# A Framework for Effective Invocation Methods of Various LLM Services

**Can Wang, Dianbo Sui, Bolin Zhang, Xiaoyu Liu, Jiabao Kang, Zhidong Qiao, Zhiying Tu***
Harbin Institute of Technology (HIT), Weihai, Shandong, China
{23B903072}@stu.hit.edu.cn
{suidianbo, brolin}@hit.edu.cn
{24S130270, 23B903065, 23S136230}@stu.hit.edu.cn
{tzy_hit}@hit.edu.cn

## Abstract

Large Language Models (LLMs) have shown impressive abilities in solving various natural language processing tasks and are now widely offered as services. LLM services enable users to accomplish tasks without requiring specialized knowledge, simply by paying service providers. However, numerous providers offer various LLM services with variations in pricing, latency, and performance. These factors are also affected by different invocation methods, such as the choice of context and the use of cache, which lead to unpredictable and uncontrollable service cost and quality. Consequently, utilizing various LLM services invocation methods to construct an effective (cost-saving, low-latency and high-performance) invocation strategy that best meets task demands becomes a pressing challenge. This paper provides a comprehensive overview of methods help LLM services to be invoked efficiently. Technically, we define the problem of constructing an effective LLM services invocation strategy, and based on this, propose a unified LLM service invocation framework. The framework classifies existing methods into four categories: input abstraction, semantic cache, solution design, and output enhancement, which can be used separately or jointly during the invocation life cycle. We discuss the methods in each category and compare them to provide valuable guidance for researchers. Finally, we emphasize the open challenges in this domain and shed light on future research.

## 1 Introduction

Large Language Models (LLM) are becoming a fundamental tool for various natural language processing tasks (Yang et al., 2024b), as they have shown amazing emergent abilities, like in-context learning (Dong et al., 2023), multi-step reasoning (Fu et al., 2023), instruction following (Lou and Yin, 2024) and tool learning (Huang et al., 2024). Due to commercial reasons, the potential risk of misuse and expensive tuning cost, LLMs, such as GPT-3 (Brown et al., 2020), GPT-4 (OpenAI et al., 2024) and Claude[1], are usually released as LLM services through application programming interface (API) instead of open sourcing model weights, which is called Language Models as a Service (LMaaS) (Zhao et al., 2021). By accessing these powerful LLMs as services through their opened API, novice users do not need to possess extensive computational resources and expertise in deep learning, as they can solve the tasks of interest by crafting task-specific input queries.

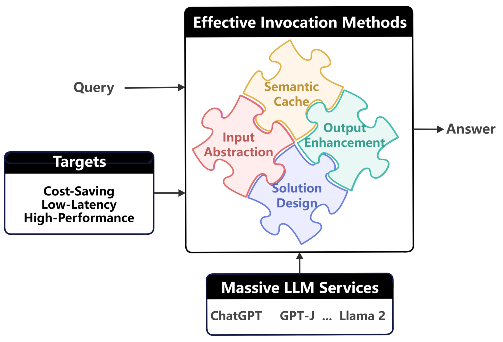

Figure 1: Vision of effective invocation strategy construction for various LLM services.

However, invoking LLM services is not free and using them for high-throughput applications can be very expensive. Estimated by Claudia Slowik[2], a business supporting 15,000 customer interactions with text-davinci-003 could have a monthly cost exceeding \$14,400. We present the cost of using 25 different LLM services from some top-tier providers (as shown in the Appendix) and find that the costs of different LLM services can vary by up to two orders of magnitude. For instance, the input

*Corresponding Author.

[1] https://claude.ai/

[2] https://neoteric.eu/blog

cost for 1 million tokens is \$10 for OpenAI's GPT-4 but only \$0.2 for Mistral 7B hosted by Textsynth.

In addition to cost considerations, various factors, including response time and performance for the same input query, can also impact the user experience using LLM services. Ahia et al. (2023) and Lai et al. (2023) find that different languages, prompt methods or the inclusion of simple enhancements can also lead to notable alterations in performance. Meanwhile, Chen et al. (2023) discover that affordable LLMs often complement expensive ones. For instance, on the CoQA (Reddy et al., 2019) dataset, GPT-4 makes errors in approximately 11% of the questions, while the more affordable GPT-J provides the correct answers.

From the fact that the heterogeneity in pricing does not necessarily correlate with the user experience, it is a great need to explore effective invocation methods for LLM services in practice. As shown in Figure 1, we expect to make use of various LLM services to construct an effective invocation strategy, which can be easily adjusted and reused according to the users' different invocation targets. To this end, we provide a comprehensive study of the development and recent advances on effective invocation methods of LMaaS. In detail, we first formalize the task of constructing an effective invocation strategy as a multi-objective optimization problem (Gunantara, 2018) , which entails simultaneous consideration of latency, performance, and cost factors. Then, we propose a taxonomy to provide a unified view on effective invocation methods of LMaaS where existing methods are categorized into: input abstraction, semantic cache, solution design, and output enhancement. These four categories can be flexibly combined and unified into a flexible framework. Finally, we highlight the challenges and potential directions.

The contributions of this survey can be concluded as follows:

- **Comprehensive Taxonomy**. We define the problem of constructing an LLM services effective invocation strategy mathematically. Based on this, we propose the taxonomy in Figure 2, which categorizes existing methods from four different aspects: input abstraction, semantic cache, solution design, and output enhancement.

- **Flexible Framework**. As shown in Figure 3, we propose a framework that unifies the four categories of methods. Our framework connects different categories during the LLM service invocation life cycle, allowing each of them to be used separately or jointly.

- **Related Resources**. To facilitate the methods of this task, the price rules of popular LMaaS products are shown in Table 1. The anonymous GitHub repository presents the existing methods available, with a demo website implementing our framework.[3]

## 2 Background

In this section, we first formalize the problem of constructing an effective invocation strategy of various LLM services. Then, we explain the problem definition from the perspective of the LLM service invocation lifecycle, which divides the invocation process into three phases. According to the different use phases and purposes when construction, we propose our taxonomy and framework.

