NOT EVERY STEP MATTERS: LEARNING MULTI-TURN LLM ROUTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents complete tasks through a sequence of model calls and environment interactions, causing inference costs to accumulate. Routing each call to an appropriate model offers a way to reduce these costs while maintaining task performance. However, the capabilities needed can change over the course of a task, and choosing a different model does not matter equally at every step. We find that the value of model selection is concentrated at a small subset of steps. We call these steps *pivotal*: differences in the capabilities of candidate models at the same agent history lead to substantially different final task outcomes. A step can be difficult or critical to the task without being pivotal if changing the model makes little difference. We identify pivotal steps through counterfactual rollout trees and use their model comparisons to guide data collection, router training, and inference. Our router matches or outperforms the strongest fixed-model baseline on five of six interactive benchmarks. It also reduces total inference cost by 47.8% across the 473-task evaluation suite.

1 INTRODUCTION

Language models differ substantially in both capability and inference cost, motivating routers that select among them to balance response quality and cost (Ong et al., 2024; Hu et al., 2024; Feng et al., 2025). Most existing routers make this choice once per query. LLM agents present a different problem: they complete a task through a sequence of model calls, actions, and environment observations, and each model output changes the history on which later decisions depend (Yao et al., 2022b; Zhang et al., 2026b; Yang et al., 2026). Effective agent routing therefore depends not only on which model is strongest, but also on *where* model choice can change the final outcome.

We call such a decision a *pivotal step*: holding the task and current agent history fixed, differences between candidate models lead to substantially different final task outcomes. Pivotality depends on whether the models' capabilities meet the needs of the current step. A difficult or task-critical step can be non-pivotal if the models preserve similar outcomes, including when an initially incorrect action can be repaired later. Conversely, a routine step can be pivotal when one model takes an unnecessary action that leads away from success. Our held-out cases illustrate recovery from an incorrect SQL query and unnecessary verification before a WebShop purchase (Appendix B.5.1); Figure 1 gives a schematic comparison.

To learn which model to choose at each step, observing successful and failed trajectories is not enough. A successful trajectory does not show whether another model could have achieved the same outcome at lower cost. A failed trajectory does not tell us whether changing the model at a particular step would have prevented the failure. In both cases, we need to compare what happens when different models continue from the same agent history.

We obtain these comparisons through counterfactual rollouts, following different model choices from the same agent history to their downstream task outcomes. These comparisons account for both the immediate consequences of a model choice and the possibility of later recovery. We use them to investigate whether routing decisions are equally consequential throughout a trajectory, or whether the value of model selection is concentrated at particular steps.

Pivotal states account for only 14.9–27.9% of the analyzed decision states across three agent domains. Table 1 characterizes their role through oracle-guided replay within the collected trees. This

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

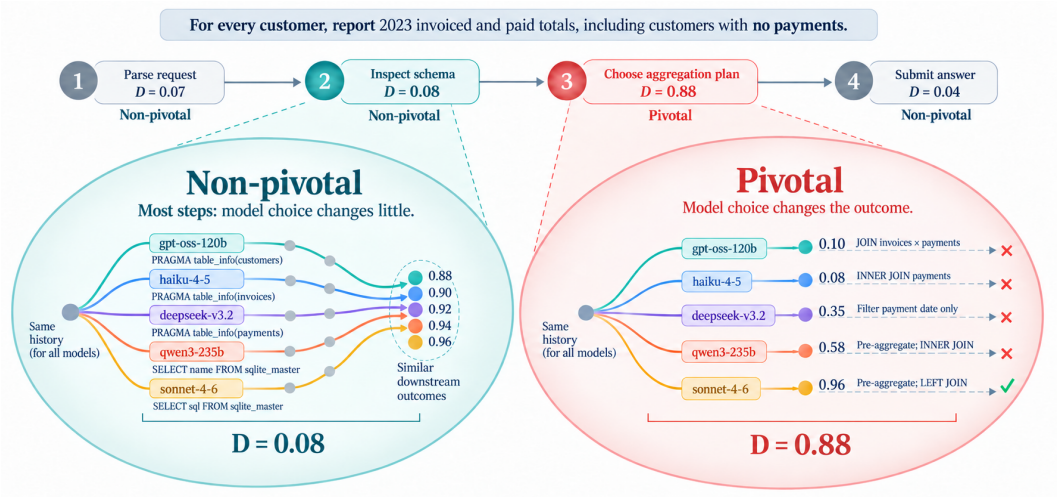

Figure 1: **Pivotal and non-pivotal model choices in a schematic SQL trajectory.** Each panel fixes the task and history. At step 2, different schema-inspection actions preserve similar downstream values; at step 3, different aggregation plans lead to sharply different values. D is the largest difference between candidate models' downstream values.

sparsity motivates a learning hypothesis: concentrating supervision on states that distinguish candidate models may be more effective than treating all decision states equally. We therefore ask whether pivotal-only supervision can match or outperform training on all decision states.

We use pivotality to guide data collection, training, and inference. During collection, we seek states that expose differences between candidate models. During training, we concentrate supervision on these comparisons to help the router learn which model to prefer at a given history. Pivotality also motivates how we balance performance and cost at inference time: when model choice is predicted to matter, we prioritize performance; when the leading models are predicted to achieve similar outcomes, we favor lower cost.

Pivotal-only training uses 32.7% of the available state–model examples and achieves higher mean performance than training on all decision states across all three source benchmarks (Section 5.1). In online evaluation across six benchmarks, including three OOD benchmarks, the resulting router matches or exceeds the strongest fixed-model baseline on five (Table 2). With performance-only inference, it also reduces total API cost by 47.8% across the evaluation suite. Pivotality-aware cost control provides further flexibility to trade performance for lower cost (Section 4.3).

2 Related Work

Per-query routing. Most LLM routers assign one model to an entire query. RouteLLM learns from preference data (Ong et al., 2024), while IPR predicts quality under a user-specified tolerance (Feng et al., 2025). Cascades escalate based on response-quality or answer-consistency signals (Chen et al., 2023; Yue et al., 2023). RouterBench (Hu et al., 2024) and ensembling methods (Wang et al., 2024) also operate at the query level. We instead route after each environment observation and study which states require a different model.

Stepwise routing for reasoning. Recent work studies reasoning capabilities (Ye et al., 2024; Xu et al., 2025; Gao et al., 2025; Ye et al., 2025) and adaptive reasoning strategies (Yue et al., 2025). Stepwise routers consider which model to use within a reasoning trace. STEER routes from a small model's confidence (Lee et al., 2025); TRIM uses process scores to identify critical reasoning steps (Kapoor et al., 2026); and Arbitrage predicts the local advantage of a target model over a draft (Maheswaran et al., 2025). These routers typically use a small–large model pair. Our router compares a broader panel at a shared agent state using downstream task outcomes.

Sequential and agentic routing. Recent work extends routing to sequential settings. DialRouter learns from search over dialogue branches (Zhang et al., 2026a); SeqRoute uses offline RL under a session-level budget (Xu et al., 2026); and MTRouter estimates per-turn utility from joint history–model embeddings (Zhang et al., 2026b). Hera compares device and cloud returns at shared states (Zhang et al., 2026c), while TwinRouterBench verifies step-level downgrades by executing mixed-model trajectories (Yang et al., 2026). These works already use downstream outcomes for routing or evaluation. We study whether routing supervision can concentrate on pivotal states, and use their within-state model contrasts to guide both data collection and router training.

Criticality and attribution. Critical-state and action-gap analyses compare the consequences of environment actions (Huang et al., 2018; Bellemare et al., 2015). ATLAS selects critical steps in expert trajectories for fine-tuning (Chen et al., 2025); CARL uses action entropy as a proxy for criticality (Shen et al., 2025); and CSO, ELPO, and Causal Agent Replay localize failures through action or continuation interventions (Li et al., 2026; Liang et al., 2026; Shah, 2026). Related attribution work studies the influence of input content on generation (Chang et al., 2025). These methods study the influence of actions or input content; we ask which *model identity* matters at a fixed agent state.

3 Pivotal Steps: Where Model Choice Matters

3.1 Defining pivotal steps

Pivotality concerns differences between candidate models at a particular step. A difficult step provides little routing opportunity if every model has a similarly poor outcome; a task-critical step can also be non-pivotal if all models handle it equally well. We therefore measure whether the models' different actions lead to different final task outcomes, including any later recovery.

At step t, state s_t contains the task and interaction history. The router chooses a model $m \in \mathcal{M}$ to generate the next action. A counterfactual rollout tree compares these choices from the same state. Each outgoing edge invokes one model; subsequent steps may use any candidate model. Let $\mathcal{A}(s)$ denote the models expanded at s and s'_m the state reached after choosing m. We back up the best terminal return found below each model choice:

$$Q^{\mathrm{perf}}(s,m) = \begin{cases} r(s,m), & \text{if the edge terminates,} \\ \max_{m' \in \mathcal{A}(s'_m)} Q^{\mathrm{perf}}(s'_m, m'), & \text{otherwise.} \end{cases} \tag{1}$$

Here $r(s,m)$ is the terminal reward. $Q^{\mathrm{perf}}(s,m)$ describes the best continuation found in the collected tree, including recovery after an incorrect action. The comparison asks what remains achievable after the current model choice, while allowing later steps to use other models. It does not require the selected model to complete the task alone. A high backed value indicates that a good continuation was found, rather than estimating how often that model would succeed. A branch without recorded success may still succeed outside the collected tree. We define

$$D(s) = \max_{m \in \mathcal{A}(s)} Q^{\mathrm{perf}}(s,m) - \min_{m \in \mathcal{A}(s)} Q^{\mathrm{perf}}(s,m), \qquad s \text{ is pivotal if } D(s) \geq \tfrac{1}{2}. \tag{2}$$

In binary-reward domains, a pivotal state has a successful recorded continuation after at least one model choice but not after another. For graded rewards, the threshold selects a return difference of at least 0.5. Appendix Table 16 examines the sensitivity of training-data selection to this threshold and an alternative pivotality criterion. Pivotality is thus relative to the candidate models and the current history; additional model branches or continuations can change the label. Section 5.3 examines the concrete capability differences behind these comparisons.

3.2 Pivotal-state sparsity and oracle-guided replay

We first ask how often model choice makes a large difference, and how much return is attainable when informed routing is restricted to those steps. We study these questions before training a router, using a separate plain-MCTS audit set. It contains one 3000-iteration tree for every task in a held-out split: 68 InterCode-SQL, 40 WebShop, and 100 SWE-bench tasks. Every task receives the same budget under standard UCB1 (Auer et al., 2002) with the same five-model panel. These audit trees are never used for router training.

Table 1: **Within-tree oracle upper bound and restricted oracle-guided replay.** Results use held-out tasks; parentheses show fractions of the all-step oracle return.

measure / replay policy	InterCode-SQL	WebShop	SWE-bench
pivotal-state share	27.9%	21.5%	14.9%
ORACLE at every step *(upper bound)*	0.956	0.960	0.730
ORACLE at pivotal steps only	**0.956** (100%)	**0.762** (79%)	**0.730** (100%)
ORACLE at non-pivotal steps only	0.343 (36%)	0.609 (63%)	0.040 (5%)

We measure sparsity as the fraction of states with at least two expanded model branches that satisfy Eq. 2. We measure where routing value lies by replaying three policies on the same trees. The all-step oracle always follows the expanded model edge with the highest backed value. The pivotal-only policy does so only at pivotal states and routes uniformly elsewhere; the non-pivotal-only policy does the converse. Table 1 reports their returns and fractions of the all-step oracle return. The all-step oracle upper-bounds return within the collected tree support. The hybrid rows report the returns of the specified replay policies; for graded rewards, they need not maximize return among all policies with the same restriction.

Pivotal states occupy only 14.9–27.9% of the multi-model states. Pivotal-only replay matches the all-step oracle return on InterCode-SQL and SWE-bench, and retains 79% on WebShop; non-pivotal-only replay retains 36%, 63%, and 5%, respectively. In binary-reward trees, the pivotal-only equality follows from max backup: all expanded branches at a non-pivotal state have identical backed values (Appendix B.7). Graded rewards can preserve subthreshold differences. We therefore interpret Table 1 as a within-tree decomposition. Whether these distinctions provide useful training supervision is tested separately through online evaluation and the data-selection ablation (Sections 4 and 5.1).

3.3 PIVOTAL-AWARE DATA COLLECTION

Useful routing examples reveal different outcomes after different model choices. Exhaustively exploring a horizon-T interaction would require $|\mathcal{M}|^T$ model sequences, so we construct partial trees with MCTS and direct the search budget toward informative comparisons. Each search iteration follows a tree policy until it expands an untried model or reaches a terminal state, then completes a rollout if needed. The terminal reward updates the search statistics along the selected path; rollout transitions are also retained in the collected tree. Trajectories are capped at $T = 10$ environment steps. After collection, Eq. 1 converts the recorded outcomes into model-choice targets.

Across tasks, a 150-iteration MCTS probe estimates p_i as the fraction of newly created terminal nodes that are successful. Inspired by GVM-RAFT (Yao et al., 2025), we allocate additional iterations as $n_i \propto 1/\sqrt{p_i + \alpha/p_i}$, with $\alpha = 0.01$. This gives more search to tasks with mixed outcomes; tasks with $p_i = 0$ receive no additional budget. The probe determines where to search, while pivotal labels still come from comparing models at a shared state. The additional search pool contains $3000|\mathcal{S}|$ iterations, where $\mathcal{S} = \{i : p_i > 0\}$, and the allocation rule redistributes this pool among retained tasks. An iteration that revisits a cached terminal does not add a new observation to p_i; its denominator therefore need not equal 150. Appendix B details the probe and its calibration.

Within a task, the final $D(s)$ is unavailable while the tree is growing. We use $U(s) = \mathrm{Var}_{m \in \mathcal{A}(s)} \hat{Q}(s, m)$ as an online proxy and max-back it through each child subtree as $U_{\mathrm{sub}}(s, m)$. Our selection rule is

$$\mathrm{score}(m) = \hat{Q}(s, m) + \big(C + \beta U_{\mathrm{sub}}(s, m)\big)\sqrt{\ln N(s)/N(s, m)}, \qquad C = 1.4,\ \beta = 2. \tag{3}$$

Here $\hat{Q}$ is the search value estimate and N denotes visit counts. The bonus favors subtrees that have exposed model differences and decays with additional visits. Max-backing the variance lets a branch receive additional exploration when informative comparisons occur deeper in its subtree, even if its immediate child shows little contrast. This online statistic guides search; the final pivotal label still uses the completed-tree return spread in Eq. 2. Appendix C compares bonus designs.

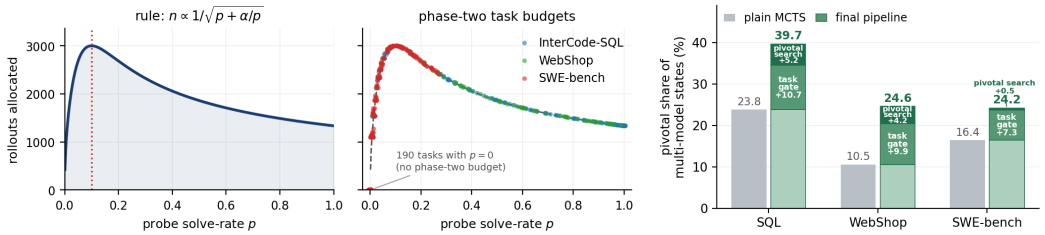

Figure 2: **Data collection targets states that distinguish models.** The two left panels show the allocation rule and task budgets. The right panel compares plain MCTS (gray) with the final collection (green), separating the task gate from the matched exploration change from $\beta = 0$ to Eq. 3. Appendix B gives the counts and controls.

