MTRouter: Cost-Aware Multi-Turn LLM Routing with History–Model Joint Embeddings

Yiqun Zhang[1,2] **Hao Li**[2] **Zihan Wang**[1] **Shi Feng**[1,†] **Xiaocui Yang**[1]
Daling Wang[1] **Bo Zhang**[2] **Lei Bai**[2] **Shuyue Hu**[2,†]
[1] School of Computer Science and Engineering, Northeastern University
Shenyang 110819, China
[2] Shanghai Artificial Intelligence Laboratory
yiqunzhang@stumail.neu.edu.cn
{fengshi,yangxiaocui,wangdaling}@cse.neu.edu.cn
{lihao4,zhangbo,bailei,hushuyue}@pjlab.org.cn

Abstract

Multi-turn, long-horizon tasks are increasingly common for large language models (LLMs), but solving them typically requires many sequential model invocations, accumulating substantial inference costs. Here, we study cost-aware multi-turn LLM routing: selecting which model to invoke at each turn from a model pool, given a fixed cost budget. We propose MTRouter, which encodes the interaction history and candidate models into joint history–model embeddings, and learns an outcome estimator from logged trajectories to predict turn-level model utility. Experiments show that MTRouter improves the performance–cost trade-off: on ScienceWorld, it surpasses GPT-5 while reducing total cost by 58.7%; on Humanity's Last Exam (HLE), it achieves competitive accuracy while reducing total cost by 43.4% relative to GPT-5, and these gains even carry over to held-out tasks. Further analyses reveal several mechanisms underlying its effectiveness: relative to prior multi-turn routers, MTRouter makes fewer model switches, is more tolerant to transient errors, and exhibits emergent specialization across models. Code: https://github.com/ZhangYiqun018/MTRouter.

1 Introduction

Large Language Models (LLMs) are increasingly deployed to solve complex, tool-using tasks that require extended sequences of interactions, such as software engineering (Jimenez et al., 2023; Jain et al., 2024) and multi-step reasoning (Wang et al., 2022; Yang et al., 2024; Team et al., 2025b; Phan and Alice Gatti, 2025). A defining characteristic of these LLMs is their long-horizon nature, often requiring dozens of sequential model calls per episode. As these trajectories grow, the cumulative inference cost—exacerbated by expanding context windows (Dao, 2023) and superlinear token consumption (Gao and Peng, 2025)—becomes a primary barrier to practical deployment and reproducible research.

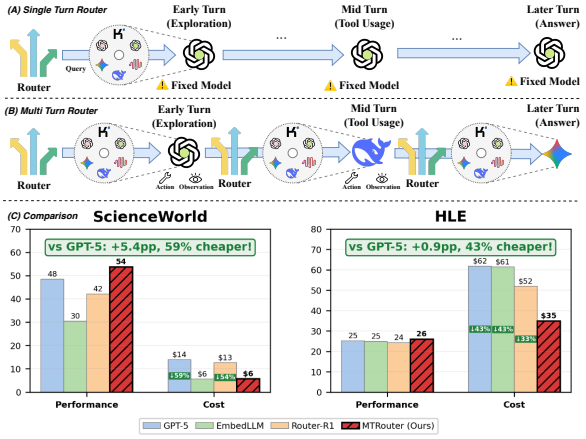

Figure 1: **Top:** a single-turn (episode-level) router selects one model and keeps it fixed throughout the episode. **Middle:** a multi-turn router can adapt the model choice across turns based on the changing interaction state. **Bottom:** MTRouter achieves a better performance–cost trade-off than representative baselines on ScienceWorld (Wang et al., 2022) and Humanity's Last Exam (HLE) (Phan and Alice Gatti, 2025).

This economic pressure is framed by a stark disparity in the current model landscape: frontier models provide state-of-the-art reasoning (like `claude-opus-4.5`[1], `gpt-5`[2]) but are often two orders of magnitude more expensive than their lightweight counterparts (like `DeepSeek-v3.2` (Liu et al., 2025), `Kimi-K2` (Team et al., 2025a)). To mitigate costs, prior work has explored single-turn (episode-level) routing, which selects a single model at the start of an episode based on the initial prompt (Figure 1, Top). However, this static approach is inherently sub-optimal for long-horizon tasks. A single episode often consists of heterogeneous steps, ranging from high-stakes strategic planning to routine

[†]Corresponding author.

[1] https://www.anthropic.com/claude/opus
[2] https://openai.com/index/gpt-5-system-card

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 44206–44226

tool invocations and data formatting. Using a frontier model for every turn is wasteful, yet relying solely on a cheap model risks catastrophic failure during critical junctures.

This observation motivates multi-turn (turn-level) routing, where the agent adaptively switches between models at each step of the interaction (Figure 1, Middle). While promising, multi-turn routing poses a significant predictive challenge: a router must assess whether a specific model choice at the current turn will jeopardize the final outcome dozens of steps later. Simple heuristics or localized error detection are often insufficient to capture the long-term impact of a model's performance on the entire trajectory.

We propose **MTRouter**, which learns an *outcome estimator* from logged trajectories. The estimator maps a history–model pair to an estimate of the eventual episode outcome (terminal score/accuracy), using an error-aware adjustment to provide a stable training signal from offline data. At inference time, the router selects the candidate model with the highest predicted outcome at each turn, and the episode is evaluated under fixed cost and turn limits.

Empirically, MTRouter delivers consistent gains in both performance and cost. On ScienceWorld (test), it improves average score from 48.4 (GPT-5) to 53.8 while reducing total cost by 58.7%; on HLE (test), it reaches competitive accuracy while reducing total cost by 43.4%. It also generalizes under semantic distribution shift (OOD), maintaining improved outcomes with substantial cost savings (Table 3). Beyond aggregate metrics, our analysis shows that multi-turn routing is not "switch more": MTRouter reaches success with fewer model switches than Router-R1 (Zhang et al., 2025a), is more tolerant to transient errors, and exhibits structured model usage and emergent specialization across tools/actions (Figures 3–6).

Our key contributions are as follows:

- We introduce **MTRouter**, which learns an outcome estimator over history–model pairs from offline trajectories and performs turn-level routing in multi-turn agent episodes.
- We evaluate on ScienceWorld and HLE (test and OOD) with a fixed candidate pool, and show consistent improvements in both performance and total cost over strong routing baselines, including Router-R1 and a representative commercial router.
- We provide analyses that connect these gains to concrete routing behaviors, including fewer unnecessary switches, improved error recovery, and emergent specialization across tools/actions.

2 Related Work

LLM Routing. The proliferation of LLMs with diverse cost-capability profiles has motivated research on intelligent model selection. FrugalGPT (Chen et al., 2023) cascades models from cheap to expensive until confidence thresholds are met. EmbedLLM (Wang et al., 2024) learns embeddings to predict model performance on specific queries. RouterDC (Chen et al., 2024) uses dual contrastive learning for effective router training. Avengers (Zhang et al., 2025c) uses a simple clustering-based routing scheme to orchestrate a pool of ten 7B models, achieving performance that surpasses GPT-4.1, while AvengersPro (Zhang et al., 2025b) extends this design to deliver Pareto-optimal performance–cost trade-offs under balanced evaluation settings. These approaches focus on *single-turn* routing: given a query, select one model to answer it. Moving towards multi-turn scenarios, Router-R1 (Zhang et al., 2025a) trained a policy LLM to interleave reasoning and routing through reinforcement learning. Unlike Router-R1 which relies on a heavy LLM-based router, our method learns a *lightweight* outcome estimator over joint history-model embeddings, enabling cost-efficient turn-level routing.

LLMs Tool Use. LLMs augmented with tools can perform complex multi-step tasks (Schick et al., 2023; Qin et al., 2023). ReAct (Yao et al., 2022) interleaves reasoning and action within a single prompt, while Toolformer (Schick et al., 2023) fine-tunes models to invoke tools autonomously. Recent studies have explored LLM-based tool use across diverse domains, including software engineering (Jimenez et al., 2023; Yang et al., 2024), web browsing (Zhou et al., 2023), operating system interaction (Xie et al., 2024), complex reasoning (Mialon et al., 2023), and autonomous research (Team et al., 2025b). These tasks are inherently interactive and typically cannot be resolved in a single turn, necessitating repeated action-observation cycles. While most tool-use frameworks rely on a fixed underlying model, our work introduces a routing layer that dynamically switches between models, treating the model selection as an adaptive decision at each step. This approach is complementary to existing tool-use methodologies and can be

generalized to multi-turn LLM system.