### 2.1 Problem Definition

In our topic, the problem is defined as how to construct an effective (cost-saving, low-latency and high-performance) invocation strategy $s$ given a task $T$ among various LLM services. The given task $T$ consists of multiple identical query-answer pairs, represented as $T = \{(q_1, a_1), (q_2, a_2), ..., (q_n, a_n)\}$, where $q_i$ represents input query and $a_i$ represents output answer. First, we consider a fixed LLM service $M$ published through API. Input a query $q$, the process of obtaining the response $\tilde{a}$ by invocation of the LLM service $M$ can be represented as:

$$\tilde{a} = M(q). \tag{1}$$

To characterize the concerned factors for the construction of effective invocation strategy with a given query $q$ and LLM service $M$, we use three metric functions: latency $f_l(M, q)$, performance $f_p(M, q)$, and cost $f_c(M, q)$. These three functions are fixed values in a specific practical invocation and can be estimated using certain methods. For example, $f_l$ could be a function of the length of the input and output sequences. $f_p$ often uses a metric function $r(\cdot, \cdot)$ to compare the difference between $a$ and $\tilde{a}$. While $f_c$ involves two different pricing parts, input cost and output cost, we adopt

[3] https://anonymous.4open.science/r/Effective-strategy-for-LMaas-BF83

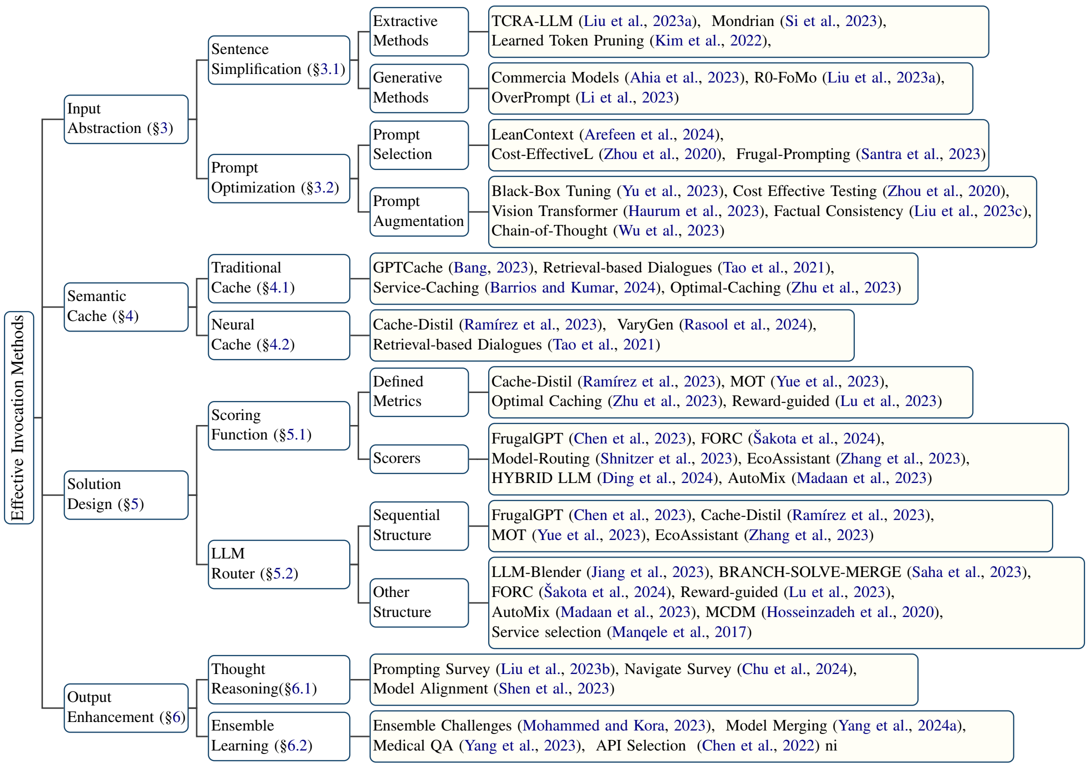
Effective Invocation Methods
Input Abstraction (§3)
Sentence Simplification (§3.1)
Extractive Methods
TCRA-LLM (Liu et al., 2023a), Mondrian (Si et al., 2023), Learned Token Pruning (Kim et al., 2022),
Generative Methods
Commercia Models (Ahia et al., 2023), R0-FoMo (Liu et al., 2023a), OverPrompt (Li et al., 2023)
Prompt Optimization (§3.2)
Prompt Selection
LeanContext (Arefeen et al., 2024), Cost-EffectiveL (Zhou et al., 2020), Frugal-Prompting (Santra et al., 2023)
Prompt Augmentation
Black-Box Tuning (Yu et al., 2023), Cost Effective Testing (Zhou et al., 2020), Vision Transformer (Haurum et al., 2023), Factual Consistency (Liu et al., 2023c), Chain-of-Thought (Wu et al., 2023)
Semantic Cache (§4)
Traditional Cache (§4.1)
GPTCache (Bang, 2023), Retrieval-based Dialogues (Tao et al., 2021), Service-Caching (Barrios and Kumar, 2024), Optimal-Caching (Zhu et al., 2023)
Neural Cache (§4.2)
Cache-Distil (Ramírez et al., 2023), VaryGen (Rasool et al., 2024), Retrieval-based Dialogues (Tao et al., 2021)
Solution Design (§5)
Scoring Function (§5.1)
Defined Metrics
Cache-Distil (Ramírez et al., 2023), MOT (Yue et al., 2023), Optimal Caching (Zhu et al., 2023), Reward-guided (Lu et al., 2023)
Scorers
FrugalGPT (Chen et al., 2023), FORC (Šakota et al., 2024), Model-Routing (Shnitzer et al., 2023), EcoAssistant (Zhang et al., 2023), HYBRID LLM (Ding et al., 2024), AutoMix (Madaan et al., 2023)
LLM Router (§5.2)
Sequential Structure
FrugalGPT (Chen et al., 2023), Cache-Distil (Ramírez et al., 2023), MOT (Yue et al., 2023), EcoAssistant (Zhang et al., 2023)
Other Structure
LLM-Blender (Jiang et al., 2023), BRANCH-SOLVE-MERGE (Saha et al., 2023), FORC (Šakota et al., 2024), Reward-guided (Lu et al., 2023), AutoMix (Madaan et al., 2023), MCDM (Hosseinzadeh et al., 2020), Service selection (Manqele et al., 2017)
Output Enhancement (§6)
Thought Reasoning(§6.1)
Prompting Survey (Liu et al., 2023b), Navigate Survey (Chu et al., 2024), Model Alignment (Shen et al., 2023)
Ensemble Learning (§6.2)
Ensemble Challenges (Mohammed and Kora, 2023), Model Merging (Yang et al., 2024a), Medical QA (Yang et al., 2023), API Selection (Chen et al., 2022) ni

Figure 2: Taxonomy of effective invocation methods of LMaaS

the definition of the prompt length multiplied by the token price as shown in the Eq. 2, where $\alpha_i$ is a constant representing the unit price.