Figure 2 shows the resulting collection. The final pivotal shares are 39.7%, 24.6%, and 24.2% on SQL, WebShop, and SWE-bench, respectively, gains of 15.9, 14.1, and 7.8 percentage points over plain MCTS. The figure separates the task gate from the matched change in within-tree exploration. The matched $\beta = 0$ and $\beta = 2$ runs reuse the same probe estimates and per-task budgets, so their difference isolates the within-task selection rule. The full improvement over plain MCTS also includes task selection and budget allocation. These collection results describe the training trees; the independent audit in Section 3.2 measures sparsity without either collection intervention. Appendix B reports the full counts and controls.

3.4 Learning which model to choose at pivotal steps

We train the router on pivotal states, grouping the models expanded from the same history. A model choice can affect both the final outcome and the cost of later interactions. We therefore predict both downstream performance and remaining cost. A shared history encoder feeds a performance head $\hat{Q}^{\text{perf}}(s, m)$ and a cost-to-go head $\hat{Q}^{\text{cost}}(s, m)$. The performance target is the backed return in Eq. 1; the cost target combines the immediate model cost with the minimum predicted continuation cost.

The pointwise baseline fits these targets separately for each state–model pair:

$$\mathcal{L}_{\text{perf}}(s, m) = \left(\hat{Q}^{\text{perf}}(s, m) - Q^{\text{perf}}(s, m)\right)^2, \tag{4}$$

$$\mathcal{L}_{\text{cost}}(s, m) = \left(\hat{Q}^{\text{cost}}(s, m) - [c(m) + \gamma \min_{m' \in \mathcal{A}(s'_m)} \bar{Q}^{\text{cost}}(s'_m, m')]\right)^2. \tag{5}$$

Here $\bar{Q}^{\text{cost}}$ is a target network; terminal edges have zero continuation cost. The baseline minimizes $\mathcal{L}_{\text{perf}} + \mathcal{L}_{\text{cost}}$.

Routing requires comparing models at the current history. We separate the value shared by all models from each model's relative advantage:

$$\bar{Q}(s) = \frac{1}{|\mathcal{A}(s)|} \sum_{m \in \mathcal{A}(s)} Q^{\text{perf}}(s, m), \qquad A(s, m) = Q^{\text{perf}}(s, m) - \bar{Q}(s). \tag{6}$$

The average $\bar{Q}(s)$ describes how promising the state is for the candidate pool as a whole; $A(s, m)$ describes how much better or worse a model's recorded continuation is than that average. Adding the same value to every model leaves their ordering unchanged, so $\arg\max_m Q^{\text{perf}}(s, m) = \arg\max_m A(s, m)$ over the expanded models. Pointwise regression fits both the common value and the model differences, while routing depends on the latter.

We therefore add a within-state ranking loss that directly compares the candidate models. Let $\hat{p}(m \mid s)$ be the softmax of predicted performance scores. We train against a soft distribution of the backed returns, which is unchanged by subtracting their common value:

$$\begin{aligned} \mathcal{L}_{\text{rank}}(s) &= -\sum_{m \in \mathcal{A}(s)} p_\tau(m \mid s) \log \hat{p}(m \mid s), \\ p_\tau(m \mid s) &= \frac{\exp(Q^{\text{perf}}(s, m)/\tau)}{\sum_{m' \in \mathcal{A}(s)} \exp(Q^{\text{perf}}(s, m')/\tau)}, \qquad \tau = 0.5. \end{aligned} \tag{7}$$

This target preserves ties and graded differences while emphasizing the comparison needed for model selection. We retain a pointwise performance loss to anchor predictions to return units and a cost loss to train the second head. The resulting *Anchored CE* objective is

$$\mathcal{L}_{\text{ours}} = \mathcal{L}_{\text{rank}} + \mathcal{L}_{\text{perf}} + \mathcal{L}_{\text{cost}}. \tag{8}$$

All three terms update the shared encoder. The performance anchor gives predicted return gaps a consistent scale for comparison with cost differences under Eq. 9. Pure CE omits $\mathcal{L}_{\text{perf}}$; the pointwise MSE baseline omits $\mathcal{L}_{\text{rank}}$. Both retain cost supervision. Appendix D analyzes both objectives and data selection.

3.5 Pivotality-aware inference

At inference time, we predict the performance and cost of choosing each model at the current history s. We compute $d(s)$ as the gap between the two highest predicted performance scores, and select

$$\pi_\lambda(s) = \arg\max_{m \in \mathcal{M}} \left[\hat{Q}^{\text{perf}}(s, m) - \lambda \frac{r}{r + d(s)} \hat{Q}^{\text{cost}}(s, m) \right], \qquad r > 0 \text{ (constant)}. \tag{9}$$

When the gap is large, $r/(r + d(s))$ becomes small. We therefore prioritize predicted performance and reduce the influence of cost on model selection. When the gap is small, the multiplier approaches one and retains more of the cost penalty. This favors lower-cost choices when their predicted performance is similar. The function adjusts this balance at each step rather than applying the same cost penalty throughout the trajectory.

We set $r = 0.05$ and vary λ to control the overall cost preference. At $\lambda = 0$, we select the model with the highest predicted performance. After each observation, we apply the same rule to the updated history. The router only sees this history, but its predictions target performance and cost beyond the next action. Appendix E compares alternative modulation rules.

4 Experiments

4.1 Experimental setup

We evaluate whether pivotal-step supervision improves routing on unseen trajectories. Evaluation is fully online: after every environment observation, the router selects a panel model and that model executes the next action in the official benchmark environment. The source domains are InterCode-SQL (Yang et al., 2023) (68 held-out tasks), WebShop (Yao et al., 2022a) (40 development tasks), and SWE-bench Verified (Jimenez et al., 2023) (100 tasks). The OOD suite contains ALFWorld (Shridhar et al., 2020) (50 tasks), InterCode CTF (100 tasks), and τ-bench retail (Yao et al., 2024) (115 tasks); none provides collection or training data. Although collection trajectories are capped at 10 steps, SWE-bench evaluation uses a configured limit of 50 agent steps, testing routing beyond the collection horizon. Appendix A gives the benchmark-specific interaction limits.

The five-model panel spans `sonnet-4-6`, `haiku-4-5`, `deepseek-v3.2`, `qwen3-235b`, and `gpt-oss-120b`. Our primary router fine-tunes Qwen3-Embedding-0.6B (Zhang et al., 2025b) over the agent history; an EmbeddingGemma-300m (Vera et al., 2025) variant tests whether the objective depends on the encoder. Baselines include uniform per-step routing, ALL-STRONG (`sonnet-4-6`), and ALL-CHEAP (`gpt-oss-120b`). Prior-router baselines cover task-level routing (ACRouter (Zhou et al., 2026)), multi-round routing (Router-R1 (Zhang et al., 2025a)), and query-level routing (PROTEUS (Bhatti et al., 2026) and RouteLLM (Ong et al., 2024)). We evaluate these adaptations in the same environments; Appendix A gives details.

We first evaluate performance-only inference with $\lambda = 0$ in Eq. 9. The main table reports this setting, whose costs arise from the learned model choices. We then vary λ to evaluate the performance–cost trade-off of pivotality-aware inference (Section 4.3).

Table 2 groups non-learned controls as *Fixed* and existing routers as *Prior*. In our rows, Gemma and Qwen name the text encoder; MSE, CE, and Anchored CE use the three objectives defined in Section 3.4, all of which train the cost head. Learned rows average three independently trained and evaluated seeds. Each *Avg.* column weights its three benchmarks equally. The table focuses on performance; Appendix A gives the harnesses, grading, full seed protocol, and cost accounting, and Appendix A.8 reports complete cost results.

Table 2: **Main online routing performance at $\lambda = 0$.** Entries are benchmark performance (mean $\pm$ std over three runs). Bold and underline mark the best and second-best column means across all methods, including fixed baselines; ties share the same mark.

method		(a) In-domain				(b) OOD			
		InterCode-SQL	WebShop	SWE-bench	**Avg.**	ALFWorld	InterCode CTF	τ-bench	**Avg.**
Baselines									
Fixed	RANDOM	0.255 $\pm$.017	0.396 $\pm$.031	0.650 $\pm$.010	0.434 $\pm$.012	0.767 $\pm$.042	0.203 $\pm$.031	0.600 $\pm$.038	0.523 $\pm$.022
	ALL-STRONG	0.358 $\pm$.009	0.551 $\pm$.046	0.650 $\pm$.053	0.520 $\pm$.024	**0.893 $\pm$.012**	0.260 $\pm$.010	**0.716 $\pm$.031**	**0.623 $\pm$.012**
	ALL-CHEAP	0.147 $\pm$.015	0.355 $\pm$.043	0.260 $\pm$.000	0.254 $\pm$.015	0.427 $\pm$.012	0.127 $\pm$.015	0.394 $\pm$.033	0.316 $\pm$.013
Prior	ACRouter	0.387 $\pm$.031	0.487 $\pm$.030	0.523 $\pm$.040	0.466 $\pm$.020	0.733 $\pm$.023	0.240 $\pm$.027	0.620 $\pm$.045	0.531 $\pm$.019
	Router-R1	0.348 $\pm$.017	0.464 $\pm$.006	0.623 $\pm$.029	0.478 $\pm$.011	0.840 $\pm$.035	0.223 $\pm$.006	0.600 $\pm$.031	0.554 $\pm$.016
	PROTEUS	0.137 $\pm$.023	0.348 $\pm$.037	0.447 $\pm$.015	0.311 $\pm$.015	0.747 $\pm$.031	0.187 $\pm$.015	0.409 $\pm$.045	0.448 $\pm$.019
	RouteLLM	0.358 $\pm$.008	0.456 $\pm$.026	0.613 $\pm$.038	0.476 $\pm$.016	**0.893 $\pm$.012**	0.270 $\pm$.020	0.548 $\pm$.023	0.570 $\pm$.011
Our methods									
Gemma	MSE	0.319 $\pm$.045	0.384 $\pm$.049	0.627 $\pm$.050	0.443 $\pm$.028	0.840 $\pm$.092	0.227 $\pm$.045	0.594 $\pm$.120	0.554 $\pm$.053
	CE	0.363 $\pm$.042	0.458 $\pm$.069	0.643 $\pm$.029	0.488 $\pm$.029	0.887 $\pm$.012	0.240 $\pm$.010	0.672 $\pm$.041	0.600 $\pm$.015
	Anchored CE	0.363 $\pm$.017	0.551 $\pm$.018	0.600 $\pm$.000	0.505 $\pm$.008	0.873 $\pm$.031	0.257 $\pm$.015	0.681 $\pm$.013	0.604 $\pm$.012
Qwen	MSE	0.299 $\pm$.047	0.484 $\pm$.021	0.587 $\pm$.137	0.457 $\pm$.049	0.787 $\pm$.129	0.230 $\pm$.044	0.678 $\pm$.046	0.565 $\pm$.048
	CE	0.377 $\pm$.009	0.545 $\pm$.015	0.600 $\pm$.020	0.507 $\pm$.009	0.873 $\pm$.012	0.263 $\pm$.006	0.649 $\pm$.013	0.595 $\pm$.006
	Anchored CE	**0.422 $\pm$.008**	**0.573 $\pm$.036**	**0.710 $\pm$.036**	**0.568 $\pm$.017**	**0.893 $\pm$.012**	**0.280 $\pm$.026**	0.681 $\pm$.035	0.618 $\pm$.015

4.2 End-to-end routing performance

At $\lambda = 0$, the primary router leads all three source benchmarks, scoring 0.422 on InterCode-SQL, 0.573 on WebShop, and 0.710 on SWE-bench. Its source-domain average is 0.568, compared with 0.520 for ALL-STRONG and 0.478 for Router-R1, the best prior router (Table 2).

The objective ordering supports the proposed learning design. Moving from MSE to CE raises the source average from 0.443 to 0.488 with Gemma and from 0.457 to 0.507 with Qwen. Adding the performance anchor raises these averages further, to 0.505 and 0.568, respectively (Table 2). Within-state ranking improves both encoders; the anchor yields the strongest Qwen router.

On the OOD suite, where no collection or training data are used, the primary router averages 0.618: only 0.005 below ALL-STRONG and 0.048 above RouteLLM, the strongest prior router. It matches ALL-STRONG on ALFWorld, exceeds it on CTF, and reaches 0.681 versus 0.716 on τ-bench; on every OOD benchmark it matches or exceeds all evaluated prior routers (Table 2).

These gains also require substantially less inference cost. The primary router costs less than ALL-STRONG on all six benchmarks, reduces OOD macro cost by 26.4%, and lowers the expected cost of the 473-task suite from $\$290.95$ to $\$151.75$, a 47.8% reduction (Appendix Tables 10 and 11).

4.3 Performance–cost trade-off

We vary λ in Eq. 9 to evaluate pivotality-aware inference at different cost levels. Figure 3 compares the resulting curve with the routing baselines and the performance-only routers.

On SQL, $\lambda = 0.4$ reaches 0.406 at $\$0.0176$ per task, exceeding ACRouter's 0.387 at $\$0.0233$ and RouteLLM's 0.358 at $\$0.0195$. On WebShop, $\lambda = 0.2$ reaches 0.501 at $\$0.0180$, compared with ACRouter's 0.487 at $\$0.0255$. On SWE-bench, $\lambda = 0.2$ reaches 0.500 at $\$0.369$, near ACRouter's 0.523 while using 62% less cost (Fig. 3). These results show that the router can trade performance for lower cost as λ increases.

5 Ablation Study

We examine the choice of training states, the effect of adjusting the inference-time cost penalty, and the model differences captured by pivotality.

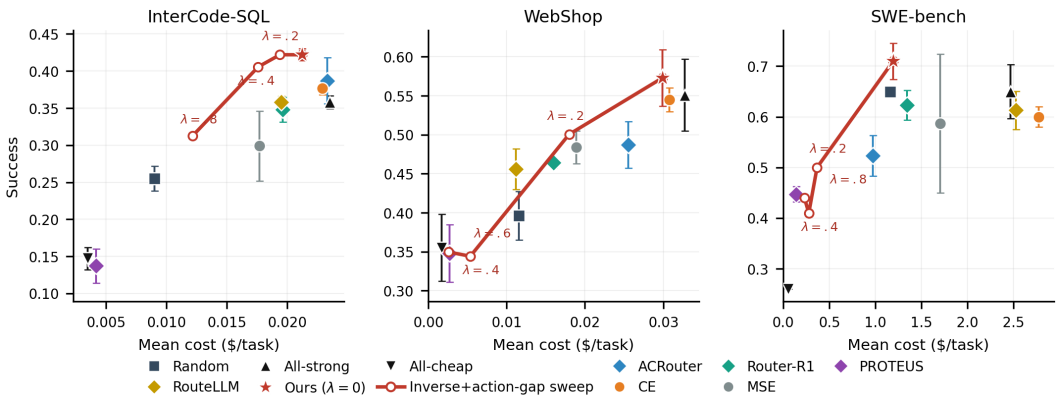

Figure 3: **Pivotality-aware inference reaches competitive low-cost operating points.** The red line varies λ in Eq. 9; markers show baselines and $\lambda = 0$ routers. Appendix Fig. 10 compares the rules.

