SkyLLM: Cross-LLM-APIs Federation for Cost-effective Query Processing

Heng Zhao[1], **Yifei Zhu**[1,2]
[1]UM-SJTU Joint Institute, Shanghai Jiao Tong University, China
[2]Cooperative Medianet Innovation Center, Shanghai Jiao Tong University, China
{zhao-heng, yifei.zhu}@sjtu.edu.cn

Abstract

Large language models (LLMs) have demonstrated exceptional capabilities across a wide range of tasks, from text generation to complex problem-solving. LLM APIs provide easy access to these models by streamlining deployment and usage. Combining LLMs with complementary strengths has been shown to yield substantial performance gains over a monolithic LLM. However, invoking a fixed set of LLM APIs for each query incurs higher API costs and increased inference latency. To address these limitations, we propose SkyLLM, a system composed of a set of estimators and an API selector, which federates multiple LLM APIs and dynamically assigns a non-empty subset of these APIs to each query prior to inference under cost and latency budgets. The selected subset consists of either a single LLM or multiple LLMs. A single LLM efficiently handles simple queries at low cost, whereas multiple LLMs are employed for more complex queries to overcome performance limitations. We evaluate SkyLLM against individual LLMs and representative ensemble LLM methods from the literature. SkyLLM achieves the highest accuracy under a high budget. It can also be cost-effective, matching the most accurate individual LLM while cutting costs by 67.8%.

1 Introduction

Large language models (LLMs) have become the dominant force in natural language processing (NLP) in recent years. Their impressive capabilities have spanned a wide range of applications e.g., question answering, code generation, text summarization, and open-ended conversation. This swift progression has been propelled by breakthroughs in model frameworks, particularly the Transformer framework (Vaswani et al., 2017), in addition to enhancements in scaling up data and training infrastructure (Ong et al., 2024).

In many cases, the massive size of modern LLMs makes local deployment both computationally and financially prohibitive, especially as many state-of-the-art models remain closed-source[1][2]. LLM APIs provide a streamlined solution, enabling users to access these powerful models via cloud-based services. By paying a usage-based fee, users can submit queries and receive responses without the overhead of managing hardware or model weights, thereby lowering the barriers to LLM adoption.

The diversity in data, architectures, and hyperparameters causes different LLMs to exhibit varying strengths and weaknesses across tasks (Jiang et al., 2023). Consequently, ensembling multiple LLMs with complementary advantages can help produce higher-quality outputs. LLM ensemble can be broadly categorized into three types: cascading, routing, and fusion. LLM cascading sequentially invokes increasingly powerful models until a satisfactory response is obtained. LLM routing directs each query to the most suitable LLM before inference, typically using a trained router. LLM fusion integrates outputs from multiple models, during or after inference.

Despite their potential advantages, each of these ensemble approaches has inherent limitations. Cascading and routing methods ultimately depend on the final output of a single LLM, thereby constraining their accuracy to the performance of the best model in the pool. Moreover, since models are invoked sequentially, LLM cascading increases inference time, which is a critical factor for user experience, particularly in real-time applications such as chatbots and voice assistants. While previous studies have shown that combining multiple LLMs can significantly enhance accuracy compared to individual models (Tekin et al., 2024), fusion-based methods generally invoke a fixed set of LLMs for

[1]https://openai.com
[2]https://www.anthropic.com/

Findings of the Association for Computational Linguistics: ACL 2025, pages 20864–20873

each query (Yu et al., 2024), resulting in higher API costs and longer inference times.

To address these limitations, we propose a dynamic ensemble approach. Given a set of LLMs, we dynamically assign a non-empty subset of these models to each query, based on query-specific characteristics such as subject and difficulty. This subset consists of a single LLM or multiple LLMs. For more complex queries, selecting multiple LLMs can improve accuracy at the cost of increased API expense and latency. Conversely, for simpler queries, opting for a single, well-suited LLM achieves cost efficiency and low latency. This flexible strategy aims to balance performance gains against the overhead of higher costs and longer inference time.

However, the dynamic ensemble framework introduces several challenges. First, the combinatorial nature of choosing subsets expands the search space exponentially as the number of models increases, especially when each query may require a distinct subset. Second, because the subset must be selected prior to inference, accurately predicting its performance (e.g., cost, latency, accuracy) is non-trivial. Moreover, we explicitly treat cost and latency as constraints in the selection process to achieve a balanced trade-off, which further complicates the selection process.

In this paper, we introduce SkyLLM, a unified system that dynamically assigns a non-empty subset of federated LLM APIs to each query prior to inference. SkyLLM is designed to maximize accuracy while adhering to cost and latency constraints. To implement SkyLLM, we leverage a set of estimators (cost, latency, and accuracy), derived from an open dataset containing responses and metadata (e.g., inference time and token count) from a diverse set of LLMs across a large collection of training queries, to assess the expected performance of each subset. SkyLLM then formulates the selection task as a multi-dimensional group knapsack problem. To reduce the exponential search space, SkyLLM employs Pareto dominance pruning to eliminate strictly inferior subsets and then applies dynamic programming to determine the optimal subset of federated LLM APIs for each query.

