HierRouter: Coordinated Routing of Specialized Large Language Models via Reinforcement Learning

Nikunj Gupta[1*] **Bill Guo**[1*] **Rajgopal Kannan**[2] **Viktor K. Prasanna**[1]
[1] University of Southern California
[2] DEVCOM Army Research Office
{nikunj, billguo, prasanna}@usc.edu
{rajgopal.kannan.civ}@army.mil

Abstract

Large Language Models (LLMs) deliver state-of-the-art performance across many tasks but impose high computational and memory costs, limiting their deployment in resource-constrained or real-time settings. To address this, we propose HierRouter, a hierarchical routing approach that dynamically assembles inference pipelines from a pool of specialized, lightweight language models. Formulated as a finite-horizon Markov Decision Process (MDP), our approach trains a Proximal Policy Optimization (PPO)-based reinforcement learning agent to iteratively select which models to invoke at each stage of multi-hop inference. The agent conditions on the evolving context and accumulated cost to make context-aware routing decisions. Experiments with three open-source candidate LLMs across six benchmarks, including QA, code generation, and mathematical reasoning, show that HierRouter improves response quality by up to **2.4**× compared to using individual models independently, while incurring only a minimal additional inference cost on average. These results highlight the promise of hierarchical routing for cost-efficient, high-performance LLM inference. All codes can be found here `https://github.com/Nikunj-Gupta/hierouter`.

1 Introduction

The ever-growing scale of large language models (LLMs) has led to escalating computation and memory demands during inference, posing significant challenges for real-time deployment and scalability. These costs are particularly prohibitive in latency-sensitive or resource-constrained settings, where the time and energy required to generate responses may outweigh the benefits of improved accuracy. While architectural techniques such as mixture-of-experts [7] and speculative decoding [44] have been proposed to reduce token-level compute, they are often tightly coupled to specific model internals and offer limited flexibility or interpretability. Alternatively, there is growing interest in higher-level coordination strategies that can operate on a pool of specialized models to maintain task performance while systematically reducing resource usage.

In parallel with efforts to optimize inference efficiency, there has been a growing trend toward the development and deployment of smaller language models that are tailored for specific domains or task types. These models leverage architectural innovations and training efficiencies to maintain high accuracy while drastically reducing computational and memory costs. Recent studies have demonstrated that, when carefully specialized, smaller models can rival or even outperform large-scale models on tasks aligned with their training objectives [29, 42, 22]. For example, compact models tuned for question answering [37, 15], code generation [39, 19, 1, 17], or mathematical reasoning [35, 6, 31, 3] often outperform general-purpose LLMs of much larger size on those respective tasks.

*Equal contribution. Corresponding author: Nikunj Gupta {nikunj@usc.edu}

Preprint. Under review.

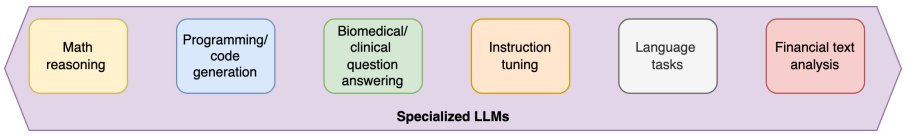

Figure 1: Specialized LLMs are smaller models fine-tuned for specific domains or task types, such as code generation, math reasoning, or biomedical QA. Unlike general-purpose LLMs, these models exhibit high efficiency and competitive accuracy on aligned tasks. By coordinating such models through routing rather than relying on monolithic architectures, systems like HierRouter can support adaptive, low-cost inference while preserving task performance.

This performance gain is attributed to task-specific representation learning and focused data curation, which allows these models to generalize efficiently within narrower operational scopes. As a result, the AI community has begun exploring ways to coordinate collections of such specialized models through routing or ensembling mechanisms rather than relying solely on increasingly larger monolithic systems.

To effectively leverage the capabilities of smaller, specialized models, two broad coordination strategies have emerged: model assembly and routing. Assembly-based approaches aggregate outputs from multiple models via voting [45], response fusion [23, 43, 25], or learned interpolation [38]. While these techniques enhance robustness by combining model strengths, they often require querying several models simultaneously, undermining efficiency and negating the benefits of using smaller models. For instance, frameworks like DeepEn [23] fuse outputs with task-specific transfer matrices but introduce substantial memory and runtime overhead. In contrast, routing-based approaches [9, 28, 12] select a single model (or a small subset) per query to minimize redundancy. However, most rely on one-shot decisions made upfront, limiting adaptability for complex queries requiring iterative reasoning or dynamic cooperation. Cascading approaches [8] offer partial adaptivity, triggering stronger models when simpler ones fall short, but still depend on rigid heuristics and lack contextual flexibility. These limitations highlight the need for a flexible, multi-step routing framework that treats inference as a sequential decision process, enabling context-aware model selection, refinement of partial answers, and improved quality-cost tradeoffs through adaptive reasoning.

