HAPS: Hierarchical LLM Routing with Joint Architecture and Parameter Search

Zihang Tian[1], **Rui Li**[1], **Jingsen Zhang**[1], **Xiaohe Bo**[1], **Wei Huo**[2], **Xu Chen**[1,*]
[1]Renmin University of China
[2]Wireless Technology Lab, Huawei Technologies Co., Ltd.
{zihangtian, xu.chen}@ruc.edu.cn

Abstract

Large language model (LLM) routing aims to exploit the specialized strengths of different LLMs for diverse tasks. However, existing approaches typically focus only on selecting LLM architectures, treating candidate models as fixed while overlooking parameter adaptation, which can substantially affect task performance. In this paper, we introduce **HAPS**, a hierarchical LLM routing framework that *jointly searches over model architectures and parameters*. Specifically, a high-level router selects among candidate LLM architectures, while a low-level router generates input-conditioned LoRA parameters for the selected architecture, expanding each candidate from a single fixed behavior into a region of reachable behaviors. The low-level router is built on the backbone of the high-level router, so that architecture selection and parameter generation share a common representation of the prompt. We further optimize the whole framework with a reinforcement learning objective. Extensive experiments show that HAPS consistently outperforms strong routing baselines across reasoning- and knowledge-intensive benchmarks, with analyses of its components, design choices, scalability, cost, and mixed open- and closed-source deployment further demonstrating the robustness of these gains. Our code is available at https://github.com/zihangtian/HAPS.

1 Introduction

Large language models (LLMs) have achieved success across a wide range of natural language processing tasks [Brown et al., 2020, Raffel et al., 2020]. However, due to differences in their architectures and parameters, individual LLMs often exhibit distinct strengths and weaknesses. To better leverage the complementary capabilities of different models, LLM routing has recently attracted increasing attention, aiming to dynamically select the most appropriate LLM for different tasks.

LLM routing typically starts by defining a pool of candidate architectures (e.g., Llama [Touvron et al., 2023a,b], Qwen [Bai et al., 2023], Mistral [Jiang et al., 2023]). A classifier is then used to map each input query to the most appropriate architecture in the pool, which subsequently generates the response. The router is usually trained with a supervision signal based on the response correctness. For example, RouteLLM [Ong et al., 2024] trains a binary router that switches between a stronger and a weaker model using preference data and data augmentation. GraphRouter [Feng et al., 2024] constructs a heterogeneous graph over tasks, queries, and LLMs, and formulates routing as inductive edge-attribute prediction. IRT-Router [Song et al., 2025] explicitly models query difficulty and model ability, yielding interpretable routing decisions and improved cold-start performance.

While the above models have shown promising performance, they all route within a fixed pool of candidates: each candidate is used with its parameters unchanged, and since parameters determine how a model behaves, each candidate offers only a single, fixed behavior, i.e., one *point* in the space

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

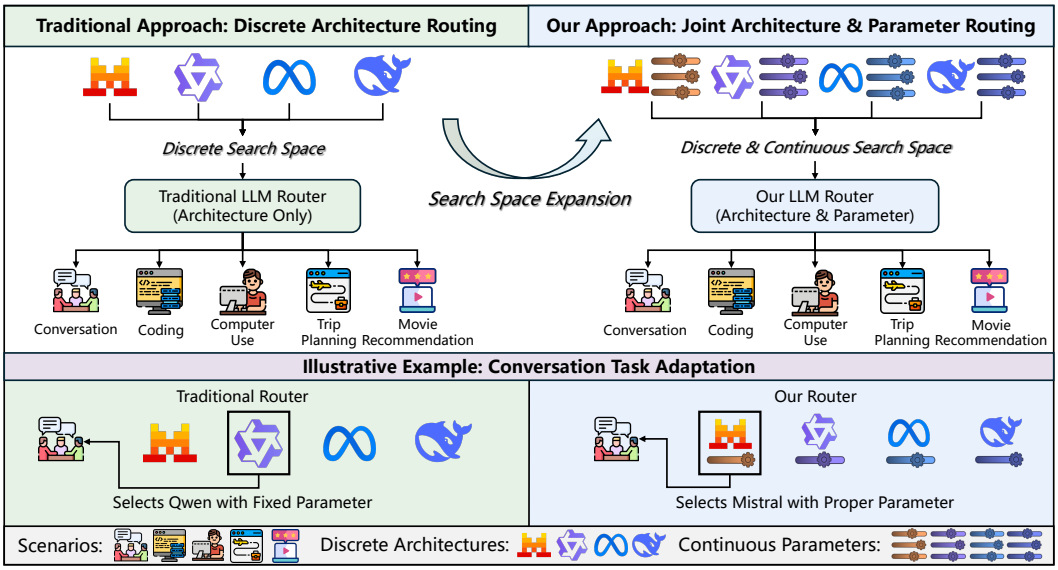

Figure 1: Illustration of the traditional LLM router versus our proposed model. Unlike standard methods that only select architectures, our framework jointly routes both LLM architectures and their specific parameters, so the preferred architecture may change once its parameters can adapt.

of behaviors an LLM can exhibit. Meanwhile, we posit that each query is best served by a set of ideal behaviors, and routing is in essence an attempt to reach one of them. The prevailing paradigm is thus, conceptually, nearest-point retrieval over a handful of fixed points: even a perfect router can only pick the point closest to this set, leaving a gap that no classifier, however well trained, can close.

This raises the question from which our work starts: why should a candidate be a single point? Since behavior comes from parameters, the natural objects to route over are the parameters themselves. Pretrained models are already strong, so we keep them as anchors rather than search parameters from scratch, and expand each anchor into a *region* around it by generating an input-conditioned low-rank update on top of the selected architecture. The pool stays the same size while the behavior it covers grows: each candidate now offers a continuum of input-specific configurations rather than a single one. Architectures differ in how far and in which directions such an update can steer them, so the anchor closest to a query by default need not be the one that can reach closest to it. As illustrated in Figure 1, for a conversation task an architecture-only router may settle on Qwen because it is the best of the fixed candidates, whereas Mistral, weaker in its default configuration, turns out to have the greater potential once its parameters are allowed to adapt. Reaching that better solution requires the two decisions to be made together, so we route over architectures and parameters *jointly*.

Although the above idea is promising, its implementation presents significant challenges. Specifically, existing LLM routers typically employ classifiers to select architectures from a discrete space, whereas LLM parameters reside in a continuous space, making them difficult to directly integrate into these routing models. Moreover, since both architecture and parameter search aim to improve the quality of responses for a given query, they should ideally be guided by shared underlying principles. However, existing methods for architecture routing (e.g., RouteLLM) and parameter tuning (e.g., LoRA [Hu et al., 2021]) follow fundamentally different strategies. Enabling these components to share common knowledge therefore requires deliberate algorithmic and architectural design.

To address the challenges outlined above, we propose a hierarchical LLM router named **HAPS**. At the high level, a Llama-based classifier is used to select the most suitable LLM architecture for each component of the system, such as each agent in a multi-agent pipeline. At the low level, instead of searching for free-form LLM parameters directly, we generate them through a neural network. With this design, both architecture selection and parameter search can be formulated as prediction problems, providing a common basis for combining them within a unified framework for LLM routing. To enable shared knowledge between architecture selection and parameter search, the low-level router reuses part of the high-level router's parameters and extends them. In the training process, we jointly optimize both high- and low-level routers using reinforcement learning, so that both levels are driven by the same task reward. We conduct extensive experiments to demonstrate

the contributions of model components and to evaluate the effectiveness of our model compared to strong routing baselines. The main contributions of this paper are summarized as follows:

- We introduce the idea of joint model architecture and parameter search for LLM routing, which lifts each candidate from a fixed point to a region of reachable behaviors.
- We implement the above idea via a hierarchical framework, designing a parameter generation network that facilitates knowledge sharing between the two levels and leveraging a reinforcement learning objective to enable joint discrete and continuous optimization.
- We conduct extensive experiments across reasoning and knowledge benchmarks, where HAPS outperforms strong routing baselines, together with analyses that validate its key design choices.

2 Preliminaries

To make our presentation clearer, we first introduce the basic concepts of LLM routing. We define a pool of candidate LLMs as $\mathcal{M} = \{m_0, \ldots, m_{K-1}\}$, where each m_k denotes an LLM with a distinct architecture (e.g., Llama or Qwen), and K is the total number of models. Given a system with N components that require routing (e.g., the agents in a multi-agent pipeline, each handling one subtask), the objective is to learn a classifier $\mathbf{r} : \mathcal{P} \to \{0, \ldots, K-1\}^N$ that maps a routing prompt to the indices of the most suitable LLMs in $\mathcal{M}$ for generating accurate responses.

Let θ denote the parameters of the router $\mathbf{r}$. The training objective is to minimize the prediction loss between the output generated by the selected models and the ground-truth label y:

$$\min_{\theta} \mathop{\mathbb{E}}_{(p,\mathbf{x},y)\sim\mathcal{D}} \left[L\left(\mathrm{G}\left(\mathbf{m}_{\mathbf{r}(p;\theta)}, \mathbf{x} \right), y \right) \right], \tag{1}$$

where $\mathcal{D}$ is the data distribution, $p \in \mathcal{P}$ is the routing prompt (e.g., the task description, the role of each component, and the list of candidate LLMs), and $\mathbf{x}=\{x_i\}_{i=0}^{N-1}$ collects the inputs issued to the LLMs, one per component. We let $\mathbf{m}_{\mathbf{r}(p;\theta)}=\{m_{[\mathbf{r}(p;\theta)]_i}\}_{i=0}^{N-1}$ denote the set of LLMs selected by the router $\mathbf{r}$. The function $\mathrm{G}(\mathbf{m}_{\mathbf{r}(p;\theta)}, \mathbf{x})$ represents the system output obtained by applying each input $x_i \in \mathbf{x}$ to its corresponding LLM $m_{[\mathbf{r}(p;\theta)]_i} \in \mathbf{m}_{\mathbf{r}(p;\theta)}$. Here, $L(\cdot,\cdot)$ is a task-specific loss.

Equation (1) has been instantiated in various forms by prior work [Ong et al., 2024, Feng et al., 2024, Song et al., 2025]. However, most existing LLM routing methods focus solely on selecting LLM architectures, neglecting the role of parameter adaptation, an important factor for task performance (as illustrated in Figure 1). To address this limitation, in the following section, we introduce a hierarchical LLM router that treats the parameters of each selected LLM as searchable variables and jointly routes the architectures and parameters to enhance task performance.

3 The HAPS Model

As shown in Figure 2, our model adopts a hierarchical structure: a high-level router selects an LLM architecture, and a low-level router generates parameters for the selected architecture as a low-rank update to its frozen weights. Both levels read the same routing prompt, and they share part of their parameters so that knowledge learned for one transfers to the other. We describe each of them below.