We evaluate SkyLLM against individual component LLMs and representative LLM ensemble methods from the literature. Under a high budget, SkyLLM achieves the highest accuracy. Furthermore, it demonstrates cost-effectiveness, attaining accuracy comparable to the most accurate individual LLM while reducing costs by 67.8%.

In summary, our contributions are as follows:

- We introduce the first system that dynamically assigns a non-empty subset(either single LLM or multiple LLMs) of federated LLM APIs for each query, effectively balancing accuracy improvements with reductions in both cost and latency.
- SkyLLM leverages a set of estimators to assess the performance of each subset and formulates the selection problem as a multi-dimensional group knapsack problem. It then employs Pareto dominance pruning to reduce the exponential search space and applies a dynamic programming solver to determine the optimal subset for each query.
- Under a high budget, SkyLLM achieves the highest accuracy compared to individual LLMs and other representative ensemble methods. Additionally, it demonstrates cost-effectiveness by matching the accuracy of the most accurate individual LLM while reducing costs by 67.8%.

2 Background

2.1 LLM as a Service

Large Language Model (LLM) API endpoints simplify access to generative AI by abstracting the complexities of model deployment and maintenance. A single API can support a wide range of tasks, including chat, summarization, and code generation. Typical interactions involve submitting a prompt and receiving a model-generated response.

For proprietary models, such as those offered by OpenAI, access is exclusively provided through official APIs, ensuring compatibility with the latest updates and provider policies. In contrast, open-source models are accessible via third-party providers such as Together AI[3] and DeepInfra[4], which offer APIs that streamline model deployment and usage.

While API-based access eliminates the operational overhead of LLM deployment, it introduces challenges related to API costs and inference latency. SkyLLM addresses these challenges by dynamically selecting subsets of federated LLM APIs tailored to cost and latency constraints, effectively balancing accuracy with reduced overhead.

[3] https://www.together.ai
[4] https://deepinfra.com/

2.2 LLM ensemble

As discussed in Introduction, LLM ensemble strategies can be broadly grouped into three categories (cascading, routing, and fusion). We now highlight additional details—particularly for LLM fusion—and compare them from three perspectives: accuracy, latency, and cost.

Accuracy. While cascading and routing ultimately rely on a single chosen model, fusion integrates outputs from multiple LLMs, demonstrating greater potential for overcoming accuracy limitations. Token-level fusion (during inference) leverages log probability information at each generation step to enhance performance (Yu et al., 2024), while example-level fusion compares and merges responses from multiple LLMs post-inference (Jiang et al., 2023). Example-level fusion operates at a coarse granularity and benefits more from larger ensemble sizes (Li et al., 2024a), whereas token-level fusion offers a fine-grained alternative, achieving accuracy gains with a smaller ensemble (Yu et al., 2024).

Latency. Cascading sequentially invokes multiple models until a satisfactory response is found, potentially increasing inference time for complex queries. Routing introduces a small delay due to the routing step (Lu et al., 2024a), while fusion is constrained by the slowest model in parallel invocation.

Cost. Ensembling multiple LLMs generally raises API usage costs. Fusion is the most expensive, as it invokes multiple models for every query. In contrast, cascading and routing may only call more powerful (and expensive) models as needed. Cascading may invoke smaller models first and then discard their unsatisfactory responses, resulting in moderate costs. Routing is often the most cost-effective, as it selects only the most suitable single model per query.

SkyLLM balances cost, latency, and accuracy by dynamically routing queries to one or multiple models based on query complexity. When combining multiple LLMs, it can leverage log probability information for fine-grained token-level fusion, reducing both ensemble size and API calls.

3 System design for SkyLLM

3.1 System overview

Consider a set of M test queries, denoted as $\mathcal{Q} = \{q_1, q_2, \ldots, q_M\}$, and a set of N large language models (LLMs), denoted as $\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$, accessible via APIs. The set

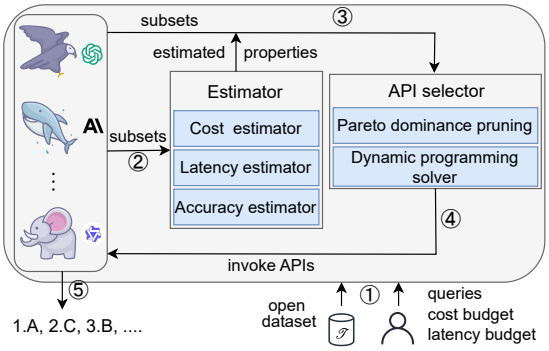


Figure 1: System overview

of possible non-empty subsets $s \subseteq \mathcal{M}$ for a given query q is denoted as $\mathcal{S}_q$, with its size given by Z_q. Initially, $Z_q = 2^N - 1$, for all $q \in \mathcal{Q}$. The objective is to assign the most suitable subset $s \in \mathcal{S}_q$ to each query prior to inference, maximizing accuracy while adhering to cost and latency constraints.