3 MTRouter

3.1 Problem Formalization

Single-turn (episode-level) routing makes one model choice at the start of an episode and keeps it fixed. We study *multi-turn* model routing, where the model choice can change over time. An episode consists of turns indexed by t. At each turn, a router selects a model $a_t \in \mathcal{A}$ to generate the agent's next output (the router itself may be implemented by an LLM). A **turn** is one round of interaction that includes exactly one invocation of the selected model: given the **history** h_t (task description, dialogue context, and the most recent observation), the selected model produces $y_t \sim p_{a_t}(\cdot \mid h_t)$, and a **parser** maps y_t to an executable action $u_t = \text{parse}(y_t)$; executing u_t yields the next observation o_{t+1}.

The episode ends when the task is completed (as indicated by the environment or by an agent submission), a turn limit is reached, or a cost budget is exhausted. The environment provides a terminal score S_{final}.

Objective. We optimize task performance under a per-episode cost budget:

$$\max \ \mathbb{E}\left[S_{\text{final}}\right] \quad \text{s.t.} \quad \sum_{t=0}^{T-1} c_t \leq B \tag{1}$$

where c_t is the cost at turn t (computed from token usage and model pricing), B is a per-episode cost budget, and $T \leq T_{\max}$ is the episode length (capped by a maximum turn limit). Episodes terminate when the budget is exhausted or the turn limit is reached.

3.2 Joint History–Model Representations

Routing decisions depend on both the interaction history and the chosen model (i.e., the pair (h_t, a_t)). We therefore embed history and candidate models separately and then combine them into a joint representation used by the router.

History Encoding. We represent the routing history as the task block plus the accumulated interaction context. Concretely, we serialize each example with a fixed template:

$$h_t = [q, \langle u_0, o_1 \rangle, \ldots, \langle u_{t-1}, o_t \rangle] \tag{2}$$

At implementation time, we do not use a fixed number of retained turns K. Instead, we apply token-budget truncation with a maximum length of 8192 tokens: we always keep the task block and retain the most recent interaction context within the remaining budget, truncating the oldest context first. The serialized history is then encoded by a frozen text encoder ϕ to obtain $z_x = \phi(h_t) \in \mathbb{R}^d$.

Model Encoding. Each candidate model $a \in \mathcal{A}$ is represented by a learned embedding $z_a = \psi(a)$ that combines (i) a vector of structured attributes attr_a (including context limits, knowledge cut-off date, and pricing) with (ii) a learned residual e_a that captures model-specific behavior not explained by metadata.The final model embedding concatenates these components through a linear projection:

$$z_a = W_{\text{proj}} \cdot [\text{MLP}(\text{attr}a); e_a] + b\text{proj} \tag{3}$$

where e_a is the residual embedding regularized to prevent overfitting.

Joint representation. We form a joint feature vector by concatenating the two embeddings, $[z_x; z_a]$, and feed it to a shared feed-forward backbone to enable model-conditioned predictions. To score multiple candidates efficiently, we compute z_x once for the current history and concatenate it with each candidate embedding in a batch.

3.3 Learning an Outcome Estimator

We learn an outcome estimator $\hat{s}_\theta(h_t, a)$ that maps a history–model pair (h_t, a) to the expected terminal outcome when selecting model a at turn t. We parameterize $\hat{s}_\theta$ as a lightweight feed-forward network (an MLP with ReLU nonlinearities and optional dropout) that takes the joint representation $[z_x; z_a]$ and outputs a single scalar:

$$\hat{s}_\theta(h_t, a) = f_\theta([z_x; z_a]) \in \mathbb{R}. \tag{4}$$

We supervise this estimator with terminal outcomes, rather than dense per-turn rewards, for two reasons: (i) in complex agent environments, faithful intermediate rewards are often unavailable, and (ii) training a stable reward model for dense feedback can be brittle. In our setting, episodes are executed under both a cost budget B and a maximum turn limit $T_{\max}$, so expensive choices and wasted turns (e.g., errors that trigger retries) are already reflected in the logged terminal scores; we therefore avoid adding a separate cost penalty to the target.

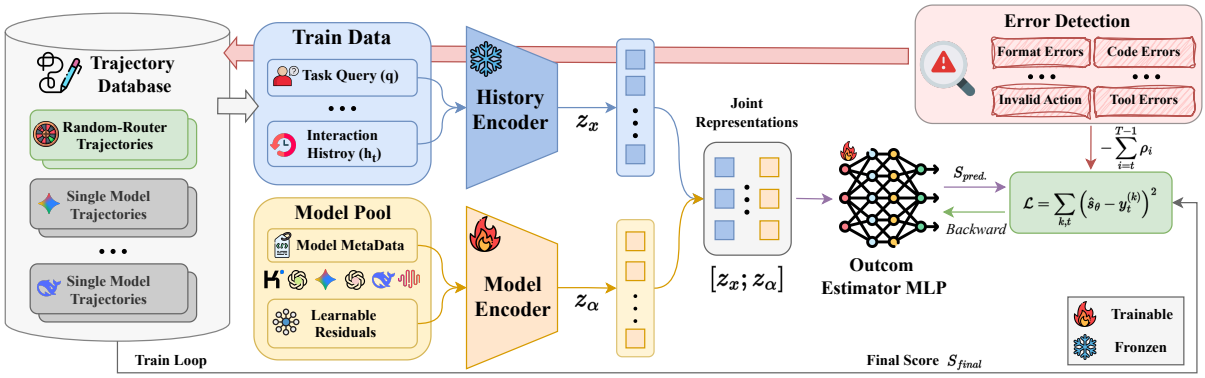

Figure 2: **Overview of MTRouter.**

Outcome target with error penalties. For each turn in a logged trajectory, we form a turn-conditional target by adjusting the terminal score S_{final} with a cumulative error penalty. Concretely, we detect errors (e.g., invalid actions and parsing/-format violations) from the trajectory logs and penalize them by severity and turn progress:

$$\tilde{S}_t = S_{\text{final}} - \sum_{i=t}^{T-1} \rho_i, \tag{5}$$

$$\rho_i = \mathbf{1}[\mathcal{E}_i \neq \emptyset] \cdot \beta_{\text{sev}(\mathcal{E}_i)} \cdot w\left(\tfrac{i+1}{T_{\max}}\right), \tag{6}$$

where $\mathcal{E}_i$ is the set of detected error types at turn i, $\text{sev}(\mathcal{E}_i)$ takes the maximum severity among errors at that turn, and β is the corresponding severity coefficient. The progress weight $w(\cdot)$ is monotone increasing (we use a simple piecewise-linear schedule), so late errors are penalized more strongly. This reflects a simple design intuition: early mistakes may be recoverable as the agent is still gathering information, whereas late mistakes more directly jeopardize task completion, so we impose lower tolerance for errors near the end of an episode. We train $\hat{s}_\theta(h_t, a)$ to approximate $\mathbb{E}[\tilde{S}_t \mid h_t, a]$.

Training Objective. We train on offline trajectories collected under a stochastic router. For each step $(h_t^{(k)}, a_t^{(k)})$ in trajectory k, we use the supervision target $y_t^{(k)} = \tilde{S}_t^{(k)}$ and minimize a squared error loss:

$$\mathcal{L} = \sum_{k,t} \left(\hat{s}_\theta(h_t^{(k)}, a_t^{(k)}) - y_t^{(k)}\right)^2 \tag{7}$$

Data Collection. We log each episode as a trajectory (a sequence of turn-level tuples) and collect training trajectories from two sources: (i) a **uniform (random) router** that samples models from the pool at each turn, and (ii) **single-model** runs where one candidate model is used for the entire episode on training tasks. The random-router data provides broad coverage of model choices, while the single-model data anchors the estimator with consistent per-model behavior. Across both benchmarks, the offline training set contains 1,291 training instances, 29,693 trajectories, and 515,221 total turns, with an estimated one-time collection cost of approximately $1,620.