We propose HierRouter, a reinforcement learning (RL)-based hierarchical routing approach that dynamically orchestrates a pool of specialized language models in a multi-hop architecture. At each step of inference, a learned router selects a single model from the candidate pool to process the current query or intermediate context. The selected model generates a response, which is appended to the context and passed forward to the next routing stage. This process continues for a fixed number of hops, enabling compositional reasoning and progressive refinement of the response. Crucially, our RL-based router is trained to optimize a reward function that balances task performance against cumulative inference cost. Unlike prior approaches that rely on static ensembling, one-shot routing, or brittle reward alignment, our router learns a context-aware policy that adapts its decisions based on both semantic progress and resource usage. By framing routing as a reinforcement learning task over a finite-horizon Markov Decision Process, we enable generalization to novel queries and dynamic model pools while ensuring robust model coordination over time. This structure captures model specialization and supports incremental inference, where earlier hops address simpler subproblems and later hops escalate complexity as needed. Compared to prior methods, HierRouter offers a more interpretable and cost-efficient coordination mechanism: it builds answers step-by-step, leverages diverse model strengths, and allocates compute resources only where necessary. This mirrors how human experts solve problems through gradual synthesis, consultation, and refinement, making HierRouter a more natural and effective strategy for LLM orchestration.

- We develop a novel hierarchical routing mechanism that iteratively selects specialized models at each hop to incrementally refine the response over a fixed inference horizon.
- Our PPO-based router is trained using a sparse reward that balances task accuracy against cumulative cost, enabling flexible adaptation to both semantic content and compute budgets.
- We conduct extensive experiments across six diverse benchmarks, including question answering (MMLU, ARC), code generation (MBPP), and mathematical reasoning (GSM8K, MATH), and show that HierRouter improves average quality score by up to **107%** compared to the best small model, while maintaining reasonably close inference cost compared to

individual small (Qwen2.5-Coder-3B, DeepSeek-R1-Distill-Qwen-1.5B, Phi-3.5-mini) and bigger LLMs (Llama-3.1-8B, Qwen2.5-14B).

2 Related works

Inference Cost and Efficiency in LLMs. While LLMs demonstrate impressive performance across a wide array of tasks, their substantial inference costs, measured in latency, memory, and energy consumption, pose significant challenges for deployment, especially in real-time or resource-constrained settings. Several efforts have focused on reducing these costs through model compression techniques such as pruning, quantization, and knowledge distillation [30, 13, 26]. Others have explored dynamic inference strategies, including early exiting [14], input-dependent routing [9], and expert selection in mixture-of-experts architectures [7]. Despite these advances, balancing task performance with compute efficiency remains a critical challenge, particularly in scenarios involving diverse or unseen inputs where static optimization methods fall short.

Rise of Small Specialized Models. The growing ecosystem of task-specialized language models reflects a broader trend toward modularity and efficiency in LLM deployment. Recent works have introduced lightweight models fine-tuned for specific domains such as programming [39, 19, 1, 17], math reasoning [35, 6, 31, 3], or retrieval-augmented QA [4, 27]. These models often outperform larger, general-purpose models in their respective domains, while requiring significantly fewer resources. This has motivated research into systems that intelligently coordinate such models. Our work builds on this paradigm by proposing a reinforcement learning-based router that learns to compose and coordinate specialized small LLMs in a cost-sensitive, multi-hop inference pipeline.

Model Coordination Strategies. Coordinating multiple models for improved inference has been explored through both ensembling and routing paradigms. Ensembling-based approaches typically aggregate outputs from multiple models, either via majority voting, confidence weighting, or learned fusion, to boost accuracy or robustness [45, 23, 43, 25, 38]. While effective, ensembles are computationally expensive due to redundant model evaluations. In contrast, routing-based approaches aim to select a single model (or a subset) per input, reducing cost while maintaining accuracy [9, 28, 12, 32]. These methods often rely on heuristics or input similarity measures to assign queries to appropriate models. Moreover, most routing strategies are shallow (one-shot) and lack mechanisms for iterative refinement or composition, limiting their expressivity and adaptability in complex reasoning tasks.

LLM Cascading. Cascading strategies execute LLMs in sequence, typically escalating queries from smaller to larger models based on fixed heuristics or confidence thresholds [8, 46, 41, 2]. These systems aim to reduce compute by resolving easy queries early, but are often rule-based, brittle to distribution shifts, and difficult to adapt to heterogeneous tasks. Moreover, model composition in cascading pipelines is typically overwrite-based: later models discard earlier outputs and reprocess the query from scratch, limiting opportunities for incremental reasoning. In contrast, our work introduces a trainable routing agent that conditions model selection on evolving context and accumulated cost, supporting multi-hop composition and adaptive refinement across inference steps.

Policy Optimization in RL. Policy gradient methods form the foundation of modern RL for decision-making tasks. Among them, Proximal Policy Optimization (PPO) [34] has emerged as a robust and sample-efficient algorithm widely used in natural language and control domains. PPO improves training stability by clipping updates to the policy objective, making it well-suited for sparse reward settings such as in this paper. In this work, we leverage PPO to train our routing agent.

3 Methodology

In this section, we describe our approach, HierRouter, for hierarchically and adaptively selecting from a pool of specialized LLMs. The goal is to learn a cost-aware policy that adaptively selects models at each inference step to balance output quality and compute efficiency. Figure 2 illustrates the high-level architecture of HierRouter. The methodology consists of three main components: (i) the formalization of the routing process as a Markov Decision Process, (ii) the design of the router policy