We focus on tasks where correctness is explicitly defined, such as multiple-choice questions (MCQs). Figure 1 provides an overview of SkyLLM, which consists of a set of estimators and an API selector. Given user-specified cost and latency budgets, SkyLLM leverages estimators to predict the properties(cost, latency and accuracy) of each subset for a given test query. Estimators are built using an open dataset $\mathcal{T}$ containing responses and metadata (e.g., inference time and token count) from all LLMs in $\mathcal{M}$ across a large set of training queries.

In API selector, we formulate this selection problem as a multi-dimensional group knapsack problem. To reduce the exponential search space, we apply Pareto dominance pruning. Then we use a dynamic programming solver to determine the optimal subset for each query. Finally, the selected APIs are invoked to generate the response.

3.2 Estimator

SkyLLM implements three estimators: an accuracy estimator, a cost estimator, and a latency estimator. These estimators predict the performance of subsets for every query, aiding the API selector in making optimal assignments.

Accuracy estimator. The accuracy estimator predicts the accuracy of subsets for a given test query by evaluating their performance on similar queries. For a query q, we first retrieve I similar queries from $\mathcal{T}$ using their embeddings. Specifically, we compute the embedding vector $\mathbf{e}_q$ with a pre-trained model and measure similarity via L2 dis-

tance. The top I closest queries are selected as reference samples.

For multi-LLM subsets, responses are aggregated using majority voting or probability-based fusion. If APIs provide log probability values for possible answers, we average these values and select the answer with the highest probability. Otherwise, majority voting determines the final response.

The accuracy of each subset $s \in \mathcal{S}_q$ on the selected I similar queries is computed and denoted as $a_q^s \in [0, 1]$, representing the estimated accuracy of subset s on query q.

Cost estimator. The cost estimator predicts the API cost for each subset given a test query. Inference cost consists of two components: input (prompt) cost and output cost, both determined by token count and API pricing[5][6][7].

The price per input and output token for m is denoted as p_m^{in} and p_m^{out}, respectively. The number of input tokens, T_q^{in}, is determined using the model's tokenizer, while the number of output tokens, T_q^{out}, is either predefined or predicted via a multilayer perceptron (MLP) based on the query embedding and prompt length if it is not fixed.

The total inference cost for query $q \in \mathcal{Q}$ using subset $s \in \mathcal{S}_q$ is given by:

$$c_q^s = \sum\nolimits_{m \in s} \left(p_m^{\text{in}} \cdot T_q^{\text{in}} + p_m^{\text{out}} \cdot T_q^{\text{out}} \right) \quad (1)$$

Latency estimator. The latency estimator predicts the inference time, defined as the duration from API invocation to response reception for a given model and test query. For a query q, the inference latency of a subset s is determined by the maximum inference time among all models in s, assuming parallel API invocation. The estimated latency is given by:

$$t_q^s = \max\nolimits_{m \in s} L(m, q) \quad (2)$$

where $L(m, q) \in \mathbb{R}$ denotes the estimated latency of LLM m on query q. This latency is predicted by an MLP trained on dataset $\mathcal{T}$, using query embeddings as well as the number of prompt and output tokens as input features.

3.3 API selector

The API selector determines the optimal subset assignment for each query based on the estimated cost, latency, and accuracy. First, it formulates the assignment problem as a multi-dimensional group knapsack problem. Then, it employs Pareto dominance pruning to reduce the exponential search space and applies dynamic programming to solve the problem. Finally, the selected APIs are invoked to generate the response.

Formulation. The objective is to maximize accuracy while keeping cost within B_c and latency within B_t. We model this as a multi-dimensional group knapsack problem, a variant of the classical knapsack problem in which items are divided into groups, and we can select at most one item from each group under multiple resource constraints.

For M test queries, each query q_i has a set of non-empty subsets $\mathcal{S}_{q_i}$ of size Z_{q_i}. Each query forms a group, and each subset is treated as an item. The value of an item is its estimated accuracy, subject to the cost and latency constraints. The optimal subset selection for every query is formulated as:

$$\max_{\mathbf{x}} \quad \sum_{i=1}^{M} \sum_{j=1}^{Z_{q_i}} a_{q_i}^{s_j} \, x_i^j \quad (3)$$

$$\text{s.t.} \quad \sum_{i=1}^{M} \sum_{j=1}^{Z_{q_i}} c_{q_i}^{s_j} \, x_i^j \leq B_c \quad (4)$$