Inference. At deployment, the router selects the model with the highest predicted outcome,

$$a_t^* = \arg\max_{a \in \mathcal{A}} \hat{s}_\theta(h_t, a), \tag{8}$$

and the episode is executed under the per-episode budget B and turn limit $T_{\max}$ (terminating when either limit is reached).

4 Experiments

We evaluate MTRouter on two challenging multi-turn benchmarks and analyze the learned routing patterns. Unless otherwise stated, we repeat each experiment three times and report the mean. Our anonymous code repository is available at https://github.com/ZhangYiqun018/MTRouter.

4.1 Experimental Setup

Benchmarks. We evaluate on two multi-turn benchmarks: **ScienceWorld** (Wang et al., 2022) and **HLE (Humanity's Last Exam)** (Phan and Alice Gatti, 2025). ScienceWorld is a text-based interactive environment requiring procedural scientific reasoning, with a terminal score $S_{\text{final}} \in [-100, 100]$. HLE is a long-context benchmark spanning academic domains, where questions require multi-step reasoning with tool use and success is binary. We consider both in-distribution (ID)

and out-of-distribution (OOD) splits. We construct OOD evaluations to be *semantically* disjoint from training and ID test (no overlap), rather than relying on random re-sampling, by holding out entire task types / subject categories as OOD. For ScienceWorld, we use 13 task types for ID and reserve 12 held-out task types for OOD; the full task-type split is listed in Table 5; we further split task *variations* within the ID task types into 60%/20%/20% train/validation/test (Appendix A). For HLE, we use 6 subject categories for ID and hold out 2 categories for OOD; the detailed category split is listed in Table 6; we report ID vs. OOD performance by partitioning the benchmark's test questions into the chosen subject groups (Table 6).

Tool Configuration. For HLE, we follow the tool-use setting of `TongYi-DeepResearch` (Team et al., 2025b) and enable four tools: `search`, `browse`, `python`, and `answer`. `search` uses Serper's Google Search API, while `browse` fetches webpage content (via Jina Reader when enabled) and optionally summarizes long pages to keep the context bounded (Appendix E.5); `python` supports deterministic computation and `answer` submits the final response. ScienceWorld does not require external web tools; the agent interacts with the simulator via a single text-action command per turn, with built-in query commands (e.g., `?navigation`, `?object`) to enumerate valid actions. Tool schemas are injected into the HLE system prompt at runtime and are listed in Appendix E.3.

Error Detection. We detect errors from environment observations to compute annealed error costs during training. Table 1 summarizes the error categories by benchmark. HLE errors span format violations, Python execution failures, and tool-specific issues; ScienceWorld only penalizes unparseable actions (environmental feedback like "door is not open" is normal exploration, not an error). Severity levels (high/medium/low) determine penalty coefficients in the AEC computation. Full rule specifications are provided in Appendix B. Unless otherwise stated, we use severity coefficients high=1.0, medium=0.8, and low=0.2 throughout.

Model Pool. We evaluate with 6 frontier LLMs spanning a 20× cost range (see in Table 2).

Implement Details. We train for 100 epochs with early stopping (patience=3) using AdamW optimizer (lr=10^{-3}, weight decay=0.01) and cosine annealing. Batch size is 64. We encode histories

Benchmark	Error Category	# Rules
HLE	Format Errors	4
	Python Execution Errors	11
	Search Tool Errors	3
	Browse Tool Errors	5
ScienceWorld	Invalid Action	1

Table 1: Error categories used for error detection. HLE has diverse tool-related errors, while ScienceWorld only penalizes unparseable actions. Full rule specifications are in Appendix B.

Model	Context	In $/M	Out $/M
GPT-5 (OpenAI, 2025)	400K	1.25	10.00
DeepSeek-V3.2 (Liu et al., 2025)	164K	0.27	0.42
MiniMax-M2 (MiniMax, 2025)	197K	0.20	1.00
Kimi-K2 (Team et al., 2025a)	131K	0.39	1.90
Gemini-2.5-Flash-Lite (Kilpatrick and Gleicher, 2025)	1M	0.10	0.40
GPT-OSS-120B (OpenAI et al., 2025)	131K	0.09	0.36

Table 2: Model pool with pricing (from OpenRouter).

with a frozen `Qwen/Qwen3-Embedding-0.6B` encoder that produces 1024-dimensional embeddings, and we set the maximum input length to 8192 tokens. The model encoder maps an 8-d metadata feature vector to a 32-d attribute embedding, concatenates it with a 16-d per-model residual embedding, and linearly projects the resulting 48-d vector to a 64-d model embedding. To prevent premature convergence, we apply an L_2 penalty ($\lambda = 0.001$) to the learnable residual embeddings. For the error penalty in Eq. 5, we instantiate the progress weight $w(\cdot)$ as a piecewise-linear warmup with p_0=0.3, p_1=0.7, $w_{\min}$=0.3, and $w_{\max}$=1.0 (Appendix B). During evaluation, we enforce a maximum horizon of 50 steps for ScienceWorld and 30 steps for HLE, with a per-episode cost cap of $2.0 for both benchmarks.

Baselines. We compare against three groups of baselines:

- **Single-model**: each candidate model from our pool is used exclusively for the entire episode.
- **Single-turn routers (episode-level)**: **RouterDC** (Chen et al., 2024), **EmbedLLM** (Wang et al., 2024), and **AvengersPro** (Zhang et al., 2025c,b). These routers make a single routing decision at the start of an episode based on the initial query context, then keep the selected model fixed for all subsequent turns.
- **Multi-turn routers (turn-level)**: **Random Router** (uniform random selection at each turn), **Router-R1** (Zhang et al., 2025a) (we train a `Qwen2.5-7B-Instruct` routing model following the Router-R1 recipe), **LLM Router** (same prompt as Router-R1 but directly us-

Method	ScienceWorld				HLE			
	Test		*OOD*		*Test*		*OOD*	
	Score↑	Total Cost ($)↓	Score↑	Total Cost ($)↓	Acc↑	Total Cost ($)↓	Acc↑	Total Cost ($)↓
Single-Model Baselines								
GPT-5	48.4±2.1	13.9	4.9±2.9	47.6	25.1±1.6	61.8	34.8±2.2	65.3
DeepSeek-V3.2	13.1±2.0	2.9	-4.2±2.4	22.8	15.6±1.3	22.4	28.7±1.6	22.2
MiniMax-M2	-0.5±1.6	3.2	0.9±2.3	10.9	7.8±1.4	18.1	8.9±1.9	9.8
Kimi-K2	5.2±1.8	2.5	0.2±2.1	8.9	11.4±1.2	12.0	20.1±1.5	9.8
Gemini-2.5-Flash-Lite	4.2±1.7	0.3	-2.1±2.2	1.5	5.6±1.1	3.0	8.4±1.5	2.1
GPT-OSS-120B	26.6±2.3	0.5	1.1±2.7	4.2	9.7±1.0	0.7	11.4±1.2	2.1
Single-Turn Routers (episode-level)								
RouterDC (Chen et al., 2024)	23.1±2.4	3.3	5.5±3.1	2.5	12.8±1.6	10.3	17.9±2.1	13.5
EmbedLLM (Wang et al., 2024)	30.4±2.8	5.6	5.0±3.4	3.0	24.8±2.0	61.4	33.6±2.5	56.8
AvengersPro (Zhang et al., 2025c)	36.8±2.5	4.1	2.4±3.2	4.1	23.7±1.8	47.5	30.6±2.3	33.3
Multi-Turn Routers (turn-level)								
Random Router	21.7±4.5	3.9	-8.1±5.4	20.3	20.0±2.8	16.9	23.8±3.4	14.7
LLM Router	19.8±3.8	12.2	-0.4±4.6	28.3	24.0±2.3	56.2	36.0±2.8	35.6
Router-R1 (Zhang et al., 2025a)	42.1±3.6	12.6	2.1±4.2	21.0	24.2±2.2	51.9	35.1±2.6	60.7
OpenRouter† (OpenRouter, 2025)	-26.4±3.2	3.0	-26.9±3.8	15.5	18.3±2.1	138.5	34.0±2.5	154.3
MTRouter (ours)	**53.8±3.2**	5.7	**9.9±3.9**	16.3	**26.0±2.3**	35.0	**38.6±3.0**	31.2
Δ vs GPT-5	**+5.4**	*saving 58.7%*	**+5.0**	*saving 65.8%*	**+0.9**	*saving 43.4%*	**+3.8**	*saving 52.3%*
Δ vs Router-R1	**+11.7**	*saving 54.4%*	**+7.8**	*saving 22.4%*	**+1.8**	*saving 32.7%*	**+3.5**	*saving 48.7%*