3.1 High-Level Router

The high-level router is responsible for selecting the appropriate LLM architectures. Intuitively, this selection should be guided by the tasks and roles we expect the LLMs to perform. Accordingly, we define p in Equation (1) as a four-part routing prompt, structured as follows: (1) *Role definition* – a brief textual specification of each agent's capabilities and expected behaviors; (2) *Task description* – a concise summary of the overall objective; (3) *Problem description* – a detailed description of the specific problem to be addressed by the LLMs (i.e., $\mathbf{x}$ in Equation (1)); (4) *Model list* – the names or symbolic identifiers of all candidate LLMs (i.e., $\mathcal{M}$). Formally, the high-level router $\mathbf{r}$ is a multi-output classifier implemented using a Llama-based architecture, defined as $\mathbf{z} = \mathbf{r}(p;\theta)$, where $\mathbf{z} = \{z_i\}_{i=0}^{N-1}$ is an N-dimensional vector, with each element indicating the index of the LLM in $\mathcal{M}$ assigned to the corresponding system component (i.e., z_i means that the ith component in the system uses m_{z_i} from $\mathcal{M}$). Here, θ is the learnable parameter vector of the high-level router. All N entries

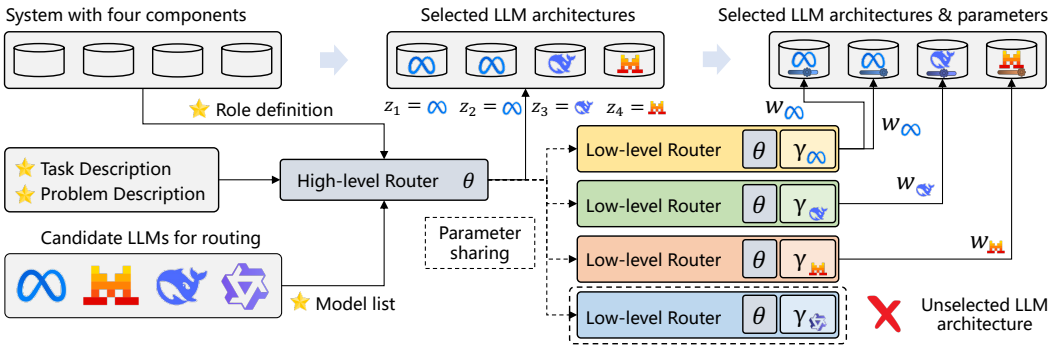

Figure 2: The overall framework of our hierarchical LLM router. The complete routing process includes two phases: (1) Architecture search based on the high-level router. (2) Parameter search based on the low-level router. The two routers share the backbone θ and thus a common prompt representation, and only the head γ_k of the selected architecture generates its parameters.

of $\mathbf{z}$ are produced from one pass over p, so the router can assign complementary models to different components, rather than choosing the model for each component independently.

3.2 Low-Level Router

The low-level router is responsible for finding suitable parameters for the selected LLMs. A straightforward approach assigns free parameters to each prompt and treats parameter search as a supervised optimization task. However, this method has two main limitations. First, assigning free parameters to each prompt implies that the parameters are independent across prompts, making it difficult to generalize the learned parameters to unseen prompts (*low generalization ability*). Second, both architecture search and parameter search aim to enhance task performance, which means they should ideally share underlying knowledge. Using free parameters, however, makes it difficult to establish such a knowledge-sharing mechanism (*poor knowledge-sharing ability*).

To address the above limitations, we propose to construct a neural network $\mathbf{s}$ to generate the LLM parameters. The parameters of $\mathbf{s}$ serve to compress and relate different prompts within a latent space, thereby allowing better generalization to unseen prompts (*enhancing the generalization ability*). Moreover, by introducing $\mathbf{s}$, both the architecture and parameter search are unified under a common prediction framework. This unification makes it possible to transfer knowledge through parameter sharing between the two tasks (*enhancing the knowledge-sharing ability*).

Formally, given K candidate LLMs in $\mathcal{M}$, we construct a dedicated parameter generation network $\mathbf{s}_k$ for each LLM m_k, since candidate architectures differ in hidden size and therefore require parameters of different shapes. All $\mathbf{s}_k$ take the same form but keep their own layers on top of a shared backbone. To facilitate knowledge sharing, each $\mathbf{s}_k$ is built by stacking several MLP layers on the pooled representation produced by the high-level router $\mathbf{r}$. To keep the generation cost low, $\mathbf{s}_k$ outputs a low-rank update rather than a full weight matrix, giving $w_k = \mathbf{s}_k(p; \phi_k)$, where p is the same routing prompt given to the high-level router, and w_k denotes the generated LoRA parameters tailored for m_k. The parameter set $\phi_k{=}\{\theta, \gamma_k\}$ includes θ, the parameters of the high-level router, which are shared across all $\mathbf{s}_k$, and γ_k, the parameters specific to $\mathbf{s}_k$ and hence to candidate m_k.

Sharing θ serves both levels. The high-level router must already encode which aspects of a prompt separate one candidate from another, since its own decision rests on that, and the same distinction tells the low-level router how the selected candidate should be adjusted. Reusing θ therefore gives each $\mathbf{s}_k$ a representation informative for parameter generation without learning it again, and keeps the added cost per candidate limited to γ_k. The two levels also move together during training: gradients from the generated parameters reach θ, so the representation driving architecture selection reflects how well that architecture can be adapted. Section 4.3 isolates this effect with a decoupled variant.

3.3 Model Optimization

The routing process of our framework consists of two phases: (i) Architecture selection, where the high-level router assigns an LLM to each system component, giving the indices $\mathbf{z} = \mathbf{r}(p; \theta)$. (ii)

Parameter search, where for the ith component we generate the LoRA parameters of its assigned LLM m_{z_i} as $w_{z_i} = \mathbf{s}_{z_i}(p; \phi_{z_i})$. Finally, we enhance the traditional LLM routing objective in Equation (1) by allowing the LLM parameters to be searchable, as follows:

$$\min_{\theta,\mathbf{\Gamma}} \mathop{\mathbb{E}}_{(p,\mathbf{x},y)\sim\mathcal{D}} \left[L\left(\mathrm{G}\left(\mathbf{m}_{\mathbf{z}}(\mathbf{w}_{\mathbf{z}}), \mathbf{x} \right), y \right) \right], \tag{2}$$

where $\mathbf{w}_{\mathbf{z}}=\{w_{z_i}=\mathbf{s}_{z_i}(p;\phi_{z_i})\}_{i=0}^{N-1}$ is the set of the generated LoRA parameters for $\mathbf{m}_{\mathbf{z}}=\{m_{z_i}\}_{i=0}^{N-1}$. The notation $\mathbf{m}_{\mathbf{z}}(\mathbf{w}_{\mathbf{z}})$ indicates that w_{z_i} is added to the frozen weights of m_{z_i}. Here, $\mathbf{\Gamma}=\{\gamma_k\}_{k=0}^{K-1}$ collects the generator parameters of all candidate LLMs in $\mathcal{M}$, excluding the shared θ.

Equation (2) cannot be optimized directly, since $\mathbf{z}$ is discrete and the task loss is only observed after the selected LLMs have produced their outputs. We therefore optimize it in two stages: we first warm up θ on routing decisions that are known to be good, and then jointly optimize θ and $\mathbf{\Gamma}$ with a policy-gradient objective that weights each sampled decision by the reward it receives.

In the warm-up stage, for each prompt p, we sample J LLM assignment vectors $\{\mathbf{z}^{(j)}\}_{j=0}^{J-1}$, or enumerate all of them when the number of components and candidates is small, and use each $\mathbf{z}^{(j)}$ to accomplish the task described in p, where the parameters of all LLMs are kept fixed. We then compute a reward R_j for each $\mathbf{z}^{(j)}$ by comparing the final predicted task output with the ground truth, and keep the best-scoring decision as the training sample for initializing θ, that is, $\{(p, \mathbf{z}^{(j^\star)}) \mid j^\star = \arg\max_j R_j\}$. After the warm-up, we replace the task loss with a reward-weighted negative log-likelihood over the routing decisions, which admits REINFORCE-style gradients:

$$\min_{\theta,\mathbf{\Gamma}} \mathop{\mathbb{E}}_{(p,y)\sim\mathcal{D}_{\text{train}}^{\text{RL}}} \left[-R(\hat{y}, y)\, \log P(\mathbf{z}, \mathbf{w}_{\mathbf{z}} \mid p) \right], \tag{3}$$

where $\mathcal{D}_{\text{train}}^{\text{RL}}$ is the dataset for joint reinforcement learning, drawn as a subset of the warm-up training prompts. $\hat{y}=\mathrm{G}(\mathbf{m}_{\mathbf{z}}(\mathbf{w}_{\mathbf{z}}), \mathbf{x})$ is the system output produced under the sampled decision, and $R(\hat{y}, y)$ is a reward function that can take different forms (e.g., task performance or the trade-off between performance and cost). $P(\mathbf{z}, \mathbf{w}_{\mathbf{z}}|p)$ is the joint probability of the LLM architectures and parameters given the prompt. The objective thus promotes the decisions that earn higher rewards and suppresses the rest. Both levels are updated from this single reward: $\mathbf{\Gamma}$ directly through the generated parameters, and θ through the shared backbone on which all K generators are built.

Remark 1 (**Extension to mixed open- and closed-source LLM routing**). While our framework is designed for parameter search, it can also be easily extended to settings involving both open- and closed-source LLMs. A simple 0-1 indicator can be introduced and multiplied with $\mathbf{w}_{\mathbf{z}}$, so that no parameters are injected into closed-source LLMs behind an inference API and no gradients flow to their generators. Formally, we can refine the objective in Equation (2) as follows:

$$\min_{\theta,\mathbf{\Gamma}} \mathop{\mathbb{E}}_{(p,\mathbf{x},y)\sim\mathcal{D}} \left[L\left(\mathrm{G}\left(\mathbf{m}_{\mathbf{z}}(\mathbf{o}_{\mathbf{z}} \odot \mathbf{w}_{\mathbf{z}}), \mathbf{x} \right), y \right) \right], \tag{4}$$

where $\odot$ is the element-wise product. $\mathbf{o}_{\mathbf{z}} = \{o_{z_i}\}_{i=0}^{N-1}$ is an N-dimensional 0-1 vector, and $o_{z_i} = 1$ means that m_{z_i} is an open-source LLM, so parameter search is confined to those components.

Remark 2 (**Cost of growing the candidate pool**). Dedicating a generator $\mathbf{s}_k$ to every candidate makes the parameter count grow linearly in K, but the slope is small: the backbone is shared, so each candidate only replicates the head on top of it, under 2% of the backbone. Inference stays batched as well. Since the generated updates are low-rank, they need not be merged into the base weights: the adapter term is evaluated as a separate batched product and added to the shared base computation, so queries with different generated parameters can still share one batched forward pass. Section 4.5 reports accuracy and overhead as the pool grows, with implementation details in Appendix F.