$$f_c \triangleq \alpha_1||q|| + \alpha_2||\tilde{a}|| + \alpha_3 \tag{2}$$

Based on that, we extend a single LLM service to $K$ different LLM services $M_s = \{M_1, M_2, ...M_k\}$. Our problem is formalized as in Eq. 3, where in the search space $S$, we seek an optimal invocation strategy $s$ that minimizes latency $f_l$, maximizes performance $f_p$, and minimizes cost $f_c$ on task $T$. The optimal strategy $s$ includes a sequence of selected LLM services, represented as $s = \{M_i\}, i \leq K$, offering flexibility in choosing one or multiple services in a specific order.

$$\begin{aligned} &\text{Minimize } \mathbf{F}(s) = \begin{bmatrix} f_l(M_i, q_j) \\ -f_p(M_i, q_j) \\ f_c(M_i, q_j) \end{bmatrix} \\ &\text{subject to } M_i \in s, q_j \in T \end{aligned} \tag{3}$$

This is a multi-objective optimization problem, and we solve it using simply weighted sum or other methods. In the construction strategy of a specific invocation, constraints may be introduced. For example, in the scenario of limited funds, the cost $f_c \leq C$ is used as a condition to obtain a strategy with high-performance $f_p$ and low-latency $f_l$, where $C$ is the maximum cost that can be used.

## 2.2 LLM Services Invocation Taxonomy and Framework

Following the idea that building an effective invocation strategy requires an understanding of the key resources involved in the LLM service life cycle (Bai et al., 2024), we organize our framework by dividing the LLM service invocation into three phases: before invocation, during invocation, and after invocation. According to the different phases and purpose, the taxonomy divides the methods to effectively invoke LLM services into four major categories, which are connected in a sequential way in our proposed framework to optimize the common goal in Eq. 3.

Before invocation, processing the input query $q$ that a user entered to express more meaningful information in a more concise way is the first step to construct the effective invocation strategy. The related methods are summarized as **input abstraction** (Section 3), which can be further divided into **sentence simplification** and **prompt optimization** according to the different ways. **Semantic cache** (Section 4) is also an important method to improve service performance, reduce latency and cost before invocation, which is divided into **traditional cache** and **neural cache** according to different structures. The semantic cache checks whether

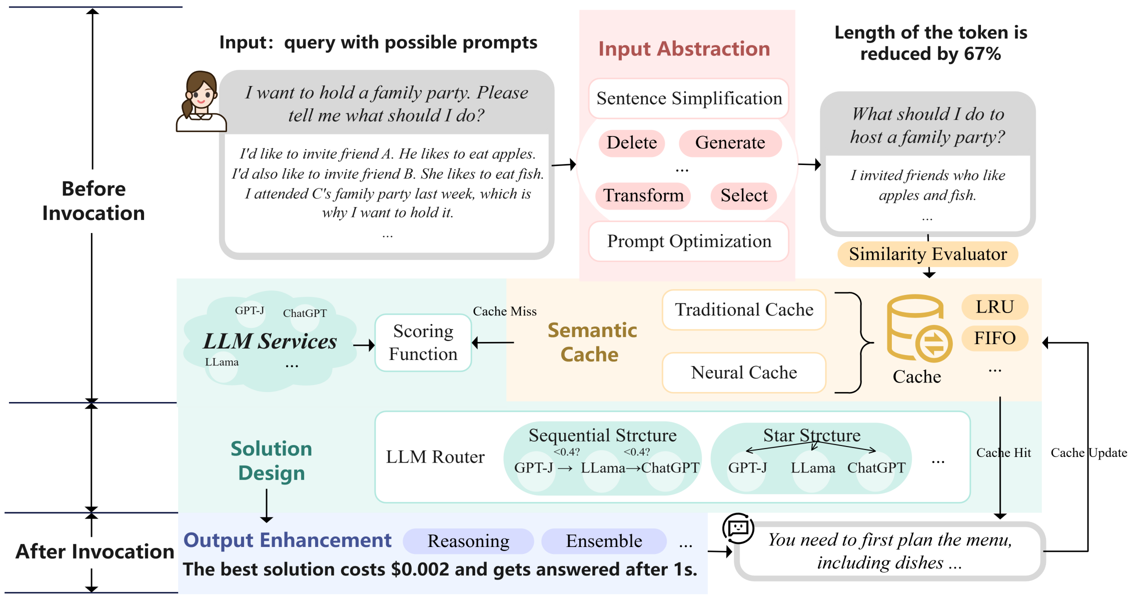
Before Invocation
After Invocation
Input: query with possible prompts
I want to hold a family party. Please tell me what should I do?
I'd like to invite friend A. He likes to eat apples. I'd also like to invite friend B. She likes to eat fish. I attended C's family party last week, which is why I want to hold it.
...
Input Abstraction
Sentence Simplification
Delete
Generate
Transform
Select
Prompt Optimization
Length of the token is reduced by 67%
What should I do to host a family party?
I invited friends who like apples and fish.
Similarity Evaluator
LLM Services
GPT-J
ChatGPT
LLama
Scoring Function
Cache Miss
Semantic Cache
Traditional Cache
Neural Cache
Cache
LRU
FIFO
Solution Design
LLM Router
Sequential Strcture
GPT-J → LLama→ChatGPT
Star Strcture
GPT-J LLama ChatGPT
Cache Hit
Cache Update
Output Enhancement
Reasoning
Ensemble
The best solution costs $0.002 and gets answered after 1s.
You need to first plan the menu, including dishes ...

Figure 3: LLM services invocation framework, shown by the phase of invocation.

there is a semantically similar query $q$ in cache. If so, it directly returns the previous answer $\tilde{a}$, and otherwise it goes into the invocation phase.

During invocation, **solution design** (Section 5) aims to construct the best invocation solution $s$ by leveraging the complementary capabilities of various LLM services $M_i$. It evaluates LLM services $M_i$ with a given query $q$, and the method of evaluation is called **scoring function**. It can also be done before the invocation. For example, the estimation of $f_c$ can be used to guide the design of a cost-saving solution. During the invocation phase, the **LLM router** is used to organize routing structures between LLM services according to the estimated results. Through different routing structures, the advantages of different services are utilized to build optimal strategies for users.

After invocation, **output enhancement** (Section 6) focuses on the information returned to the user. The output $\tilde{a}$ is enhanced to improve clarity and accuracy, allowing it to better meet different targets and enhance the overall user experience. Output enhancement can be implemented in two main ways: **thought reasoning** and **ensemble learning technology**. In addition, the input and output of this invocation are stored into the semantic cache for future invocations.