Table 3: Main results on ScienceWorld and HLE benchmarks (Test and OOD splits). We report mean±std over three runs for the performance metrics; total cost is summed over evaluated episodes. OOD evaluations use held-out task types (ScienceWorld) and held-out subject categories (HLE). The Δ rows show score gains and relative cost savings compared to GPT-5 and Router-R1. †OpenRouter uses a fixed provider-side routing API with a different model pool (Appendix D).

ing `DeepSeek-V3.2` as the routing model, no training), and **OpenRouter** (OpenRouter, 2025) (a representative commercial router via OpenRouter's automatic routing API). OpenRouter is included to contextualize MTRouter against an off-the-shelf production routing system; however, its routing API does not allow us to customize the candidate model pool, and it selects from a substantially larger pool than our fixed 6-model setting (Appendix D).

4.2 Does MTRouter Work?

Table 3 presents in-distribution (test) and out-of-distribution (OOD) results across MTRouter and the other baselines.

Test. On ScienceWorld, MTRouter achieves the best average score (53.8) while cutting total cost by 58.7% vs. GPT-5; compared to Router-R1, it gains +11.7 points with 54.4% lower total cost. Episode-level routers (single-turn) that commit to one model per episode consistently lag behind, supporting the necessity of *multi-turn* routing in interactive settings where phases and errors evolve over time. Notably, OpenRouter produces negative scores on ScienceWorld because it underestimates task difficulty and over-relies on lightweight models (Appendix D). On HLE, MTRouter attains the best accuracy (26.0%) while remaining cost-efficient (43.4% lower total cost than GPT-5 and 32.7% lower than Router-R1); Router-R1 and LLM Router reach similar accuracy but at higher cost. Overall, MTRouter delivers a better accuracy–cost trade-off than both strong single-model baselines and existing routing baselines on both benchmarks.

Variant	SW Avg Score↑	HLE Acc. (%)↑
MTRouter (full setting)	53.8 ± 3.2	26.0 ± 2.3
Ridge instead of MLP	49.1 ± 3.5	23.4 ± 2.2
w/o Random-Router data	47.2 ± 4.1	22.6 ± 2.1
w/o error penalties	48.5 ± 3.6	23.8 ± 2.1
w/o routing history*	44.6 ± 3.8	21.3 ± 2.0
Hardcoded model encoder	41.3 ± 4.4	19.7 ± 2.1

Table 4: Ablation study on ScienceWorld and HLE test sets. We report performance with 95% confidence intervals. *This ablation removes *router* history: routing conditions only on the current turn (the chosen model still receives the full conversation history).

OOD. We next examine the OOD columns of Table 3, which evaluate semantic distribution shift (held-out ScienceWorld task types and held-out HLE subject categories). On ScienceWorld OOD, MTRouter improves over GPT-5 by +5.0 points while using 65.8% lower total cost; on HLE OOD, it reaches 38.57% accuracy with 52.3% lower total cost than GPT-5. These results show that

MTRouter not only generalizes under distribution shift but also preserves its cost efficiency, establishing the strongest overall performance among the compared methods.

Ablation Studies Table 4 summarizes ablations on ScienceWorld and HLE. Performance degrades when we replace the MLP with a simpler regressor, remove random-router data, or remove the error-penalty adjustment, indicating that each component contributes to learning reliable routing preferences from offline trajectories. The largest drops come from removing routing history or replacing the learned model encoder with hardcoded features, highlighting the importance of modeling both the evolving interaction context and model behavior. We use Ridge as a standard regularized linear baseline to test whether the gains come from nonlinear modeling rather than feature construction alone. Additional robustness checks on budget sensitivity, candidate-pool size, and history token budget are reported in Appendix C.

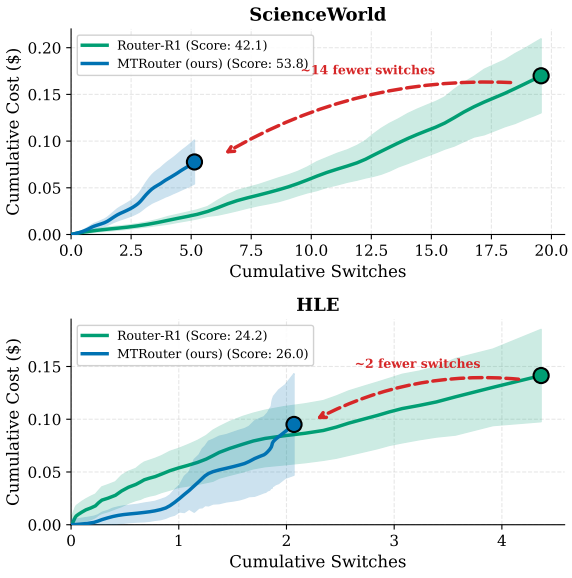

Figure 3: Cumulative cost vs. cumulative model switches over successful episodes, comparing MTRouter against Router-R1 on ScienceWorld and HLE (constructed by replaying logged trajectories).

4.3 Why Does MTRouter Work?

While the in-domain and out-of-domain results (Table 3) and ablations (Table 4) establish MTRouter's effectiveness, we next ask a more diagnostic question: *why* does it work? We use a sequence of complementary analyses to connect the performance gains to concrete routing behaviors.

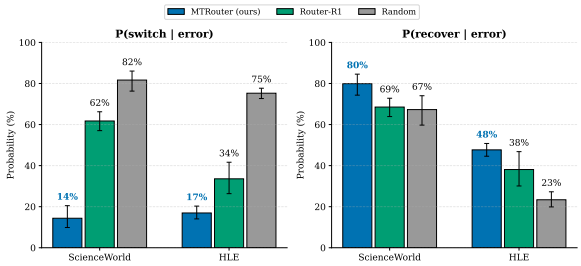

Figure 4: Error-triggered switching and recovery. **Left**: probability of switching models after an error (format errors / invalid actions). **Right**: probability that the next turn recovers (no error) conditioned on an error.

Start from a simple diagnostic: switching vs. cost. If multi-turn routing is "just switch more often," then a router that switches frequently should dominate. We find the opposite. Figure 3 plots, over *successful* episodes, how cumulative API cost grows as the router makes additional model switches. Each curve is constructed by replaying logged trajectories from MTRouter and Router-R1, accumulating per-turn cost and counting switches along the episode. Despite MTRouter achieving better end performance (Table 3), its trajectories typically reach success with *fewer* switches and *lower* cumulative cost (e.g., on ScienceWorld: ∼5 switches for MTRouter vs. ∼20 for Router-R1).

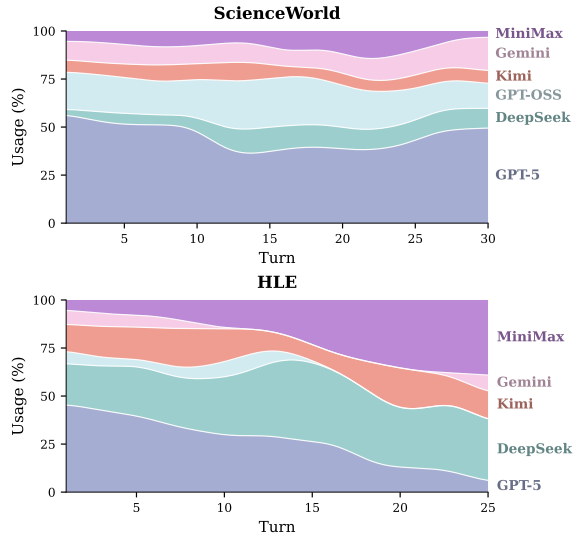

Figure 5: Model usage by turn on ScienceWorld and HLE. MTRouter exhibits structured, benchmark-specific routing behavior rather than uniformly switching models across turns. In the HLE, GPT-OSS-120B is used primarily in early turns and then decays to near-zero usage in later turns, which makes its band difficult to notice visually.