4 Experiments

We empirically evaluate our framework around the following research questions (**RQs**):

- **RQ1**: Does HAPS achieve superior performance compared to existing LLM routing methods?
- **RQ2**: Are the key components of HAPS necessary for its gains?
- **RQ3**: How do router backbone and LoRA design choices affect HAPS?
- **RQ4**: As the candidate pool grows, how do performance and computational overhead change?
- **RQ5**: Can HAPS effectively balance the trade-off between task performance and cost?
- **RQ6**: Is our framework effective in mixed scenarios involving open- and closed-source LLMs?

In the following, we answer and analyze the above questions in detail based on empirical results.

Table 1: **Main Results.** HAPS compared with routing baselines on HotpotQA and MMLU. Best and second-best results are bold and underlined. All values are percentages with "%" omitted.

Method	HotpotQA (F1)			MMLU (Acc)		
	L-Q	M-Q	L-M	L-Q	M-Q	L-M
Random	38.94	11.44	23.26	64.48	60.00	58.97
RouteLLM	40.67	<u>36.10</u>	<u>38.95</u>	66.55	<u>65.86</u>	62.76
GraphRouter	<u>41.31</u>	23.07	26.75	<u>66.90</u>	65.52	63.10
IRT-Router	37.34	34.09	38.92	65.52	65.17	<u>63.79</u>
HAPS	**43.16**	**39.70**	**40.58**	**68.97**	**67.24**	**65.86**

4.1 Experimental Setup

Datasets. We evaluate HAPS on two representative benchmarks: **HotpotQA** [Yang et al., 2018] and **MMLU** [Hendrycks et al., 2020].[1] HotpotQA serves as a testbed for multi-hop reasoning, requiring models to synthesize information from multiple supporting facts to answer complex questions. In contrast, MMLU evaluates broad knowledge coverage across 57 subjects, ranging from STEM to the humanities. These two datasets allow us to evaluate HAPS from complementary perspectives: complex reasoning generation and broad knowledge selection. Each benchmark is split into training, validation, and test sets. The warm-up stage trains on the full training set and uses the validation set for checkpoint selection, while the reinforcement learning stage draws a small subset of each. The test set is held out and used only for the final results. Appendix C details the splits.

Baselines. We compare HAPS against a comprehensive set of routing strategies with complementary inductive biases: (1) **Random** samples LLMs uniformly; (2) **RouteLLM** [Ong et al., 2024] learns a binary routing policy from preference data and augmentation; (3) **GraphRouter** [Feng et al., 2024] models task–query–LLM interactions using a heterogeneous graph; and (4) **IRT-Router** [Song et al., 2025] uses item response theory to estimate query difficulty and model ability. To fit our two-agent setting, we apply the protocol-level adjustments summarized in Appendix C.

Implementation Details. We establish a two-agent environment (i.e., $N{=}2$), which comprises a **Student** for solution generation and a **Teacher** for feedback [Bo et al., 2024, Chen et al., 2024c, 2025]. We evaluate our method across three open-source candidate pairs: **L-Q** (Llama-3.1-8B-Instruct & Qwen2.5-7B-Instruct), **M-Q** (Mistral-7B-Instruct-v0.3 & Qwen2.5-7B-Instruct), and **L-M** (Llama-3.1-8B-Instruct & Mistral-7B-Instruct-v0.3). We further extend the evaluation to two mixed-source settings in Section 4.7: **L-G** (Llama-3.1-8B-Instruct & GPT-4.1 Nano) and **Q-D** (Qwen2.5-7B-Instruct & DeepSeek V3 [Liu et al., 2024]). Our high-level router $\mathbf{r}$ employs a Llama-3.2-1B-Instruct backbone to keep routing overhead low. For each low-level router $\mathbf{s}_k$, we append a parameter generation network, structured as a 2-layer MLP with 256 hidden units, to the backbone. During training, we empirically set the LoRA scaling factor $\alpha_{\text{lora}}{=}0.01$ and LoRA rank $r{=}8$. The generated LoRA parameters are injected into the output projection (`o_proj`) of the final Transformer layer[2]. Following common practice, we use token-level F1 for HotpotQA and accuracy for MMLU.

4.2 Overall Performance (RQ1)

We compare **HAPS** with four baselines in Table 1. Random is generally weak, while RouteLLM, GraphRouter, and IRT-Router fluctuate across candidate pairs: none is consistently the runner-up, and GraphRouter, second on HotpotQA L-Q, drops to 23.07 F1 on M-Q. This instability suggests that fixed routing criteria do not generalize across pairs; useful routing signals depend on both the models and the input distribution. HAPS achieves the best performance in all six settings, suggesting that joint search is robust across both complex reasoning generation and broad knowledge selection. On HotpotQA, HAPS outperforms the runner-up by 1.85, 3.60, and 1.63 F1 points on L-Q, M-Q, and L-M, respectively. The largest gain appears on M-Q, the pair on which every baseline scores lowest, suggesting that HAPS helps most when one candidate is much weaker than the other. Here, the margin corresponds to a relative improvement of nearly 10% over the runner-up, RouteLLM. On MMLU, HAPS improves over the strongest baseline by 2.07, 1.38, and 2.07 accuracy points,

[1] Appendix E additionally evaluates HAPS on **IFEval** [Zhou et al., 2023] in a single-agent setting.

[2] RQ3 (Section 4.4) analyzes the effects of backbone choice and alternative LoRA injection designs.

Table 2: **Ablation Studies.** Comparison between HAPS variants and the full model. "Left" and "Right" denote using the first or second model of the candidate pair (e.g., L in L-Q) as the static assignment applied to every test query. Marking and units follow Table 1.

Variant	HotpotQA (F1)			MMLU (Acc)		
	L-Q	M-Q	L-M	L-Q	M-Q	L-M
High-Level Router Ablation						
w/o High-Level (Random)	40.13	26.37	25.44	64.83	61.03	58.62
w/o High-Level (Fixed):						
Teacher=Left, Student=Left	38.48	11.55	<u>37.84</u>	62.07	56.21	<u>64.48</u>
Teacher=Right, Student=Right	34.23	<u>35.51</u>	7.10	<u>68.28</u>	<u>66.90</u>	54.48
Teacher=Left, Student=Right	34.06	32.25	8.89	66.90	66.55	56.55
Teacher=Right, Student=Left	<u>42.45</u>	9.54	35.04	65.52	60.69	62.76
Low-Level Router Ablation						
w/o Low-Level Router	33.93	33.67	36.32	66.55	64.83	61.03
HAPS	**43.16**	**39.70**	**40.58**	**68.97**	**67.24**	**65.86**

respectively. Overall, these gains suggest that jointly searching over architectures and parameters is more flexible than selecting among fixed candidate models without adapting their parameters.

4.3 Component Analysis (RQ2)

To identify the sources of HAPS's gains, we compare it with four variants in Table 2 and Figure 3: (1) **w/o High-Level (Random)** uses uniform sampling; (2) **w/o High-Level (Fixed)** enforces static assignments; (3) **w/o Low-Level** removes LoRA parameter generation; and (4) **w/o Parameter Sharing** freezes the high-level router after initialization and trains the low-level router with its own backbone. Random selection lowers performance substantially, confirming that a learned high-level policy is necessary. Yet under the same random routing, this variant still outperforms the parameter-free Random baseline in Table 1 in five of six settings (e.g., from 11.44 to 26.37 F1 on HotpotQA M-Q), showing that parameter generation partly compensates for poor routing decisions. Static assignments are also insufficient: even the best one for each pair, chosen in hindsight, falls below HAPS in all six settings, indicating that no single assignment suits every query. Removing the low-level module consistently degrades performance, by up to 9.23 F1 points on HotpotQA, indicating that architecture routing alone cannot fully exploit candidate-specific strengths. Removing parameter sharing also weakens HAPS on every pair, by up to 4.33 F1 points on HotpotQA and 4.48 accuracy points on MMLU, confirming that the two routers benefit from a shared backbone (Section 3.2). Appendix H further toggles only the generated parameters on a fixed routing decision to show how they change the behavior of the selected agents.

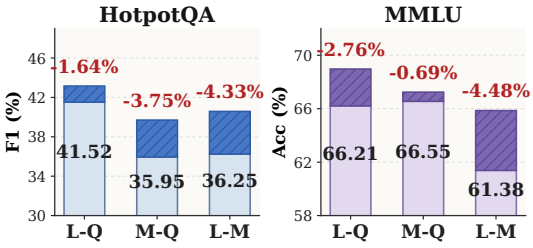

Figure 3: **Parameter sharing analysis.** We compare HAPS with its no-sharing variant on HotpotQA and MMLU; red labels mark the performance drop of the no-sharing variant.

4.4 Effects of Architectural Choices (RQ3)

We analyze three architectural choices on HotpotQA: the router backbone, the number of adapted LoRA layers, and the target module in the final layer. Figure 4 shows that router capacity has a clear effect. Scaling the backbone from Qwen2.5-0.5B-Instruct to Llama-3.2-1B-Instruct (our default) and Qwen2.5-1.5B-Instruct raises the average F1 from 36.63 to 38.33 and 40.96, indicating that HAPS benefits from a stronger representation for prompt-level routing. The preferred adaptation depth depends on the candidate pair: L-Q and L-M peak at l_2, while M-Q continues to improve at l_3. Increasing the depth from l_1 to l_2 improves F1 for all three pairs, with gains of up to 4.56 points on L-M. The target-module results further show that `o_proj` is the strongest final-layer target on L-Q, reaching 44.04 F1 and outperforming `gate_proj` (40.86) and `v_proj` (38.91), while FFN modules are on average stronger than the query, key, and value projections. Overall, these results

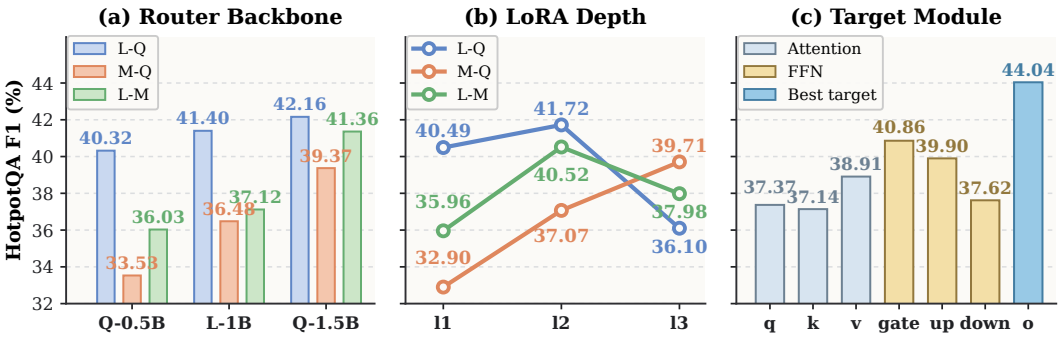

Figure 4: **Architectural choices in HAPS.** We test three design factors on HotpotQA: **(a)** router backbone size, where Q and L denote Qwen and Llama backbones; **(b)** the number of adapted final layers; and **(c)** the final-layer target module. Each panel is an independent sweep that varies only the factor shown, with all other hyperparameters held fixed within that panel.