$$\sum_{i=1}^{M} \sum_{j=1}^{Z_{q_i}} t_{q_i}^{s_j} \, x_i^j \leq B_t \quad (5)$$

$$\sum_{j=1}^{Z_{q_i}} x_i^j \leq 1 \quad \forall i \in [M] \quad (6)$$

$$x_i^j \in \{0, 1\} \quad \forall i \in [M], j \in [Z_{q_i}] \quad (7)$$

where $[X] = \{1, 2, \ldots, X\}$. The vector $\mathbf{x} = [\mathbf{x_1}, \ldots, \mathbf{x_M}]$ concatenates sub-vectors $\mathbf{x_i}$, each of length Z_{q_i}. Here, $\mathbf{x_i} = (x_i^1, \ldots, x_i^{Z_{q_i}}) \in \{0, 1\}^{Z_{q_i}}$ indicates the selected subset (if any) for query q_i. Constraints (4) and (5) enforce cost and latency limits, respectively, while constraint (6) ensures that at most one subset is chosen per query.

Pareto dominance pruning. During selection, the $2^N - 1$ possible subsets make exhaustive search infeasible. To reduce the exponential search space, we apply Pareto dominance pruning, discarding a subset if another exists that is equal or superior in all relevant metrics (e.g., accuracy, cost, and latency) and strictly better in at least one.

To evaluate the efficiency of this approach, we use data from the HELM project (Liang et al., 2023), which queries multiple models on various

[5] https://openai.com/api/pricing
[6] https://www.together.ai/pricing
[7] https://deepinfra.com/pricing

datasets via API. This dataset includes model outputs, question-answer pairs, and metadata such as inference time and token count. We apply the pruning strategy to the MMLU dataset (Hendrycks et al., 2021), significantly reducing the search space, as shown in Table 1.

N	Average # of subsets per query	
	Before Pruning	After Pruning
10	1,023	7
13	8,191	9
15	32,767	12

Table 1: Comparison of average number of subsets (Z_q) per query before and after pruning.

Algorithm. We solve the multi-dimensional group knapsack problem via a dynamic programming (DP) approach. Let $dp[c][l]$ be the highest achievable accuracy under cost $c \leq B_c$ and latency $l \leq B_t$. We initialize $dp[c][l]$ to zero. For each query q_i, we enumerate each subset $s_j \in \mathcal{S}_{q_i}$, updating:

$$dp[c][l] \leftarrow \max\left(dp[c][l],\, dp[c - c_{q_i}^{s_j}][l - t_{q_i}^{s_j}] + a_{q_i}^{s_j}\right)$$

whenever $c \geq c_{q_i}^{s_j}$ and $l \geq t_{q_i}^{s_j}$. We record chosen subsets in a backtracking table. After processing all queries, we trace back from (B_c, B_t) to recover each query's selected subset, yielding $\mathbf{x_1}, \ldots, \mathbf{x_M}$.

Complexity analysis. The algorithm runs in $\mathcal{O}(MB_cB_tZ_m)$ time, where $Z_m = \max_i Z_{q_i}$ is the maximum number of subsets across all queries. This complexity arises from iterating over each subset for every cost and latency state for every query. The space complexity is $\mathcal{O}(MB_cB_t)$, primarily due to the backtracking table. If the estimators are perfectly accurate, this approach guarantees an optimal solution within the specified budgets.

Algorithm 1: Dynamic programming for subset selection

Input : Cost budget B_c, latency budget B_t; queries $\mathcal{Q}$; subset sets $[\mathcal{S}_{q_i}]_{i\in[M]}$
Output : $\mathbf{x} = [\mathbf{x_1}, \ldots, \mathbf{x_M}]$

$dp \leftarrow 0^{(B_c+1)\times(B_t+1)}$;
$bt \leftarrow -1^{M\times(B_c+1)\times(B_t+1)}$;
for $i \leftarrow 1$ **to** M **do**
  Retrieve($\{c_{q_i}^{s_j}, t_{q_i}^{s_j}, a_{q_i}^{s_j}\}$ *for all* $s_j \in \mathcal{S}_{q_i}$);
  $prev\_dp \leftarrow dp$;
  for $s_j \in \mathcal{S}_{q_i}$ **do**
    for $c \leftarrow c_{q_i}^{s_j}$ **to** B_c **do**
      for $l \leftarrow t_{q_i}^{s_j}$ **to** B_t **do**
        $val \leftarrow prev\_dp[c - c_{q_i}^{s_j}][l - t_{q_i}^{s_j}] + a_{q_i}^{s_j}$;
        if $val > dp[c][l]$ **then**
          $dp[c][l] \leftarrow val$;
          $bt[i][c][l] \leftarrow j$;
$\mathbf{x_i} \leftarrow \mathbf{0}^{Z_{q_i}}$ for $i = 1, \ldots, M$;
$(c, l) \leftarrow (B_c, B_t)$;
for $i \leftarrow M$ **to** 1 **do**
  $j \leftarrow bt[i][c][l]$;
  if $j \neq -1$ **then**
    $x_i^j \leftarrow 1$;
    $c \leftarrow c - c_{q_i}^{s_j}$;
    $l \leftarrow l - t_{q_i}^{s_j}$;
return $\mathbf{x} = [\mathbf{x_1}, \ldots, \mathbf{x_M}]$;

4 Implementation

We implement the SkyLLM system in Python. This section details the implementation of each component, including the LLMs, estimators, and API selector.