In summary, we categorize effective invocation methods for LLM services into four groups, as shown in Figure 2, based on different phases of use and their construction purposes. We discuss these categories and compare the methods in later sections, providing advantages, disadvantages, and applicable scenarios. We propose the LLM services invocation framework that can unify these methods as illustrated in Figure 3, where different categories of methods can be used separately or jointly.

## 3 Input Abstraction

Input abstraction is designed for better performance at lower cost and latency to invoke a given LLM. The generalization and in-context capabilities allow LLM services to obtain good answers on untrained samples (Dong et al., 2023), and input content directly affects the cost, latency or performance of the service. For example, concatenating the prompt “Just tell me the option" with the question as input to the LLM will generate shorter output, reducing invocation cost and latency. However, it may cause the LLM to lose its ability to think step by step, resulting in performance degradation.

We group the methods into two categories based on different goals, sentence simplification and prompt optimization. The input query $q$ typically consists of a question (representing the user's task) and multiple prompts (optional information to aid in task completion). Sentence simplification reduce the length of the input query without changing the semantics, while prompt optimization ensures the quality of the query and improves the performance of the invocation by optimizing prompts.

## 3.1 Sentence Simplification

Sentence simplification is the process of making input more concise and simple while retaining its core meaning by modifying, removing, or replacing words, phrases, or structures in a sentence. The process is similar to the summarization task, and many methods used in summarization can be applied (Huang et al., 2021; Watanangura et al., 2024; Antony et al., 2023; Mridha et al., 2021). We collate the methods available for LMaaS, and classify them into extractive and generative methods based on whether derived entirely from the original input.

**Extractive methods.** Extractive methods select sentences from long original input by extracting key sentences or phrases, where the content is entirely sourced from the origin. Pruning semantically irrelevant tokens according to the relevance to the contexts is an effecitve approach (Liu et al., 2023a). Iterative deletion and substitution of tokens are another methods (Si et al., 2023; Kim et al., 2022), removing unimportant tokens based on the attention mechanism.

Extractive methods are straightforward and efficient, making it convenient for the real-time use. However, the extractive methods may ignore some global information. And it has limitations in some tasks such as language translation, as it cannot discern which parts need to be translated or deleted.

**Generative methods**. The generative methods refer to rewriting based on the original input, allowing for the generation of new words. Language encoding is a simple processing method. Ahia et al. (2023) conduct extensive experiments with different languages and tokenizers, where the cost varied by up to 5 times. AE.studio[4] employs encryption to provide an online platform that sacrifices readability, reducing the length of input tokens by half. Besides, utilizing fast and low cost generative models (Li et al., 2023) presents a viable option for sentence simplification.

Generative methods are flexible, as the generated sentences contain less redundant information while preserving the main content. However, it may introduce grammatical or factual errors.

## 3.2 Prompt Optimization

Prompt optimization is the adjustment of user-provided input prompts to guide LLM to produce more accurate, useful, and tailored output. The effectiveness of prompt optimization stems from the LLM's ability to learn from few-shot demonstrations (Liu et al., 2023b), where appropriate prompts can complement the context of a task, highlight key information, or improve the explainability.

Based on the different granularity of optimization objective, we distinguish two types of prompt optimization methods. Prompt selection selects or combines some sentences in prompts to guide invocations more effectively. Prompt augmentation is concerned with the quality of the content and aims to maximize the potential of the context.

**Prompt selection.** Prompt selection selects the most meaningful prompt from possible prompts to accurately guide LLM. It removes the interference of irrelevant prompts, and helps in efficient invocation. Zhou et al. (2020) select representative samples that can be highly beneficial in few-shot tasks. Santra et al. (2023) combine various methods involving instructions, examples, and additional context to propose a more compact method for providing prompts in dialogues. Arefeen et al. (2024) consider the concatenation of prompts and retrieves the most important $k$ sentences, enabling the shared use of prompts for similar questions.

Prompt selection directly guide LLM to focus on specific information and understand user needs more accurately without too much personalization. However, this approach cannot maximize the potential of LLM capability for complex prompts because no additional knowledge is introduced.

**Prompt augmentation.** Prompt augmentation considers the understanding ability of LLM to elicit more accurate and desirable responses. Knowledge retrieval is a direct method of augmentation, as it helps achieving a comprehensive understanding during the invocation. Haurum et al. (2023) respond the limitations of factual knowledge in LLM and optimizes the reasoning process with minimal retrieval cost. Yu et al. (2023) and Zhou et al. (2020) propose black-box fine-tuning methods to optimize continuous prompts using non-derivative methods. Model alignment (Liu et al., 2023c) and chain of thought reasoning (Wu et al., 2023), are also key focuses in prompt optimization.

The improvement in invocation performance through prompt augmentation is significant, despite it may resulting in more complex processing procedures. However, the general method is difficult to explore, requiring professional knowledge.

[4]Prompt Reducer: `https://www.promptreducer.com/`

## 4 Semantic Cache

Semantic cache is an approach to improve LLM invocation efficiency and performance by storing and quickly retrieving information. Unlike traditional data cache, semantic cache focuses on storing high-level semantics information such as meaning, relationship, rather than just raw data. The semantic cache is checked before the LLM service is invoked. If cache hit, the output given by the cache is returned, avoiding subsequent costly invocations while responding faster. With the gradual increase in the scale of LLM, the semantic cache plays a more important role in accelerating computation, reducing data transmission costs, and supporting high concurrent requests (Miao et al., 2023), providing users with low-latency, high-performance, and cost-saving services.

There are two typical structures for implementing semantic cache in LMaaS, and unlike other subsections, these two structures generally cannot be used jointly. Traditional caches use key-value pairs for storage and retrieval, returning the same value for similar input. Neural caches, on the other hand, use neural networks to respond in a predictive manner, learning semantic relationships between inputs without relying on a fixed storage structure.

### 4.1 Traditional Cache

The current paradigm of traditional cache consists of three parts (Bang, 2023): the cache manager, similarity evaluator and post processor. The cache manager is responsible for storing content in the form key-value pairs, and managing eviction. The similarity evaluator is used to determine if any of the keys in the cache match the input query. The post processor organizes the final response to be returned to the user. If no similar query is found in the cache, the LLM service is invoked by the post processor to generate the output and then the generated output is stored in the cache.

Bang (2023) represents a typical implementation of traditional cache, which utilizes question embedding for similarity matching and provides various matching methods. The open-source application Zep[5] supports storage LLM applications, storing information in the database. Through theoretical proof, Zhu et al. (2023) introduce the cache scheme with minimum expected cost considering the query frequency. Besides, methods for query and conversations cache (Tao et al., 2021; Barrios and Kumar, 2024) can be migrated to LLM services.