Beyond differences in per-token pricing, frequent switching can also reduce the effectiveness

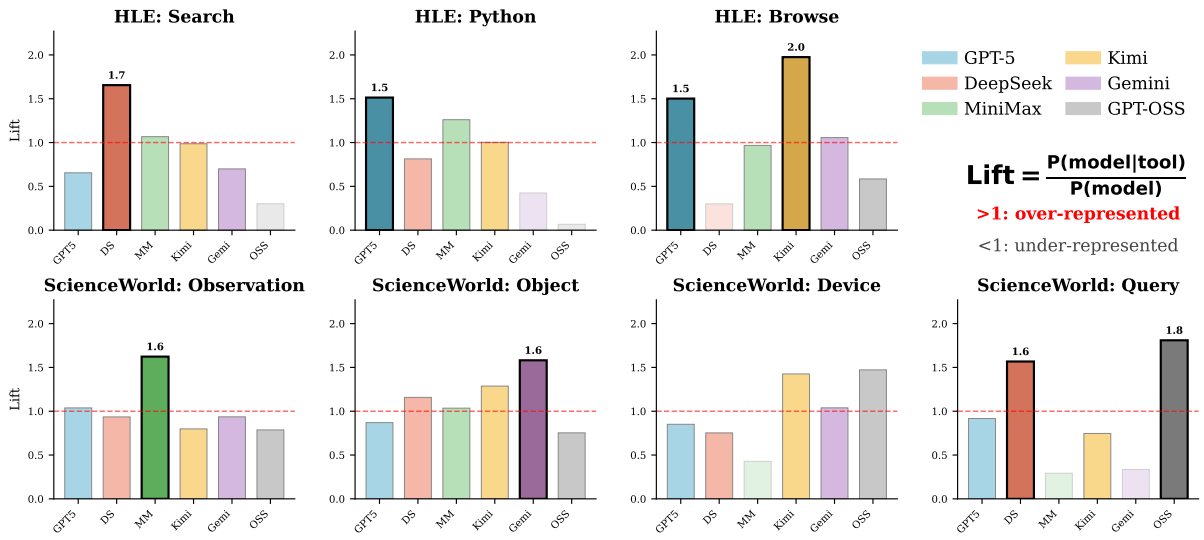

Figure 6: Tool/action specialization by model measured via *lift* (Lift $= P(\text{model} \mid \text{tool})/P(\text{model})$). Values > 1 indicate a model is over-represented for a tool/action (specialization), while values < 1 indicate under-use.

of prompt caching in multi-turn settings, lowering cache hit rates and increasing the effective cost of serving long histories (Hu et al., 2025). This immediately raises a more specific question: *when* does Router-R1 choose to switch, and are those switches actually helpful?

Switching less by being more tolerant to errors. Figure 4 shows that Router-R1 switches aggressively after errors, while MTRouter is less reactive: it often keeps the current model and tries to continue. Crucially, this is not "ignoring" errors: the right panel shows a higher probability of recovery on the next turn under MTRouter. Consistent with this, after an error MTRouter stays with the same model ≈90.2% of the time on ScienceWorld and ≈80.9% on HLE, substantially higher than Router-R1 (38.3% and 66.4%, respectively). Together with Figure 3, these trends suggest that MTRouter avoids a large fraction of error-triggered switches that appear low-yield, helping control cumulative cost without sacrificing performance. We hypothesize this gap stems from the learning signal: Router-R1 largely relies on a natural-language router prompt to infer when switching helps, whereas MTRouter is trained directly on trajectory outcomes (terminal scores with annealed error penalties), providing more direct supervision for effective switching.

Not "never switch"—but switch with structure. One might worry that the previous results simply reflect a conservative router that rarely changes models. Figure 5 rules this out: MTRouter uses multiple models throughout an episode, but in a stable, benchmark-specific way rather than as a reflex to errors. This suggests that the router is learning a *strategy* (which models to rely on, and when), not just a generic "upgrade on failure" heuristic. For instance, on ScienceWorld, GPT-5 accounts for 50.8% of early turns, while GPT-OSS increases to 24.3% in the final turns; in contrast, Router-R1 largely concentrates on DeepSeek and Gemini at roughly ∼45% each across phases, exhibiting much less structured diversity.

A concrete form of strategy: emergent specialization. To make this structure explicit, Figure 6 measures *lift*: how much a model is over-used for a tool/action relative to its overall frequency. We observe clear specialization patterns (lift > 1) that align with complementary strengths—e.g., on HLE, DeepSeek is over-represented on `search` (lift 1.66), GPT-5 on `python` (1.51), and Kimi on `browse` (1.98). On ScienceWorld, we observe analogous specialization across action types, such as MiniMax on observation-heavy actions (1.62), Gemini on object interactions (1.58), and GPT-OSS on query commands (1.81). These findings connect back to the main results: MTRouter wins not by switching more, but by switching *selectively* and assigning stable roles to models over the course of an episode.

Learned model embeddings. Figure 7 visualizes the learned model embeddings after training. The encoder learns to distinguish the candidate models and organizes them by cost tier, suggest-

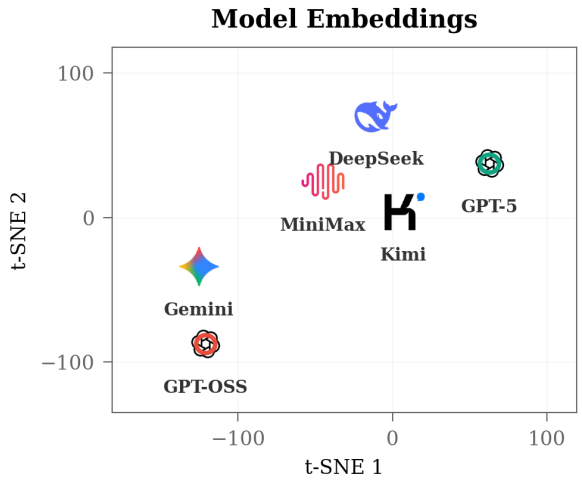

Figure 7: t-SNE visualization of learned model embeddings from the model encoder. The embeddings separate models by identity and form a clear cost-tier structure, with low-cost models (e.g., GPT-OSS, Gemini) distinct from higher-cost frontier models (e.g., GPT-5).

ing it captures meaningful capability–cost structure beyond raw attributes.

5 Conclusion

In this paper, we presented MTRouter, a multi-turn routing framework that optimizes the performance–cost trade-off in agentic workflows via an offline-learned outcome estimator. By enabling turn-level model selection, our approach allows agents to adaptively allocate computational resources based on the evolving interaction state. Our experiments on ScienceWorld and HLE demonstrate the empirical effectiveness of MTRouter. It improves the average score on ScienceWorld from 48.4 to 53.8 while reducing total costs by 58.7%. On HLE, it achieves a 43.4% cost reduction while maintaining competitive accuracy. Beyond aggregate performance, our analysis of the routing trajectories shows that MTRouter exhibits structured model usage across different actions and achieves success with fewer model switches than existing baselines.

Limitations

Despite the performance and cost gains demonstrated by MTRouter, several limitations remain that offer avenues for future work.

Scalability of Data Collection. The primary bottleneck for training turn-level routers is the cost of collecting diverse, long-horizon trajectories across multiple candidate models. Due to computational budget constraints, we evaluate episodes with horizons of up to 50 steps on ScienceWorld and 30 steps on HLE.

Lack of Online Adaptation. Currently, MTRouter operates in an offline learning paradigm. While this significantly reduces the cost of training, the router cannot adapt its strategy in real-time to novel or rapidly evolving environments. Implementing online updates via reinforcement learning would likely yield further gains in robustness, but the prohibitive cost of continuous online interaction with frontier models makes this challenging for current academic research.

Prompt Caching and Switching Overhead. A technical challenge inherent to multi-turn routing is the potential loss of prompt caching (e.g., KV cache) when switching between different models. In many API-based deployments, frequent switching requires the new model to re-process the entire trajectory prefix, which could inadvertently increase latency and per-turn costs. However, our analysis suggests that MTRouter partially mitigates this issue; unlike reactive baselines that switch models at the first sign of a transient error, MTRouter exhibits a more "composed" switching pattern, maintaining model stability over longer sequences of turns. This structured model usage, as reflected in our cost-efficiency results, helps balance the trade-off between adaptive model selection and the benefits of context caching.