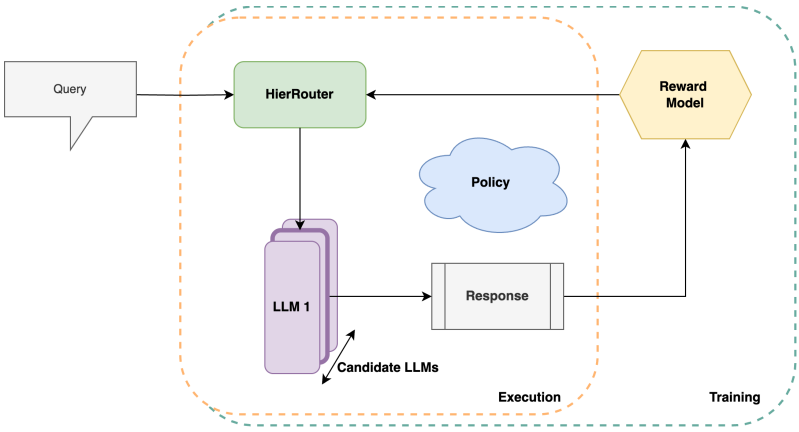

Figure 2: Overview of the HierRouter architecture. At each of the L inference stages, a PPO-based (RL) router selects one model from a pool of specialized LLMs to respond to the current context. The model's output is appended to the evolving context, which is passed to the next router layer. This multi-hop process enables compositional reasoning: earlier hops handle simpler sub-tasks, while later hops refine the output. The routing policy is trained to optimize response quality under inference cost constraints, balancing accuracy and efficiency through sequential model coordination.

network for model selection, and (iii) the PPO-based training procedure for learning cost-sensitive, context-aware routing strategies.

3.1 MDP Formulation

To jointly optimize response quality and computational efficiency in the context of hierarchical LLM routing, we formalize the problem as a finite-horizon Markov Decision Process (MDP):

$$\mathbb{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, L \rangle,$$

where the agent operates over a structured decision space to construct a compositional inference pipeline. At each decision step, the agent selects one model from a fixed pool of M specialized LLMs. The process unfolds over a fixed number of L stages, reflecting the bounded depth of the routing hierarchy. This formulation captures the essence of dynamic inference in multi-model systems, where partial outputs at each stage serve as intermediate context for future decisions. The agent must reason over evolving semantic representations while balancing two competing objectives: (i) maximizing the quality of the final output, as judged by an external or learned evaluator, and (ii) minimizing the total computational cost, typically measured via token usage or latency.

State space. At each decision stage t, the agent observes a state $s_t = (c_t, \ell_t, C_t)$, which compactly encodes all information necessary for selecting the next routing action. The component c_t denotes the context history - a growing textual sequence formed by concatenating the original user query with the responses generated by each selected model up to the current point. This evolving context serves both as the input to subsequent models and as a representation of the semantic progress made so far. The second component, $\ell_t \in \{0, \ldots, L-1\}$, indicates the current depth in the decision process, reflecting how many stages of the routing hierarchy have been executed. This acts as a surrogate for the residual budget or inference horizon. Finally, $C_t \in \mathbb{R}_{\geq 0}$ captures the cumulative inference cost incurred so far, such as token-FLOPs or latency-normalized resource usage, which directly influences the reward signal. Together, these three components define a state space that balances semantic reasoning and budget awareness. The agent must learn to interpret c_t to assess what aspects of the query have been resolved, use ℓ_t to track the decision horizon, and consider C_t to control computational efficiency.

Action space. At each decision point t, the agent selects an action $a_t \in \mathcal{A}$, which corresponds to choosing a model $m_t \in \{0, \ldots, M-1\}$ from the pool of M available language models to invoke next. Since the routing process proceeds for a fixed number of stages L, the action space is simply of size $|\mathcal{A}| = M$. This formulation enables the agent to focus on model selection at each hop without needing to reason about termination decisions. Crucially, by learning context-dependent model choices across

stages, the agent can implement non-myopic routing strategies that optimize for long-term trade-offs between inference cost and final answer quality, leveraging the diverse capabilities of the underlying model ensemble.

Transition dynamics. The environment evolves deterministically according to the agent's action and the behavior of black-box components [2] for text generation and cost estimation (algorithm included in Appendix 2). We define the `Gen` function as a black-box model inference interface that takes the current context c_t and a selected model m_t, and returns a textual response $r_t = \texttt{Gen}(m_t, c_t)$. Each model has a predefined or dynamically inferred inference schema, including instruction formatting and tokenizer handling, which is abstracted from the agent. The `Cost` function computes the normalized inference cost $\delta_t = \texttt{Cost}(m_t, c_t)$, using token-level statistics and per-model pricing metadata. For each call, we approximate the cost as:

$$\delta_t = \text{base\_rate}(m_t) \cdot (\texttt{num\_tokens\_in} + \texttt{num\_tokens\_out}),$$

where the base rate is obtained from either a user-supplied configuration file or estimated heuristically based on model size (e.g., 3B, 7B, 13B) when exact pricing is unavailable. The environment ensures cost units are normalized (e.g., USD/token or FLOPs/token), making the resulting penalty comparable across diverse architectures. This abstraction enables the router to reason over both semantic and budgetary signals in a unified reward formulation.

Specifically, upon selecting action $a_t = m_t$ at stage t, the agent queries model m_t with the current context c_t. This results in a new textual response $r_t = \texttt{Gen}(m_t, c_t)$, generated by the black-box function `Gen`, and an associated computational cost $\delta_t = \texttt{Cost}(m_t, c_t)$, computed by a black-box estimator `Cost`. The cost function typically reflects a proxy for inference effort such as normalized token-level FLOPs, model-specific latency, or memory usage. The environment then transitions to the next state $s_{t+1} = (c_{t+1}, \ell_{t+1}, C_{t+1})$ where: $c_{t+1} = c_t \,\|\, r_t, \quad \ell_{t+1} = \ell_t + 1, \quad C_{t+1} = C_t + \delta_t$. That is, the context is extended via concatenation with the new response r_t, the layer index is incremented by one to reflect deeper traversal into the inference pipeline, and the cumulative cost is updated to include the additional compute incurred by m_t. An episode ends when the routing depth reaches its predefined maximum $\ell_{t+1} = L$. These transitions enforce a monotonic structure: context and cost only grow, and depth strictly increases. Conceptually, this models a progressive assembly of the final output, where each model's contribution builds upon the prior state and incurs additive inference cost. By encoding both semantic and resource dynamics in state transitions, the agent can learn to balance output refinement against compute efficiency over time.