Table 3: **Scalability and efficiency on larger candidate pools (HotpotQA).** HAPS training time is reported as *Warm-up* + *RL*; inference latency is measured on 1,000 queries with $K=5$. Training time excludes the offline data sampling shared by all methods. Marking and units follow Table 1.

Method	Scalability			Training Time		Inference
	F1@K=5	F1@K=6	Δ	K=5	K=6	K=5 (1k)
Random	23.28	19.43	-3.85	–	–	–
RouteLLM	34.09	28.05	-6.04	6m 36s	6m 32s	52.24s
GraphRouter	<u>35.35</u>	<u>33.98</u>	<u>-1.37</u>	6m 04s	7m 33s	14.71s
HAPS	**37.14**	**41.41**	**+4.27**	5m 19s + 32m 28s	4m 21s + 21m 37s	25.60s

suggest that HAPS gains most from a sufficiently strong router backbone and a well-chosen injection site, with `o_proj` working best among all final-layer modules we tested.

4.5 Scaling with Candidate Pool Size (RQ4)

We next study how performance and cost change as the candidate pool grows. For $K=5$, the pool contains Mistral-7B-Instruct-v0.3, Llama-3.1-8B-Instruct, Qwen2.5-7B-Instruct, Vicuna-7B-v1.5, and DeepSeek-LLM-7B-Chat; $K=6$ additionally includes Yi-1.5-6B-Chat. Since the $\sim$1B router backbone is shared, each extra candidate adds only a 17.4M-parameter low-level head (0.6M MLP + 16.8M projector), about 1.7% of the backbone. Table 3 shows that baselines without parameter adaptation degrade as K increases, whereas HAPS improves from 37.14 to 41.41 F1 (+4.27 points). In particular, RouteLLM and GraphRouter lose 6.04 and 1.37 F1 points when the sixth candidate is added, while HAPS gains 4.27 points, suggesting that HAPS can exploit the added candidate through parameter adaptation. HAPS requires more training time than the baselines, mainly because its RL stage samples multi-agent interactions online; this stage uses only 300 training and 100 validation queries with early stopping (Appendix C). On $K=5$, HAPS routes 1,000 queries in 25.60s, faster than RouteLLM (52.24s) though slower than GraphRouter (14.71s). Appendix F details how LoRA generation and injection are vectorized and batched across requests at inference.

4.6 Performance & Cost Trade-off (RQ5)

In practical LLM routing, candidate models often differ in size, and larger models are typically more capable but also more expensive, so an explicit performance–cost trade-off is necessary. We evaluate HAPS on HotpotQA with a smaller Llama-3.1-8B-Instruct and a larger Qwen2.5-14B-Instruct, pricing each Teacher–Student assignment by a simulated three-tier scheme that reflects relative cost levels rather than real API prices: \$0.2 per 1M input or output tokens when both agents use Llama, \$0.8 when both use Qwen, and \$0.5 when the two are mixed. Following prior work [Song et al., 2025, Feng et al., 2024], we incorporate cost using a weighted reward $R=\alpha\cdot\text{Perf}-\beta\cdot\text{Cost}_{\text{norm}}$, where Perf is task performance and $\text{Cost}_{\text{norm}}$ is the min-max normalized cost under this simulated scheme. We report three regimes: **Performance-First** ($\alpha=0.8, \beta=0.2$), **Balanced** ($\alpha=0.5, \beta=0.5$),

Table 4: **Performance–Cost Trade-off on HotpotQA.** Reward is $\alpha \cdot \text{Perf} - \beta \cdot \text{Cost}_{\text{norm}}$, where Perf is the F1 score and Cost is the normalized cost. All Weak and All Strong assign Llama-3.1-8B-Instruct and Qwen2.5-14B-Instruct, respectively, to both agents. Marking follows Table 1.

Method	$\alpha = 0.8,\ \beta = 0.2$			$\alpha = 0.5,\ \beta = 0.5$			$\alpha = 0.2,\ \beta = 0.8$		
	Perf ↑	**Cost** ↓	**Reward** ↑	**Perf** ↑	**Cost** ↓	**Reward** ↑	**Perf** ↑	**Cost** ↓	**Reward** ↑
Static									
All Weak	0.3458	0.0923	0.2582	0.3458	0.0923	0.1267	0.3458	0.0923	-0.0047
All Strong	0.3697	0.3362	0.2285	0.3697	0.3362	0.0168	0.3697	0.3362	-0.1950
Baselines									
Random	0.3433	0.2160	0.2314	0.3433	0.2160	0.0636	0.3433	0.2160	-0.1042
RouteLLM	**0.4335**	0.2111	0.3046	0.3558	0.0939	0.1310	0.3458	0.0923	-0.0047
GraphRouter	0.3458	0.0923	0.2582	0.3458	0.0923	0.1267	0.3458	0.0923	-0.0047
IRT-Router	0.4299	0.1377	0.3164	0.3604	0.0921	0.1341	0.3552	0.0881	0.0005
HAPS	0.4194	**0.0886**	**0.3178**	**0.3709**	**0.0857**	**0.1426**	**0.4181**	**0.0861**	**0.0148**

and **Cost-First** (α=0.2, β=0.8). As shown in Table 4, HAPS attains the highest reward and lowest cost in all three regimes. Even in the Performance-First setting, it remains competitive while reducing cost. In the Cost-First setting, most baselines approach the All Weak strategy at around 35 F1, while HAPS still reaches 41.81 F1. Notably, HAPS costs even less than All Weak, which routes both agents to the cheaper model, indicating that part of the saving comes from parameter adaptation shortening generations rather than from model selection alone. Further details are provided in Appendix G.

4.7 Routing Across Mixed Open- and Closed-Source LLMs (RQ6)

To validate the extensibility discussed in Remark 1 (Equation (4)), we evaluate HAPS on two mixed-source pairs on HotpotQA: **L-G** (Llama-3.1-8B-Instruct & GPT-4.1 Nano) and **Q-D** (Qwen2.5-7B-Instruct & DeepSeek V3). As shown in Figure 5, HAPS achieves the best F1 on both pairs. On L-G, it reaches 47.19 F1, outperforming the strongest baseline, RouteLLM, by 3.45 points. Since the API parameters are fixed, HAPS improves the system only through routing and by adapting the open-source model to complement the API. On Q-D, HAPS also achieves the highest F1 of 58.52. Here the much stronger DeepSeek V3 handles most queries, so the open-source Qwen is selected less often and leaves less room for parameter adaptation. HAPS still outperforms all baselines, showing that it remains effective even when the open-source model only supplements a powerful API. In both pairs, closed-source models are excluded from parameter generation through the indicator in Equation (4), so the same training procedure applies without modification.

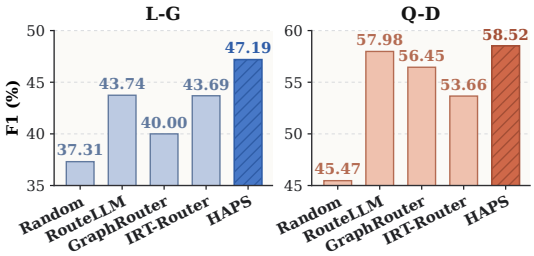

Figure 5: **Mixed-source routing.** HAPS on HotpotQA with open-source models and fixed proprietary APIs; generated parameters are applied only to the open-source model of each pair.

5 Related Work

LLM Routing. LLM routing aims to optimize the performance–cost trade-off via dynamic model selection. Existing approaches are typically categorized into cascade systems, which invoke models sequentially based on estimated response quality [Chen et al., 2023, Dekoninck et al., 2024], and predictive routers that directly map queries to suitable models [Ding et al., 2024] using preference data [Ong et al., 2024], embeddings [Mohammadshahi et al., 2024, Chen et al., 2024b, Jitkrittum et al., 2025], graphs [Feng et al., 2024], or item response theory [Song et al., 2025], and recent work further assigns LLMs to agent roles in multi-agent systems [Yue et al., 2025]. Together with supporting work on serving [Stripelis et al., 2024] and benchmarking [Hu et al., 2024, Huang et al., 2025], these methods treat candidate LLMs as *fixed* and are limited to discrete architecture selection. HAPS bridges this gap by extending the decision space to jointly searchable parameter configurations.

Dynamic Parameter Adaptation. Parameter-Efficient Fine-Tuning (PEFT) is the de facto approach for adapting LLMs, spanning adapters, soft prompts, and LoRA [Houlsby et al., 2019, Pfeiffer et al.,

2020, Li and Liang, 2021, Lester et al., 2021, Hu et al., 2021, Dettmers et al., 2023]. However, conventional PEFT learns *fixed* dataset-level parameters, limiting flexibility. To enable input-conditioned adaptation, prior work uses hypernetworks [Ha et al., 2016] to generate parameters from inputs, improving transfer across tasks and domains [Mahabadi et al., 2021, Ivison et al., 2023]. This extends naturally to LoRA, generating low-rank updates for cross-task generalization [Lv et al., 2024, Charakorn et al., 2025]. HAPS brings this paradigm to routing: rather than adapting a single fixed model, it generates LoRA parameters for whichever candidate is selected, so that model selection and parameter adaptation are optimized jointly under a single shared reward signal.

6 Limitations

HAPS is primarily evaluated in two-agent settings with small candidate pools of up to six models; scaling to larger multi-agent workflows or more heterogeneous pools may require further routing and batching optimization. Parameter generation also requires white-box access, so for closed-source APIs HAPS can only adjust routing, not behavior. In addition, the RL stage samples multi-agent interactions online, which makes training slower than for routers trained purely offline; more sample-efficient optimization could reduce this cost. Finally, we study only a subset of LoRA injection designs; future work could explore deeper or finer-grained parameter generation, such as injecting adapters into all layers or assigning layer-specific target modules to each candidate model.

7 Conclusion

We presented HAPS, a hierarchical routing framework that jointly selects LLM architectures and generates input-conditioned LoRA parameters for the selected candidates. Rather than treating each candidate as a fixed point in behavior space, HAPS lifts it to a region of reachable behaviors and routes over these regions. Across HotpotQA and MMLU, multiple candidate pairs, larger candidate pools, cost-aware rewards, and mixed open- and closed-source deployments, HAPS consistently outperforms strong routing baselines. We hope this perspective encourages a shift in LLM routing from routing among fixed models to routing over their reachable behaviors.

Acknowledgments and Disclosure of Funding

The authors report no third-party funding or support for this work and no competing interests.

References

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

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant transformer adaption. *arXiv preprint arXiv:2506.06105*, 2025.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024a.

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

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. RouterDC: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024b.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pages 20094–20136, 2024c.

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing effectiveness and efficiency for LLM-based multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11534–11557, 2025.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for LLMs. *arXiv preprint arXiv:2410.10347*, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. *Advances in neural information processing systems*, 36:10088–10115, 2023.