LLMs. We adopt LLMs from OpenAI (GPT-4o, GPT-4o-mini, and GPT-3.5-Turbo) and Qwen (Qwen-max, Qwen-plus, and Qwen-turbo). Both OpenAI and Qwen APIs return log probabilities for a limited number of top-ranked tokens at each generated token position: OpenAI provides up to 20 tokens, while Qwen provides up to 5. We average these probabilities across selected models to obtain a fused distribution over possible answers and select the answer with the highest average probability. If the desired answer is not among these tokens, its probability is set to 0.

Estimators. For the accuracy estimator, we employ the sentence transformer *all-MiniLM-L6-v2* from Hugging Face[8] to map each query to a 384-dimensional dense vector. To efficiently search for similar queries in dataset $\mathcal{T}$, we use Faiss[9] to retrieve I similar queries for each query.

In the cost estimator, model-specific API pricing is obtained from respective providers. The number of prompt tokens is estimated using Tiktoken[10]. We apply SkyLLM to the MMLU dataset (Hendrycks et al., 2021). Since MMLU consists of multiple-

[8] https://huggingface.co
[9] https://github.com/facebookresearch/faiss
[10] https://github.com/openai/tiktoken

choice questions, the number of output tokens is fixed at one.

In the latency estimator, inference time is predicted using a multilayer perceptron (MLP) regressor consisting of a two-layer feedforward neural network with 50 neurons per hidden layer. The model takes the model ID, query embedding, number of prompt tokens, and number of output tokens as input features.

After evaluating our estimators on the MMLU dataset, we observe that the accuracy estimator achieves a cross-entropy loss of 0.44. The average cost per query is $\$7.3 \times 10^{-4}$. For cost estimation, the mean squared error (MSE) is $\$3.1 \times 10^{-10}$, indicating highly precise cost predictions. Meanwhile, since the maximum number of output tokens is restricted to one, the overall inference time remains relatively short, with a mean inference time of 0.54 s. Consequently, even modest network fluctuations result in a comparatively large estimation error, leading to an MSE of 0.16 s for the latency estimator.

API selector. The API selector first enumerates all $2^N - 1$ possible subsets for each query, estimates their cost, latency, and accuracy, and applies Pareto dominance pruning to discard inferior subsets. After pruning, we solve the multi-dimensional group knapsack problem via dynamic programming (DP). Since DP uses cost and latency as discrete indices, we convert floating-point cost and latency values to integers by multiplying by enlargement factors (α_c and α_t) and applying the ceiling function. Larger α_c and α_t values provide finer granularity but increase DP's time and space complexity, as the budgets are also enlarged. To mitigate excessive memory usage, we employ LZ4[11], a high-speed lossless compression algorithm that reduces memory consumption during runtime. Once DP completes, we retrieve the chosen subsets, invoke APIs, and ensemble outputs for each query, ultimately returning the final answer.

5 Experiment

5.1 Evaluation setup

Dataset. We evaluate SkyLLM on the MMLU dataset (Hendrycks et al., 2021), which consists of 14,042 questions spanning 57 subjects. To maintain the original query distribution, we randomly partition the dataset into training and test sets at an 8:2 ratio based on subjects.

[11] https://lz4.org/

Benchmarks. To evaluate the efficiency of our system, we compare it not only against individual component LLMs but also with established ensemble methods from the literature. Specifically, we consider FrugalGPT (Chen et al., 2023), RouteLLM (Ong et al., 2024), and LLM-TOPLA (Tekin et al., 2024), which are representative approaches for LLM cascading, routing, and fusion, respectively.

FrugalGPT employs a router to select component LLMs in the cascade and utilizes a scoring function to assess output reliability. If the output meets a predefined reliability threshold, it is accepted; otherwise, the next model in the cascade is queried. RouteLLM dynamically routes queries between a strong and a weak LLM, balancing cost and response quality based on query complexity. In our setup, we designate Qwen-max as the strong LLM and Qwen-turbo as the weak LLM, based on their respective cost and accuracy. LLM-TOPLA integrates responses from multiple LLMs at the token level, selecting and combining the most diverse and complementary outputs to generate an optimal response. In our configuration, LLM-TOPLA ensembles output from all individual LLMs.