Reward function. We define a sparse terminal reward that captures the trade-off between response quality and inference cost. The agent receives no intermediate rewards and is evaluated only at the final step $T = L$, corresponding to the last stage of the multi-hop routing process. The reward function is given by:

$$R(s_T, a_T) = Q(c_T) - \alpha C_T,$$

where $Q(c_T) \in [0, 1]$ denotes the quality of the final response as assigned by an external evaluator (e.g., a reward model, task metric, or oracle label), and $C_T \in \mathbb{R}_{\geq 0}$ represents the total cumulative cost incurred over the trajectory. The cost penalty coefficient $\alpha > 0$ controls the balance between performance and efficiency. This formulation encourages policies that produce high-quality answers while minimizing inference cost, and supports deployment in settings constrained by latency, memory, or energy budgets. The reward is agnostic to the specific form of the evaluator $Q(\cdot)$, allowing flexibility in incorporating ground-truth metrics or learned preference models across different tasks.

3.2 HierRouter: Multi-Hop Model Routing

We adopt an episodic learning framework in which the agent receives reward only at the terminal timestep. Each episode consists of exactly L decision steps, corresponding to the fixed depth of the hierarchical routing procedure. This encourages the agent to be judicious in model selection at each step, as all hops contribute to the final reward. To learn routing behaviors that optimize the trade-off between response quality and computational cost, we train a parameterized stochastic policy $\pi_\theta(a \mid s)$, along with a value function baseline $V_\phi(s)$, using Proximal Policy Optimisation

[2] By "black-box functions" we mean the policy interacts with them only via inputs/outputs, without access to their internal mechanisms. This allows flexible integration of heterogeneous models and cost estimators.

Algorithm 1: HierRouter: Hierarchical Multi-Hop Routing with PPO

Input : Pool of models $\mathcal{M} = \{m_1, \ldots, m_M\}$, max hops L, reward weight α
Output Trained policy $\pi_\theta(a|s)$ and value function $V_\phi(s)$
:

foreach *training episode* **do**
  Initialize context c_0, cost $C_0 \leftarrow 0$, depth $\ell_0 \leftarrow 0$;
  for $t = 0$ **to** $L - 1$ **do**
    Construct state $s_t = (c_t, \ell_t, C_t)$;
    Sample model $m_t \sim \pi_\theta(\cdot \mid s_t)$;
    Query model: $r_t \leftarrow \texttt{Gen}(m_t, c_t)$;
    Compute cost: $\delta_t \leftarrow \texttt{Cost}(m_t, c_t)$;
    Update context: $c_{t+1} \leftarrow c_t \| r_t$;
    Update cost: $C_{t+1} \leftarrow C_t + \delta_t$;
    Update depth: $\ell_{t+1} \leftarrow \ell_t + 1$;
    Store transition $(s_t, m_t, \delta_t, r_t)$;
  Evaluate final output $Q(c_L) \in [0, 1]$;
  Compute terminal reward: $R \leftarrow Q(c_L) - \alpha C_L$;
  Compute advantage estimates using GAE [33];
  Update π_θ and V_ϕ using PPO loss [34];

(PPO) [34] (where θ and ϕ represent the parameters of the policy and value networks, respectively). The objective is to maximize the expected episodic return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t)\right],$$

where trajectories $(s_0, a_0, \ldots, s_T, a_T)$ are sampled under the current policy π_θ. Since rewards are sparse and delayed until termination, policy gradients must propagate credit through sequences of compute-augmenting decisions. This formulation enables the agent to discover adaptive routing schemes that leverage the heterogeneous strengths of the model ensemble, while automatically adapting to task difficulty and remaining within stringent compute budgets (see Algorithm 1).

We implement the routing agent as a shared neural network that jointly parameterizes the policy $\pi_\theta(a \mid s)$ and value function $V_\phi(s)$. For each input state $s_t = (c_t, \ell_t, C_t)$, we encode c_t using a frozen Sentence-Transformer [36], and represent ℓ_t via a learned stage embedding. These are concatenated to form a joint state representation. This representation is fed into a two-layer MLP to produce (i) action logits over the model pool, and (ii) a scalar value estimate.

At inference time the router executes a *multi-hop* decision process governed by the learned policy $\pi_\theta(a_t \mid s_t)$. Starting from the initial state $s_0 = (c_0, \ell_0{=}0, C_0{=}0)$, the agent iteratively selects an action $a_t = m_t \in \{0, \ldots, M-1\}$, where m_t indexes the next LLM to query. The chosen model receives the current context c_t and produces a reply $r_t = \texttt{Gen}(m_t, c_t)$. We append this reply to yield the updated context $c_{t+1} = c_t \,\|\, r_t$, increment the stage counter $\ell_{t+1} = \ell_t + 1$, and accumulate cost $C_{t+1} = C_t + \texttt{Cost}(m_t, c_t)$. As the context is continually enriched by earlier replies, deeper hops operate on a progressively more informative input. This feedback enables *refinement*: specialised models correct or elaborate on partial answers produced by shallower models, while the router monitors accumulated cost C_t to decide whether further improvement justifies additional computation. Figure 2 visualises the resulting decision stack. The episode concludes after L hops, at which point the reward $R(s_T, a_T)$ defined in Section 3.1 is issued (only when training). The sparse, final-state reward encourages the policy to discover routing strategies that attain high-quality answers with minimal compute by selecting efficient sequences of model invocations.