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

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A graph-based router for LLM selections. *arXiv preprint arXiv:2410.03834*, 2024.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Zhongzhan Huang, Guoming Ling, Yupei Lin, Yandong Chen, Shanshan Zhong, Hefeng Wu, and Liang Lin. RouterEval: A comprehensive benchmark for routing LLMs to explore model-level scaling up in LLMs. *arXiv preprint arXiv:2503.10657*, 2025.

Hamish Ivison, Akshita Bhagia, Yizhong Wang, Hannaneh Hajishirzi, and Matthew E Peters. HINT: Hypernetwork instruction tuning for efficient zero- and few-shot generalisation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11272–11288, 2023.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, 2023. URL `https://arxiv.org/abs/2310.06825`.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, et al. Universal model routing for efficient LLM inference. *arXiv preprint arXiv:2502.08773*, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 3045–3059, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Chuancheng Lv, Lei Li, Shitou Zhang, Gang Chen, Fanchao Qi, Ningyu Zhang, and Hai-Tao Zheng. HyperLoRA: Efficient cross-task generalization via constrained low-rank adapters generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16376–16393, 2024.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576, 2021.

Alireza Mohammadshahi, Arshad Rafiq Shaikh, and Majid Yazdani. Routoo: Learning to route to large language models effectively. *arXiv preprint arXiv:2401.13979*, 2024.

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

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 46–54, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. S-LoRA: Serving thousands of concurrent LoRA adapters. *arXiv preprint arXiv:2311.03285*, 2023.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. IRT-Router: Effective and interpretable multi-LLM routing via item response theory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15629–15644, 2025.

Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. TensorOpera router: A multi-model router for efficient LLM inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 452–462, 2024.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, 2025.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Architectural Details of HAPS

In this section, we provide a detailed specification of the high-level and low-level routing components within our hierarchical framework, complementing the overview in Section 3.

A.1 High-Level Router

To minimize inference overhead, we employ a lightweight instruction-tuned LLM (Llama-3.2-1B-Instruct) as the high-level router $\mathbf{r}$, which reads the natural language routing prompt and expresses its decision as a JSON object assigning one candidate model to each agent in the system.

Since the set of valid assignments is finite, the router defines a distribution over them: during RL, assignments are sampled from this distribution, and at inference we select the most probable one. The chosen assignment is parsed into the vector $\mathbf{z} = \{z_i\}_{i=0}^{N-1}$, which dictates the model selection for each agent in the system and determines which low-level generators are invoked to produce the corresponding input-conditioned LoRA parameters for the selected candidates.

A.2 Low-Level Router

The role of the low-level router is to generate suitable LoRA parameters for the base models selected by the high-level router. We formulate this as a hypernetwork-based parameter generation task. Since candidate models in the pool possess distinct architectures (e.g., varying hidden state sizes), we instantiate a dedicated low-level routing head $\mathbf{s}_k$ for each candidate base model m_k.

Structurally, each low-level router $\mathbf{s}_k$ shares the backbone of the high-level router $\mathbf{r}$ (Section 3.2). The routing prompt is encoded by this backbone, and its final-layer hidden states are mean-pooled over non-padding tokens. On top of this pooled representation, $\mathbf{s}_k$ applies a 2-layer MLP with 256 hidden units and ReLU activations. This is subsequently followed by two parallel linear heads, which are responsible for independently projecting the latent vector into the LoRA rank-decomposition matrices A and B, whose shapes depend on the hidden size of the target candidate.

Formally, for a selected base model m_{z_i} with a hidden dimension of d_{z_i}, these heads generate $A \in \mathbb{R}^{d_{z_i} \times r}$ and $B \in \mathbb{R}^{r \times d_{z_i}}$, where the rank r is a hyperparameter set to 8 based on empirical findings. These generated parameters are injected into the output projection layer (`o_proj`) of the final Transformer block in m_{z_i}, yielding the effective weight

$$\tilde{W}^o_{z_i} = W^o_{z_i} + \alpha_{\text{lora}} AB, \tag{5}$$

where $W^o_{z_i}$ denotes the frozen weights of the final `o_proj` layer, and α_{lora} is the scaling factor. This equation is only an equivalent form: in practice the update is never merged into $W^o_{z_i}$, and the adapter term is computed separately in batch to keep the per-query overhead low (Appendix F).

B Detailed Multi-Agent System Configuration

Following the ReAct paradigm [Yao et al., 2022], both benchmarks use a two-agent Teacher–Student system with minor task-specific adaptations in the action space and termination rule. All agent and router prompts used on both benchmarks are listed verbatim in Appendix I.

HotpotQA Protocol. For the HotpotQA dataset, which necessitates multi-hop reasoning, the system is equipped with an external retrieval interface. The interaction protocol is formalized as follows:

The Student agent operates with an action space consisting of two discrete primitives:

- `Search[Entity]`: Retrieves the single paragraph most similar to `Entity` among those provided with the question, using an unsupervised SimCSE (RoBERTa-base) encoder.
- `Finish[Answer]`: Terminates the episode and submits the final `Answer` for evaluation.

The Teacher provides feedback by selecting `Continue` (validating the utility of the previous retrieval) or `Rethink` (prompting a revised search strategy with accompanying justification).

Both agents generate a reasoning trace before outputting their action. Each agent also maintains a running summary of the interaction, which it rewrites every round to incorporate the newly acquired

Algorithm 1: HotpotQA Pipeline

Input: Question q, Budget B, Knowledge $\mathcal{I}$
Output: Final Answer a

$H_S, H_T \leftarrow \varnothing, \varnothing$
for $t \leftarrow 1$ **to** $B + 1$ **do**
  // Student Phase: Reasoning
  $s_{\text{obs}} \leftarrow \text{Observe}(t, H_S, H_T)$
  $s_{\text{thought}} \leftarrow \text{Think}(q, s_{\text{obs}})$
  $s_{\text{act}} \leftarrow \text{Action}(q, s_{\text{obs}}, s_{\text{thought}})$
  // Execution & Termination
  if $s_{\text{act}} \in \text{Search}$ **then**
    $s_{\text{res}} \leftarrow \text{Retrieve}(s_{\text{act}}, \mathcal{I})$
    $H_S \leftarrow H_S \oplus [(s_{\text{act}}, s_{\text{res}})]$
  end
  else if $s_{\text{act}} = \text{Finish}[a]$ **then**
    return a
  end
  // Teacher Phase: Feedback
  $t_{\text{obs}} \leftarrow \text{Observe}(t, H_S, H_T)$
  $t_{\text{thought}} \leftarrow \text{Think}(q, t_{\text{obs}})$
  $t_{\text{act}} \leftarrow \text{Action}(q, t_{\text{obs}}, t_{\text{thought}})$
  $H_T \leftarrow H_T \oplus [(t_{\text{act}}, t_{\text{thought}})]$
end
return Null

Algorithm 2: MMLU Pipeline

Input: Problem p, Agents S, T, Limit K_{round}
Output: Final Answer a

$S_m, T_m \leftarrow [\,], [\,]$
for $t \leftarrow 1$ **to** K_{round} **do**
  // Student Phase: Reasoning
  $s_{\text{obs}} \leftarrow S.\text{Observe}(t, S_m, T_m)$
  $s_{\text{ans}} \leftarrow S.\text{Answer}(p, s_{\text{obs}})$
  $S_m.\text{append}(s_{\text{ans}})$
  // Teacher Phase: Feedback
  $t_{\text{obs}} \leftarrow T.\text{Observe}(t, S_m)$
  $t_{\text{feed}}, \epsilon \leftarrow T.\text{Feedback}(p, t_{\text{obs}})$
  if $\epsilon = \text{True}$ **then**
    return s_{ans}
  end
  $T_m.\text{append}(t_{\text{feed}})$
end
return *Last item of* S_m

information. Each episode allows a search budget of B turns (set to 3 in our experiments), after which the Student must submit an answer in the final, $(B+1)$-th round.

To compute the reward, we normalize both the predicted answer $\hat{a}$ and the ground truth g following standard protocols: (i) lower-casing; (ii) punctuation removal; (iii) article stripping (i.e., removing "a", "an", "the"); and (iv) whitespace normalization. We then compute the token-level F1 between the normalized strings as the task score $s \in [0, 1]$, which serves as the reward $R(\hat{y}, y)$ in Equation (3) after the shaping in Appendix D. Algorithm 1 details the complete pipeline.

MMLU Protocol. Unlike HotpotQA, the MMLU Student uses *no* external retrieval tools. Each episode proceeds for at most K_{round} interaction rounds (default $K_{\text{round}} = 3$). At every round, the Student observes the Teacher's feedback from the previous round and generates a revised solution with answer. The Teacher reviews the solution, produces feedback and chooses between two discrete actions: `Continue` (requesting another revision) or `Submit` (terminating the episode and submitting the current solution as the final answer). If `Submit` is issued or K_{round} rounds are reached, the episode ends and the last solution and answer produced by the Student are evaluated.

The predicted answer is a single option in $\{\text{A}, \text{B}, \text{C}, \text{D}\}$ and is correct only if it exactly matches the ground-truth label, so the task score is binary, $s \in \{0, 1\}$, and is shaped into the reward as above. Algorithm 2 details the complete pipeline, including the Teacher's early-submission rule.

C Data Collection & Implementation

Benchmarks and Splits. For HotpotQA, we sample 3,000 training ($\mathcal{D}_{\text{train}}$), 1,000 validation ($\mathcal{D}_{\text{valid}}$), and 100 test ($\mathcal{D}_{\text{test}}$) questions from the `hotpot_dev_distractor_v1.json` split. For MMLU, we apply stratified sampling over all subjects to maintain broad coverage, obtaining 3,000 training, 1,000 validation, and 290 test questions, each with four answer options.

Action-Level Dataset Construction. With two agent roles (Teacher/Student) and two candidate LLMs, a routing decision for each instance can be represented as an assignment vector $\mathbf{z} = (z_T, z_S)$, where each entry specifies the selected model for one agent. We define the routing action space as the set of all feasible assignment vectors, denoted by $\mathcal{A}_{\text{act}}$; in our setting, $|\mathcal{A}_{\text{act}}| = 4$.

To obtain full-information supervision signals, we exhaustively execute every routing decision $\mathbf{z} \in \mathcal{A}_{\text{act}}$ for each problem p in the train/validation splits, and record the task score $s(p, \mathbf{z})$ defined in Appendix B. This yields an expanded *action-level* dataset:

$$\widetilde{\mathcal{D}}_{\text{split}} = \{(p, \mathbf{z}, s) \mid p \in \mathcal{D}_{\text{split}}, \mathbf{z} \in \mathcal{A}_{\text{act}}\}, \tag{6}$$

where $\text{split} \in \{\text{train}, \text{valid}\}$. Thus, each expanded dataset is $|\mathcal{A}_{\text{act}}|$ times larger than its problem-level counterpart, covering all four routing decisions of every problem.