Implementing traditional cache is usually simple, requiring only basic data structures such as hash set. This approach is general, but it may not capture semantic similarity between inputs because it relies heavily on the key matching.

### 4.2 Neural Cache

Neural cache uses neural networks or deep learning models to learn and store data representations. Neural cache maps input data into a high-dimensional space by learning the representation of the data, which can capture the semantic similarity of the input. Unlike the compositional paradigm of traditional cache, neural cache has no specific structure.

Ramírez et al. (2023) train a student model using T5-base[6] for providing early feedback in classification tasks. To address the semantic cache missing issue, Rasool et al. (2024) generate similar input to hit the cache as much as possible. Furthermore, a retrieval-based dialogue response selection model (Tao et al., 2021) also can serve as an alternative choice for neural cache.

The neural cache often outperforms the traditional cache, especially in domain-specific problems. However, its implementation and updates can be relatively complex. As such, it is important to carefully consider the effectiveness of the cache to avoid incurring unnecessary resource waste.

## 5 Solution Design

Solution design considers different scenarios and targets, dynamically selecting one or more LLM services that are most suitable for the invocation, and organizing them to provide flexible and efficient solutions. The solution design helps to select LLM services that best solve the task, taking advantage of LLM services' heterogeneous costs and performance. When new queries arrive or requirements change, the solution can be flexibly updated to achieve optimal performance and cost savings.

Solution design has two main parts working together to achieve dynamic LLM service selection and routing. The scoring function is responsible for evaluating the performance of each available LLM service, which reflects the concerned factors for invocation such as quality and speed. The router, based on the evaluation results of the scoring function, performs query routing between services, and

[5] https://github.com/getzep/zep

[6] https://huggingface.co/docs/transformers/model_doc/t5

selects the appropriate one in a dynamic manner.

### 5.1 Scoring Function

The scoring function is a comprehensive evaluation of LLM services given a specific task, considering both targets and scenarios, which is used to guide the routing path in the solution. The scoring function may be influenced by multiple factors, such as response time, query cost, and accuracy of answers. The scoring function plays a decision-making role, and helps to understand the relative performance of each LLM service. The scoring function can be implemented in two different ways.

**Defined metrics.** Defined metrics provide a measurable way to achieve direct quantification of factors that affect invocation. For instance, accuracy in classification tasks, BLEU score in generation tasks, packet loss in web, and quality of service (QoS) are all applicable indicators. Ramírez et al. (2023) use interval sampling and predictive entropy to determine whether to invoke LLM services. Considering three sources of consistency, decision-making for LLM services is performed through sampling and voting (Yue et al., 2023). Calculating the cost expectations between two models, Zhu et al. (2023) extend the decision when invoking multiple LLM services. Reward ranking from answers provided by different services is used as an evaluation criterion by Lu et al. (2023), incurring minimal computational cost in the solution.

The defined metrics are intuitive and easily understandable, which are often based on statistical data or experiments, being less susceptible to subjective factors. However, setting thresholds for evaluating the quality of LLM services can be challenging and may not adapt well to the dynamic and changing environment. Additionally, certain complex factors may be difficult to capture with defined metrics, leading to limitations in scoring.

**Scorers.** A scorer is a tool for scoring LLM services based on metrics that cannot be defined by a particular formula. The scorer utilizes prior knowledge, training data, or rules to provide scores in a less interpretable manner, typically using smaller neural networks (Chen et al., 2023). For example, ALBERT (Lan et al., 2020) is used as a scorer (Šakota et al., 2024), with the query and predicted output as $x$, and the accuracy of invocation as $y$. Using DistilBERT (Sanh et al., 2020) as a scorer, with query and model ID as $x$, Shnitzer et al. (2023) predict whether an LLM can solve the problem. A comparison of LLM performances on different benchmark datasets is conducted, with Zhang et al. (2023) modeling it as a binary selection problem and providing guiding suggestions. For specific tasks, such as the predictor of execution results in the code generation task (Zhang et al., 2023), the classifier of query difficulty in the task of question and answer (Ding et al., 2024; Madaan et al., 2023), and estimator of LLM service capability in the dataset benchmark test task (Shnitzer et al., 2023) are all reasonable scorers.

Compared with metrics defined by formulas, the scorers can be updated based on real-time data and feedback, demonstrating strong generalization across different scenarios. However, it is equivalent to using a more powerful model and incurring its training and usage costs. Moreover, it still requires labeled examples, which is applicable only when the query dataset is larger than the training dataset.

### 5.2 LLM Router

The LLM router emphasizes the organizational structure between LLM services, connecting multiple independent services in a specific order logically. It focuses on constructing a flexible and reusable solution to address continuously changing queries or targets. Depending on the scoring function and position used, the LLM router can construct various target-oriented solutions, such as cost-oriented or performance-oriented solutions.

**Sequential Structure**. Sequential structure is the simplest structure, which selects one or several available services from the extensive pool of LLM services and invokes them in sequence. The scoring function is used to decide whether to accept the answer or proceed to the next step (Chen et al., 2023). The number of models is typically limited to three in a sequential structure, and possible options are determined through permutation, with pruning techniques applied (Ramírez et al., 2023). The use of small models as a cache, with large models being invoked in sequence when cache misses occur, can also be regraded as a fixed sequential structure (Yue et al., 2023). For problems like code generation (Zhang et al., 2023), an initial response is obtained using a cost-saving LLM, which is tracked as context for subsequent queries.

This structure is simple and effective, and a limited number of permutations can be searched quickly in the entire space. However, the sequential structure may result in the invocation of all LLM services in the sequential structure. Thus, it is difficult to extend, because all LLM services need to

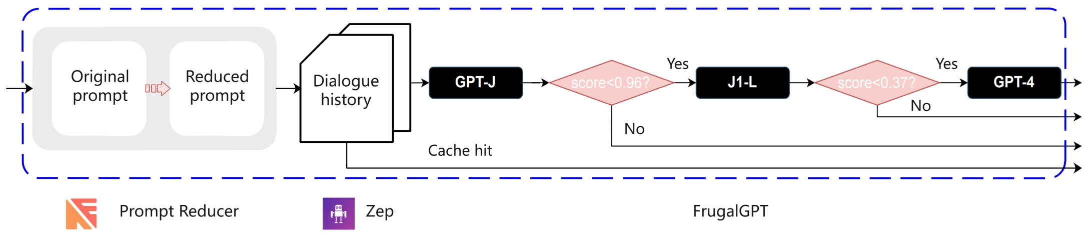

Figure 4: A simple invocation strategy composed of existing methods, using Prompt Reducer in input abstraction, Zep in semantic cache, FrugalGPT in solution design, and nothing in output enhancement.

be rearranged when adapting to new requirements.