Acknowledgements

The work was supported by the National Natural Science Foundation of China (Nos. 62272092, 62172086, 62506186), and the Fundamental Research Funds for the Central Universities under Grants (N25XQD004). This work was supported by Shanghai Artificial Intelligence Laboratory. This work was done during Yiqun Zhang, Hao Li's internships at Shanghai Artificial Intelligence Laboratory.

References

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

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

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Pengfei Gao and Chao Peng. 2025. More with less: An empirical study of turn-control strategies for efficient coding agents. *Preprint*, arXiv:2510.16786.

Shuyue Hu, Siyue Ren, Yang Chen, Chunjiang Mu, Jinyi Liu, Zhiyao Cui, Yiqun Zhang, Hao Li, Dongzhan Zhou, Jia Xu, and 1 others. 2025. Hands-on llm-based agents: A tutorial for general audiences.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Logan Kilpatrick and Zach Gleicher. 2025. Gemini 2.5 Flash-Lite is now stable and generally available. Official Google Developers blog. Accessed: 2026-04-18.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *Preprint*, arXiv:2311.12983.

MiniMax. 2025. Minimax m2 & agent: Ingenious in simplicity. Official MiniMax release post. Accessed: 2026-04-18.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

OpenAI. 2025. Introducing GPT-5 for developers. Official OpenAI release. Accessed: 2026-04-18.

OpenRouter. 2025. Openrouter: Unified api for llms and automatic routing. `https://openrouter.ai`. Accessed 2026-01-03.

Long Phan and et.al. Alice Gatti. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025a. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, and 1 others. 2025b. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*.

Qianxin Wang, Liqi Chen, Souradip Chakraborty, Qing Dong, Haoran Zhang, Yang Liu, Zhenghai Qi, Tianyi Guo, and Meng Liu. 2024. Embedllm: Learning compact representations of large language models. *arXiv preprint arXiv:2401.11623*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025a. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yiqun Zhang, Hao Li, Jianhao Chen, Hangfan Zhang, Peng Ye, Lei Bai, and Shuyue Hu. 2025b. Beyond gpt-5: Making llms cheaper and better via

performance-efficiency optimized routing. In *Proceedings of the 2025 7th International Conference on Distributed Artificial Intelligence*, pages 122–129.

Yiqun Zhang, Hao Li, Chenxu Wang, Linyao Chen, Qiaosheng Zhang, Peng Ye, Shi Feng, Daling Wang, Zhen Wang, Xinrun Wang, and 1 others. 2025c. The avengers: A simple recipe for uniting smaller language models to challenge proprietary giants. *arXiv preprint arXiv:2505.19797*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Dataset and Split Details

A.1 ScienceWorld Task Types

Table 5 lists the ScienceWorld task types used for training and out-of-distribution evaluation.

Split	Task Types
Train (13)	boil, melt, chemistry-mix, find-animal, find-plant, grow-plant, identify-life-stages-1, lifespan-longest-lived, inclined-plane-angle, measure-melting-point, power-component, test-conductivity, mendelian-genetics
OOD Test (12)	freeze, change-state-of-matter, chemistry-mix-paint, find-non-living, grow-fruit, identify-life-stages-2, lifespan-shortest, inclined-plane-friction, use-thermometer, power-renewable, test-conductivity-unknown, genetics-unknown

Table 5: ScienceWorld task type splits.

For each task type, we sample up to 30 variations (to bound collection time). Variations are split 60%/20%/20% into train/validation/test using a fixed random seed (42).

A.2 HLE Subject Categories

Category	N (Test)
In-Distribution (Training)	
Math	200
Physics	46
Chemistry	23
Biology/Medicine	37
Engineering	16
Computer Science/AI	37
Out-of-Distribution	
Humanities/Social Science	193
Other	176

Table 6: HLE test set distribution by category.

B Error Detection Rules

Table 7 provides the full specification of error detection rules used to compute annealed error costs (AEC) during training. Each rule consists of pattern strings matched against environment observations, a severity level (high/medium/low), and a description.

Design Principles. For HLE, we distinguish between model errors (format violations, Python exceptions) and external failures (HTTP errors, paywalls). Model errors receive higher severity since they reflect controllable mistakes; external failures are marked low severity as they depend on third-party services. For ScienceWorld, we only penalize truly invalid actions (commands not recognized by the environment parser). Environmental feedback such as "the door is not open" or "the object is already in your inventory" represents normal exploration and is not treated as an error.

Severity Coefficients. Severity levels map to penalty coefficients in AEC: high (1.0), medium (0.8), low (0.2). The warmup schedule uses p_0=0.3, p_1=0.7, $w_{\min}$=0.3, and $w_{\max}$=1.0. These coefficients modulate the base penalty $1/N$ where N is the expected episode length for the task category.

C Additional Experiments

D OpenRouter Baseline Details

Motivation. OpenRouter (OpenRouter, 2025) provides an automatic routing API commonly used in commercial deployments. We include OpenRouter as a representative commercial router baseline to contextualize MTRouter against an off-the-shelf production routing system.

Model pool. Unlike our setting, which restricts routing to a fixed 6-model candidate pool (Table 2), OpenRouter's automatic routing can choose from a much broader set of models. In our evaluation, the OpenRouter baseline routes over the following pool (as reported by the API at routing time):

Implications for comparison. Because OpenRouter can select from a superset of models (including multiple frontier and provider-specific options not present in our pool), it represents a stronger routing setting than ours. We still include it as a practical reference point, but comparisons should

Category	Rule Name	Severity	Description
HLE Benchmark			
Format Errors	format_error	medium	Tool call format error—model did not follow required format
	tool_invalid_args	medium	Invalid tool arguments or missing required parameters
	tool_parse_error	medium	Tool call parsing failure
	tool_unknown	high	Model called a non-existent tool name
Python Execution Errors	python_traceback	high	Python execution exception with traceback
	python_name_error	high	Undefined variable reference
	python_syntax_error	high	Python syntax error
	python_indentation_error	high	Python indentation error
	python_type_error	high	Type mismatch error
	python_value_error	medium	Invalid value error
	python_index_error	medium	Index out of bounds
	python_key_error	medium	Dictionary key not found
	python_attribute_error	medium	Attribute access error
	python_import_error	medium	Module import failure
	python_zero_division	medium	Division by zero
	python_timeout	high	Code execution timeout
Search Tool Errors	search_no_results	high	Search returned no results
	search_http_error	low	HTTP connection error during search
	search_rate_limit	low	Search rate limit exceeded
Browse Tool Errors	browse_403	low	HTTP 403 Forbidden
	browse_404	low	HTTP 404 Not Found
	browse_access_denied	low	Access denied to resource
	browse_paywall	low	Paywall or subscription block
	browse_timeout	low	Browse request timeout
ScienceWorld Benchmark			
Invalid Action	no_known_action	high	Invalid action command not recognized by environment

Table 7: Detailed error detection rules used for annealed error cost (AEC) computation. Severity levels determine penalty coefficients: high (1.0), medium (0.8), low (0.2). Browse tool errors are marked low severity as they reflect external failures rather than model errors.

Benchmark	Budget	Score/Acc↑	Total Cost ($)↓
ScienceWorld	$0.5 \times B$	45.2±3.8	3.6
ScienceWorld	$1.0 \times B$	53.8±3.2	5.7
ScienceWorld	$2.0 \times B$	55.6±3.1	8.1
HLE	$0.5 \times B$	20.6±2.8	23.4
HLE	$1.0 \times B$	26.0±2.3	35.0
HLE	$2.0 \times B$	27.3±2.2	44.8

Table 8: Budget sensitivity. Relaxing the budget improves performance, but with clear diminishing returns.