SFT Initialization Dataset. We first construct a warm-up dataset $\mathcal{D}_{\text{warm}}$ to supervise the high-level router by selecting, for each training problem, the highest-reward routing decision under exhaustive evaluation: $\mathcal{D}_{\text{warm}}=\{(p, \mathbf{z}^\star) \mid p \in \mathcal{D}_{\text{train}},\ \mathbf{z}^\star \in \arg\max_{\mathbf{z} \in \mathcal{A}_{\text{act}}} s(p, \mathbf{z})\}$. Our code supports several strategies for handling tied optimal decisions and label imbalance, such as keeping all tied decisions or over- and undersampling, which can be chosen according to the training budget.

Joint RL Dataset. To simulate a data-efficient online adaptation setting for Phase 2, we randomly select 300 problems from $\mathcal{D}_{\text{train}}$ and 100 from $\mathcal{D}_{\text{valid}}$ for each benchmark, forming $\mathcal{D}^{\text{RL}}_{\text{train}}$ and $\mathcal{D}^{\text{RL}}_{\text{valid}}$, respectively; the latter is used for checkpoint selection and early stopping. During Phase 2, $\mathcal{D}^{\text{RL}}_{\text{train}}$ is used to jointly optimize the high-level router $\mathbf{r}_{\boldsymbol{\theta}}$ and low-level generators $\mathbf{s}_{\boldsymbol{\Gamma}}$. The held-out $\mathcal{D}_{\text{test}}$ is strictly reserved for final evaluation of the complete routing framework.

Baseline Adaptations. To align all the baselines with our main experimental protocol, which evaluates routing decisions solely by performance, we keep their original architectures and training procedures unchanged, and introduce only the minimal modifications required by our two-agent routing setup. For the performance–cost trade-off in Section 4.6, we instead enable each baseline's native cost-aware routing. **RouteLLM.** Since RouteLLM makes binary decisions, we train an independent router for each agent role. Each router ignores cost and fixes the win-probability

threshold at τ=0.5 (other hyperparameters unchanged), selecting the strong model only when its predicted win probability is at least 0.5. **GraphRouter.** GraphRouter builds a heterogeneous task–query–LLM graph with textual model descriptors; we instantiate descriptors for our two candidates and use the "Performance First" setting to obtain routing utilities. **IRT-Router.** We adopt the *MIRT* variant and set β=0 (thus α=1) to remove cost from its objective; we enable cold-start warm-up and use λ=0.3 following the sensitivity analysis in the original IRT-Router paper.

D Training & Evaluation Details

Algorithm 3 summarizes our two-phase protocol: (i) supervised warm-up (SFT) of the high-level router on $\mathcal{D}_{\text{warm}}$, and (ii) joint on-policy optimization on $\mathcal{D}^{\text{RL}}_{\text{train}}$ with periodic validation on $\mathcal{D}^{\text{RL}}_{\text{valid}}$ for checkpoint selection and early stopping; the selected checkpoint is evaluated once on $\mathcal{D}_{\text{test}}$ after training. All main HAPS training runs are conducted on NVIDIA A800 80GB GPUs. Table 3 reports representative training time and inference latency measurements. In Phase 2, for each problem prompt p, the high-level router selects a joint assignment $\mathbf{z} = (z_T, z_S)$ for the Teacher and Student, and the corresponding low-level generators produce input-conditioned LoRA parameters for the selected base models. We then run the multi-agent interaction with the LoRA-injected models, sampling tokens during training and decoding greedily at evaluation, and record the log-probability of every token generated by *both* agents. In implementation, the likelihood $\log P(\mathbf{z}, \mathbf{w}_{\mathbf{z}} \mid p)$ in Equation (3) is computed from these token log-probabilities, which depend on the generated parameters and, through the shared backbone, on θ. The per-episode loss is

$$\mathcal{L}_{\text{rl}} = -\hat{R} \sum_{t \in \mathcal{T}} \log \pi_{\theta,\mathbf{\Gamma}}(a_t \mid c_t)\,, \tag{7}$$

which is averaged over a batch of episodes before each update. Here, $\mathcal{T}$ is the index set of all decoding steps produced by both agents in an episode; a_t is the token emitted at step t by the active agent, and c_t is its conditioning context, consisting of the problem prompt, the interaction history, any retrieved evidence available up to that step, and role indicators. $\pi_{\theta,\mathbf{\Gamma}}(a_t \mid c_t)$ is the probability assigned to a_t by the active agent's LLM after injecting the generated LoRA parameters, which depends on the high-level router parameters θ and the low-level generator parameters $\mathbf{\Gamma}$. $\hat{R}$ is the scalar episode-level reward derived from the task score s (Appendix B); we apply a shaping rule $\hat{R} = -1$ if $s = 0$, and otherwise $\hat{R} = s$. The hyperparameters specific to HAPS are the LoRA rank (r=8), scaling factor (α_{lora}=0.01), injection site (final-layer `o_proj`), generator width (256 hidden units), and interaction budgets (B=3 search turns for HotpotQA and K_{round}=3 rounds for MMLU).

E Single-Agent Routing on Instruction Following

To test HAPS beyond multi-agent coordination and reasoning tasks, we evaluate it in a *single-agent* setting (N=1) on IFEval-style instruction following, using verifiable instructions from the RLVR-IFeval pool [Lambert et al., 2024] built on the IFEval constraint set [Zhou et al., 2023]. For each instruction, the high-level router selects one model to answer it directly and the low-level router generates LoRA parameters for that model; no Teacher–Student interaction takes place. The reward is binary and given by programmatic constraint checkers rather than a judge model.

We use 2,000 instances for training, 500 for validation, and 500 for testing, and report the mean reward on the test split over three seeds. Beyond two-candidate pools, we also route over a three-candidate pool (**L-Q-M**) to see whether the benefit survives a larger action space. All methods share the same data splits, candidate pools, and constraint checkers for training and evaluation.

As shown in Table 5, HAPS attains the highest mean in all three pools, outperforming the strongest baseline by 0.40, 0.93, and 1.86 points on L-Q, M-Q, and L-Q-M. The margin is largest on the three-candidate pool, where the error bars of HAPS and the runner-up do not overlap. The benefit of joint architecture-and-parameter routing thus carries over to single-agent instruction following.

F Efficient Inference Implementation

This appendix describes how HAPS supports *per-input* LoRA parameters while preserving GPU parallelism. The key challenge is that naively applying different adapters to different samples

Algorithm 3: HAPS Training: SFT Warm-Up & Joint RL

Input: Warm-Up Data $\mathcal{D}_{\text{warm}}$; RL Data $\mathcal{D}_{\text{train}}^{\text{RL}}, \mathcal{D}_{\text{valid}}^{\text{RL}}$; Test Data $\mathcal{D}_{\text{test}}$
Input: Model Pool $\mathcal{M}$; Router $\mathbf{r}_{\boldsymbol{\theta}}$; Generators $\{\mathbf{s}_{\boldsymbol{\gamma}_m}\}_{m\in\mathcal{M}}$
Input: LRs $\eta_{\text{sft}}, \eta_\theta, \eta_\gamma$; Scale α_{lora}; Epochs $E_{\text{sft}}, E_{\text{rl}}$; Eval interval K_{eval}; Patience P
Output: Optimized Parameters $\boldsymbol{\theta}^*, \boldsymbol{\Gamma}^*$ and test score $\text{score}_{\text{test}}$

Initialize parameters $\boldsymbol{\theta}, \boldsymbol{\Gamma} = \{\boldsymbol{\gamma}_m\}_{m\in\mathcal{M}}$
$\text{score}_{\text{best}} \leftarrow -\infty$; $\text{step} \leftarrow 0$; $\text{stall} \leftarrow 0$
// Phase 1: Supervised Warm-Up (SFT)
for epoch $\leftarrow 1$ **to** E_{sft} **do**
  foreach $(p, \mathbf{z}^\star) \in \text{Shuffle}(\mathcal{D}_{\text{warm}})$ **do**
    $\mathbf{z}_{\text{pred}} \leftarrow \mathbf{r}_{\boldsymbol{\theta}}(p)$
    $\mathcal{L}_{\text{sft}} \leftarrow \text{CrossEntropy}(\mathbf{z}_{\text{pred}}, \mathbf{z}^\star)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\text{sft}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{sft}}$
  end
end
// Phase 2: Joint RL Optimization
for epoch $\leftarrow 1$ **to** E_{rl} **do**
  foreach *batch* $\mathcal{B} \subset \text{Shuffle}(\mathcal{D}_{\text{train}}^{\text{RL}})$ **do**
    foreach $p \in \mathcal{B}$ **do**
      `// 1. Hierarchical Routing`
      $(m_T, m_S) \leftarrow \mathbf{r}_{\boldsymbol{\theta}}(p)$
      `// 2. Adaptive Parameter Generation`
      $\mathbf{w}_T \leftarrow \mathbf{s}_{\boldsymbol{\gamma}_{m_T}}(p), \quad \mathbf{w}_S \leftarrow \mathbf{s}_{\boldsymbol{\gamma}_{m_S}}(p)$
      `// 3. Vectorized Injection & Interaction`
      $\hat{\mathbf{y}}_p, \log \boldsymbol{\pi}_{T,p}, \log \boldsymbol{\pi}_{S,p} \leftarrow \text{RunMAS}(p \mid m_T^{\mathbf{w}_T}, m_S^{\mathbf{w}_S})$
      $s_p \leftarrow \text{TaskMetric}(\hat{\mathbf{y}}_p, \mathbf{y}_p^*)$
      if $s_p = 0$ **then** $\hat{R}_p \leftarrow -1$
      else $\hat{R}_p \leftarrow s_p$
    end
    `// 4. Optimization Step`
    $\mathcal{L}_{\text{rl}} \leftarrow -\frac{1}{|\mathcal{B}|} \sum_{p\in\mathcal{B}} \hat{R}_p \cdot (\sum \log \boldsymbol{\pi}_{T,p} + \sum \log \boldsymbol{\pi}_{S,p})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_\theta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{rl}}$
    foreach *model* m *selected in* $\mathcal{B}$ **do**
      $\boldsymbol{\gamma}_m \leftarrow \boldsymbol{\gamma}_m - \eta_\gamma \nabla_{\boldsymbol{\gamma}_m} \mathcal{L}_{\text{rl}}$
    end
    `// 5. Interval Validation & Early Stopping`
    $\text{step} \leftarrow \text{step} + 1$
    if $\text{step} \bmod K_{\text{eval}} = 0$ **then**
      $\text{score}_{\text{val}} \leftarrow \text{EvaluateBucketed}(\mathcal{D}_{\text{valid}}^{\text{RL}}, \boldsymbol{\theta}, \boldsymbol{\Gamma}, \alpha_{\text{lora}})$
      if $\text{score}_{\text{val}} > \text{score}_{\text{best}}$ **then**
        $\text{score}_{\text{best}} \leftarrow \text{score}_{\text{val}}$; Save checkpoint $\boldsymbol{\theta}^*, \boldsymbol{\Gamma}^*$; $\text{stall} \leftarrow 0$
      else
        $\text{stall} \leftarrow \text{stall} + 1$
      end
      if $\text{stall} \geq P$ **then** stop training
    end
  end
end
// Final Evaluation
$\text{score}_{\text{test}} \leftarrow \text{EvaluateBucketed}(\mathcal{D}_{\text{test}}, \boldsymbol{\theta}^*, \boldsymbol{\Gamma}^*, \alpha_{\text{lora}})$

Table 5: **Single-agent instruction-following results on IFEval.** L-Q and M-Q are two-candidate pools; L-Q-M adds Mistral-7B-Instruct-v0.3 as a third candidate. Values are mean ± standard deviation over seeds $\{42, 43, 44\}$, in percentages with "%" omitted. Marking follows Table 1.