**Other Structure.** Parallel structure, similar to the bagging in machine learning, can enhance the performance of LLM services, with task decomposition and merging being key aspects (Jiang et al., 2023). Star structure, as seen in Šakota et al. (2024); Lu et al. (2023), involves decision-making by a meta-model, allocating the current query to the most suitable model. For the third category of unsolvable queries, pruning is applied by Madaan et al. (2023) to prevent unnecessary costs for particularly challenging problems. The tree structure is considered to be promising, combining advantages of both star-shaped and sequential structures. It initially routes query to the most probable branch, and then invokes services in sequence. Additionally, certain selection solutions specific to HTTP services (Hosseinzadeh et al., 2020; Manqele et al., 2017) are also noteworthy to be explored.

Other structure offers more flexible and extensible solutions, while also introducing complexity into the routing process. Moreover, designing and implementing these structures can be challenging, particularly when dealing with dynamic query environments or evolving system requirements

## 6 Output Enhancement

Output enhancement refers to the process of further optimizing and adjusting the output generated by the invocation. This process improve the syntactic correctness, semantic accuracy, and overall fluency of the generated output to meet needs of the user under the specific scenario. Depending on the method, output enhancement can be classified into thought reasoning and ensemble learning.

The thought reasoning focuses on the internal reasoning process of the LLM to improve the organization and explanation of outputs. Ensemble learning technology emphasizes the cooperation of multiple service invocations. By combining the strengths of different models, it enhances the stability and robustness of the LLM, and reduces the bias and error of the output of a single service.

### 6.1 Thought Reasoning

Thought reasoning improve reasoning clarity and logical flow in outputs by changing the path of reasoning inside LLM services. Liu et al. (2023b) guide an LLM to give concise answers using carefully designed prompt, reducing unnecessary output tokens. Chu et al. (2024) use X-of-Thought to refer to the chain of thought in a broad sense, systematically organize the current research according to the classification of methods. In addition, work on model alignment identifies (Shen et al., 2023) and reduces the bias or unsafe behavior that LLM may produce in inference, reducing the need for subsequent human intervention.

The thought reasoning is easy to implement and obviously enhances the logicality and organization of the output through simple prompt optimization. However, thought reasoning relies on carefully designed prompts, and the effect is limited by the ability of the underlying LLM service.

### 6.2 Ensemble Learning

Ensemble learning is a powerful technique that combines multiple LLM services to improve performance over individual LLM services. Mohammed and Kora (2023) and Yang et al. (2024a) discusse the work and challenges of model merging, combining structure or knowledge from multiple models into a single model. To solve domain-specific tasks, Yang et al. (2023) enhance medical question answering capabilities based on boosting and clustering methods, and (Chen et al., 2022) aggregate responses from multiple low-cost LLM services to improve the precision of multi-label prediction tasks.

By combining the predictions of multiple LLM services, ensemble learning techniques significantly improve the accuracy and stability of the output, reduce the risk of overfitting, and flexibly adapt to multi-domain tasks. However, this approach requires more computational resources and time, leading to high costs, while the complexity of the model also makes the system difficult to debug and interpret.

## 7 Conclusion and Challenges

In conclusion, this paper provide a comprehensive overview of effective invocation methods in the realm of LMaaS. Through the establishment of a taxonomy, we categorize existing methods into four categories: input abstraction, semantic cache, solution design, and output enhancement. We formalize the problem of effective LLM services strategy construction, and propose an LLM services invocation framework. Methods of different categories in the framework can be used separately or jointly to form the effective strategy invocation that are low-latency, high-performance, and cost-saving.

Most existing methods focus only on one category in the framework, and the methods in different categories can be used as plugins. A practical example of a simple invocation strategy built from three existing methods to save money is shown in Figure 4. In addition, other factors may also be important when calling LLM services, including but not limited to interpretability, security, automation rate, etc. Our framework is open and flexible, allowing for easy expansion to these aspects. We look forward to future research further advancing the field, and here are some open challenges.

**Input Abstraction.** One of the main challenges faced in the input abstraction category is the processing of multi-modal input (Yin et al., 2023). More comprehensive and balanced methods (Zhang et al., 2024) are needed to optimize multiple types of input such as text, image, and speech. Input abstraction methods for dynamically changing queries are also worth exploring, such as real-time data streaming (Räth et al., 2023) or user interaction with the system.

**Semantic Cache.** In the semantic cache category, how to design and select cache methods (Brais et al., 2021) more efficiently to accommodate different inputs and queries is the main challenge faced in traditional cache, while semantic representation (Brito, 2023) can be achieved by neural cache. The study of more efficient algorithms for cache storage and update (Jin et al., 2024) is also a technique worth discussing.

**Solution Design.** In terms of solution design, a quantitative evaluation of LLM services (Chang et al., 2024) is an extension of the scoring function, which adaptation and interpretability need to be paid more attention to in the future. The LLM router needs to focus on designing more powerful service integration methods that not only focus on a task, but also take into account requirements of different resources (Xu et al., 2024).

**Output Enhancement.** The importance of output enhancement is gradually seen by people. The balance between specification and diversity (Chung et al., 2023) of output is a key issue. When a task is completed, the user's satisfaction is an important indicator to measure the quality of service, and future research may focus on building more intelligent and user-oriented (Chisari et al., 2022) output enhancement methods.

**Other Challenges.** Basic work such as qualitative description and quantitative comparison in experiments still has a gap to be filled, and the lack of baselines results in no uniform standard for the comparison of the LLM services invocation methods. Futher studies, such as how to choose the tokenizer (Alyafeai et al., 2023) with the shortest input, how to set the best suitable size of cache (Vavouliotis et al., 2022), and the choice of different pricing methods for the LLM service, need to be explored. In additionally, We specifically call for attention to fairness (Sah et al., 2024) and privacy issues (Luo et al., 2024; Utpala et al., 2023) of LMaaS.

## Limitations

First, in defining the construction of an effective invocation strategy for LLM services, we select three key factors and model the topic as a multi-objective optimization problem. It is not comprehensive because there are many other factors in the actual invocation of LLM services, which also affect each other. Second, we summarize the effective LLM service invocation methods, aiming to provide a general framework to help users build the cost-saving, low-latency and high-performance invocation strategy. However, some of these methods are limited because we do not consider whether they can be directly applied to black-box LLM services that are only published through APIs. Furthermore, due to the variety of experimental datasets and evaluation metrics, we are unable to conclude a unified baseline for this topic, which would be a strong support for future research.