	ScienceWorld Test		ScienceWorld OOD		HLE Test		HLE OOD	
Pool	Score↑	Cost↓	Score↑	Cost↓	Acc↑	Cost↓	Acc↑	Cost↓
2-model	50.6±3.6	8.9	7.2±4.3	28.5	25.4±2.5	45.2	36.4±3.1	43.5
6-model	53.8±3.2	5.7	9.9±3.9	16.3	26.0±2.3	35.0	38.6±3.0	31.2
8-model	54.1±3.3	5.3	10.3±4.1	16.8	25.8±2.4	34.2	39.1±2.9	29.5

Table 9: Candidate-pool sensitivity under 2-, 6-, and 8-model settings.

History Budget	ScienceWorld↑	HLE↑
`max_tokens=2048`	49.3±3.7	23.8±2.6
`max_tokens=4096`	52.1±3.4	25.3±2.4
`max_tokens=8192`	53.8±3.2	26.0±2.3
`max_tokens=16384`	53.5±3.3	26.2±2.3

Table 10: Sensitivity to the history token budget.

be interpreted with this mismatch in candidate pools in mind.

Why does OpenRouter perform poorly on ScienceWorld? Figure 8 shows the model usage distribution of OpenRouter on both benchmarks. On ScienceWorld, OpenRouter predominantly selects lightweight models: Mistral-Nemo accounts for 68% of all model calls, followed by GPT-5-nano (24%) and Claude-4.5 (7%). These models, while cost-efficient, lack the reasoning capability required for ScienceWorld's procedural scientific tasks. In contrast, on HLE, OpenRouter allocates 54% of calls to Claude-4.5—a much stronger frontier model—along with Sonar (23%) and Mistral-Nemo (15%), reflecting a more appropriate difficulty assessment for that benchmark.

This discrepancy reveals a key limitation of general-purpose commercial routers: without task-specific training, they may underestimate the difficulty of unfamiliar domains and over-rely on cheaper models. ScienceWorld's text-based interface and seemingly simple commands may mislead the router into treating it as an easy task, when in fact it requires multi-step planning and precise action sequencing that weaker models struggle to execute. The resulting negative scores (-26.4 on Test, -26.9 on OOD) indicate that the selected models frequently fail to make meaningful progress toward task completion, as judged by the environment's scoring function.

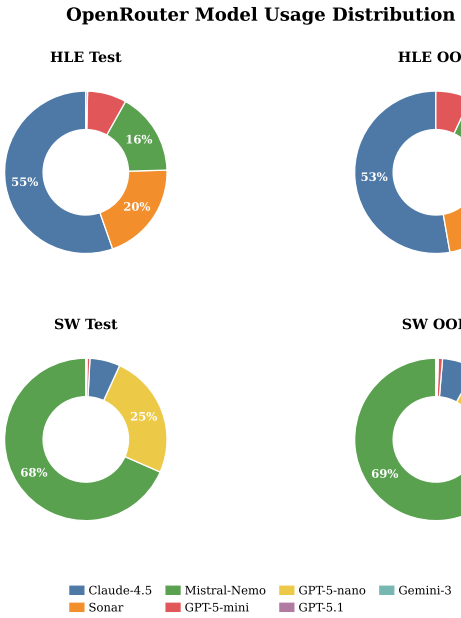

Figure 8: Model usage distribution of OpenRouter on ScienceWorld and HLE. On ScienceWorld, OpenRouter predominantly selects lightweight models (Mistral-Nemo: 68%, GPT-5-nano: 24%), while on HLE it primarily uses stronger models (Claude-4.5: 54%), reflecting different difficulty assessments.

E Prompts and Tool Schemas

We report the prompts used in our evaluation setup. For brevity, we omit the format-error correction prompts.

E.1 ScienceWorld Agent Prompt

E.2 HLE Agent Prompt

The HLE system prompt injects tool schemas via a template variable. We list each tool schema explicitly in Appendix E.3.

E.3 Tool Schemas

We list each tool's JSON schema (name, description, and parameter schema) used by the HLE agent.

E.4 Routing Model Prompt for LLM Router / Router-R1

We report the turn-level routing prompt used by the LLM Router baseline and by Router-R1. In both baselines, the routing model is queried at each turn to produce a `<select>` decision. They

ScienceWorld System Prompt

You are a helpful assistant interacting with a text-based science simulation environment. Your goal is to complete science experiments by issuing text commands.

How to Interact

You issue commands by writing them in a “‘text code block. The environment will respond with observations.

<format_example>
THOUGHT: I should explore my surroundings first.
“‘text
look around
“‘
</format_example>

Available Command Types

Common commands include:

- Movement: `go to [location]`, `open door`, `go through door`
- Interaction: `pick up [object]`, `put [object] in [container]`, `activate [object]`
- Observation: `look around`, `look at [object]`, `inventory`
- Task-specific: `focus on [object]`, `use [tool] on [object]`

Query Commands (Free Actions)

You can query available actions without consuming a game turn:

- `?navigation`: show movement actions (go, walk, move)
- `?object`: show object manipulation (pick up, put, pour)
- `?observation`: show observation actions (look, examine, inventory)
- `?device`: show device control (activate, turn on/off, use)
- `?door`: show door/container actions (open, close)
- `?electrical`: show electrical actions (connect, disconnect)
- `?interaction`: show interaction actions (mix, eat, focus)
- `?all`: show all valid actions
- `?categories`: show query help

Use queries to explore available actions before deciding your next move.

Important Notes

- Issue ONE command per response
- Include a THOUGHT section explaining your reasoning
- The environment is turn-based—wait for observations before issuing the next command
- Some tasks require multiple steps to complete

ScienceWorld Instance Prompt

<task_description>
{{task_description}}
</task_description>

<initial_observation>
{{initial_observation}}
</initial_observation>

<instructions>
Complete the science experiment described above. You are interacting with a simulated environment. Issue commands one at a time and observe the results.

Tip: Use query commands like `?navigation` or `?object` to explore available actions without consuming a turn.

Strategy hints:
- First explore: use `open door to [room]` then `go to [room]` to navigate
- Look around each room to find objects you need
- Pick up objects with `pick up [object]`
- For heating: find a stove, turn it on with `activate [stove]`, place container on it
- For cooling: use a freezer or refrigerator
- Use `focus on [object]` to examine substances

To complete the task, perform the necessary actions described in the task description. When you believe the task is complete, issue the command:

```text
task completed
```

Remember:
- Think step by step about what actions are needed
- Explore your environment to find needed objects
- Some actions may require prerequisites (e.g., picking up an object before using it)
</instructions>

HLE System Prompt

You are an expert problem solver tackling challenging questions from Humanity's Last Exam. These questions require deep reasoning, careful research, and precise answers.

Available Tools

```
<tools>
{% for tool in tool_schemas %}
{{ tool.to_xml() }}
{% endfor %}
</tools>
```

Tool Call Formats

You can use either XML or JSON format inside `<tool_call>` tags:

Format A: XML

```
<tool_call>
<tool_name>
<param1>value1</param1>
<param2>value2</param2>
</tool_name>
</tool_call>
```

Format B: JSON

```
<tool_call>{"name": "tool_name", "arguments": {"param1": "value1"}}</tool_call>
```

Examples

```
<tool_call>
<search>
<query>maximum likelihood estimation</query>
</search>
</tool_call>
<tool_call>{"name": "python", "arguments": {"code": "print(2 + 2)"}}</tool_call>
```

Strategy for HLE Questions

1. **Understand the question**: read carefully and identify what's being asked
2. **Break down the problem**: decompose into sub-problems if needed
3. **Research if needed**: use `search` for factual information you're unsure about
4. **Verify sources**: use `browse` to read primary sources when accuracy matters
5. **Compute when needed**: use `python` for calculations and data analysis
6. **Synthesize**: combine information from multiple sources
7. **Verify your answer**: double-check before submitting
8. **Submit**: use `answer` with your final answer

Important Notes

- These are challenging questions—take your time
- Be precise—exact answers are often required
- Show your reasoning before using tools
- If uncertain, express confidence level in your answer

HLE Instance Prompt

HLE Question

```
{% if subject %}
```

Subject: `{{ subject }}`

```
{% endif %}
```

```
{{ question | default(task) }}
```

Solve this step by step. Use tools to research and verify. Submit your final answer using the `answer` tool: `<tool_call>{"name": "answer", "arguments": {"answer": "your final answer"}}</tool_call>`