Method	L-Q	M-Q	L-Q-M
Random	45.87±0.52	39.53±1.05	42.93±0.09
RouteLLM	46.20±0.75	44.93±0.52	46.87±0.96
GraphRouter	<u>47.00</u>±0.71	45.33±0.19	<u>47.47</u>±0.09
IRT-Router	46.40±0.28	<u>45.80</u>±0.16	47.00±0.00
HAPS	**47.40**±0.90	**46.73**±0.75	**49.33**±0.74

can introduce per-sample loops (thus $O(b)$ serial latency). We avoid this by (i) separating the shared base computation from the adapter computation, and (ii) vectorizing the adapter term with batched GEMM/BMM kernels. This design follows multi-adapter serving systems that batch LoRA computations across heterogeneous requests [Sheng et al., 2023, Chen et al., 2024a].

F.1 Vectorized Dynamic LoRA Injection

Consider a linear layer with frozen base weight $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. LoRA adds a low-rank update $\Delta W = AB$, where $A \in \mathbb{R}^{d_{\text{out}} \times r}$ and $B \in \mathbb{R}^{r \times d_{\text{in}}}$ with $r \ll \min(d_{\text{in}}, d_{\text{out}})$. Let the batched hidden states be $H^{\text{in}} \in \mathbb{R}^{b \times L \times d_{\text{in}}}$. The layer output becomes

$$\begin{aligned} H^{\text{out}} &= H^{\text{in}} W_0^\top + \alpha_{\text{lora}}\, H^{\text{in}} (AB)^\top \\ &= H^{\text{in}} W_0^\top + \alpha_{\text{lora}}\, (H^{\text{in}} B^\top) A^\top, \end{aligned} \tag{8}$$

where α_{lora} is the LoRA scaling factor; the second form never materializes the $d_{\text{out}} \times d_{\text{in}}$ update AB.

In HAPS, each sample i has its own (A_i, B_i) produced on the fly by the low-level generator. Instead of looping over samples, we stack per-sample adapters into batch tensors $\mathcal{A} \in \mathbb{R}^{b \times d_{\text{out}} \times r}$ and $\mathcal{B} \in \mathbb{R}^{b \times r \times d_{\text{in}}}$, where $^\top$ transposes the last two dimensions. We then compute the adapter term via two batched matrix multiplications:

$$H_{\text{lora}} = \alpha_{\text{lora}} \cdot \text{BMM}\big(\text{BMM}(H^{\text{in}}, \mathcal{B}^\top), \mathcal{A}^\top\big), \tag{9}$$

which yields $H_{\text{lora}} \in \mathbb{R}^{b \times L \times d_{\text{out}}}$ in parallel across the batch. Intuitively, the hidden states of each sample i are first projected to a rank-r subspace by B_i and then mapped back by A_i, while the base term $H^{\text{in}} W_0^\top$ remains a standard highly optimized GEMM shared by all samples. Because the frozen weights are never swapped in memory, this avoids per-request weight materialization and retains the throughput benefits of batched execution even under per-input adapters.

F.2 Request Bucketing for Low Latency

To further improve runtime efficiency, we bucket incoming queries by the routed base model. The high-level router first predicts the base model identity for each request; requests assigned to the same base model are then grouped and executed as a sub-batch. Within each bucket, the low-level generator produces all per-request adapter parameters in one forward pass, and subsequent decoding runs on a single loaded base model with vectorized dynamic injection. This reduces repeated model loading so that the base-model forward pass dominates end-to-end latency, while the per-request adapter computation remains fully batched within each bucket and adds little overhead.

G Trade-off Analysis

G.1 Experimental Setup and Cost Simulation

Following the protocols established in prior studies [Song et al., 2025, Feng et al., 2024], we quantify the trade-off between effectiveness and efficiency under explicit performance–cost reward configurations. For each episode, we define the raw inference cost of a query, summed over both agents and all interaction rounds, as:

$$C_{\text{raw}} = \frac{L_{\text{in}} P_{\text{in}} + L_{\text{out}} P_{\text{out}}}{10^6}. \tag{10}$$

where L_{in} and L_{out} are the input and output token counts, and P_{in} and P_{out} are the corresponding unit prices (per 1M tokens) under the three-tier pricing scheme used in Section 4.6: \$0.2/\$0.2 for Llama+Llama, \$0.5/\$0.5 for mixed Llama+Qwen routing, and \$0.8/\$0.8 for Qwen+Qwen.

To make cost commensurate with the performance metric (e.g., Perf $\in [0, 1]$ for F1), we apply min-max normalization using training-set statistics:

$$\text{Cost}_{\text{norm}} = \frac{C_{\text{raw}} - C_{\min}}{C_{\max} - C_{\min}}, \tag{11}$$

where $C_{\min}$ and $C_{\max}$ are computed on the training split and then fixed for validation/test evaluation.

Reward Definition. We evaluate performance–cost trade-offs via a scalar reward:

$$\text{Reward} = \alpha \cdot \text{Perf} - \beta \cdot \text{Cost}_{\text{norm}}, \tag{12}$$

where (α, β) specifies the deployment preference and Perf is the F1 score on HotpotQA. For each (α, β), HAPS is trained with Equation (12) as its RL reward, while the baselines use their native cost-aware routing under the same (α, β) weights, as described in Appendix C.

The cost excludes LoRA generation and injection, whose overhead is small relative to the base-model forward pass under batched execution (Appendix F), so it reflects token usage and model selection.

G.2 Performance Analysis across Scenarios

We evaluate HAPS under three configurations (α, β) that reflect different deployment priorities.

Performance-First (α=0.8, β=0.2). In this regime, the system prioritizes accuracy. As shown in Table 4, RouteLLM attains 43.35 F1 but incurs substantial cost (0.2111) due to frequent selection of the expensive model. In contrast, HAPS achieves a comparable 41.94 F1 while reducing the cost to 0.0886, a 58% reduction relative to RouteLLM. This suggests HAPS selectively leverages the cheaper model when it suffices, aided by input-conditioned parameter generation.

Balanced (α=0.5, β=0.5). In the Balanced regime, HAPS achieves the highest aggregate reward (0.1426), outperforming the runner-up IRT-Router (0.1341). These results suggest that coupling discrete model selection with continuous parameter adaptation achieves a better balance between performance and cost than routing among fixed candidate models alone.

Cost-First (α=0.2, β=0.8). As discussed in Section 4.6, HAPS costs less than All Weak under the same decoding setting (0.0861 vs. 0.0923) while maintaining 41.81 F1 (vs. 34.58). Since no routing decision can be cheaper than assigning the cheaper model to both agents, the remaining saving must come from fewer generated tokens, e.g., through more concise trajectories.

H What the Generated Parameters Change

We examine how the generated parameters change agent behavior by holding the high-level decision fixed and toggling only the generated parameters: for each test question we take the architecture that HAPS itself selected, then run the same multi-step agent twice with greedy decoding, once on the frozen base models and once with the generated LoRA injected. Any difference is therefore attributable to the parameters alone. The example below is from the M-Q pair, where HAPS routes the Teacher to Mistral-7B-Instruct-v0.3 and the Student to Qwen2.5-7B-Instruct.

In the example of Figure 6, the first retrieval is uninformative: the query returns a paragraph about a television film on Pete Rose that does not name the lawyer. Both runs receive this paragraph in Round 1 and are told by the Teacher to rethink. Without the generated parameters, the Student repeats the identical query in every remaining round, receives the same paragraph each time, and finally answers with a name that appears in none of the retrieved text. With them, the Student rephrases the query, which retrieves the paragraph on the Dowd Report stating that it was prepared by John M. Dowd; the Teacher confirms, and the Student answers correctly in the next round. The generated parameters thus change how the agents respond when a retrieval returns irrelevant evidence.

Question. *What was the name of the lawyer who took the main role in the report that helped land Pete Rose a lifetime suspension from baseball?* **Answer:** `John M. Dowd`

Round	Frozen base models	With generated parameters
1	`Search[Pete Rose lifetime suspension lawyer]` → a television film about Pete Rose; no lawyer named *Teacher:* `[Rethink]`	`Search[Pete Rose lifetime suspension lawyer]` → same paragraph *Teacher:* `[Rethink]`
2	`Search[Pete Rose lifetime suspension lawyer]` *identical query repeated* → same paragraph *Teacher:* `[Rethink]`	`Search[lawyer in Pete Rose report]` *query rephrased* → *"The Dowd Report . . . was prepared by Special Counsel to the Commissioner, John M. Dowd . . . "* *Teacher:* `[Continue]`
3	`Search[Pete Rose lifetime suspension lawyer]` *identical query repeated* → same paragraph *Teacher:* `[Rethink]`	`Finish[John M. Dowd]`
4	`Finish[Judge Kenesaw Mountain Landis]` *name absent from all retrieved text*	–
Final	*wrong answer* F1 = 0.0	`John M. Dowd` (correct) F1 = 1.0

Figure 6: Case study on HotpotQA (M-Q pair) with the high-level routing decision held fixed, comparing the frozen base models with the same models after injecting the generated LoRA. Each cell shows the Student's action in that round, followed by the Teacher's response to it.

I Prompt Templates

Figures 7–15 list the prompts used in our experiments: the Thought & Action and Context prompts of the HotpotQA agents, the prompts of the MMLU agents, and the system and user prompts that the high-level router receives on both benchmarks. Placeholders in braces are filled at run time.

HotpotQA Teacher Thought & Action Prompt

You are the Teacher agent. Your task is to guide the Student agent to answer the question. You should analyze whether the student's step is logically helpful, provide an analysis, and give your final action. You can also give advice on the student's future step.
Context refers to the summary of previous interactions.
Question is the question you need to guide the Student to answer.
Observation is the last round of what you guided and what the Student did after you guided.
Thought should reason about the current situation and determine what action to take.
Action can be of two types:
(1) [Continue], which means the student made a good decision and should continue the process.
(2) [Rethink], which means the student's previous step is wrong and should consider another step.