Metrics. We evaluate our system based on *cost* (C), *latency* (L), and *mean accuracy* (MA). Specifically, *cost* refers to the total API cost for processing all test queries, while *latency* represents the total inference time. To assess the system's ability to handle challenging queries, we introduce *Hardness-Weighted Accuracy (HA)*, defined as:

$$\text{HA} = \frac{\sum_h h \cdot Acc_h}{\sum_h h} \times 100\% \tag{8}$$

where $h \in \mathbb{N}$ denotes query hardness, defined as the number of individual LLMs that fail to answer the query correctly, and Acc_h is the mean accuracy of the ensemble system on queries of hardness h. Although the number of queries varies across hardness levels, we do not incorporate query counts into calculation. Instead, we weigh by hardness directly to emphasize performance on difficult queries.

Hyperparameter settings. In the accuracy estimator, the parameter I is set to one-tenth of the training set size. The latency estimator is trained using the Adam optimizer for up to 1,000 iterations. Additionally, the enlargement factors α_c and α_t are set to 10,000 and 10, respectively, in the API selector to balance granularity and computational complexity.

5.2 Overall performance

	C($)	L(s)	MA(%)	HA(%)
qwen-max	5.84	1399	85.0	40.1
qwen-plus	0.23	1285	83.8	36.9
qwen-turbo	0.09	1363	76.0	24.1
gpt-4o	4.90	1549	82.2	36.6
gpt-4o-mini	0.29	1771	58.0	7.7
gpt-3.5-turbo	0.98	1742	69.9	21.2
FrugalGPT	6.78	2001	86.5	44.5
RouteLLM	3.18	**1370**	79.6	29.5
TOPLA	12.33	1771	86.3	39.9
SkyLLM-H	5.37	1601	**87.1**	**46.0**
SkyLLM-L	**1.88**	1559	84.5	40.6

Table 2: Comparison with individual LLMs and representative ensemble methods in LLM cascading, routing and fusion

Table 2 compares cost, latency, mean accuracy, and hardness-weighted accuracy across SkyLLM, individual LLMs, and other ensemble methods. SkyLLM-H represents the high-performance version with a larger cost budget, while SkyLLM-L is the cost-effective version with a lower budget.

Among individual LLMs, Qwen-max achieves the highest accuracy. Without exceeding its cost, SkyLLM-H attains the highest mean and hardness-weighted accuracy, outperforming both individual LLMs and representative ensemble methods. This demonstrates its strong overall performance and effectiveness in handling complex queries. Meanwhile, SkyLLM-L achieves accuracy comparable to Qwen-max while reducing costs by 67.8%. Additionally, SkyLLM-L offers cost advantages over other ensemble methods.

However, SkyLLM provides limited latency benefits compared to individual LLMs. A stricter latency budget reduces its accuracy, as shown in the sensitivity analysis. This is likely because, although LLMs are invoked in parallel, the slowest model determines overall efficiency. Nonetheless, SkyLLM retains certain advantages over other ensemble methods, possibly due to its ability to assign simple queries to individual LLMs, thereby reducing inference time.

5.3 Sensitivity analysis

Impact of parameter I. Parameter I represents the number of similar queries whose accuracy is used to estimate the accuracy of subsets for a given test query. To account for variations in training set size, we define R_I as the ratio of I to the training set size. As shown in Figure 2(a), mean accuracy initially increases with R_I. This trend occurs because larger values of I enable a more accurate and unbiased evaluation of subsets by leveraging a broader set of similar queries. However, when R_I exceeds 0.1, accuracy begins to decline, likely due to the inclusion of unrelated queries as I increases, given that MMLU spans a wide range of topics.

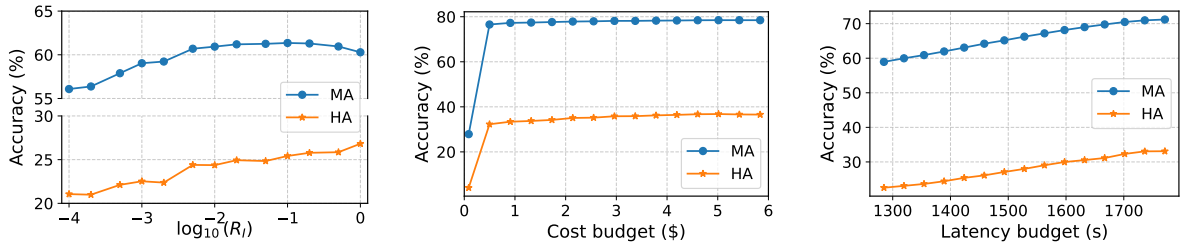
In contrast, hardness-weighted accuracy continues to improve. This may be attributed to the fact that subsets with higher overall performance tend to possess stronger problem-solving abilities, making them more effective in handling difficult queries.