## Acknowledgements

The work is supported by the National Key R&D Program of China (Grant No.2022YFF0902703), the National Natural Science Foundation of China (Grant No.62472121), the National Natural Science Foundation of China (Grant No.62306087), and the Special Funding Program of Shandong Taishan Scholars Project.

## References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R. Mortensen, Noah A. Smith, and Yulia Tsvetkov. 2023. Do all languages cost the same? tokenization in the era of commercial language models. In *Proc. of EMNLP*.

Zaid Alyafeai, Maged Saeed AlShaibani, Mustafa Ghaleb, and Irfan Ahmad. 2023. Evaluating various tokenizers for arabic text classification. *Neural Process. Lett.*

Dinu Antony, Sumit Abhishek, Sujata Singh, Siddu Kodagali, Narayana Darapaneni, Mukesh Rao, Anwesh Reddy Paduri, and BG Sudha. 2023. A survey of advanced methods for efficient text summarization. In *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*.

Md Adnan Arefeen, Biplob Debnath, and Srimat Chakradhar. 2024. Leancontext: Cost-efficient domain-specific question answering using llms. *Natural Language Processing Journal*.

Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, Carl Yang, Yue Cheng, and Liang Zhao. 2024. Beyond efficiency: A systematic survey of resource-efficient large language models. *ArXiv*.

Fu Bang. 2023. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings.

Carlos Barrios and Mohan Kumar. 2024. Service caching and computation reuse strategies at the edge: A survey. *ACM Comput. Surv.*

Hadi Brais, Rajshekar Kalayappan, and Preeti Ranjan Panda. 2021. A survey of cache simulators. *ACM Comput. Surv.*

Eduardo Brito. 2023. *Explainable Resource-Aware Representation Learning via Semantic Similarity*. Ph.D. thesis.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Proc. of NeurIPS*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2022. Efficient online ML API selection for multi-label classification tasks. In *Proc. of ICML*.

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*.

Eugenio Chisari, Tim Welschehold, Joschka Boedecker, Wolfram Burgard, and Abhinav Valada. 2022. Correct me if i am wrong: Interactive learning for robotic manipulation. *IEEE Robotics and Automation Letters*.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through enigmatic labyrinth A survey of chain of thought reasoning: Advances, frontiers and future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 1173–1203. Association for Computational Linguistics.

John Chung, Ece Kamar, and Saleema Amershi. 2023. Increasing diversity while maintaining accuracy: Text data generation with large language models and human interventions. In *Proc. of ACL*.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A survey on in-context learning. *Preprint*, arXiv:2301.00234.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2023. Complexity-based prompting for multi-step reasoning. In *Proc. of ICLR*.

Nyoman Gunantara. 2018. A review of multi-objective optimization: Methods and its applications. *Cogent Engineering*.

Joakim Bruslund Haurum, Sergio Escalera, Graham W. Taylor, and Thomas B. Moeslund. 2023. Which tokens to use? investigating token reduction in vision transformers. In *Proc. of ICCV*.

Mehdi Hosseinzadeh, Hawkar Kamaran Hama, Marwan Yassin Ghafour, Mohammad Masdari, Omed Hassan Ahmed, and Hemn Khezri. 2020. Service selection using multi-criteria decision making: a comprehensive overview. *Journal of Network and Systems Management*.

Jerry Huang, Prasanna Parthasarathi, Mehdi Rezagholizadeh, and Sarath Chandar. 2024. Towards practical tool usage for continually learning llms. *arXiv preprint arXiv:2404.09339*.

Yichong Huang, Xiachong Feng, Xiaocheng Feng, and Bing Qin. 2021. The factual inconsistency problem in abstractive text summarization: A survey. *arXiv preprint arXiv:2104.14839*.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proc. of ACL*.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *Preprint*, arXiv:2404.12457.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *Proc. of KDD*.

Viet Dac Lai, Nghia Trung Ngo, Amir Pouran Ben Veyseh, Hieu Man, Franck Dernoncourt, Trung Bui, and Thien Nguyen. 2023. Chatgpt beyond english: Towards a comprehensive evaluation of large language models in multilingual learning. In *Proc. of EMNLP Findings*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. *Preprint*, arXiv:1909.11942.

Jiazheng Li, Runcong Zhao, Yongxin Yang, Yulan He, and Lin Gui. 2023. Overprompt: Enhancing chatgpt through efficient in-context learning. In *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*.

Junyi Liu, Liangzhi Li, Tong Xiang, Bowen Wang, and Yiming Qian. 2023a. TCRA-LLM: token compression retrieval augmented large language model for inference cost reduction. In *Proc. of EMNLP Findings*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023b. Pretrain, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*

Yixin Liu, Budhaditya Deb, Milagro Teruel, Aaron Halfaker, Dragomir Radev, and Ahmed Hassan Awadallah. 2023c. On improving summarization factual consistency from natural language feedback. In *Proc. of ACL*.

Renze Lou and Wenpeng Yin. 2024. Toward zero-shot instruction following. In *Proc. of EACL*.

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*.

Jinglong Luo, Yehong Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. 2024. Secformer: Towards fast and accurate privacy-preserving inference for large language models. *arXiv preprint arXiv:2401.00793*.

Aman Madaan, Pranjal Aggarwal, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. 2023. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*.

Lindelweyizizwe Manqele, Mqhele E. Dlodlo, Louis Coetzee, and George Sibiya. 2017. A survey for service selection approaches in dynamic environments. In *IEEE AFRICON 2017, Cape Town, South Africa, September 18-20, 2017*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*.

Ammar Mohammed and Rania Kora. 2023. A comprehensive review on ensemble deep learning: Opportunities and challenges. *Journal of King Saud University - Computer and Information Sciences*, 35(2):757–774.

Muhammad F. Mridha, Aklima Akter Lima, Kamruddin Nur, Sujoy Chandra Das, Mahmud Hasan, and Muhammad Mohsin Kabir. 2021. A survey of automatic text summarization: Progress, process and challenges. *IEEE Access*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, and Shyamal Anadkat et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Guillem Ramírez, Matthias Lindemann, Alexandra Birch, and Ivan Titov. 2023. Cache & distil: Optimising api calls to large language models. *arXiv preprint arXiv:2310.13561*.

Zafaryab Rasool, Scott Barnett, David Willie, Stefanus Kurniawan, Sherwin Balugo, Srikanth Thudumu, and Mohamed Abdelrazek. 2024. Llms for test input generation for semantic caches. *Preprint*, arXiv:2401.08138.