Table 3: **Pivotal-only training outperforms the all-state alternatives.** Results are mean $\pm$ standard deviation over three seeds. The middle row reports the columnwise best results among fifteen pivotal-aware mechanisms that retain all states; Appendix Table 33 reports every configuration.

training states	data share	InterCode-SQL	WebShop	SWE-bench
all decision states	100%	$0.373 \pm .023$	$0.481 \pm .024$	$0.653 \pm .021$
best all-state attempt	100%	$0.402 \pm .022$	$0.566 \pm .013$	$0.703 \pm .006$
pivotal only	32.7%	$\mathbf{0.422 \pm .008}$	$\mathbf{0.573 \pm .036}$	$\mathbf{0.710 \pm .036}$

5.1 Does pivotal-only training outperform all-state training?

We hold the Qwen encoder, Anchored CE objective, optimization protocol, and $\lambda = 0$ evaluation fixed and vary only which decision states enter training. The resulting comparison directly tests the value of pivotality as a training-state filter.

Pivotal-only training uses 32.7% of the examples yet improves performance by 0.049, 0.092, and 0.057 on SQL, WebShop, and SWE-bench. We also test fifteen ways to retain all states while emphasizing pivotal ones through reweighting, gating, adaptive temperatures, or objective selection. Their best results also remain below pivotal-only training (Table 3).

5.2 Effect of inference-time modulation

We keep the trained router fixed and compare our state-dependent penalty with a fixed cost coefficient at every state. Both rules share the $\lambda = 0$ endpoint. We sweep $\lambda \in \{0.01, 0.1, 0.2\}$ for the fixed rule and $\{0.2, 0.4, 0.8\}$ for ours.

The fixed rule is highly sensitive to λ: performance and cost change substantially within this small coefficient range (Fig. 4). On SQL, increasing λ from 0 to 0.01 reduces success from 0.422 to 0.377, with little cost reduction. On SWE-bench, the same change reduces cost sharply but lowers success from 0.710 to 0.440. Because the fixed rule already produces such large changes over this small range, we did not extend its sweep to larger coefficients. Our state-dependent rule weakens the cost penalty when the leading models have a larger predicted performance gap.

At similar realized costs, our rule matches or improves on the fixed-penalty curve throughout their shared cost range on SQL. On WebShop and SWE-bench, our $\lambda = 0.2$ setting also improves on the fixed curve at similar costs. The benefit is not uniform: stronger penalties can still reduce performance, and SWE-bench is non-monotonic. Appendix E compares modulation rules and scales.

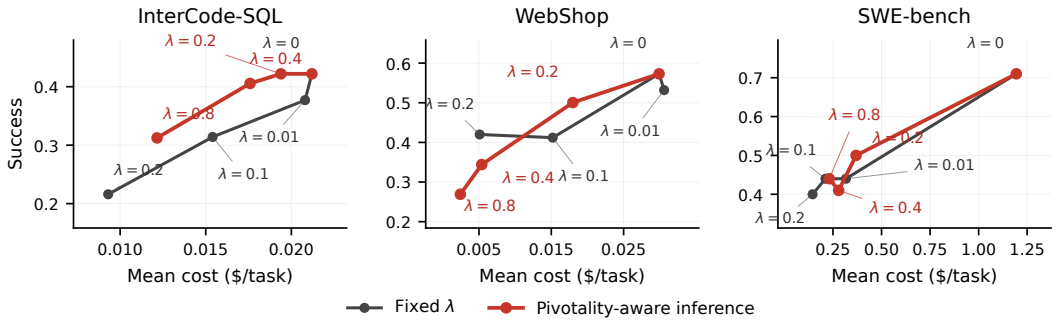

Figure 4: **State-dependent versus fixed cost penalties.** Gray uses one fixed cost coefficient at all states; red uses Eq. 9. Labels give each point's λ; both rules share the $\lambda = 0$ endpoint.

Table 4: **Different actions can preserve or change the final outcome.** Rows within each case share the same history. Outcomes describe recorded continuations, which may use other models later; full actions appear in Appendix B.5.1.

model choice	next action	recorded outcome
SQL 126: non-pivotal — query the molecule ID		
Qwen3-235B	Select the label instead	Corrected later; success
Other four models	Return molecule IDs	Success, directly or later
WebShop 157: pivotal — product and options already selected		
Sonnet 4.6	`click[features]`	No recorded success
Other four models	`click[buy now]`	Success

5.3 Pivotality reflects differences between candidate models

A step's difficulty or importance alone does not determine whether routing matters. Table 4 compares all five models at two histories to show when their actions change the outcome.

At SQL 126, the agent must query a molecule ID to answer the question. Qwen3-235B initially selects the label instead, but a later action corrects the output. The other four models also preserve success, either immediately or through a later action. This query is necessary to the task, and one model makes a local error, yet the step is non-pivotal: changing the model does not change the successful outcome available in the recorded tree.

A simple step can nevertheless be pivotal. At WebShop 157, the product and its options have already been selected; the remaining action is a single click to buy. Four models take this action and succeed, while Sonnet checks the product features again and has no successful recorded continuation. Here, the models differ in recognizing when to stop checking and complete the purchase. The action itself is simple, but this difference changes the final outcome and makes the step pivotal.

Appendix B.5.1 gives full actions and further examples: SQL states where no model branch succeeds, and a SWE-bench case with a shared diagnosis but different execution.

6 Conclusion

Model choice matters when candidate models differ in their ability to meet the needs of the current agent step. We identify these pivotal steps through their effects on final task outcomes, including later recovery, rather than through difficulty alone.

We use pivotality to guide data collection, training, and inference. Pivotal-only training uses 32.7% of the available state–model examples and matches or exceeds the evaluated all-state alternatives. Across six interactive benchmarks, the router matches or exceeds always using the strongest model on five, while reducing total expected cost by 47.8% over the 473-task suite at $\lambda = 0$. Varying λ yields competitive performance at lower cost than prior routers. These results support using pivotality to guide both learning and inference.

AI USE STATEMENT

We used generative AI tools to aid and polish writing, assist with retrieval and discovery of related work, and draft portions of the manuscript. These uses included improving the clarity and flow of the text, identifying potentially relevant literature, and producing initial drafts that the authors subsequently reviewed and revised. The authors take full responsibility for the final manuscript, including its citations, claims, and conclusions.

REPRODUCIBILITY STATEMENT

We define model-choice value and pivotal states in Section 3.1, and the collection and ranking methods in Sections 3.3 and 3.4. Appendix A consolidates router training, benchmark grading, seeds, and cost accounting; Appendix B gives tree construction and collection budgets; and Appendix D gives training diagnostics and data-selection ablations.

ETHICS STATEMENT

This work routes among existing, publicly available or commercially served language models on standard agentic benchmarks; it does not involve human subjects, new data collection from people, or personally identifying information. The intended effect is to reduce the compute and monetary cost of running multi-turn agents. We note that a cost-aware router inherits the behavior of the models it selects, so a router tuned aggressively toward cheap models may degrade reliability in ways that are unevenly distributed across task types; our cost-quality analysis is meant in part to make that trade-off explicit rather than implicit.

REFERENCES

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002. doi: 10.1023/A:1013689704352.

Marc G. Bellemare, Georg Ostrovski, Arthur Guez, Philip S. Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. *arXiv preprint arXiv:1512.04860*, 2015.

Amit Singh Bhatti, Vishal Vaddina, and Dagnachew Birru. PROTEUS: SLA-aware routing via lagrangian RL for multi-LLM serving systems. *arXiv preprint arXiv:2601.19402*, 2026.

Yurui Chang, Bochuan Cao, Yujia Wang, Jinghui Chen, and Lu Lin. JoPA: Explaining large language model's generation via joint prompt attribution. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22106–22122. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1074.

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

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. ATLAS: Agent tuning via learning critical steps. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25334–25349. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1299.

Aosong Feng, Balasubramaniam Srinivasan, Yun Zhou, Zhichao Xu, Kang Zhou, Sheng Guan, Yueyan Chen, Xian Wu, Ninad Kulkarni, Yi Zhang, Zhengyuan Shen, Dmitriy Bespalov, Soumya Smruti Mishra, Yifei Teng, Darren Yow-Bang Wang, Haibo Ding, and Lin Lee Cheong. IPR: Intelligent prompt routing with user-controlled quality-cost trade-offs. *arXiv preprint arXiv:2509.06274*, 2025.

Zijun Gao, Zhikun Xu, Xiao Ye, and Ben Zhou. CORE: Concept-oriented reinforcement for bridging the definition-application gap in mathematical reasoning. *arXiv preprint arXiv:2512.18857*, 2025.

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

Sandy H. Huang, Kush Bhatia, Pieter Abbeel, and Anca D. Dragan. Establishing appropriate trust via critical states. *arXiv preprint arXiv:1810.08174*, 2018. IROS 2018.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*, 2023. ICLR 2024.

Vansh Kapoor, Aman Gupta, Hao Chen, Anurag Beniwal, Jing Huang, and Aviral Kumar. TRIM: Hybrid inference via targeted stepwise routing in multi-step reasoning tasks. *arXiv preprint arXiv:2601.10245*, 2026. ICLR 2026.

Sangmook Lee, Dohyung Kim, Hyukhun Koh, Nakyeong Yang, and Kyomin Jung. Confidence-guided stepwise model routing for cost-efficient reasoning. *arXiv preprint arXiv:2511.06190*, 2025.

Mukai Li, Qingcheng Zeng, Tianqing Fang, Zhenwen Liang, Linfeng Song, Qi Liu, Haitao Mi, and Dong Yu. Verified critical step optimization for LLM agents. *arXiv preprint arXiv:2602.03412*, 2026. Findings of ACL 2026.

Qiao Liang, Yuke Zhu, Chao Ge, Lei Yang, Ying Shen, Bo Zheng, and Sheng Guo. Learning from the irrecoverable: Error-localized policy optimization for tool-integrated LLM reasoning. *arXiv preprint arXiv:2602.09598*, 2026.

Monishwaran Maheswaran, Rishabh Tiwari, Yuezhou Hu, Kerem Dilmen, Coleman Hooper, Haocheng Xi, Nicholas Lee, Mehrdad Farajtabar, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Arbitrage: Efficient reasoning via advantage-aware speculation. *arXiv preprint arXiv:2512.05033*, 2025.

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

Jaineet Shah. Causal agent replay: Counterfactual attribution for LLM-agent failures. *arXiv preprint arXiv:2606.08275*, 2026.

Leyang Shen, Yang Zhang, Chun Kai Ling, Xiaoyan Zhao, and Tat-Seng Chua. CARL: Criticality-aware agentic reinforcement learning. *arXiv preprint arXiv:2512.04949*, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020. ICLR 2021.

Henrique Schechter Vera, Sahil Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, Daniel Cer, Alice Lisak, Min Choi, Lucas Gonzalez, Omar Sanseviero, Glenn Cameron, Ian Ballantyne, Kat Black, Kaifeng Chen, Weiyi Wang, Zhe Li, Gus Martins, Jinhyuk Lee, Mark Sherwood, Juyeong Ji, Renjie Wu, Jingxiao Zheng, Jyotinder Singh, Abheesht Sharma, Divyashree Sreepathihalli, Aashi Jain, Adham Elarabawy, AJ Co, Andreas Doumanoglou, Babak Samari, Ben Hora, Brian Potetz, Dahun Kim, Enrique Alfonseca, Fedor Moiseev, Feng Han, Frank Palma Gomez, Gustavo Hernández Ábrego, Hesen Zhang, Hui Hui, Jay Han, Karan Gill, Ke Chen, Koert Chen, Madhuri Shanbhogue, Michael Boratko, Paul Suganthan, Sai Meher Karthik Duddu, Sandeep Mariserla, Setareh Ariafar, Shanfeng Zhang, Shijie Zhang, Simon Baumgartner, Sonam Goenka, Steve Qiu, Tanmaya Dabral, Trevor Walker, Vikram Rao, Waleed Khawaja, Wenlei Zhou, Xiaoqi Ren, Ye Xia, Yichang Chen, Yi-Ting Chen, Zhe Dong, Zhongli Ding, Francesco Visin, Gaël Liu, Jiageng Zhang, Kathleen Kenealy, Michelle Casbon, Ravin Kumar, Thomas Mesnard, Zach Gleicher, Cormac Brick, Olivier Lacombe, Adam Roberts, Qin Yin, Yunhsuan Sung, Raphael Hoffmann, Tris Warkentin, Armand Joulin, Tom Duerig, and Mojtaba Seyedhosseini. EmbeddingGemma: Powerful and lightweight text representations. *arXiv preprint arXiv:2509.20354*, 2025.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

Zhikun Xu, Ming Shen, Jacob Dineen, Zhaonan Li, Xiao Ye, Shijie Lu, Aswin RRV, Chitta Baral, and Ben Zhou. ToW: Thoughts of words improve reasoning in large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3057–3075. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.157.

Zhongling Xu, Shunan Zheng, and Wei Wang. SeqRoute: Global budget-aware sequential LLM routing via offline reinforcement learning. *arXiv preprint arXiv:2605.25424*, 2026.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. *arXiv preprint arXiv:2306.14898*, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

Pei Yang, Wanyi Chen, Tongyun Yang, Pengbin Feng, Jiarong Xing, Wentao Guo, Yuhang Yao, Yuhang Han, Hanchen Li, Xu Wang, Zeyu Wang, Jie Xiao, Anjie Yang, Liang Tian, Lynn Ai, Eric Yang, and Tianyu Shi. TwinRouterBench: Fast static and live dynamic evaluation for realistic agentic LLM routing. *arXiv preprint arXiv:2605.18859*, 2026.

Jiarui Yao, Yifan Hao, Hanning Zhang, Hanze Dong, Wei Xiong, Nan Jiang, and Tong Zhang. Optimizing chain-of-thought reasoners via gradient variance minimization in rejection sampling and RL. *arXiv preprint arXiv:2505.02391*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b. ICLR 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Xiao Ye, Andrew Wang, Jacob Choi, Yining Lu, Shreya Sharma, Lingfeng Shen, Vijay Murari Tiyyala, Nicholas Andrews, and Daniel Khashabi. AnaloBench: Benchmarking the identification of abstract and long-context analogies. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13060–13082. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.725.

Xiao Ye, Shaswat Shrivastava, Zhaonan Li, Jacob Dineen, Shijie Lu, Avneet Ahuja, Ming Shen, Zhikun Xu, and Ben Zhou. CC-LEARN: Cohort-based consistency learning. *arXiv preprint arXiv:2506.15662*, 2025.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *arXiv preprint arXiv:2310.03094*, 2023. ICLR 2024.