4 Experiments

Candidate LLMs. The router operates over a curated pool of three specialized instruction-tuned LLMs, each offering complementary strengths: *Qwen2.5-Coder-3B* [24], tailored for efficient code understanding and generation; *DeepSeek-R1-Distill-Qwen-1.5B* [18], a general-purpose model distilled

for lightweight reasoning (math and programming); and *Phi-3.5-mini* [1], optimized for compact instruction following and broad generalization. These serve as the decision space for the hierarchical policy during both training and inference. To contextualize the performance and cost trade-offs, we additionally evaluate the router against two larger models: *Llama-3.1-8B* [16] and *Qwen2.5-14B* [40]. These are not part of the routing pool but are included in evaluation to highlight the benefits of routing over smaller, specialized models versus direct usage of larger monolithic ones. All models are queried under a consistent configuration (see appendix).

Dataset Construction. We train the router on a curated set of six tasks spanning math reasoning, program synthesis, and general QA. Each dataset is capped at 300 examples (randomly sampled) to ensure balanced task coverage while keeping the overall training lightweight. This setup is sufficient for learning a performant routing policy due to the shared representation backbone and task-agnostic reward function. The dataset pool includes: *GSM8K* [11], *MATH* [21], *MBPP* [5], and *ARC* [10] (easy and challenge), and *MMLU* [20]. MMLU queries are reformatted to use consistent multiple-choice labels (A/B/C/D). We use a random ∼70%-30% train-test split, ensuring consistent evaluation without overlap. Evaluation is conducted on the disjoint held-out test sets from the same task domains to ensure fair comparisons across all experiments, including static routing with small or large LLMs and our hierarchical router. While the experiments are run with relatively smaller subsets of the data points for demonstration, the framework is designed for seamless scalability to larger datasets and tasks. The reported metric values are interpreted comparatively to highlight trade-offs in quality and cost across strategies.

Baselines. To assess the effectiveness of our router, we compare it directly against each constituent model in isolation, both from the router's internal pool of smaller LLMs and a set of larger, higher-capacity LLMs. These comparisons reveal how the router's selective multi-hop coordination compares to running small or large models in a fixed, single-shot manner. Through these experiments, our focus is to demonstrate that: (i) a lightweight router trained over small, specialized models can match or surpass larger models in performance, and (ii) the same setup provides reasonable efficiency gains under identical conditions. This setup is necessary and sufficient to validate the router's ability to dynamically trade off quality and cost across tasks of varying complexity.

Implementation details. The router is implemented as a two-layer feedforward neural network that jointly parameterizes both the policy and value function. At each decision step, the input state consists of three components: (i) an embedding of the evolving context c_t, obtained using a frozen Sentence-Transformer encoder [36]; (ii) a learned stage embedding representing the current routing depth ℓ_t; and (iii) a scalar indicating cumulative cost C_t. These features are concatenated into a unified state vector and passed through a multilayer perceptron to produce model-selection logits and value predictions. Training is performed using PPO with generalized advantage estimation (GAE) [33], using $\gamma = 0.99$, bias-variance trade-off parameter $\lambda = 0.95$, clip parameter 0.2, and entropy regularization coefficient 0.01. At the end of each routing episode the agent receives a terminal reward $R = Q(c_T) - \alpha C_T$. The task-specific quality score $Q(c_T)$ is computed by a ground-truth evaluator, which compares the final output to reference answers using an appropriate metric for the task. The evaluator includes normalization routines (e.g., punctuation stripping, case folding) and pattern-matching logic for robust label extraction. This design ensures metric consistency across heterogeneous benchmarks while maintaining reward granularity sufficient for policy optimization. The total inference cost accumulated over the episode, C_T, is normalized using static weights proportional to model size (i.e., parameter count), for fair comparison of compute efficiency across models of varying scale. The cost penalty $\alpha = 0.005$ is a hyperparameter and was tuned empirically.

5 Results

We evaluate HierRouter across six diverse benchmark datasets: **MMLU**, **GSM8K**, **MBPP**, **MATH**, **ARC-Easy**, and **ARC-Challenge**, covering general knowledge, mathematical reasoning, and code generation. Table 1 reports the average test quality score per dataset for three model groups: (i) *large LLMs* (Qwen2.5-14B and Llama-3.1-8B), (ii) *candidate small LLMs* included in the routing pool (Qwen2.5-Coder, DeepSeek-R1-Distill-Qwen-1.5B, and Phi-3.5), and (iii) our proposed method, *HierRouter*, which dynamically selects a model per query via a multi-hop routing policy.

For all experiments, we report the *quality score* (Table 1) as the token-level F1 score between the model-generated answer and the reference answer, consistent with common evaluation practices for open-ended QA and reasoning tasks. F1 provides a more robust and informative signal than an exact match, which can be too brittle in some cases (e.g., math derivations or code outputs with lexical variation). The evaluation is performed using a unified reward interface with dataset-specific ground truth answers. The *inference cost* is computed via a normalized pricing scheme that scales with the number of tokens and a model-specific base rate. The router's average cost per dataset (Table 2) reflects its actual per-query compute usage, offering a realistic picture of test-time efficiency.