Timo Räth, Ngozichukwuka Onah, and Kai-Uwe Sattler. 2023. Interactive data cleaning for real-time streaming applications. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. Coqa: A conversational question answering challenge. *Trans. Assoc. Comput. Linguistics*.

Chandan Kumar Sah, Dr Lian Xiaoli, and Muhammad Mirajul Islam. 2024. Unveiling bias in fairness evaluations of large language models: A critical literature review of music and movie recommendation systems. *arXiv preprint arXiv:2401.04057*.

Swarnadeep Saha, Omer Levy, Asli Celikyilmaz, Mohit Bansal, Jason Weston, and Xian Li. 2023. Branch-solve-merge improves large language model evaluation and generation. *arXiv preprint arXiv:2310.15123*.

Marija Šakota, Maxime Peyrard, and Robert West. 2024. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proc. of WSDM*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *Preprint*, arXiv:1910.01108.

Bishal Santra, Sakya Basak, Abhinandan De, Manish Gupta, and Pawan Goyal. 2023. Frugal prompting for dialog models. In *Proc. of EMNLP Findings*.

Tianhao Shen, Renren Jin, Yufei Huang, Chuang Liu, Weilong Dong, Zishan Guo, Xinwei Wu, Yan Liu, and Deyi Xiong. 2023. Large language model alignment: A survey. *arXiv preprint arXiv:2309.15025*.

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*.

Wai Man Si, Michael Backes, and Yang Zhang. 2023. Mondrian: Prompt abstraction attack against large language models for cheaper api pricing. *arXiv preprint arXiv:2308.03558*.

Chongyang Tao, Jiazhan Feng, Rui Yan, Wei Wu, and Daxin Jiang. 2021. A survey on response selection for retrieval-based dialogues. In *Proc. of IJCAI*.

Saiteja Utpala, Sara Hooker, and Pin-Yu Chen. 2023. Locally differentially private document generation using zero shot prompting. In *Proc. of EMNLP Findings*.

Georgios Vavouliotis, Gino Chacon, Lluc Alvarez, Paul V. Gratz, Daniel A. Jiménez, and Marc Casas. 2022. Page size aware cache prefetching. In *Proc. of MICRO*.

Patcharapruek Watanangura, Sukit Vanichrudee, On Minteer, Theeranat Sringamdee, Nattapong Thanngam, and Thitirat Siriborvornratanakul. 2024. A comparative survey of text summarization techniques. *SN Comput. Sci.*

Dingjun Wu, Jing Zhang, and Xinmei Huang. 2023. Chain of thought prompting elicits knowledge augmentation. In *Proc. of ACL Findings*.

Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, et al. 2024. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092*.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024a. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *Preprint*, arXiv:2408.07666.

Han Yang, Mingchen Li, Huixue Zhou, Yongkang Xiao, Qian Fang, and Rui Zhang. 2023. One llm is not enough: Harnessing the power of ensemble learning for medical question answering. *medRxiv*.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024b. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2023. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*.

Lang Yu, Qin Chen, Jiaju Lin, and Liang He. 2023. Black-box prompt tuning for vision-language model as a service. In *Proc. of IJCAI*.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2023. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *arXiv preprint arXiv:2310.03094*.

Duzhen Zhang, Yahan Yu, Chenxing Li, Jiahua Dong, Dan Su, Chenhui Chu, and Dong Yu. 2024. Mm-llms: Recent advances in multimodal large language models. *arXiv preprint arXiv:2401.13601*.

Jieyu Zhang, Ranjay Krishna, Ahmed H Awadallah, and Chi Wang. 2023. Ecoassistant: Using llm assistant more affordably and accurately. *arXiv preprint arXiv:2310.03046*.

Mengjie Zhao, Fei Mi, Yasheng Wang, Minglei Li, Xin Jiang, Qun Liu, and Hinrich Schütze. 2021. Lmturk: Few-shot learners as crowdsourcing workers in a language-model-as-a-service framework. *arXiv preprint arXiv:2112.07522*.

Jianyi Zhou, Feng Li, Jinhao Dong, Hongyu Zhang, and Dan Hao. 2020. Cost-effective testing of a deep learning model through input reduction. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*.

Banghua Zhu, Ying Sheng, Lianmin Zheng, Clark Barrett, Michael I Jordan, and Jiantao Jiao. 2023. On optimal caching and model multiplexing for large model inference. *arXiv preprint arXiv:2306.02003*.

## Appendix

| **Provider** | **LLM** | **Input Cost** | **Output Cost** |
|---|---|---|---|
| OpenAI | gpt-4 | $30.0 | $60.0 |
| | gpt-4-turbo | $10.0 | $30.0 |
| | gpt-3.5-turbo-1106 | $1.00 | $2.00 |
| Anthropic | Claude-2.0 | $11.02 | $32.68 |
| | Claude-instant-1.2 | $1.63 | $5.51 |
| AI21 | Jurassic-2 Ultra | $15.0 | $15.0 |
| | Jurassic-2 Mid | $10.0 | $10.0 |
| | Jurassic-2 Light | $3.00 | $3.00 |
| Textsynth | M2M100 1.2B | $0.15 | $3.00 |
| | GPT-J 6B | $0.20 | $5.00 |
| | Falcon 7B | $0.20 | $5.00 |
| | Mistral 7B | $0.20 | $2.00 |
| | Llama2 7B | $0.20 | $2.00 |
| | Flan-T5-XXL | $0.20 | $5.00 |
| | Falcon 40B | $3.30 | $10.00 |
| Cohere | command | $1.00 | $2.00 |
| | command-light | $0.30 | $0.60 |
| Baidu | Llama-2-13B-Chat | ¥6.00 | ¥6.00 |
| | Llama-2-70B-Chat | ¥35.0 | ¥35.0 |
| | ERNIE-Bot 4.0 | ¥150 | ¥300 |
| | ChatGLM2-6B-32K | ¥4.00 | ¥4.00 |
| | Llama-2-7B-Chat | ¥4.00 | ¥4.00 |
| | ERNIE-Bot | ¥12.0 | ¥12.0 |
| | BLOOMZ-7B | ¥4.00 | ¥4.00 |
| | ERNIE-Bot-turbo-0922 | ¥8.00 | ¥12.0 |

Table 1: Price list of different LLM services. The cost is priced per 1 million tokens. Typically, the cost of invoking LLM services consists of two components: (1) input cost (proportional to the length of the input prompt), (2) output cost (proportional to the length of the generated sequence). Note that Baidu's LLM services are priced in Chinese Yuan (¥), while the other LLM services are priced in US Dollars ($). The data updated to May 2024.