Murong Yue, Wenlin Yao, Haitao Mi, Dian Yu, Ziyu Yao, and Dong Yu. DOTS: Learning to reason dynamically in LLMs via optimal reasoning trajectories search. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tn2mjzjSyR.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-R1: Teaching LLMs multi-round routing and aggregation via reinforcement learning. *arXiv preprint arXiv:2506.09033*, 2025a. NeurIPS 2025.

Jiarui Zhang, Xiangyu Liu, Yong Hu, Chaoyue Niu, Hang Zeng, Shaojie Tang, Fan Wu, and Guihai Chen. From myopic selection to long-horizon awareness: Sequential LLM routing for multi-turn dialogue. *arXiv preprint arXiv:2604.12385*, 2026a.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Yiqun Zhang, Hao Li, Zihan Wang, Shi Feng, Xiaocui Yang, Daling Wang, Bo Zhang, Lei Bai, and Shuyue Hu. MTRouter: Cost-aware multi-turn LLM routing with history-model joint embeddings. *arXiv preprint arXiv:2604.23530*, 2026b. ACL 2026.

Yuxin Zhang, Mengxue Hu, Zheng Lin, Xiaoyi Fan, Fan Xie, Zihan Fang, Jing Yang, Wenjun Zhu, Zhiwen Chen, Chengfei Lv, and Zhe Chen. Hera: Learning long-horizon coordination for device-cloud collaborative LLM agents. *arXiv preprint arXiv:2605.24598*, 2026c.

Pengfei Zhou, Zhiwei Tang, Yixing Ma, Jiasheng Tang, Yizeng Han, Zhenglin Wan, Fanqing Meng, Wei Wang, Bohan Zhuang, Wangbo Zhao, and Yang You. Agent-as-a-router: Agentic model routing for coding tasks. *arXiv preprint arXiv:2606.22902*, 2026.

A Implementation and Evaluation Details

A.1 Data partitions and model panel

We partition the root tasks of each source benchmark into train, development, and test sets in a $60/20/20$ ratio (Table 5). All states descending from a root task remain in the same partition, and collection and router training use only training roots.

The routed panel is `sonnet-4-6`, `haiku-4-5`, `deepseek-v3.2`, `qwen3-235b`, and `gpt-oss-120b`. We freeze the AWS Bedrock `us-east-2` on-demand Standard rates in Table 6, checked June 8, 2026.[1] Reported dollar costs apply these rates to each call's realized token

[1] https://aws.amazon.com/bedrock/pricing/

Table 5: **Source-domain task partitions.** Splits are defined over root tasks, before collecting any descendant states.

benchmark	universe	train	dev	test
InterCode-SQL	338	202	68	68
WebShop	200	120	40	40
SWE-bench Verified	500	300	100	100

counts. The cost-head step target $c(m)$ instead evaluates the same price formula at the fixed observed SQL mean token mix shown in the table and normalizes by Sonnet's blended cost. The larger ratio for GPT-OSS than for Qwen therefore reflects GPT-OSS's longer mean output, not a higher per-token price.

Table 6: **Model prices and training cost ratios.** Input and output prices are US dollars per million tokens. The fixed mix is mean input/output tokens per call and is used only to compute the dimensionless, Sonnet-normalized cost-head target $c(m)$; online dollar costs use realized usage.

model	input price	output price	fixed mix (in/out)	$c(m)$
`sonnet-4-6`	3.0000	15.0000	912/23	1.0000
`haiku-4-5`	1.0000	5.0000	908/20	0.3272
`deepseek-v3.2`	0.6200	1.8500	846/13	0.1780
`qwen3-235b`	0.2266	0.9064	855/16	0.0676
`gpt-oss-120b`	0.1545	0.6180	908/313	0.1083

ALFWorld, InterCode CTF and τ-bench retail have no collection or training roots; their 50, 100 and 115 evaluation tasks, respectively, form the OOD suite. On τ-bench the simulated user is pinned to `haiku-4-5` in every row, so it is identical across methods, and its tokens are excluded from the reported router cost. Because Bedrock exposes different tool-calling interfaces across the five panel models, we render τ-bench's own tool schemas into the system prompt as a JSON action grammar, uniformly for all five; absolute scores are therefore comparable across our rows.

A.2 Router architecture and optimization

The primary router encodes the task and agent history with Qwen3-Embedding-0.6B (Zhang et al., 2025b). Inputs are limited to 2048 tokens and left-truncated so that the most recent agent steps are retained. The encoder is fully fine-tuned, and no separate model-history feature is appended. Its 1024-dimensional embedding passes through a shared $1024 \to 512 \to 256$ MLP with ReLU and LayerNorm after each hidden layer. Separate linear heads output five performance scores (with sigmoid activation) and five nonnegative cost scores (with softplus activation). EmbeddingGemma-300m (Vera et al., 2025) replaces only the text encoder in the encoder ablation.

The completed trees contain 761,656 state–model examples at decision states (states with at least two expanded models). The primary training base retains the 248,734 examples whose state satisfies $D(s) \geq \frac{1}{2}$. For each retained state, Eq. 7 normalizes only over models expanded from the same history. The target temperature is $\tau = 0.5$. Eq. 8 assigns unit weight to the ranking, performance, and cost terms. Pure in-state CE omits only $\mathcal{L}_{\text{perf}}$, while the regression baseline omits only $\mathcal{L}_{\text{rank}}$; every objective trains the cost head on the same examples.

For the cost target in Eq. 5, $\gamma = 0.99$, and $c(m)$ is a fixed, dimensionless blended input/output price ratio normalized so that `sonnet` has cost one (Table 6); it does not use the realized token count of an individual call. The target network is hard-updated every 500 optimizer steps.

We train for one epoch with AdamW at learning rate 10^{-5} and cosine decay to 10^{-6}. The eight-GPU run uses eight examples per GPU, no gradient accumulation, and an effective batch size of 64. Training uses bfloat16 and gradient checkpointing. Every learned configuration is trained under three seeds, which change model initialization and data order while holding the split and examples fixed. At evaluation time, Eq. 9 is applied before every environment action; all rows in Table 2 use $\lambda = 0$.

A.3 EVALUATION HARNESSES

We follow the official evaluation harness for each benchmark. We leave the task instances, environment dynamics, agent scaffold, action parser, stopping conditions, and grader unchanged; the only intervention is that the router selects which panel model handles each model call. InterCode-SQL and WebShop use their ReAct-style scaffolds (Yao et al., 2022b), and SWE-bench Verified uses SWE-agent (Yang et al., 2024). All panel models use temperature 0. The selected model's output is passed to the original action parser and environment, so the reported outcomes are end-to-end agent results rather than offline routing accuracy. Table 7 lists the task counts and interaction limits.

Table 7: **Online evaluation protocols.** All methods use the same tasks, agent harnesses, and limits within each benchmark.

benchmark	tasks	interaction limit
InterCode-SQL	68	10 steps
WebShop	40	10 steps
SWE-bench Verified	100	50 configured calls
ALFWorld	50	35 steps
InterCode CTF	100	10 steps

A.4 METRICS

For InterCode-SQL, reported online success is one only when the final submitted query returns the complete correct result set; partial result-set overlap does not count as evaluation success. Tree collection uses the same criterion for stopping and task gating, but preserves the environment's terminal overlap score when a trajectory reaches the depth cap (Appendix B). WebShop performance is the official graded terminal reward in $[0, 1]$. SWE-bench performance is the resolved rate from the official `FAIL_TO_PASS` and `PASS_TO_PASS` tests. ALFWorld and InterCode CTF use their official terminal task-success indicators. Benchmark performance is the arithmetic mean over evaluation tasks.

Cost is the mean API price in US dollars per task, accumulated over every model call in the agent trajectory. Micro averages weight benchmark means by their evaluation-set sizes, whereas macro averages weight benchmarks equally; the OOD averages cover ALFWorld, InterCode CTF and τ-bench retail. All reported means and sample standard deviations in the online result tables are computed over the same three seeds. For aggregate columns with normalized benchmark weights w_b, uncertainty is propagated as $\sqrt{\sum_b w_b^2 \sigma_b^2}$ from the benchmark-level standard deviations.

A.5 PUBLISHED-ROUTER ADAPTATIONS

ACRouter. ACRouter (Zhou et al., 2026) is a per-task method: it selects one model at the root and we cache that model for the rest of the episode. We warm-start its Qwen-embedding memory with the 409 collected train tasks and counterfactual fixed-model labels for all five panel models. When a fixed-model path leaves the collected tree, its continuation is estimated by the subtree Monte Carlo mean. We use $k = 8$ neighbors, prior weight 0.5, similarity temperature 0.1, and a $20{,}000$-record FIFO cap. The UCB bonus is zero because the counterfactual warm start already observes every arm.

Router-R1. The original Router-R1 (Zhang et al., 2025a) generates subqueries with a 3B language-model policy; it does not directly implement environment-step routing. We therefore report an offline-tree adaptation, not a faithful reproduction. A trainable Qwen state encoder and five-way policy are optimized with REINFORCE and a learned value baseline on recorded tree trajectories. Training uses decision states, an expanded-action mask, depth 12, six epochs, Adam at 10^{-5}, and 16 tasks per batch. The cost coefficient is $\alpha = 0.1$, entropy coefficient is 0.01, value coefficient is 0.5, and each batch receives one policy epoch.

PROTEUS. Our PROTEUS (Bhatti et al., 2026) adaptation follows the published description and holds the state encoder fixed. A frozen Qwen embedding is projected by a 512-hidden-layer encoder

to a 256-dimensional latent; the policy has one 128-unit hidden layer. It trains on pivotal states for eight epochs with Adam at 10^{-3} and batch size 256. Quality floors are sampled uniformly over 21 dual bins, the policy gradient uses the exact action expectation, and an automatically normalized oracle cost head supplies cost-to-go. The main-table setting $\lambda = 0$ maps to its native quality floor $\tau = 0$.

A.6 Training seeds and cost accounting

Every online result in Tables 2–11 reports the mean and sample standard deviation over three runs. All learned routers, including our methods and the trained adaptations of published-router baselines, use three independent *training* seeds. The seeds change initialization and data order while holding each method's split and training examples fixed; each resulting checkpoint is evaluated online once. The reported standard deviations therefore include training-seed variability, as well as any variability in online evaluation. For the fixed controls, the three runs repeat online evaluation without training. Diagnostic sweeps that use a different protocol are marked explicitly. Tree replay uses a state-path-keyed random choice and reports one deterministic point estimate.

Dollar cost is recorded by each benchmark harness over the same live runs used for quality. For SWE-bench, this is SWE-agent's `model_stats.instance_cost`, accumulated per API call at the price of the selected model. All tables report mean dollars per task. Cost standard deviations are computed over the same three seeds as performance.

A.7 Runtime-length and KV-cache audit

We additionally audit the available raw online outputs for the source-domain methods. For SQL and WebShop, a step is the recorded `n_steps`; for SWE-bench, it is the billed `model_stats.api_calls`. Table 8 reports the mean and sample standard deviation across the three run-level means.

Table 8: **Recorded interaction length in the online outputs.** Entries are mean ± sample standard deviation over three runs; SWE-bench reports billed API calls.

method	InterCode-SQL steps	WebShop steps	SWE-bench calls
RANDOM	$8.59 \pm .28$	$6.73 \pm .21$	$43.52 \pm .43$
ALL-STRONG	$7.76 \pm .06$	$6.30 \pm .07$	$35.51 \pm .31$
ALL-CHEAP	$9.35 \pm .01$	$6.53 \pm .27$	$25.44 \pm .58$
ACRouter	$7.70 \pm .26$	$6.54 \pm .13$	33.68 ± 1.45
Router-R1	$8.03 \pm .04$	$6.50 \pm .33$	49.96 ± 1.04
PROTEUS	$9.25 \pm .14$	$6.90 \pm .26$	28.09 ± 1.92
RouteLLM	$8.00 \pm .14$	$6.08 \pm .28$	$36.26 \pm .29$
Gemma / MSE	$8.14 \pm .31$	$6.59 \pm .40$	39.16 ± 4.24
Gemma / CE	$8.01 \pm .20$	$6.40 \pm .50$	41.86 ± 7.05
Gemma / Anchored CE	$8.00 \pm .02$	$6.72 \pm .26$	37.43 ± 3.25
Qwen / MSE	$8.32 \pm .42$	$6.27 \pm .32$	42.98 ± 3.12
Qwen / CE	$7.84 \pm .14$	$6.36 \pm .11$	46.49 ± 4.27
Qwen / Anchored CE	$7.79 \pm .10$	$6.56 \pm .18$	$44.80 \pm .82$

To estimate the opportunity for cross-step KV reuse, we call step t cache-eligible when it uses the same model as step $t-1$ in the same episode. The denominator is all model calls, including the first call of each episode, which is necessarily ineligible. This adjacent-step definition assumes model-specific caches and does not count a switch away and later return to a model. Thus, even fixed policies have eligibility below 100%: their fraction is $1 - N_{\text{episodes}}/N_{\text{calls}}$ because every episode starts with an ineligible call.

Table 9: **Potential cross-step KV-cache reuse estimated from routed model sequences.** Eligible is the percentage of all calls immediately preceded by a call to the same model. Each episode's first call is included in the denominator and is ineligible. Daggers mark fixed-policy values inferred from recorded step totals.

	SQL eligible	WebShop eligible	SWE eligible
RANDOM	–	–	19.9%
ALL-STRONG	87.1%†	84.1%†	97.2%
ALL-CHEAP	89.3%†	84.7%†	96.1%
ACRouter	87.0%	84.7%	97.0%
Router-R1	70.8%	65.4%	94.6%
PROTEUS	71.7%	80.7%	88.3%
RouteLLM	61.4%	54.4%	94.3%
Gemma / MSE	63.4%	49.3%	81.3%
Gemma / CE	61.0%	63.8%	79.7%
Gemma / Anchored CE	61.6%	65.1%	73.6%
Qwen / CE	78.8%	74.6%	77.5%
Qwen / Anchored CE	**82.3%**	**78.9%**	**73.8%**

These percentages describe potential cache opportunities, not observed cache hits. Actual reuse also requires matching prompt prefixes and provider cache support. The evaluated outputs report no provider cache reads or writes.

