily consists of questions related to timeliness. We collected question-answer examples from public QA websites. We prioritize privacy protection and data integrity, ensuring that the dataset contains only open-domain QA content and excludes any harmful, or unethical material. PrivateTimeQA is used to evaluate LLMs' capability to answer timeliness questions. Due to confidentiality requirements, we are unable to publicly disclose the dataset.

Data Processing For WebQA and CDQA, we use the official data split. For each example, we first use different search tools and LLMs for retrieval-augmented generation. We discard examples with invalid search results. After obtaining labels, we merge training examples of CDQA and WebQA, and randomly split the Train/Dev partitions with a ratio of 0.85/0.15. We train the model on the mixed train part and test our model on the testsets of WebQA, PrivateTimeQA, and CDQA. The statistics of used datasets are shown in Table 3.

	Language	Train	Dev	Test
WebQA	Chinese	8102	1409	2924
PrivateTimeQA	Chinese	N/A	N/A	1831
CDQA	Chinese	8067	1444	1056

Table 3: Statistcs of used datasets.

Training Details We adopt *damo/nlp_roberta_backbone_large_std* as the backbone. These pre-trained models can be downloaded from the modelscope[4]. We use the Adam optimizer and linearly decrease the learning rate to zero with a 10% warmup ratio. We use PyTorch toolkit to conduct all experiments on the Ubuntu server with a V100 (32G) GPU. All the hyperparameters for are searched according to the final QA accuracy on the development set. For reproduction, we set the random seed to 42 for all experiments. The searched parameters are shown in Table 4.

batch size	16
num-epochs	20
lr	5e-5
α	0.8
k	2

Table 4: The used hyperparameters in our experiments.

[4]https://www.modelscope.cn/home