Impact of cost budget B_c. Figure 2(b) illustrates SkyLLM's performance as the cost budget increases, ranging from the minimum to the maximum cost of individual LLMs. For each budget, we average results over four latency budgets, evenly spaced between their minimum and maximum values. We observe that SkyLLM converges quickly, with minimal gains in both mean accuracy and hardness-weighted accuracy beyond a moderate cost budget. A key factor is the strong performance of Qwen-plus, which handles most queries at low cost. Further accuracy improvements are limited, as the remaining queries are more difficult and require more powerful (and expensive) LLM combinations.

Impact of latency budget B_t. Figure 2(c) shows a steady increase in both mean accuracy and hardness-weighted accuracy as the latency budget expands from minimum to maximum. For each latency budget, we similarly average results over four different cost budgets. This upward trend suggests that allowing more time for inference particularly benefits accuracy improvements. While models like Qwen-plus achieve comparable accuracy at a fraction of the cost (e.g., tens of times lower), their latency advantage is less pronounced. Consequently, under strict latency constraints, these models cannot be fully leveraged, preventing SkyLLM from converging efficiently at low latency budgets.

6 Related Work

LLM cascading. LLM cascading sequentially invokes increasingly large and powerful LLMs until a satisfactory response is obtained (Chen et al., 2023; Yue et al., 2024; Aggarwal et al., 2024; Wang et al., 2023; Chen et al., 2024). Yue et al. (Yue et al., 2024) propose using the consistency of a weaker



(a) Impact of parameter I (b) Impact of cost budget B_c (c) Impact of latency budget B_t

Figure 2: Sensitivity analysis

LLM's answer as an indicator of question difficulty and whether to query a stronger LLM. Similarly, Aggarwal et al. (Aggarwal et al., 2024) employ self-verification in smaller models to determine whether to route queries to a larger model. Despite its potential benefits, cascading significantly increases inference time due to sequential invocation, and its accuracy remains constrained by the individual models in the chain. In contrast, SkyLLM introduces lower latency through its routing step and parallel invocation while achieving higher accuracy by dynamically combining multiple LLMs.

LLM routing. LLM routing directs each query to the most suitable model before inference (Srivatsa et al., 2024; Shnitzer et al., 2023; Lu et al., 2023, 2024b; Šakota et al., 2024; Ong et al., 2024). Unlike cascading, which sequentially invokes models, routing employs a pre-inference decision mechanism. Shnitzer et al. (Shnitzer et al., 2023) propose a routing algorithm that leverages benchmark datasets to train binary classifiers, enabling the selection of the most suitable LLM for a given task by learning model strengths and weaknesses from past evaluations. Sakota et al. (Šakota et al., 2024) introduce the FORC framework, which assigns input queries to the most appropriate language model by predicting performance via a meta-model and estimating cost through LLM API pricing, optimizing the trade-off between cost and accuracy. Unlike these approaches, our system flexibly routes queries to either a single LLM or a combination of LLMs, demonstrating greater potential for handling complex queries.

LLM fusion. LLM fusion combines the outputs of multiple LLMs, either during or after inference, to generate a more robust final response. Fusion during inference applies ensembling techniques at each step of the decoding process (Yu et al., 2024; Li et al., 2024b; Mavromatis et al., 2024; Xu et al., 2024; Huang et al., 2024; Hoang et al., 2024). Yu et al. (Yu et al., 2024) treat token generation as a classification task, enabling token-level ensembling and mitigating the error snowball effect during inference. In fusion after inference, models generate responses independently, which are then aggregated into a final output (Jiang et al., 2023; Wang et al., 2024; Chen et al., 2025; Schoenegger et al., 2024). Jiang et al. (Jiang et al., 2023) ensemble multiple LLMs by ranking subset outputs using a pairwise comparison model and generating a final response through generative fusion, leveraging the strengths of top-ranked outputs.

While model fusion enhances accuracy, invoking a fixed set of LLMs incurs high API costs and inference time. In contrast, SkyLLM offers greater flexibility by assigning a single LLM to easy queries and multiple LLMs to difficult queries, achieving high accuracy while maintaining relatively low cost and inference latency.

7 Conclusion

In this paper, we investigate the query handling problem under cost and latency constraints. We propose SkyLLM, a system composed of a set of estimators and an API selector, which federates multiple LLM APIs and dynamically assigns a non-empty subset of these APIs to each query prior to inference. The selected subset consists of either a single LLM or multiple LLMs. A single LLM efficiently handles simple queries at low cost, whereas multiple LLMs are employed for more complex queries to overcome performance limitations. We evaluate SkyLLM against individual component LLMs and representative LLM ensemble methods from the literature. SkyLLM achieves the highest accuracy under a high budget. It can also be cost-effective, matching the most accurate individual LLM while cutting costs by 67.8%.