A.8 COMPLETE ONLINE RESULTS

Table 10: **Complete benchmark-level online results.** Performance and cost are shown in separate columns as mean ± sample standard deviation over three training seeds. Cost is US dollars per task; performance means match Table 2, while costs and their uncertainty are reported here. Light shading marks cost columns and vertical labels identify method families. Bold and underline mark the best and second-best performance means across all methods, including fixed baselines; ties share the same mark. On τ-bench the simulated user is pinned to `haiku-4-5` for every row and its tokens are excluded from the router cost.

method	(a) In-domain						(b) OOD					
	InterCode-SQL		WebShop		SWE-bench		ALFWorld		InterCode CTF		τ-bench	
	success ↑	cost ↓	success ↑	cost ↓	success ↑	cost ↓	success ↑	cost ↓	success ↑	cost ↓	success ↑	cost ↓
Baselines												
Fixed RANDOM	0.255 ±.017	0.0090 ±.0011	0.396 ±.031	0.0116 ±.0010	0.650 ±.010	1.1612 ±.0637	0.767 ±.042	0.0314 ±.0034	0.203 ±.031	0.0356 ±.0062	0.600 ±.038	0.0737 ±.0035
Fixed ALL-STRONG	0.358 ±.009	0.0235 ±.0001	0.551 ±.046	0.0327 ±.0006	0.650 ±.053	2.4670 ±.0222	**0.893 ±.012**	0.0686 ±.0018	0.260 ±.010	0.1053 ±.0016	**0.716 ±.031**	0.2381 ±.0015
Fixed ALL-CHEAP	0.147 ±.015	0.0035 ±.0001	0.355 ±.043	0.0017 ±.0001	0.260 ±.000	0.0560 ±.0020	0.427 ±.012	0.0059 ±.0004	0.127 ±.015	0.0049 ±.0003	0.394 ±.033	0.0124 ±.0006
Prior ACRouter	0.387 ±.031	0.0233 ±.0011	0.487 ±.030	0.0255 ±.0010	0.523 ±.040	0.9730 ±.1164	0.733 ±.023	0.0276 ±.0007	0.240 ±.027	0.1043 ±.0011	0.620 ±.045	0.1584 ±.0046
Prior Router-R1	0.348 ±.017	0.0196 ±.0001	0.464 ±.006	0.0160 ±.0012	0.623 ±.029	1.3420 ±.2623	0.840 ±.035	0.0410 ±.0029	0.223 ±.006	0.0503 ±.0128	0.600 ±.031	0.1097 ±.0030
Prior PROTEUS	0.137 ±.023	0.0042 ±.0002	0.348 ±.037	0.0027 ±.0001	0.447 ±.015	0.1420 ±.0175	0.747 ±.031	0.0180 ±.0008	0.187 ±.015	0.0085 ±.0001	0.409 ±.045	0.0124 ±.0004
Prior RouteLLM	0.358 ±.008	0.0195 ±.0009	0.456 ±.026	0.0112 ±.0006	0.613 ±.038	2.5259 ±.0191	**0.893 ±.012**	0.0378 ±.0005	0.270 ±.020	0.1014 ±.0006	0.548 ±.023	0.0838 ±.0016
Our methods												
Gemma MSE	0.319 ±.045	0.0180 ±.0035	0.384 ±.049	0.0085 ±.0017	0.627 ±.050	1.5480 ±1.9116	0.840 ±.092	0.0651 ±.0076	0.227 ±.045	0.0463 ±.0162	0.594 ±.120	0.0938 ±.0737
Gemma CE	0.363 ±.042	0.0201 ±.0013	0.458 ±.069	0.0216 ±.0080	0.643 ±.029	1.8860 ±1.5599	0.887 ±.012	0.0700 ±.0006	0.240 ±.010	0.0939 ±.0117	0.672 ±.041	0.2065 ±.0202
Gemma Anchored CE	0.363 ±.017	0.0201 ±.0018	0.551 ±.018	0.0226 ±.0083	0.600 ±.000	0.7290 ±.2761	0.873 ±.031	0.0665 ±.0118	0.257 ±.015	0.0796 ±.0206	0.681 ±.013	0.2091 ±.0047
Qwen MSE	0.299 ±.047	0.0177 ±.0037	0.484 ±.021	0.0189 ±.0019	0.587 ±.137	1.7050 ±1.7886	0.787 ±.129	0.0448 ±.0139	0.230 ±.044	0.0324 ±.0268	0.678 ±.046	0.1954 ±.0358
Qwen CE	0.377 ±.009	0.0229 ±.0004	0.545 ±.015	0.0308 ±.0025	0.600 ±.020	2.7740 ±.7349	0.873 ±.012	0.0666 ±.0019	0.263 ±.006	0.0945 ±.0108	0.649 ±.013	0.2217 ±.0112
Qwen **Anchored CE**	**0.422 ±.008**	0.0212 ±.0016	**0.573 ±.036**	0.0299 ±.0018	**0.710 ±.036**	1.1940 ±.4965	**0.893 ±.012**	0.0631 ±.0033	**0.280 ±.026**	0.0697 ±.0120	0.681 ±.035	0.1703 ±.0333

Table 11: **In-domain and OOD aggregate online results.** The layout and best/second-best performance marks follow Table 10. Micro averages weight benchmark means by evaluation-set size ($68/40/100$ in-domain and $50/100/115$ OOD); macro averages weight benchmarks equally. For normalized weights w_b, uncertainty is propagated as $\sqrt{\sum_b w_b^2 \sigma_b^2}$.

method		(a) In-domain				(b) OOD			
		micro		macro		micro		macro	
		success ↑	cost ↓	success ↑	cost ↓	success ↑	cost ↓	success ↑	cost ↓
Baselines									
Fixed	RANDOM	0.472 ±.009	0.5634 ±.0306	0.434 ±.012	0.3939 ±.0212	0.482 ±.022	0.0513 ±.0029	0.523 ±.022	0.0469 ±.0026
Fixed	ALL-STRONG	0.536 ±.027	1.2000 ±.0107	0.520 ±.024	0.8411 ±.0074	**0.577 ±.014**	0.1560 ±.0010	**0.623 ±.012**	0.1373 ±.0009
Fixed	ALL-CHEAP	0.241 ±.010	0.0284 ±.0009	0.254 ±.015	0.0204 ±.0007	0.300 ±.016	0.0083 ±.0003	0.316 ±.013	0.0077 ±.0003
Prior	ACRouter	0.472 ±.022	0.4803 ±.0560	0.466 ±.020	0.3406 ±.0388	0.498 ±.022	0.1133 ±.0020	0.531 ±.019	0.0968 ±.0016
Prior	Router-R1	0.503 ±.015	0.6547 ±.1261	0.478 ±.011	0.4592 ±.0874	0.503 ±.015	0.0743 ±.0013	0.554 ±.016	0.0670 ±.0010
Prior	PROTEUS	0.327 ±.013	0.0702 ±.0084	0.311 ±.015	0.0496 ±.0058	0.389 ±.021	0.0120 ±.0002	0.448 ±.019	0.0130 ±.0001
Prior	RouteLLM	0.499 ±.019	1.2229 ±.0092	0.476 ±.016	0.8522 ±.0064	0.508 ±.013	0.0818 ±.0007	0.570 ±.011	0.0743 ±.0005
Our methods									
Gemma	MSE	0.480 ±.030	0.7518 ±.9190	0.443 ±.028	0.5248 ±.6372	0.502 ±.057	0.0705 ±.0320	0.554 ±.053	0.0684 ±.0246
Gemma	CE	0.516 ±.024	0.9175 ±.7500	0.488 ±.029	0.6426 ±.5200	0.550 ±.018	0.1383 ±.0088	0.600 ±.015	0.1235 ±.0067
Gemma	Anchored CE	0.513 ±.007	0.3614 ±.1328	0.505 ±.008	0.2572 ±.0921	0.557 ±.010	0.1333 ±.0020	0.604 ±.012	0.1184 ±.0016
Qwen	MSE	0.473 ±.068	0.8291 ±.8599	0.457 ±.049	0.5805 ±.5962	0.530 ±.036	0.1055 ±.0155	0.565 ±.048	0.0909 ±.0119
Qwen	CE	0.517 ±.010	1.3471 ±.3533	0.507 ±.009	0.9426 ±.2450	0.546 ±.006	0.1444 ±.0049	0.595 ±.006	0.1276 ±.0037
Qwen	**Anchored CE**	**0.590 ±.019**	0.5867 ±.2387	**0.568 ±.017**	0.4150 ±.1655	0.570 ±.018	0.1121 ±.0145	0.618 ±.015	0.1010 ±.0111

The headline total weights each benchmark mean by its evaluation-set size. Under this accounting, one complete 473-task evaluation costs an expected $\$151.75$, compared with $\$290.95$ for ALL-STRONG, a saving of $\$139.20$ or 47.8%. Alternative aggregations give the same direction but different magnitudes. The ID macro cost falls from $\$0.8411$ to $\$0.4150$ (50.7%), and the ID micro cost falls from $\$1.2000$ to $\$0.5867$ (51.1%). On the OOD benchmarks, the corresponding reductions are 26.4% macro and 28.1% micro. The task-weighted total is dominated by SWE-bench's larger per-task cost; the equal-benchmark mean of the six relative reductions is 23.4%.

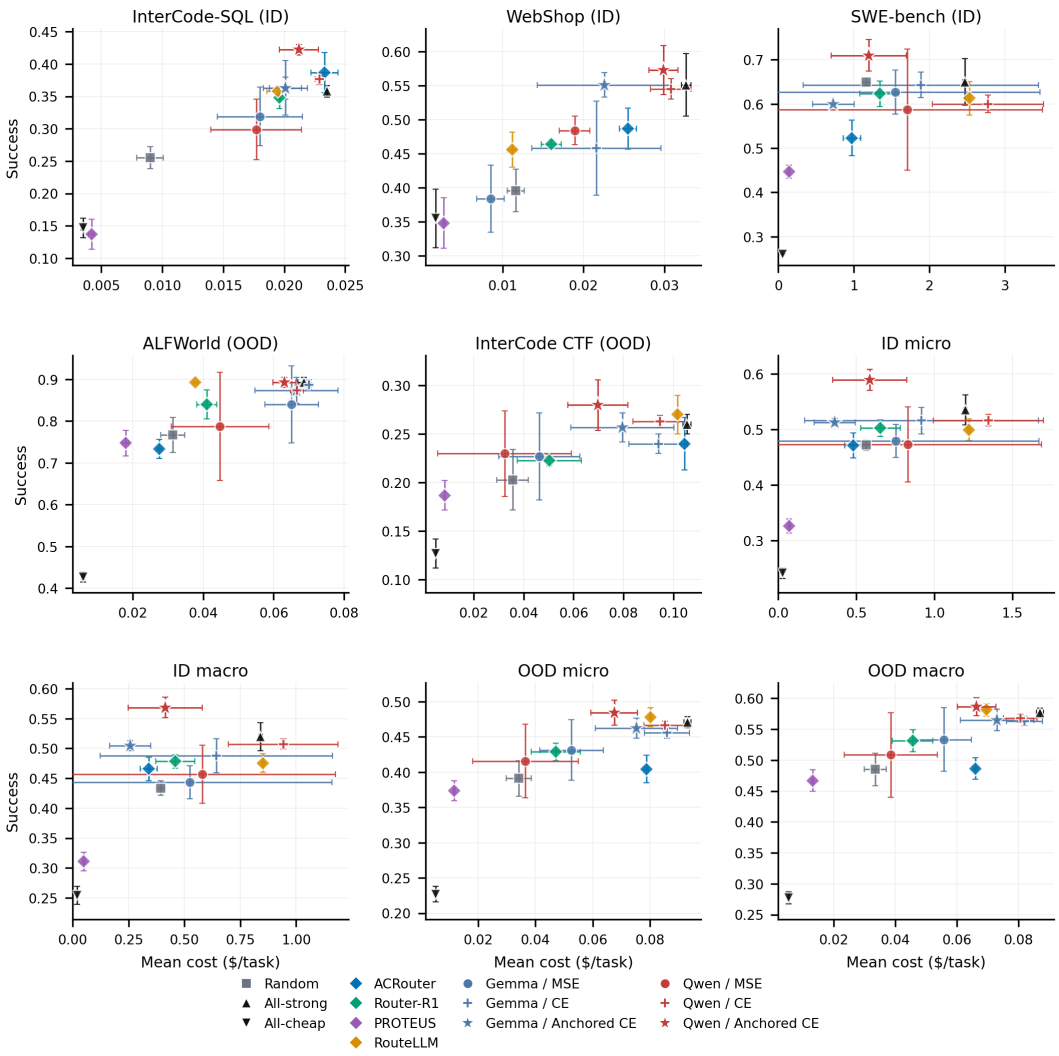

Figure 5: **Complete benchmark and aggregate quality–cost results.** The first five panels show every benchmark; the remaining panels show the ID/OOD micro and macro averages from Table 11. For our variants, color distinguishes the Gemma and Qwen encoders, while marker shape distinguishes MSE, CE, and Anchored CE. Horizontal and vertical bars show one cost and success standard deviation, respectively. All cost axes are linear.

A.9 Cost-Aware Tree-Oracle Upper Bound

We use the held-out replay trees to estimate the best cost–performance trade-off represented by the collected support. For a cost weight λ, backward induction computes

$$V_\lambda(s) = \max_{m \in \mathcal{A}(s)} \left[-\lambda c(s,m) + \begin{cases} R(s,m), & \text{if the recorded edge terminates,} \\ V_\lambda(s'_m), & \text{otherwise,} \end{cases} \right] \tag{10}$$

where $c(s,m)$ is the recorded API cost in dollars. We follow the maximizing edge from the root, record its terminal return and realized path cost, and average both quantities across held-out tasks. Sweeping $\lambda \in \{0, 1, 2, 5, 10, 25, 50, 100, 250, 500, 1000, 5000\}$ and retaining the non-dominated points gives the purple curves in Figure 6.

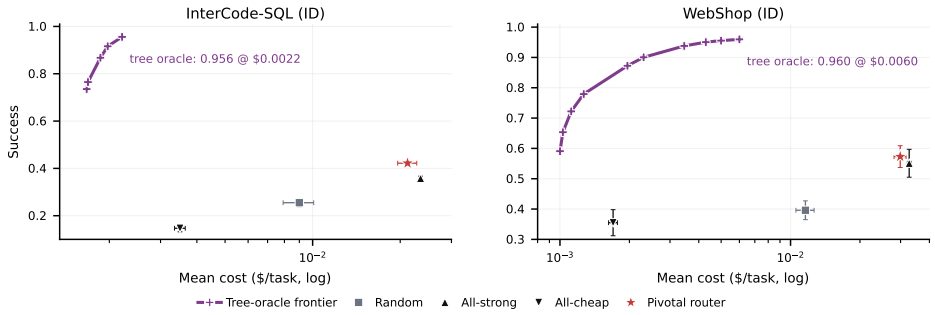

Figure 6: **Cost-aware tree-oracle frontiers on held-out evaluation tasks.** Purple curves show the non-dominated points from the exact replay sweep in Eq. 10; selected online references are shown for context, and the complete set of methods appears in Figure 5. SQL and WebShop collection and online evaluation both use a 10-step limit, so their recorded dollar costs are directly comparable. We omit SWE-bench because a comparable frontier would require collecting 50-step counterfactual trees, whose generation cost is prohibitive. Cost axes are logarithmic.

The highest-return non-dominated oracle points are 0.956 at $\$0.0022$ per SQL task and 0.960 at $\$0.0060$ per WebShop task. Both are well above the primary online router, which obtains 0.422 at $\$0.0212$ and 0.573 at $\$0.0299$, respectively. SWE-agent permits 50 calls in our online evaluation, while the available SWE-bench counterfactual trees stop at 10 steps. Extending that branching collection to 50 steps is too costly, and the existing 10-step replay costs cannot support a like-for-like online comparison; we therefore exclude SWE-bench from Figure 6.

This oracle has access to every recorded continuation and is not deployable. It upper-bounds policies restricted to the same tree support under the Lagrangian objective in Eq. 10; it is not an upper bound over unobserved environment continuations. Accordingly, the gap measures available headroom in model selection within collected support, not the performance promised by additional router training. Appendix B.7 further analyzes how the performance-only endpoint depends on exploration coverage.