Table 1: Test quality score of HierRouter candidate small LLMs, and selected large LLMs across six benchmark datasets. HierRouter dynamically routes across models to optimize performance.

Model	MMLU	GSM8K	MBPP	MATH	ARC-Easy	ARC-Challenge
Bigger LLMs						
Qwen2.5-14B-Instruct	0.003	0.077	0.028	0.076	0.003	0.003
Llama-3.1-8B-Instruct	0.005	0.06	0.025	0.057	0.006	0.003
Candidate LLMs						
Qwen2.5-Coder-3B-Instruct	0.002	0.067	0.030	0.069	0.003	0.003
DeepSeek-R1-Distill-Qwen-1.5B	0.003	0.054	0.016	0.053	0.002	0.002
Phi-3.5-mini-instruct	0.004	0.067	**0.044**	0.068	0.003	0.003
HierRouter	**0.005**	**0.139**	0.029	**0.105**	**0.009**	**0.010**

Quality Gains from Routing. As seen in Table 1, HierRouter consistently outperforms all candidate small LLMs and often even the larger ones. On GSM8K, HierRouter achieves a quality score of **0.139**, more than doubling the best candidate baseline (Phi-3.5 at 0.067) and outperforming Llama-3.1-8B (0.060) and Qwen2.5-14B (0.077). On MATH, HierRouter yields a score of **0.105**, outperforming all small and large models, including Qwen2.5-14B (0.076). On MBPP, a code generation benchmark, HierRouter matches Qwen2.5-14B (0.028) and surpasses Llama-3.1-8B (0.025), while trailing the highest static score (Phi-3.5 at 0.044) by a modest margin. These results highlight how routing enables HierRouter to exploit complementarity across models without needing to re-train or ensemble them. The benefits of routing are especially clear in low-signal regimes like ARC-Challenge and ARC-Easy, where static models offer limited improvements beyond chance (e.g., scores around 0.002-0.003), HierRouter yields **0.010** and **0.009**, respectively, more than 3× better than the best-performing static candidate. These results highlight how multi-hop routing enables HierRouter to adapt to task difficulty and model strengths without requiring additional supervision or fine-tuning. Taken together, the results support our central hypothesis: dynamically routing queries over a diverse model pool yields higher quality responses than any fixed single-model strategy, including larger LLMs, while maintaining compute efficiency. **Comparison with bigger LLMs:** Table 1 also shows that even larger-scale models like Llama-3.1-8B and Qwen2.5-14B fall short of HierRouter on several datasets. For example, Llama-3.1-8B lags behind HierRouter by over 0.08 points on GSM8K and nearly 0.05 on MATH.

Table 2: Average inference cost of HierRouter, candidate small LLMs, and large LLMs.

Model	MMLU	GSM8K	MBPP	MATH	ARC-Easy	ARC-Challenge
Bigger LLMs						
Qwen2.5-14B-Instruct	0.0095	0.0089	0.0092	0.0098	0.0086	0.0100
Llama-3.1-8B-Instruct	0.0071	0.0078	0.0048	0.0077	0.0071	0.0096
Candidate LLMs						
Qwen2.5-Coder-3B-Instruct	0.0025	0.0030	0.0028	0.0031	0.0029	0.0030
DeepSeek-R1-Distill-Qwen-1.5B	0.0013	0.0015	0.0012	0.0016	0.0016	0.0017
Phi-3.5-mini-instruct	0.0025	0.0027	0.0021	0.0029	0.0027	0.0030
HierRouter	**0.0178**	**0.0163**	**0.0183**	**0.0159**	**0.0184**	**0.0188**

Efficiency and Cost Analysis. Each model is associated with a token-dependent inference cost, as defined earlier (in subsection 3.1) via a normalized pricing formulation that scales with the number of input/output tokens and a model-specific base rate. This dynamic cost is integrated into the reward

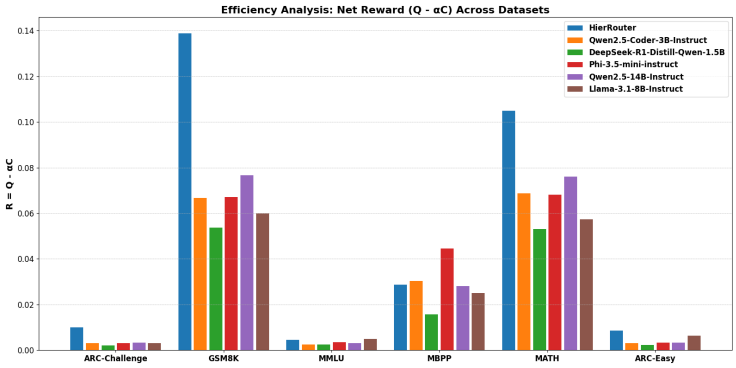

Figure 3: Comparing met score or reward comparison ($R = Q - \alpha C$). Q denotes average F1 quality score, C denotes average cost per query (token-based), and α is the cost penalty coefficient used during training. Despite slightly higher costs, HierRouter achieves superior or comparable net reward in all datasets, demonstrating efficient tradeoffs between performance and compute usage.