Tool Schema: search

```
{
  "name": "search",
  "description": "Search the web for information using Google",
  "parameters": {
    "type": "object",
    "properties": {
      "query": {
        "oneOf": [
          { "type": "string" },
          { "type": "array", "items": { "type": "string" } }
        ],
        "description": "Search query or list of queries for batch search"
      },
      "num_results": {
        "type": "integer",
        "description": "Number of results to return (default: 10)",
        "minimum": 1,
        "maximum": 100
      }
    }
  },
  "required": [ "query" ]
}
```

Tool Schema: browse

```
{
  "name": "browse",
  "description": "Visit webpage(s) and extract information based on a goal",
  "parameters": {
    "type": "object",
    "properties": {
      "url": {
        "type": "string",
        "description": "URL to visit. Can be a single URL or array of URLs."
      },
      "goal": {
        "type": "string",
        "description": "The goal of the visit - what information to extract from the webpage(s
    )."
      },
      "extract_mode": {
        "type": "string",
        "enum": [ "text", "markdown", "html" ],
        "description": "Content extraction mode (default: text). Only used when LLM summary is
     disabled."
      }
    }
  },
  "required": [ "url", "goal" ]
}
```

Tool Schema: python

```
{
  "name": "python",
  "description": "Execute Python code and return the output",
  "parameters": {
    "type": "object",
    "properties": {
      "code": {
        "type": "string",
        "description": "Python code to execute"
      }
    }
  },
  "required": [ "code" ]
}
```

Tool Schema: answer

```
{
  "name": "answer",
  "description": "Submit your final answer to complete the task",
  "parameters": {
    "type": "object",
    "properties": {
      "answer": {
        "type": "string",
        "description": "The final answer to submit"
      },
      "confidence": {
        "type": "number",
        "description": "Confidence level from 0 to 1 (optional)",
        "minimum": 0,
        "maximum": 1
      },
      "reasoning": {
        "type": "string",
        "description": "Brief explanation of how you arrived at the answer (optional)"
      }
    }
  },
  "required": [ "answer" ]
}
```

Provider/Model	Provider/Model
openai/gpt-5.1	openai/gpt-5
openai/gpt-5-mini	openai/gpt-5-nano
openai/gpt-4.1	openai/gpt-4.1-mini
openai/gpt-4.1-nano	openai/gpt-4o
openai/gpt-4o-2024-05-13	openai/gpt-4o-2024-08-06
openai/gpt-4o-2024-11-20	openai/gpt-4o-mini
openai/gpt-4o-mini-2024-07-18	openai/gpt-4-turbo
openai/gpt-4-turbo-preview	openai/gpt-4-1106-preview
openai/gpt-4	openai/gpt-3.5-turbo
openai/gpt-oss-120b	anthropic/claude-opus-4.5
anthropic/claude-opus-4.1	anthropic/claude-opus-4
anthropic/claude-sonnet-4.5	anthropic/claude-sonnet-4
anthropic/claude-3.7-sonnet	anthropic/claude-haiku-4.5
anthropic/claude-3.5-haiku	anthropic/claude-3-haiku
google/gemini-3-pro-preview	google/gemini-2.5-pro
google/gemini-2.0-flash-001	google/gemini-2.5-flash
mistralai/mistral-large	mistralai/mistral-large-2407
mistralai/mistral-large-2411	mistralai/mistral-medium-3.1
mistralai/mistral-nemo	mistralai/mistral-7b-instruct
mistralai/mixtral-8x7b-instruct	mistralai/mixtral-8x22b-instruct
mistralai/codestral-2508	x-ai/grok-4
x-ai/grok-3	x-ai/grok-3-mini
deepseek/deepseek-r1	meta-llama/llama-3.3-70b-instruct
meta-llama/llama-3.1-405b-instruct	meta-llama/llama-3.1-70b-instruct
meta-llama/llama-3.1-8b-instruct	meta-llama/llama-3-70b-instruct
meta-llama/llama-3-8b-instruct	qwen/qwen3-235b-a22b
qwen/qwen3-32b	qwen/qwen3-14b
cohere/command-r-plus-08-2024	cohere/command-r-08-2024
moonshotai/kimi-k2-thinking	perplexity/sonar

Table 11: OpenRouter automatic routing model pool used by our OpenRouter baseline (as reported by the API at routing time).

use the same prompt template; Router-R1 uses a trained `Qwen2.5-7B-Instruct` routing model, while LLM Router directly uses `DeepSeek-V3.2` (no training).

E.5 Browse Extractor Prompt

The HLE `browse` tool can optionally use an LLM to extract and summarize relevant evidence from retrieved webpages. We report the extractor prompt used for this LLM-based summarization.

E.6 HLE Judge Prompt

HLE scoring uses an LLM-as-a-judge component aligned with the official HLE evaluation. We report the judge prompt used to compare a model response against the provided reference answer.

LLM Router / Router-R1 System Prompt (Policy LLM)

You are a model routing assistant. Your job is to select the best model for the given task.

Available models:
`{model_list}`

Instructions

1. Analyze the task in `<think>...</think>` tags
2. Select the best model using `<select>model_name</select>`

Example
`<think>This is a simple math question. A cheaper model would suffice.</think>`
`<select>deepseek/deepseek-v3.2</select>`

Rules
- You MUST output exactly one `<select>` tag
- The model name must match exactly from the available list
- Consider: task complexity, model strengths, cost-effectiveness

LLM Router / Router-R1 Model Descriptors

`openai/gpt-5`: Cost 1.25/10. 400K context. Top performer with SW 48% SR and HLE 25% SR. Best for complex multi-step reasoning and hard science problems.
`openai/gpt-oss-120b`: Cost 0.09/0.36. 131K context. Lowest cost with SW 33% SR and HLE 10% SR. Excellent cost-efficiency for moderate difficulty tasks.
`moonshotai/kimi-k2-0905`: Cost 0.39/1.9. 131K context. SW 31% SR and HLE 11% SR. Balanced option for general-purpose tasks with good instruction following.
`deepseek/deepseek-v3.2`: Cost 0.27/0.42. 164K context. SW 16% SR and HLE 16% SR. Strong on math and coding. Good for structured reasoning tasks.
`google/gemini-2.5-flash-lite`: Cost 0.1/0.4. 1M context. SW 23% SR and HLE 6% SR. Largest context window. Best for long document analysis.
`minimax/minimax-m2`: Cost 0.2/1. 196K context. SW 22% SR and HLE 8% SR. Fast inference. Suitable for simple QA and straightforward tasks.

Browse Extractor Prompt (LLM Summary)

Please process the following webpage content and user goal to extract relevant information:

Webpage Content
{webpage_content}

User Goal
{goal}

Task Guidelines

1. **Content Scanning for Rationale**: Locate the **specific sections/data** directly related to the user's goal within the webpage content
2. **Key Extraction for Evidence**: Identify and extract the **most relevant information** from the content; do not miss important information. Output the **full original context** as much as possible (it can exceed three paragraphs).
3. **Summary Output for Summary**: Organize into a concise paragraph with logical flow, prioritizing clarity and judging the contribution of the information to the goal.

Output Format: JSON format containing `"rational"`, `"evidence"`, and `"summary"` fields.

HLE Judge Prompt

Judge whether the following `[response]` to `[question]` is correct or not based on the precise and unambiguous `[correct_answer]` below.

```
[question]: {question}
[response]: {response}
```

Your judgement must be in the format and criteria specified below:

extracted_final_answer: The final exact answer extracted from the `[response]`. Put the extracted answer as `None` if there is no exact, final answer to extract from the response.

`[correct_answer]: {correct_answer}`

reasoning: Explain why the extracted_final_answer is correct or incorrect based on `[correct_answer]`, focusing only on whether there are meaningful differences. Do not comment on background, do not attempt to solve the problem, do not argue for any answer different than `[correct_answer]`; focus only on whether the answers match.

correct: Answer `yes` if extracted_final_answer matches the `[correct_answer]` given above, or is within a small margin of error for numerical problems. Answer `no` otherwise.

confidence: The extracted confidence score between 0% and 100% from `[response]`. Put 100 if there is no confidence score available.

Respond with a JSON object containing these four fields: extracted_final_answer, reasoning, correct, confidence.