B Data Generation and Pivotal-Step Analysis

B.1 Tree construction

Each node is an environment state indexed by its model path from the root, and each edge records one model executing one step in the benchmark environment. Collection backs up task-level terminal rewards. InterCode-SQL stops early only on a complete result-set match, which has reward one. If no exact match is found by the $T = 10$ depth cap, the collector records the environment's overlap score at that terminal leaf; rewards observed on earlier nonterminal queries are not backed up. Consequently, SQL tree labels are mostly binary but can be fractional at capped leaves. The phase-one gate still counts only full-success terminals. WebShop retains its official graded terminal reward, and SWE-bench uses the resolved outcome from the official test harness. Backward induction over the completed tree gives $Q^{\mathrm{perf}}(s, m)$ as defined in Eq. 1. The cost target instead applies the self-min backup in Eq. 5: the model's blended price ratio plus the cheapest expanded continuation. A state with at least two expanded models is a *decision state*; it is pivotal when $D(s) \geq \frac{1}{2}$ (Eq. 2). This max–min definition is used for the primary router.

The base collector runs MCTS with UCB1 (Auer et al., 2002) and $C = 1.4$. Each iteration follows the tree policy until reaching a terminal node or expanding an untried model, then rolls out to a terminal outcome. The resulting reward updates N and W only along the selected search-tree chain, including the newly expanded child. Rollout suffixes are retained as collected transitions, but their nodes do not receive N/W updates unless a later tree-policy traversal explicitly selects them. Collection trajectories are capped at $T = 10$ environment steps.

B.2 TREE POPULATIONS

We maintain three separate tree populations. The archived $\beta = 0$ training population uses standard UCB; its full population defines the gray endpoint in Fig. 2, while its phase-two subset provides the matched control in Table 13. Final $\beta = 2$ phase-two trees provide router training data and the descriptive analysis in Table 17. The third population is a fixed-budget plain-MCTS collection over 68, 40, and 100 held-out tasks from SQL, WebShop, and SWE-bench, respectively. It uses standard UCB1 for 3000 iterations per task with no probe, GVM allocation, or pivotal bonus. These trees are never used for training; they provide both the independent plain-MCTS audit below and the replay analyses.

B.3 TASK-LEVEL BUDGET ALLOCATION

For task i, let $\mathcal{T}_i^{\text{probe}}$ be the terminal nodes first created during its 150-iteration probe, and let $Y(t)$ indicate benchmark-level success (full reward on WebShop and binary success on SQL and SWE-bench). We estimate

$$p_i = \frac{\sum_{t \in \mathcal{T}_i^{\text{probe}}} \mathbb{1}[Y(t) = 1]}{|\mathcal{T}_i^{\text{probe}}|}. \tag{11}$$

Thus, p_i is the fraction of *created terminal nodes* that are successful. An iteration that revisits a cached terminal node still updates the search statistics, but it neither invokes a new model edge nor contributes another observation to Eq. 11; consequently, the denominator need not equal 150. We set $p_i = 0$ if the probe creates no terminal node.

Why 150 probe iterations? The design is calibrated by the Rule of Three. Under an idealized stationary model with independent Bernoulli rollout outcomes of success probability p, the probability of observing no success in n trials is $(1-p)^n$. The exact one-sided 95% upper bound after zero successes is therefore

$$p_{0.95} = 1 - 0.05^{1/n} \simeq \frac{-\log 0.05}{n} \simeq \frac{3}{n}. \tag{12}$$

For $n = 150$, this is $p_{0.95} = 0.0198$. Equivalently, to miss a task whose rollout success probability is at least $\delta = 0.02$ with probability at most 0.05 requires $n \geq \log(0.05)/\log(0.98) = 148.3$, so we round up to 150. Because each probe starts from a fresh tree, the gate $p_i = 0$ implies that none of its 150 MCTS iterations reached a successful terminal, even though Eq. 11 counts only newly created terminal nodes.

This calculation is a calibration, not an exact confidence guarantee for MCTS: UCB changes the rollout policy over time, cached paths induce dependence, and task success probabilities need not be stationary. We therefore also checked the gate empirically. An earlier, less conservative 40-iteration audit compared probe decisions with the completed collections on 199 tasks (Table 12). It assigned zero budget to seven rare-but-solvable tasks, losing about 13 positive terminals in total (less than 0.3% of all positives), and retained no task that remained all-failure in the completed collection. The mean absolute error $|p_{\text{probe}} - p_{\text{full}}|$ was 0.02–0.05. This audit predates the final 150-iteration run and does not include SWE-bench, so it is supporting evidence rather than a formal validation.

Table 12: **Empirical false-zero audit with the earlier 40-iteration probe.** Lost positives count successful terminal nodes in the corresponding completed collection.

surface	false zero	lost positives	false keep
InterCode-SQL	3/50	~ 3	0
InterCode CTF	2/50	~ 4	0
DBBench	1/49	~ 4	0
WebShop	1/50	~ 2	0

This is a GVM-inspired task allocation rather than a direct implementation of GVM-RAFT (Yao et al., 2025). GVM-RAFT estimates acceptance from a fixed number N' of independently sampled responses per prompt and allocates $n_i \propto G_i / \sqrt{p_i + \alpha_{\text{alloc}} / p_i^{\beta_{\text{alloc}}-1}}$, where G_i is an estimated

gradient-norm factor. Our terminal nodes are produced adaptively by MCTS, we omit G_i, and we use $\alpha_{\text{alloc}} = 0.01$ and $\beta_{\text{alloc}} = 2$. The latter is distinct from the within-task exploration coefficient β in Eq. 3. We therefore use the resulting curve as a hardness-aware collection heuristic and do not inherit the optimal-allocation claim of the independent-sampling analysis.

If $\mathcal{S} = \{i : p_i > 0\}$, the phase-two pool contains $3000|\mathcal{S}|$ iterations and is allocated according to

$$n_i \propto \frac{1}{\sqrt{p_i + 0.01/p_i}}, \qquad i \in \mathcal{S}. \tag{13}$$

Tasks with $p_i = 0$ receive no phase-two budget. The final $\beta = 2$ collection reuses the $\beta = 0$ probe estimates and integer task budgets rather than probing again. Phase two uses a separate random seed equal to the probe seed plus 777. Thus, the matched $\beta = 0$ and $\beta = 2$ comparison changes the within-task selection rule while holding task eligibility and allocation fixed. The two left panels of Figure 2 show the allocation curve and assigned phase-two budgets.

Overall gain and matched decomposition. Table 13 separates the end-to-end comparison in Fig. 2 from the matched phase-two-only effect of β. All entries use the same pivotal definition in Eq. 2. The first row is the plain-MCTS endpoint with the phase-one zero-success population retained. That population contains the real probe edges and synthetic zero-performance labels for unexpanded model actions at the same hopeless states. Applying the phase-one retention gate gives the second row; changing only the within-task selection rule from $\beta = 0$ to $\beta = 2$ gives the final row.

Table 13: **Pivotal-share decomposition (%).** The Figure 2 gain compares the first and last rows; the last two rows isolate the exploration-bonus effect on matched phase-two populations and task budgets.

collection endpoint	SQL	WebShop	SWE-bench
plain UCB1, phase one retained	23.8	10.5	16.4
after task gate: $\beta = 0$ phase two	34.5	20.4	23.7
after pivotal search: final $\beta = 2$	39.7	24.6	24.2

The total improvements are therefore 15.9, 14.1, and 7.8 percentage points. Of these, 5.2, 4.2, and 0.5 points come from the matched β change; the remainder comes from excluding the phase-one zero-success population: the task gate contributes 10.7, 9.9, and 7.3 points, and the pivotal search contributes 5.2, 4.2, and 0.5 points. These are the segments drawn inside the green bars. The first row is not a uniform-budget control because its phase-two iterations use the GVM-inspired task allocation; the last two rows isolate the β change with matched task eligibility and budgets.

Independent plain-MCTS run without phase one. We also archived a true no-probe baseline on the held-out split. Every task receives 3000 standard-UCB1 iterations with $C = 1.4$ and the same five-model panel; there is no phase-one probe, task filtering, GVM allocation, or pivotal-aware bonus. Recomputing max-backed values with the paper's $D(s) \geq \frac{1}{2}$ definition gives Table 14.

Table 14: **Measured fixed-budget plain MCTS without phase one.** The last column reports the same completed trees after excluding tasks with no full-success terminal anywhere in the tree. This is a retrospective audit of the gate mechanism, not an additional collection arm.

benchmark	tasks	decision states	pivotal states	raw share	success-filtered share
InterCode-SQL	68	38,334	10,711	27.9%	30.0%
WebShop	40	20,886	4,492	21.5%	23.8%
SWE-bench	100	58,850	8,787	14.9%	22.2%

Three SQL tasks, five WebShop tasks, and 27 SWE-bench tasks have no fully successful terminal in their completed fixed-budget tree. They contribute 2577, 3157, and 19,301 decision states, of which 0, 273, and 0 are pivotal. The three SQL tasks may retain depth-cap overlap scores, but none induces a backed gap of at least one half. The WebShop exception arises because its graded partial rewards

can differ by at least one half even when no branch reaches full reward. Removing these tasks raises the held-out pivotal share by 2.0, 2.3, and 7.3 points, confirming that the gate removes a lower-yield state population. This is a retrospective completed-tree audit, not the exact 150-iteration probe decision. We do not substitute these values into Fig. 2, because they use held-out tasks whereas the three plotted endpoints use the training population; differences in task mix would otherwise be mislabeled as component effects.

B.4 Training dataset

Only phase-two transitions are used to train the router; probe rollouts are excluded. The final $\beta = 2$ population contains 2,155,832 such transitions over 409 train tasks (Table 15). After tree backup, it yields 761,656 decision-state training rows from 204,816 decision states. Equation 2 retains 248,734 rows from 62,393 pivotal states.

Table 15: **Final $\beta = 2$ phase-two collection.** Counts are after merging all collection workers and before decision-state or pivotal filtering.

benchmark	tasks	transitions
InterCode-SQL	172	984,108
WebShop	87	413,997
SWE-bench Verified	150	757,727
total	409	2,155,832

Sensitivity to the pivotal definition. We re-filter the same backed decision states under nearby range thresholds and a standard-deviation criterion. Table 16 reports selected-state and state–model-row counts plus row-level Jaccard overlap with the paper's $D(s) \geq 0.5$ definition. Here $\sigma_Q(s)$ is the population standard deviation of the available $Q^{\text{perf}}(s, m)$ values; the 0.20 cutoff approximately matches the reference-set size.

Table 16: **Training-set overlap under alternative pivotal definitions.** Overlap columns are row-level Jaccard indices relative to $D(s) \geq 0.5$.

definition	selected states	selected rows	overall	SQL	WebShop	SWE-bench
$D(s) \geq 0.4$	62,642	249,705	99.61%	99.48%	99.34%	100.00%
$D(s) \geq 0.5$	62,393	248,734	100.00%	100.00%	100.00%	100.00%
$D(s) \geq 0.6$	61,522	245,288	98.61%	99.46%	93.99%	100.00%
$\sigma_Q(s) \geq 0.20$	62,484	248,978	99.81%	99.73%	99.72%	100.00%

Overall row-level Jaccard is at least 98.61% for nearby range thresholds and 99.81% for the approximately size-matched standard-deviation criterion. SWE-bench is unchanged; WebShop's lower 93.99% at $D(s) \geq 0.6$ reflects graded rewards. This audits data selection only; no router is retrained.

B.5 Detailed pivotal-step characterization

We characterize pivotal steps from two complementary views: how many models preserve a successful continuation, and which benchmark-native agent actions occur disproportionately at these states. For interpretability, both analyses use fully expanded states with at least one high-return and one low-return model ($\max_m Q^{\text{perf}}(s, m) > 0.9$ and $\min_m Q^{\text{perf}}(s, m) < 0.1$). This stricter subset is used only for analysis, not for defining pivotal labels or training the router.

Model-panel hardness. This is an operational property of the candidate-model panel, not a human difficulty label. We call a pivotal state *hard* when only one of the five models has high return, *polar* when two or three do, and a *trap* when four do. A trap can be handled by merely avoiding one failing model; hard and polar states instead require choosing among materially different model capabilities.

Table 17: **Hardness structure of pivotal steps.** Percentages use fully expanded high-versus-low states from the final training trees.

benchmark	hard (one wins)	polar (two–three win)	trap (four win)
InterCode-SQL	43%	39%	17%
WebShop	36%	42%	21%
SWE-bench	56%	33%	11%

Hard and polar states together account for 78–89% of the analyzed pivotal states. Thus, pivotality is not mainly an artifact of one anomalous model failing an otherwise routine step. It usually separates several plausible model choices, and on SWE-bench more than half of the states can be preserved by only one model.

Native-action enrichment. Hardness describes the model panel but not the agent behavior at the state. We therefore classify each recorded action using the benchmark's native action space and compute

$$\mathrm{enrich}(a) = \frac{\Pr(\text{action class} = a \mid \text{pivotal})}{\Pr(\text{action class} = a \mid \text{all collected steps})}. \tag{14}$$

Values above one identify action classes that are overrepresented at pivotal steps; values below one identify common but comparatively routine actions. Table 18 reports the complete benchmark-native action breakdown.

Table 18: **Benchmark-native action enrichment at pivotal steps.** Enrichment compares an action class's share among pivotal steps with its share among all collected steps. Shares are percentages. For InterCode-SQL, SHOW includes USE; *empty/answer* is the residual class with no executable SQL action.

benchmark	native action class	pivotal share	all-step share	enrichment
InterCode-SQL	SELECT (query)	54.6	37.0	1.48×
	DESC (schema)	28.7	32.2	0.89×
	SHOW/USE (list)	16.0	25.2	0.64×
	empty/answer	0.7	5.6	0.13×
WebShop	`click[< prev]`	19.8	11.7	1.70×
	`search`	15.1	8.9	1.69×
	`click[back to search]`	14.8	8.1	1.84×
	`click[buy now]`	14.4	35.9	0.40×
	`click[option/other]`	11.3	18.6	0.61×
	`click[item(asin)]`	7.1	6.0	1.19×
	`click[features]`	6.4	3.9	1.64×
	`click[next >]`	4.2	2.6	1.63×
	malformed/stuck	3.6	1.9	1.83×
	`click[description]`	2.9	2.3	1.24×
	`click[reviews]`	0.3	0.2	1.84×
SWE-bench	`apply_patch`	2.2	1.6	1.41×
	`run/modify-bash`	10.4	7.8	1.34×
	`submit`	7.9	6.8	1.17×
	`read/nav-bash`	66.0	67.5	0.98×
	`other-bash`	12.3	14.6	0.84×
	`run-test-bash`	1.1	1.8	0.65×

The pattern is consistent across environments. Actions that determine direction or produce the task's key content are enriched: constructing the SQL query, navigating or recovering in WebShop, and modifying code in SWE-bench. By contrast, schema inspection, the final purchase click, and test execution are underrepresented. Pivotal steps therefore concentrate on agent decisions that shape what remains attainable, rather than simply on the most frequent actions.