function used to train HierRouter, guiding it to balance output quality with inference efficiency. Table 2 presents the average token-normalized inference cost (per 1000 tokens) incurred by each model. As expected, the large models like Qwen2.5-14B and Llama-3.1-8B consistently incur higher compute costs (up to 0.01), while smaller candidates remain within the 0.001-0.003 range. In contrast, HierRouter incurs a moderate cost, between **0.0159** and **0.0188**, across all datasets. These values are higher than any single small model but substantially lower than invoking a very large LLM even once. This difference is a natural result of HierRouter's multi-hop routing framework. The router operates in two layers: the first generates initial decisions, and the second layer consumes enriched inputs that include prior outputs, embeddings, and query context. This increased input footprint leads to more tokens being processed overall, raising cost moderately. However, the benefit is clear: HierRouter's routing flexibility enables it to consistently deliver better-quality answers (Table 1), even outperforming the bigger models. To further quantify this tradeoff, Figure 3 reports the net score or reward $R = Q - \alpha C$, combining quality score Q and cost penalty C using a training-time cost coefficient α. Across nearly all tasks, HierRouter achieves the highest net reward, despite incurring slightly more cost, demonstrating that its dynamic policy makes efficient, context-sensitive use of compute. We argue that this tradeoff is both expected and desirable. The router pays a marginal premium for harder queries but avoids blanket overuse of high-cost models. In contrast, static baselines must commit to either efficiency (but low quality) or brute-force accuracy (with high, fixed cost). HierRouter sidesteps this tension with input-adaptive routing.

These results demonstrate the value of hierarchical, cost-aware routing. HierRouter's multi-hop policy architecture enables it to reason over both model specialization and budget constraints, leading to improved performance on a diverse set of tasks. Despite moderate increases in inference cost relative to small models, HierRouter achieves consistent quality gains, often exceeding the performance of individual candidates or bigger LLMs. This positions it as a scalable and practical alternative to brute-force LLM usage, making it particularly well-suited for deployment.

6 Conclusion

We introduced HierRouter, a hierarchical, RL-driven routing framework for composing specialized language models in a multi-hop inference pipeline. Unlike typical one-shot model invocation strategies, our method learns to make sequential, context-aware decisions that balance response quality with compute efficiency. By formulating the problem as an MDP and leveraging PPO for policy optimization, HierRouter effectively integrates task semantics and budget constraints into its routing logic. Extensive experiments across six diverse benchmarks demonstrate that HierRouter benefits from staged reasoning and model specialization and delivers substantial quality gains with minimal cost overhead. This paper affirms that RL-trained multi-hop routing is a promising alternative to brute-force scaling, enabling efficient and adaptive LLM deployment. Several future directions emerge. First, extending to dynamic model pools could enable lifelong adaptation to evolving domains. Second, integrating uncertainty-aware decision-making may enable robustness by deferring to stronger

models under low confidence. Finally, combining routing with early-exit strategies could further amplify efficiency by stopping inference when adequate certainty is reached early.

References

[1] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034, 2024.

[3] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

[4] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

[5] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[6] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024.

[7] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*, 2024.

[8] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024.

[9] 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, 2024.

[10] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[12] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2024.

[13] Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. Exploiting LLM quantization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[14] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, 2024.

[15] Alex Gichamba, Tewodros Kederalah Idris, Brian Ebiyau, Eric Nyberg, and Teruko Mitamura. Colbert retrieval and ensemble response scoring for language model question answering. *arXiv preprint arXiv:2408.10808*, 2024.

[16] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[17] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

[18] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[19] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[20] 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.

[21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

[22] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

[23] Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Ting Liu, and Bing Qin. Ensemble learning for heterogeneous large language models with deep parallel collaboration. *Advances in Neural Information Processing Systems*, 37:119838–119860, 2024.

[24] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[25] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics.

[26] Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, Jie Liu, Ge Zhang, Yanan Wu, Congnan Liu, et al. Ddk: Distilling domain knowledge for efficient large language models. *Advances in Neural Information Processing Systems*, 37:98297–98319, 2024.

[27] Suqing Liu, Zezhu Yu, Feiran Huang, Yousef Bulbulia, Andreas Bergen, and Michael Liut. Can small language models with retrieval-augmented generation replace large language models when learning computer science? In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 388–393, New York, NY, USA, 2024. Association for Computing Machinery.

[28] Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1964–1974, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[29] Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D Lane, and Mengwei Xu. Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790*, 2024.

[30] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

[31] Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. LILA: A unified benchmark for mathematical reasoning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[32] 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 from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025.

[33] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[35] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[36] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 33:16857–16867, 2020.

[37] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

[38] Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. UL2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2023.

[39] CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, et al. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*, 2024.

[40] Qwen Team. Qwen2.5: A party of foundation models, September 2024.

[41] Neeraj Varshney and Chitta Baral. Model cascading: Towards jointly improving efficiency and accuracy of NLP systems. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11007–11021, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[42] Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, et al. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350*, 2024.

[43] Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. Fusing models with complementary expertise. In *The Twelfth International Conference on Learning Representations*, 2024.

[44] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*, 2024.

[45] Joshua C Yang, Damian Dalisan, Marcin Korecki, Carina I Hausladen, and Dirk Helbing. Llm voting: Human choices and ai collective decision-making. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, volume 7, pages 1696–1708, 2024.

[46] Jieyu Zhang, Ranjay Krishna, Ahmed Hassan Awadallah, and Chi Wang. Ecoassistant: Using LLM assistants more affordably and accurately. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

Appendix

Appendix A: Environment Dynamics. Details the environment step function used to simulate multi-hop model invocation, reward calculation, and sequential decision-making.