8 Limitations

When combining multiple LLMs, we utilize log probability information. However, many APIs do not provide token log probability data for token-level fusion, particularly third-party providers such as TogetherAI and DeepInfra. As a result, we must employ majority voting to combine these models, which typically benefits from a larger ensemble size but may significantly increase API costs. A potential solution is to expand the model pool to include more diverse and cost-effective LLMs, a strategy demonstrated to be effective through simulations using data from the HELM project (Liang et al., 2023).

9 Acknowledgement

This work is supported by the National Key R&D Program of China (Grant No. 2023YFB2704400) and the Fundamental Research Funds for the Central Universities. Yifei Zhu is the corresponding author.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. 2024. AutoMix: Automatically Mixing Language Models. *arXiv preprint*. ArXiv:2310.12963 [cs].

Boyuan Chen, Mingzhi Zhu, Brendan Dolan-Gavitt, Muhammad Shafique, and Siddharth Garg. 2024. Model Cascading for Code: Reducing Inference Costs with Model Cascading for LLM Based Code Generation. *arXiv preprint*. ArXiv:2405.15842 [cs].

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei A Zaharia, and James Y Zou. 2025. Are more llm calls all you need? towards the scaling properties of compound ai systems. *Advances in Neural Information Processing Systems*, 37:45767–45790.

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 [cs].

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. *arXiv preprint*. ArXiv:2009.03300 [cs].

Hieu Hoang, Huda Khayrallah, and Marcin Junczys-Dowmunt. 2024. On-the-Fly Fusion of Large Language Models and Machine Translation. *arXiv preprint*. ArXiv:2311.08306.

Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Bing Qin, and Ting Liu. 2024. Ensemble Learning for Heterogeneous Large Language Models with Deep Parallel Collaboration. *arXiv preprint*. ArXiv:2404.12715.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, Toronto, Canada. Association for Computational Linguistics.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024a. More agents is all you need. *Preprint*, arXiv:2402.05120.

Tianlin Li, Qian Liu, Tianyu Pang, Chao Du, Qing Guo, Yang Liu, and Min Lin. 2024b. Purifying Large Language Models by Ensembling a Small Language Model. *arXiv preprint*. ArXiv:2402.14845.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic Evaluation of Language Models. *arXiv preprint*. ArXiv:2211.09110 [cs].

Jinliang Lu, Ziliang Pang, Min Xiao, Yaochen Zhu, Rui Xia, and Jiajun Zhang. 2024a. Merge, Ensemble, and Cooperate! A Survey on Collaborative Strategies in the Era of Large Language Models. *arXiv preprint*. ArXiv:2407.06089.

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 [cs].

Xiaoding Lu, Zongyi Liu, Adian Liusie, Vyas Raina, Vineet Mudupalli, Yuwen Zhang, and William Beauchamp. 2024b. Blending Is All You Need: Cheaper, Better Alternative to Trillion-Parameters LLM. *arXiv preprint*. ArXiv:2401.02994 [cs].

Costas Mavromatis, Petros Karypis, and George Karypis. 2024. Pack of LLMs: Model Fusion at Test-Time via Perplexity Optimization. *arXiv preprint*. ArXiv:2404.11531 [cs].

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2024. RouteLLM: Learning to Route LLMs with Preference Data. *arXiv preprint*. ArXiv:2406.18665.

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.

Philipp Schoenegger, Indre Tuminauskaite, Peter S. Park, and Philip E. Tetlock. 2024. Wisdom of the Silicon Crowd: LLM Ensemble Prediction Capabilities Rival Human Crowd Accuracy. *arXiv preprint*. ArXiv:2402.19379 [cs].

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 [cs].

KV Aditya Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. 2024. Harnessing the Power of Multiple Minds: Lessons Learned from LLM Routing. *arXiv preprint*. ArXiv:2405.00467.

Selim Furkan Tekin, Fatih Ilhan, Tiansheng Huang, Sihao Hu, and Ling Liu. 2024. LLM-TOPLA: Efficient LLM Ensemble by Maximising Diversity. *arXiv preprint*. ArXiv:2410.03953 [cs].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. 2024. Fusing Models with Complementary Expertise. *arXiv preprint*. ArXiv:2310.01542 [cs].

Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. 2023. Tabi: An Efficient Multi-Level Inference System for Large Language Models. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 233–248, Rome Italy. ACM.

Yangyifan Xu, Jinliang Lu, and Jiajun Zhang. 2024. Bridging the Gap between Different Vocabularies for LLM Ensemble. *arXiv preprint*. ArXiv:2404.09492 [cs].

Yao-Ching Yu, Chun-Chih Kuo, Ziqi Ye, Yu-Cheng Chang, and Yueh-Se Li. 2024. Breaking the Ceiling of the LLM Community by Treating Token Generation as a Classification for Ensembling. *arXiv preprint*. ArXiv:2406.12585 [cs].

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. 2024. Large Language Model Cascades with Mixture of Thoughts Representations for Cost-efficient Reasoning. *arXiv preprint*. ArXiv:2310.03094 [cs].