Action-conditional pivotal rates on held-out trees. Enrichment describes the composition of the pivotal set. To ask whether an interpretable action class is itself sufficient to identify pivotality, we additionally compute $\Pr(\text{pivotal} \mid \text{action class})$ on the held-out replay trees. Each generated edge is classified by its native action and inherits the pivotal label of its parent decision state, using the

primary $D(s) \geq \frac{1}{2}$ definition. We use edge-level rates because models at the same state can emit different action classes.

Table 19: **Core decision classes are more often pivotal, but do not determine pivotality.** Rates are measured on the $68/40/100$ held-out SQL, WebShop, and SWE-bench trees. The paired classes illustrate a higher-leverage decision and a more routine action in each native action space.

benchmark	core decision class	pivotal rate	routine class	pivotal rate
InterCode-SQL	SELECT query	29.9%	SHOW/USE	17.5%
WebShop	`back to search`	34.1%	`buy now`	10.3%
SWE-bench	`apply_patch`	43.6%	read/navigation	13.0%

The direction is consistent with the training-tree enrichment analysis, but the absolute rates show why semantic difficulty cannot replace the counterfactual definition. Query construction, search recovery, and patch generation are substantially more likely to be pivotal than their routine counterparts, yet a majority of edges in every listed core class are still non-pivotal.

Why a task-critical query may not be pivotal. To resolve this apparent mismatch, Table 20 partitions all held-out SQL query edges by the backed values at their parent state. Nearly all non-pivotal query edges fall into two consensus regimes: every expanded model branch retains high return, or every branch has low return. The former provides no need to distinguish models; the latter provides no successful model choice to exploit. Together these regimes account for 96.2% of non-pivotal query edges; the small remainder exhibits model differences that do not cross the pivotal threshold.

Table 20: **Decomposition of held-out SQL query edges.** The analysis includes 77,113 SELECT/WITH edges whose parent state expands at least two models. Each edge inherits its parent state's pivotal label. Values and labels are computed only over continuations represented in the collected tree.

state regime	condition	query edges	share
pivotal contrast	$D(s) \geq \frac{1}{2}$	23,068	29.9%
consensus high return	$\min_m Q^{\mathrm{perf}}(s,m) > 0.9$	22,802	29.6%
consensus low return	$\max_m Q^{\mathrm{perf}}(s,m) < 0.1$	29,174	37.8%
other subthreshold contrast	remaining $D(s) < \frac{1}{2}$	2,069	2.7%

B.5.1 COMPLETE HELD-OUT CASE STUDIES

We next instantiate the taxonomy with complete state records from the held-out trees. Each record fixes the task, state path, common history, all five counterfactual model actions, their immediate environment observations, and the backed value of each branch. The collectors store the exact parsed environment action and the first 500 characters of the free-form model message and observation. We therefore reproduce the exact executed action, the outcome-relevant part of the recorded observation, and terminal evaluation statistics. For a nonterminal branch whose backed value is central to the example, we also give an explicit successful continuation.

We abbreviate Sonnet, Haiku, DeepSeek, Qwen3-235B, and GPT-OSS-120B as S/H/D/Q/G. Vectors below follow this order.

Table 21: **Overview of the complete held-out cases.** A one indicates that the branch has a successful continuation in the completed tree.

task/state	regime	$Q^{\text{perf}}_{\text{S/H/D/Q/G}}$	routing distinction
SQL 126	consensus success	$(1, 1, 1, 1, 1)$	all five branches preserve the answer
SQL 102-A	recoverable error	$(1, 1, 1, 1, 1)$	even malformed or empty-result actions recover
SQL 102-B	hard pivotal	$(1, 0, 0, 0, 0)$	only the immediately correct query survives
SQL 111	consensus failure	$(0, 0, 0, 0, 0)$	one plausible result, but no successful completion
WebShop 157	trap	$(0, 1, 1, 1, 1)$	one model over-verifies instead of buying
SWE `pytest-10081`	hard pivotal	$(0, 0, 0, 0, 1)$	one branch turns the shared diagnosis into a passing patch

Case 1: consensus success on InterCode-SQL 126. **Task:** "What is the molecule of atom ID `TR000_2` and atom ID `TR000_4`?" The state path is `(haiku, sonnet, haiku)`. Its complete executed history is: (1) H: `SHOW TABLES`, returning {`atom`, `bond`, `connected`, `molecule`}; (2) S: `DESC atom`, returning {`id_fake`, `atom_id`, `molecule_id`, `element`}; and (3) H: `DESC molecule`, returning {`id_fake`, `molecule_id`, `label`}.

Table 22: **All five counterfactual actions at SQL 126.** Three branches answer immediately; the other two expose enough information for their next action to return `TR000`.

model	executed action	immediate observation	Q^{perf}
S	`SELECT T1.molecule_id FROM atom AS T1 WHERE T1.atom_id='TR000_2' INTERSECT SELECT T2.molecule_id FROM atom AS T2 WHERE T2.atom_id='TR000_4'`	`TR000`; terminal success	1
H	`SELECT DISTINCT a.molecule_id FROM atom a WHERE a.atom_id IN ('TR000_2','TR000_4')`	`TR000`; terminal success	1
D	`SELECT DISTINCT a1.molecule_id FROM atom a1 JOIN atom a2 ON a1.molecule_id=a2.molecule_id WHERE a1.atom_id='TR000_2' AND a2.atom_id='TR000_4'`	`TR000`; terminal success	1
Q	`SELECT m.label FROM molecule m JOIN atom a1 ON m.molecule_id=a1.molecule_id JOIN atom a2 ON m.molecule_id=a2.molecule_id WHERE a1.atom_id='TR000_2' AND a2.atom_id='TR000_4'`	+; then H queries the ID and succeeds	1
G	`SELECT a.atom_id,a.molecule_id,m.label FROM atom a LEFT JOIN molecule m ON a.molecule_id=m.molecule_id WHERE a.atom_id IN ('TR000_2','TR000_4')`	`(TR000_2,TR000,+)`, `(TR000_4,TR000,+)`; then succeeds	1

The surface forms differ, and Q initially selects the wrong output column, but every branch preserves a short route to the answer. This is a task-critical query with zero model-choice leverage.

Case 2: a paired counterexample on InterCode-SQL 102. **Task:** "Among the atoms that contain carbon, which ones do not belong to a carcinogenic compound?" States A and B have seen exactly the same database evidence: `SHOW TABLES`, `DESC atom`, and `DESC molecule`, with the observations listed in Case 1. Their model paths differ: A is `(gptoss120b, sonnet, deepseek)`, whereas B is `(deepseek, deepseek, haiku)`.

Table 23: **The same immediate mistake can be recoverable in one subtree and fatal in another.** Every row is an actually expanded edge.

state	model	executed action	immediate observation	Q^{perf}
A	S	`SELECT DISTINCT a.atom_id FROM atom a JOIN molecule m ON a.molecule_id=m.molecule_id WHERE a.element='c' AND m.label!='+'`	benchmark answer; terminal success	1
A	H	`DESC bond`	bond schema	1
A	D	`DESC bond`	bond schema	1
A	Q	`SELECT a.atom_id,a.molecule_id FROM atom a JOIN molecule m ON a.molecule_id=m.molecule_id WHERE a.element='carbon' AND m.label!='carcinogenic'`	empty result	1
A	G	`DESC bondDESC bond`	`bonddesc` table error	1
B	S	`SELECT DISTINCT a.atom_id FROM atom a JOIN molecule m ON a.molecule_id=m.molecule_id WHERE a.element='c' AND m.label!='+'`	benchmark answer; terminal success	1
B	H	`DESC bond`	bond schema	0
B	D	`SELECT DISTINCT label FROM molecule`	+, −	0
B	Q	`SELECT a.atom_id,a.molecule_id FROM atom a JOIN molecule m ON a.molecule_id=m.molecule_id WHERE a.element='carbon' AND m.label!='carcinogenic'`	empty result	0
B	G	`SELECT * FROM atom LIMIT 5`, reasoning text, then concatenated `SELECT *` queries over `molecule`, `bond`, and `connected`	SQL syntax error	0

For completeness, the successful witness below is the shortest recorded completion after Q's empty result in state A.

Table 24: **Successful witness continuation for the apparently wrong SQL 102-A branch.**

step	model	executed action	observation
1	Q	query literal `'carbon'` and `'carcinogenic'`	empty result
2	G	`SELECT DISTINCT label FROM molecule`	+, −
3	Q	inspect the current database's information-schema tables	four task tables
4	D	`DESC atom`	atom schema
5	D	`DESC molecule`	molecule schema
6	S	query `element='c'` and `label!='+'`	terminal success

The pair separates action correctness from downstream value. In A, even an empty result or a malformed query leaves a discovered successful continuation. In B, only Sonnet's direct answer does so. Pivotality therefore supervises the model choice that changes the reachable outcome, not a local classifier of whether the next action looks correct.

Case 3: consensus failure on InterCode-SQL 111. **Task:** "Which molecules have triple bonds, and what elements do they contain?" The path `(qwen235b, haiku, qwen235b, sonnet)` has executed `SHOW TABLES`, `DESC molecule`, `DESC bond`, and `DESC atom`. Thus every model sees the relevant schemas, but not a sample of the encoded bond values.

Table 25: **All five counterfactual actions at SQL 111.** Sonnet decodes the bond symbol and returns plausible rows, but no branch completes the benchmark answer.

model	executed action	immediate observation	Q^{perf}
S	`SELECT DISTINCT b.molecule_id,a.element FROM bond b JOIN atom a ON b.molecule_id=a.molecule_id WHERE b.bond_type='#' ORDER BY b.molecule_id,a.element`	13 molecule–element rows, including `TR041`, `TR377`, and `TR447`	0
H	`SELECT DISTINCT b.molecule_id,GROUP_CONCAT(DISTINCT a.element ORDER BY a.element) FROM bond b JOIN atom a ON b.molecule_id=a.molecule_id WHERE b.bond_type='triple' GROUP BY b.molecule_id`	empty result	0
D	`SELECT DISTINCT molecule_id FROM bond WHERE bond_type='triple'`	empty result	0
Q	`SELECT m.molecule_id,GROUP_CONCAT(a.element) FROM molecule m JOIN bond b ON m.molecule_id=b.molecule_id JOIN atom a ON m.molecule_id=a.molecule_id WHERE b.bond_type='triple' GROUP BY m.molecule_id`	empty result	0
G	`SELECT DISTINCT m.label,a.element FROM bond b JOIN molecule m ON m.molecule_id=b.molecule_id JOIN atom a ON a.molecule_id=b.molecule_id WHERE b.bond_type='triple'`	`No database selected`	0

This is not a five-model copy of the same lexical error: Sonnet already recovers the # encoding and makes substantive progress. The state remains non-pivotal because none of the five best recorded continuations reaches the execution-match answer. There is no better model edge for a router to exploit at this state.

Case 4: an over-verification trap on WebShop 157. Task: buy women's straight-leg jeans in large and `black06` for less than $120. The path `(sonnet, sonnet, haiku, sonnet)` executes: (1) `search[ladies large size black jeans straight leg fit]`; (2) `click[b09b9yjx14]`, opening "539 Womens ... Straight Leg Denim Pants" at $35.99; (3) `click[black]`; and (4) `click[large]`. The current page confirms the product, price, and selected options.

Table 26: **All five counterfactual actions at WebShop 157.** The exact parsed action disambiguates the losing model: Sonnet, not DeepSeek, is the sole failing branch at this state.

model	exact executed action	rationale and immediate observation	Q^{perf}
S	`click[features]`	worries that the listed `black` option may not be the requested string `black06`; opens product features, and no recorded continuation purchases successfully	0
H	`click[buy now]`	purchase records ASIN `B09B9YJX14`, color `black`, size `large`; reward 1	1
D	`click[buy now]`	same terminal purchase and reward 1	1
Q	`click[buy now]`	same terminal purchase and reward 1	1
G	`click[buy now]`	same terminal purchase and reward 1	1

The model that performs the most additional checking is the only loser. Pivotality here captures stopping and execution timing: the relevant capability is recognizing that the environment's `black` option already instantiates the natural-language color request.

Case 5: shared diagnosis, different execution on SWE-bench. Task: fix `pytest-dev__pytest-10081`, where `unittest.TestCase.tearDown` runs under `--pdb` even when the test class has `@unittest.skip`. At path `(sonnet, sonnet)`, Sonnet first greps `src/_pytest/unittest.py`, then prints lines 305–335. The observation exposes the faulty condition `usepdb and not _is_skipped(self.obj)` and the delayed-`tearDown` logic.

Table 27: **All five counterfactual actions at SWE-bench `pytest-10081`.** F2P and P2P denote fail-to-pass and pass-to-pass tests.

model	executed operation	immediate observation / branch outcome	Q^{perf}
S	`submit`	claims the class-aware fix is present, but submits without an effective edit; F2P $0/1$, P2P $63/63$, unresolved	0
H	`sed -n '315,325p' ...`	continues inspection; no successful continuation in this subtree	0
D	`submit`	immediate unchanged submission; F2P $0/1$, P2P $63/63$, unresolved	0
Q	`submit`	describes a class-aware `sed` edit, but the parsed action is submission; F2P $0/1$, unresolved	0
G	class-aware `apply_patch` command	shell reports `apply_patch: command not found`; the branch then recovers and resolves	1

The complete successful witness for G's nonterminal edge is: (1) G attempts to add `and not _is_skipped(self._testcase.__class__)`, but the helper command is absent; (2) H applies the same replacement with a Python file-edit script, which reports `File updated successfully`; and (3) D submits, yielding F2P $1/1$, P2P $63/63$, and resolved reward 1. All five branches have enough semantic context to state the diagnosis. The pivotal distinction is whether that diagnosis becomes an effective repository edit.

B.6 Where the learned router differs from the tree oracle

The all-step oracle in Table 1 supplies a ceiling, but the errors below show what the learned policy has not yet captured. We enumerate strict pivotal states with all five model branches expanded, stratify them by benchmark and oracle-winning model, and use the seed-1 Qwen/Anchored-CE checkpoint at $\lambda = 0$ to identify high-regret states. We then rescore the selected cases with all three final checkpoints. Table 28 gives one behaviorally interpretable state per source benchmark. The predicted-score vector is from seed 1; the last column reports the actual greedy choice for all three independently trained seeds.

Table 28: **Three learned-router gaps to the tree oracle.** Vectors use S/H/D/Q/G order. A bold choice is oracle-optimal.

state	tree Q^{perf}	seed-1 $\hat{Q}^{\text{perf}}$	choices (seeds 1/2/3)	missed distinction
SQL 27	$(0, 1, 1, 0, 1)$	$(.795, .559, .473, .562, .541)$	S/S/S	signed difference versus absolute difference
WebShop 157	$(0, 1, 1, 1, 1)$	$(.651, .623, .563, .537, .541)$	S/**Q**/S	purchase now versus unnecessary verification
SWE `pytest-10081`	$(0, 0, 0, 0, 1)$	$(.500, .574, .671, .544, .512)$	D/D/S	stating the diagnosis versus applying a working edit

SQL gap: a one-token semantic distinction. For SQL task 27, the question asks for "the difference between the number of carcinogenic and non-carcinogenic molecules." At path `(sonnet, gptoss120b, sonnet, haiku)`, the history has exposed the `molecule(label)` schema and sample labels +/−.