Appendix B: PPO and Router Configuration. Covers training hyperparameters, architecture setup, routing prompts, and cost-based reward shaping.

Appendix C: Evaluation. Presents the token-level F1 score computation used to evaluate open-ended QA responses, including normalization and precision-recall logic.

Appendix A: Additional details on Environment Dynamics

This section outlines the custom environment step logic used to simulate hierarchical LLM routing as a sequential decision-making process. The design abstracts multi-hop routing as a finite-horizon MDP, where each decision selects a model and optionally terminates inference. The environment tracks the evolving query context, accumulates cost, and computes a final reward based on output quality and incurred cost.

Algorithm 2: HierRouter Environment Step Function

Input : Query q, action a_t at time t, ground truth (optional)
Output Next state s_{t+1}, reward r_t, done flag, info
:

if *first step* **then**
  Initialize context $c_0 \leftarrow q$, layer index $\ell_0 \leftarrow 0$, total cost $C_0 \leftarrow 0$;
if *multi-layer* **then**
  Parse action a_t to get model index m_t and stop flag $\texttt{done}_t$;
else
  Set $m_t \leftarrow a_t$, $\texttt{done}_t \leftarrow \texttt{True}$;
Obtain model response: $r_t \leftarrow \texttt{Gen}(m_t, c_t)$;
Compute cost: $\delta_t \leftarrow \texttt{Cost}(m_t, c_t)$;
Update total cost: $C_{t+1} \leftarrow C_t + \delta_t$;
Update context: $c_{t+1} \leftarrow c_t \| r_t$;
Increment layer: $\ell_{t+1} \leftarrow \ell_t + 1$;
if $\texttt{done}_t$ *or* $\ell_{t+1} = L$ **then**
  Evaluate response quality: $q_t \leftarrow Q(c_{t+1})$;
  Compute reward: $r_t \leftarrow q_t - \alpha C_{t+1}$;
  Set $\texttt{done} \leftarrow \texttt{True}$;
else
  Set $r_t \leftarrow 0$, $\texttt{done} \leftarrow \texttt{False}$;
Return state $s_{t+1} = (c_{t+1}, \ell_{t+1}, C_{t+1})$, reward r_t, done, and info;

Appendix B: PPO and Router Training Configuration

This section details the configuration used to train HierRouter via Proximal Policy Optimization (PPO). The hierarchical router operates with two layers and a semantic encoder to guide multi-hop decision-making. The reward incorporates token-level quality (e.g., F1) and model-specific cost penalties.

PPO Training Configuration. The policy is trained using PPO with the following key parameters: 8 iterations, 128 rollouts per iteration, 16 mini-batches, 4 training epochs per iteration, $\gamma = 0.99$,

GAE-$\lambda = 0.95$, clipping parameter 0.2, value loss coefficient 0.5, entropy coefficient 0.01, maximum gradient norm 0.3, and learning rate 10^{-4}. The optimizer uses Adam with $\epsilon = 10^{-4}$.

Router Architecture. The router comprises two decision layers. Inputs are embedded using a pre-trained encoder `sentence-transformers/all-mpnet-base-v2`[3] with a maximum sequence length of 512. Each layer is prompted with task-specific instructions:

- **Layer 0:** *"Describe the problem in detail, then plan how you would solve it. Analyze the problem step by step, identifying key constraints and requirements."*
- **Layer 1:** *"Using the previous analysis and plan, verify if the approach is correct and solve the problem methodically. Ensure completeness and correctness in your solution."*

Training Protocol. Training is performed on CUDA-enabled devices with a batch size of 96 for 200 epochs. A warm-up phase of 100 steps is used, with evaluations every 3 epochs and checkpoints every 5 epochs. A fixed seed of 42 ensures reproducibility.

Model Cost Configuration. Each LLM is associated with a normalized cost value used in computing the penalty term c:

Model	Cost
Qwen2.5-Coder-3B-Instruct	0.003
DeepSeek-R1-Distill-Qwen-1.5B	0.002
Phi-3.5-mini-instruct	0.003
Llama-3.1-8B-Instruct	0.008
Qwen-14B-Instruct	0.014

The reward function encourages economical routing while maintaining high answer quality.

Appendix C: Ground Truth Evaluator (F1 Score)

To evaluate model responses against known correct answers in tasks such as QA and math reasoning, we implement a token-level F1 score evaluator. This serves as the primary external reward model during both training and evaluation.

Algorithm 3: Evaluate(query, response, ground_truth)

Input : Query q, Model Response r, Ground Truth Answer(s) g
Output F1 score $\in [0, 1]$
:

Normalize r and g: lowercase, strip punctuation, standardize whitespace;
Tokenize $r \rightarrow r_{\text{tokens}}$, $g \rightarrow g_{\text{tokens}}$;
Compute intersection: $I \leftarrow r_{\text{tokens}} \cap g_{\text{tokens}}$;
if $|r_{tokens}| = 0$ *or* $|g_{tokens}| = 0$ *or* $|I| = 0$ **then**
    return 0.0
Compute precision: $p = \frac{|I|}{|r_{\text{tokens}}|}$;
Compute recall: $r = \frac{|I|}{|g_{\text{tokens}}|}$;
return $F1 = \frac{2 \cdot p \cdot r}{p + r}$

Notes. If multiple ground truth answers are available, we return the maximum F1 score among them. This formulation ensures robustness to lexical variation and partial correctness in free-form answers. The score serves as a reliable and differentiable reward signal for PPO optimization in our router framework.

[3] `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`