Please note:
You can analyze the correctness of the Student agent's action, summarize the useful information the Student found, or provide suggestions for subsequent steps.
Your action step has only two types: [Continue] or [Rethink]. You can only take one of the above actions. Please be an encouraging teacher most of the time; that is, unless the student is completely wrong, use the [Continue] action more often.

[INPUT]
Context: {context}
Question: {question}
Observation: {observation}

[OUTPUT REQUIREMENTS]
Your output must strictly follow the format:

Thought: <Your thinking process on your action.>
Action: <Your action on the current situation. [Continue] or [Rethink]>

You must output exactly in the format, beginning with "Thought:" when you output your thought, and beginning with "Action:" when you output your action.
Notice that the action **MUST** be wrapped by brackets "[]".
Keep your entire output within 256 tokens.

Figure 7: The Thought & Action prompt of the Teacher agent on HotpotQA.

HotpotQA Teacher Context Prompt

You are the Teacher agent, and your task is to guide the Student agent to answer the question. Now you are provided with a previous summary, as well as new messages that were not included in the original summary. You need to make a new summary.
Your summary should encapsulate the main points of the new messages and integrate them into the existing summary to create a comprehensive recap. Highlight the key issues discussed, decisions made, and any actions assigned. Record the helpful factual information given by the search engine.

[INPUT]
Question: {question}
Previous Summary: {context}
New Observation: {observation}
New Thought: {thought}
New Action: {action}

[OUTPUT REQUIREMENTS]
Your output must strictly follow the format:

Summary: <Your new summary based on the information.>

You must output exactly in the format, beginning with "Summary:" when you output your new summary.
Keep your entire summary within 256 tokens.

Figure 8: The Context prompt of the Teacher agent on HotpotQA.

HotpotQA Student Thought & Action Prompt

You are the Student agent. Your task is to answer the question under the guidance of the Teacher agent and the given information. First, you should make a reasonable plan and choose the appropriate action.
Context refers to the summary of previous interactions.
Question is the question you need to find the answer to.
Observation is the last action you took and the guidance of the Teacher agent.
Budget refers whether you can search or must submit the answer.
Thought should reason about the current situation and determine what action to take.
Action can be of two types:
(1) Search[entity], which invokes a local searcher to provide relevant information about the entity.
(2) Finish[answer], which returns the final answer and completes the task.

Please note:
If you need more information to answer the question, output "Search[entity]".
If you find enough information to answer the question, output "Finish[answer]".
When submitting your answer, please try to submit the full answer if you think it is ambiguous, e.g. "movie director" is better than "director"!
The answer to the question should be as accurate and concise as possible, i.e. try not to answer the question with long sentences. Please answer the yes-no question with either "yes" or "no".
Be aware of your search time limit. If you are told in the "Budget" that you must answer the question, return the action "Finish[answer]".

[INPUT]
Context: {context}
Question: {question}
Observation: {observation}
Budget: {budget}

[OUTPUT REQUIREMENTS]
Your output must strictly follow the format:

Thought: <Your thinking process on your action.>
Action: <Your action on the current situation.>

You must output exactly in the format, beginning with "Thought:" when you output your thought, and beginning with "Action:" when you output your action.
Keep your entire output within 256 tokens.

Figure 9: The Thought & Action prompt of the Student agent on HotpotQA.

HotpotQA Student Context Prompt

You are the Student agent. Your task is to answer the question under the guidance of the Teacher agent.
Now you are provided with a previous summary, as well as new messages that were not included in the original summary. You need to make a new summary.
Your summary should encapsulate the main points of the new messages and integrate them into the existing summary to create a comprehensive recap. Highlight the key issues discussed, decisions made, and any actions assigned. Record the helpful factual information given by the search engine.

[INPUT]
Question: {question}
Previous Summary: {context}
New Observation: {observation}
New Thought: {thought}
New Action: {action}

[OUTPUT REQUIREMENTS]
Your output must strictly follow the format:

Summary: <Your new summary based on the information.>

You must output exactly in the format, beginning with "Summary:" when you output your new summary.
Keep your entire summary within 256 tokens.

Figure 10: The Context prompt of the Student agent on HotpotQA.

MMLU Teacher Prompt

You are the Teacher agent. Your goal is to review the Student agent's current solution to a multiple-choice question and provide targeted feedback, and decide whether the solution is correct and ready for submission.

[INPUT]
Here is the multiple-choice question and the options:
{question}
Here is the Student agent's current choice and rationale:
{current_solution}

[OUTPUT REQUIREMENTS]
Your output must strictly follow this format:

Feedback: <Your feedback on the Student's last round choice and rationale.>
Submit: <Decide whether the choice is correct to be submitted, 0 (not enough to submit) or 1 (enough to submit).>

You must output exactly in the format, beginning with "Feedback:" and "Submit:" when you output feedback and submit.
Keep your entire output within 512 tokens.

Figure 11: The prompt of the Teacher agent on MMLU.

MMLU Student Prompt

You are the Student agent. Your task is to read a multiple-choice question and provide the most likely correct answer based on the options given, revising your previous answer according to the Teacher's feedback.

[INPUT]
Here is the multiple-choice question and options:
{question}
Here is your last round choice and rationale:
{last_solution}
Here is the Teacher agent's feedback of your last round choice:
{last_feedback}

[OUTPUT REQUIREMENTS]
Your output must strictly follow the format:

Rationale: <Your analysis process on the question, incorporating previous rationale and feedback.>
Choice: <Your final choice, A or B or C or D.>

You must output exactly in the format, beginning with "Rationale:" and "Choice:" when you output rationale and choice.
Keep your entire output within 512 tokens.

Figure 12: The prompt of the Student agent on MMLU.

HotpotQA & MMLU System Prompt

You are a router that must output only a single JSON object that matches the schema {"teacher": <id>, "student": <id>}.
The integers must be valid model IDs listed in the prompt.
No explanation, no extra text. If unsure, still output a valid JSON object.

Figure 13: The system prompt for routing on both HotpotQA and MMLU.

HotpotQA User Prompt

Dataset Background
The dataset we use is HotPotQA, where each data point includes a question and relevant information. We employ a multi-agent system comprising a Teacher and a Student agent to answer these questions. The agents alternate actions until either the search budget is exhausted or the Student provides a final answer.

Student Agent Introduction
The Student agent is responsible for searching for information and answering the question, guided by the Teacher agent. The Student's action space includes:
- Search[entity]: This action uses a retrieval tool to search for relevant information from the dataset, constructed using the SimCSE model (unsup-simcse-roberta-base).
- Finish[answer]: This action returns the final answer and concludes the task.

Teacher Agent Introduction
The Teacher agent evaluates and guides the Student agent by analyzing its steps and providing feedback. The Teacher can:
- [Continue]: Indicate that the Student's current step is correct and they should proceed.
- [Rethink]: Indicate that the Student's action is incorrect and they should reconsider their approach.

Specific Question
{question}

Candidates
0: {model_0}
1: {model_1}

Task
Assign a model to each agent to maximize performance.
Return exactly one JSON object on a single line:
{{"teacher": <id>, "student": <id>}}
where <id> is an integer, 0 or 1. Do not include any extra text.

Figure 14: The user prompt for routing on HotpotQA.

MMLU User Prompt

Dataset Background
The dataset we use is MMLU, where each data point includes a multiple-choice question with the options. We employ a multi-agent system comprising a Teacher and a Student agent to solve these questions. The agents alternate actions until either round limit 3 is exhausted or the Teacher thinks the current choice of the Student is correct enough to be submitted.

Student Agent Introduction
The Student agent is responsible for solving the question, guided by the Teacher agent. It will be given the question with the options, its last round solution and Teacher's feedback on its last round solution. The Student needs to provide a new better solution.

Teacher Agent Introduction
The Teacher agent evaluates and guides the Student agent by analyzing its current solution and providing feedback. It will be given the question with the options and Student's current solution. The Teacher needs to provide feedback on the solution.

Specific Question With the Options
{question}

Candidates
0: {model_0}
1: {model_1}

Task
Assign a model to each agent to maximize performance.
Return exactly one JSON object on a single line:
{{"teacher": <id>, "student": <id>}}
where <id> is an integer, 0 or 1. Do not include any extra text.

Figure 15: The user prompt for routing on MMLU.

J Broader Impacts

HAPS is intended to improve the effectiveness and efficiency of multi-model LLM deployment by routing queries to suitable models and adapting selected open-source models with lightweight generated parameters. This may reduce unnecessary use of stronger or more expensive models and make adaptive routing systems more accessible. At the same time, stronger automated LLM routing systems inherit the risks of their underlying models, including incorrect, biased, or harmful outputs when deployed without appropriate monitoring. Practical deployments should therefore combine HAPS with task-specific safety checks, model-use policies, and human oversight, especially when used in sensitive domains. Since HAPS modifies model behavior through generated parameters, such checks should be applied to the adapted models rather than only to their frozen base versions.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction clearly state the main contribution of HAPS: jointly routing over LLM architectures and input-conditioned parameter adaptations. They also summarize the hierarchical design, training objective, and experimental scope, which are supported by the main results and ablation studies in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: We include a dedicated Limitations section in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A].

Justification: This paper proposes an empirical LLM routing framework and does not present any formal theoretical results, theorems, assumptions, or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes].

Justification: We provide the information needed to reproduce the main results in Sections 2–3.3 and Appendices A–D and I, including the model architecture, data construction, prompts, baseline adaptations, and the training and evaluation procedures used in all experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: We release our code at the link given in the abstract, and describe the data, implementation details, prompts, and evaluation protocols in the appendix.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: We specify the experimental setup in Section 4, provide additional details on datasets, model pairs, hyperparameters, training, and evaluation in Appendices B–D, and list all agent and router prompts used on both benchmarks in Appendix I.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

   Justification: We do not report error bars or formal significance tests because the main experiments require expensive multi-agent LLM inference. Following common practice in comparable LLM routing studies, we instead report results on a separate test set together with extensive ablations and diagnostic analyses; Appendix E additionally reports the mean and standard deviation over three random seeds for the single-agent experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: We report compute-related information in Appendix D and Table 3, including the GPU type used for main training runs, representative training time, and inference latency.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The research uses public benchmarks and existing LLMs, does not involve human subjects, and fully conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes].

Justification: We discuss potential positive and negative societal impacts in Appendix J.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: We do not release pretrained models or datasets that pose a high risk of misuse; the released code only builds on existing publicly available models and benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

    Justification: We cite them in the paper properly.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes].

    Justification: The paper introduces code and experimental artifacts for HAPS; the method, prompts, training procedure, and evaluation protocol are documented in the paper and appendix, and the code is released in our public GitHub repository linked in the abstract.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: This work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: This work does not involve crowdsourcing or research with human subjects, so IRB approval or equivalent review is not applicable to this work.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes].

    Justification: LLMs are core components of the proposed routing framework and experimental environment. Their roles as candidate models, high-level router backbone, and Teacher–Student agents are described in Sections 2–3.3 and Appendix B.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.