Table 29: **Complete SQL 27 oracle-gap state.** The database contains 152 positive and 191 negative labels, so the ordered difference is -39.

model	executed action	immediate observation	Q^{perf}
S	`SELECT ABS(SUM(CASE WHEN label='+' THEN 1 ELSE -1 END)) AS difference FROM molecule`	39; no successful continuation	0
H	`SELECT SUM(CASE WHEN label='+' THEN 1 ELSE 0 END) - SUM(CASE WHEN label='-' THEN 1 ELSE 0 END) AS difference FROM molecule`	-39; next query terminates successfully	1
D	`SELECT SUM(CASE WHEN label='+' THEN 1 ELSE 0 END) AS positive, SUM(CASE WHEN label='-' THEN 1 ELSE 0 END) AS negative, ABS(SUM(CASE WHEN label='+' THEN 1 ELSE 0 END) - SUM(CASE WHEN label='-' THEN 1 ELSE 0 END)) AS difference FROM molecule`	152, 191, 39; next query corrects to -39	1
Q	`SELECT SUM(CASE WHEN label='+' THEN 1 ELSE 0 END) AS positive, SUM(CASE WHEN label='-' THEN 1 ELSE 0 END) AS negative FROM molecule`	152, 191; no successful continuation	0
G	`SELECT (SELECT COUNT(*) FROM molecule WHERE label='+') - (SELECT COUNT(*) FROM molecule WHERE label='-') AS difference`	-39; terminal success	1

All three checkpoints rank Sonnet first, even though its `ABS` removes the sign requested by the ordered wording. This is not a failure to recognize SQL difficulty; it is under-sensitivity to a small semantic operator that reverses the backed preference. The large seed-1 margin (0.795 versus 0.562 for the highest-scored winning branch) also shows a strong model prior overriding the state-specific contrast.

WebShop gap: over-verification. The complete state is Table 26. Seeds 1 and 3 select Sonnet's `click[features]` and lose one full reward; seed 2 ranks Qwen first and buys successfully. Unlike SQL 27, the top two seed-1 scores are close (0.651 versus 0.623). This gap is therefore both a stopping-policy error and a calibration-sensitive decision: small training variation is enough to switch from the sole losing action to a winning one.

SWE-bench gap: implementation state, not diagnosis. The complete state and successful witness are in Table 27. Every checkpoint misses the unique winning model branch. Seeds 1 and 2 choose DeepSeek's immediate unchanged submission; seed 3 chooses Sonnet, which likewise submits without a passing edit. GPT-OSS itself first invokes an unavailable patch helper, but its subtree is the only one that persists through an alternative file edit and reaches F2P $1/1$. The router must therefore predict execution reliability after a shared diagnosis, a harder distinction than matching issue text to a model's general coding capability.

What the three gaps reveal. The failures are not one phenomenon. SQL 27 is a confident fine-grained semantic misranking; WebShop 157 is a low-margin stopping decision; and SWE `pytest-10081` separates verbal diagnosis from effective environment mutation. In contrast, all three checkpoints choose the oracle-optimal Sonnet branch in both SQL 102 states, while model choice is immaterial in the all-win SQL 126 and all-fail SQL 111 states. The remaining oracle gap is thus concentrated in state-conditioned model contrasts of exactly the kind pivotal supervision targets, but the examples identify where more targeted hard negatives are still needed: signed aggregations, terminal-action timing, and edit-and-test execution.

B.7 Tree replay

Table 1 uses the held-out population of 68 SQL, 40 WebShop, and 100 SWE-bench trees. Each task receives 3000 MCTS iterations with UCB1 ($C = 1.4$), random continuation rollouts, and depth cap 10, without task-level reallocation or a pivotal bonus. Replay begins at the root, follows the edge selected by the replay policy, and reads the recorded terminal reward. The random policy samples uniformly from expanded edges using a state-path-keyed pseudo-random choice; Table 1 therefore reports one fixed point estimate and requires no new model calls.

Replay is restricted to expanded edges and thus measures value concentration under collected support. Moreover, the oracle and pivotal labels are computed from the same completed tree. The

analysis should therefore be interpreted as a within-tree decomposition, not an unbiased estimate of an unconstrained policy.

Binary-reward replay property. On a finite tree with terminal rewards in $\{0, 1\}$, Eq. 1 gives backed values in $\{0, 1\}$. Under Eq. 2, every expanded branch at a non-pivotal state has the same backed value, so any choice preserves the state's maximum value. At a pivotal state, the replay policy selects a maximizing branch and also preserves that value. Applying this argument along the finite path shows that pivotal-only replay reaches the all-step oracle return regardless of its random choices at non-pivotal states. Thus the equality in the binary case is a structural property, not independent evidence of learned-router generalization. The argument does not apply directly when capped leaves or terminal rewards are fractional.

Oracle coverage sensitivity. The all-oracle row in Table 1 is an upper bound only within the continuations exposed by collection. We quantify its dependence on tree coverage by retaining the first B recorded edges of every held-out tree, in collection order, and recomputing the backward-induction oracle. Figure 7 shows a substantial increase with coverage. At $B = 100$, the oracle reaches 0.662, 0.800, and 0.380 on SQL, WebShop, and SWE-bench; the complete trees raise these values to 0.956, 0.960, and 0.730, respectively.

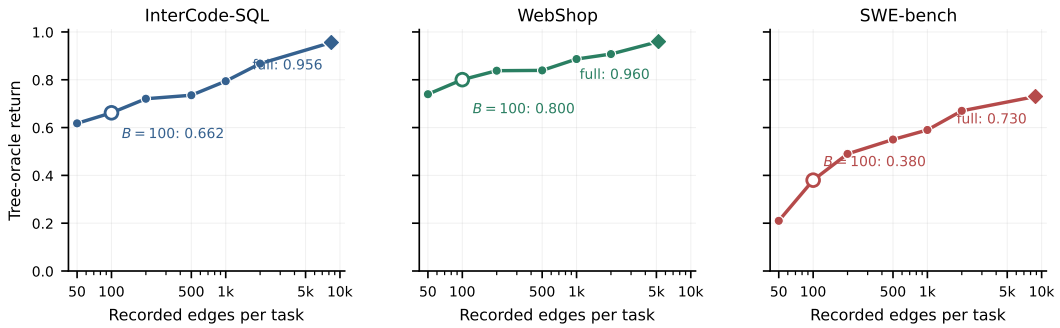

Figure 7: **The tree-oracle ceiling depends on exploration coverage.** Prefix points retain the first B recorded edges per held-out task and recompute the oracle from the resulting partial tree. Diamonds use each complete tree; because tree sizes vary, their horizontal coordinates are the mean full-tree edge counts for the corresponding benchmark. InterCode-SQL uses binary exact execution match, WebShop uses its official graded terminal reward, and SWE-bench uses the resolved outcome.

This dependence limits how the absolute oracle value should be read: it measures the best return represented by a particular search budget, rather than the value of an unconstrained online policy. It does not affect the pivotal-versus-non-pivotal comparison in Table 1, whose policies are evaluated on exactly the same completed trees.

C Design of the Pivotal-Aware Exploration Bonus

Equation 3 uses the online statistic U, which is distinct from the completed-tree label D. The design has two axes: whether disagreement is measured at the parent, immediate child, or child subtree; and whether the bonus is added directly to value or placed inside the UCB exploration coefficient. We tested the combinations in Table 30 in sequence. Their scopes differ: the small sweeps are one-seed mechanism studies, whereas only the selected rule is used in the matched full collection. The comparison identifies two useful properties: the statistic should distinguish children and its influence should decay as a branch is sampled.

The two β ranges are not directly comparable. In the additive variants, βU is measured against value gaps of roughly 0.1–0.3; in the coefficient variants it is measured against $C = 1.4$.

Table 30: **All data-generation variants and their experimental scope.** Pilot sweeps use one collection seed and diagnose search behavior, not downstream router variance. DB denotes the DBBench pilot, which is not part of final router training.

statistic	placement	sweep and scope	principal observation
(i) parent $U(s)$	additive	algebraic control	cancels inside the within-state $\arg\max$
(ii) parent $U(s)$	UCB coeff.	$\beta = \{0, 2, 4, 6, 8\}$; SQL/DB/SWE, 5 tasks each, 200 iterations	SQL pivotal share $48.6\% \rightarrow 24.4\%$ and edges $5407 \rightarrow 2058$; no gain on DB/SWE
(iii) immediate-child $U(s, m)$	additive	$\beta = \{0, .25, .5, 1, 2\}$; SQL, 5 tasks, 200 iterations	edges remain 5280–5641, but pivotal coverage is non-monotonic
(iv) subtree U_{sub}	additive	$\beta = \{0, .25, .5, 1, 2, 4\}$ on the SQL pilot; then $\beta = 1$ on 10 tasks/domain	SQL pilot rises $36.0\% \rightarrow 60.5\%$, but larger budgets reveal benchmark-dependent over-concentration
(v) subtree U_{sub}	UCB coeff.	shared $\beta = 2$; matched full SQL/WebShop/SWE collection	**selected**: child-specific signal with visit-dependent decay

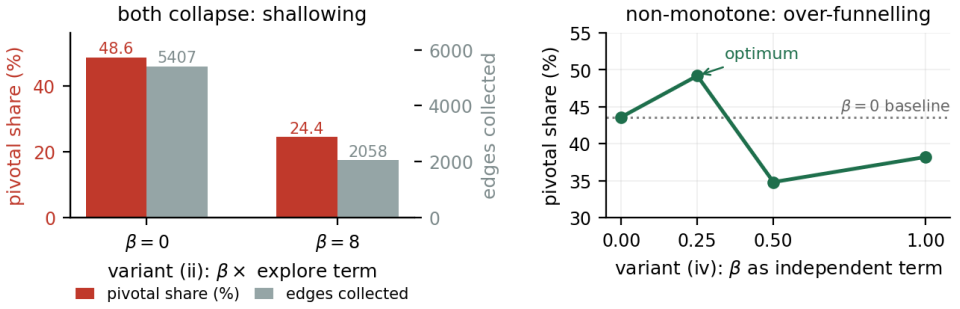

Figure 8: **Behavior of two alternative bonus placements in SQL pilot collections.** *Left:* parent variance in the exploration coefficient reduces tree depth at large β. *Right:* under the larger task-level budget, an additive subtree bonus concentrates rollouts and produces a non-monotonic pivotal share. These pilots use different collection budgets from the full-scale phase-two comparison in Table 13.

C.1 Parent and immediate-child statistics

An additive parent statistic,

$$\text{score}(m) = \hat{Q}(s, m) + \beta\, U(s) + C\sqrt{\ln N(s)/N(s, m)} \tag{15}$$

cannot alter model selection: $U(s) = \text{Var}_{m'}\, \hat{Q}(s, m')$ is constant across all children in the same $\arg\max$. Placing the same statistic in the exploration coefficient does change selection:

$$\text{score}(m) = \hat{Q}(s, m) + \big(C + \beta\, U(s)\big)\sqrt{\ln N(s)/N(s, m)}, \qquad \beta \in \{0, 2, 4, 6, 8\} \tag{16}$$

because the visit term varies across children. However, increasing exploration at every high-variance parent spends more calls balancing its immediate edges and leaves fewer calls for deeper states. In the SQL pilot, increasing β from 0 to 8 reduces the pivotal share from 48.6% to 24.4% and the number of collected edges from 5407 to 2058 (Fig. 8, left); DBBench and SWE-bench show no consistent gain. Error against a fixed label reference remains nearly constant at 0.02–0.03 MAE, indicating that the main effect is tree shape rather than improved label estimation.

A child-specific statistic avoids applying the same exploration pressure to every outgoing edge:

$$\text{score}(m) = \hat{Q}(s, m) + \beta\, U(s, m) + C\sqrt{\ln N(s)/N(s, m)}, \qquad \beta \in \{0, 0.25, 0.5, 1, 2\} \tag{17}$$

where $U(s, m)$ is the variance at the immediate child reached by m. Edge counts remain stable, between 5280 and 5641, but pivotal coverage changes non-monotonically. Immediate-child variance is too local to identify branches that lead to disagreement several steps later.

C.2 SUBTREE STATISTICS

The subtree statistic propagates the largest observed disagreement below each child:

$$\text{score}(m) = \hat{Q}(s,m) + \beta\, U_{\text{sub}}(s,m) + C\sqrt{\ln N(s)/N(s,m)}, \tag{18}$$

$$U_{\text{sub}}(s,m) = \max_{u \in \text{subtree}(s')} U(u) \tag{19}$$

where s' is the child reached by m. The statistic is maintained during the existing backup pass and requires no additional model calls. In the five-task, 200-iteration SQL pilot, increasing the additive coefficient from $\beta = 0$ to 4 raises pivotal share monotonically from 36.0% to 60.5%. The same mechanism is not uniformly beneficial at larger budgets. With 10 tasks per benchmark and $\beta = 1$, pivotal share changes by $+7.1$ percentage points on DBBench, $+8.5$ on SWE-bench, $+0.8$ on WebShop, and -5.8 on SQL.

In every domain, the additive subtree bonus increases the fraction of rollouts entering the highest-bonus branch. On the narrow, deep SQL trees, this can repeatedly extend an already solvable subtree and convert potential comparisons into all-success states. A SQL follow-up finds that $\beta = 0.25$ raises pivotal share from 43.6% to 49.2%, whereas $\beta \in \{0.5, 1\}$ falls below the matched baseline (Fig. 8, right).

Tree shape does not provide a reliable rule for choosing an additive coefficient: SWE-bench has the lowest average branching factor (1.37) yet benefits most from $\beta = 1$, while SQL has a higher factor (1.82) and declines. A benchmark-specific coefficient would also require target collection for tuning. We therefore use a shared rule whose bonus decays with branch visits:

$$\text{score}(m) = \hat{Q}(s,m) + \big(C + \beta\, U_{\text{sub}}(s,m)\big)\sqrt{\ln N(s)/N(s,m)}, \qquad C = 1.4,\ \beta = 2 \tag{3 revisited}$$

Here U_{sub} remains child-specific, while its contribution decreases as $N(s,m)^{-1/2}$. This combines directional information about downstream disagreement with UCB's visit-dependent decay. We use $\beta = 2$ for the final phase-two endpoint in Fig. 2. The final placement was not swept at full collection scale, so the matched last-two-row comparison in Table 13 establishes an improvement over $\beta = 0$, not optimality of the selected coefficient.

All variants above choose which model to expand after reaching a state. They do not directly allocate rollouts among previously discovered states. A state-level scheduler could instead sample expanded nodes according to disagreement before reconstructing and backing up their ancestor paths, but its depth–breadth trade-off lies outside the present collection design.

D OBJECTIVE AND DATA-SELECTION ANALYSIS

D.1 WHY USE A WITHIN-STATE RANKING OBJECTIVE?

At a pivotal state, the learning problem is comparative: the router must identify which candidate model preserves the best continuation from the *same* agent history. Pointwise MSE instead treats every state–model value as an independent absolute target. When histories are difficult to separate in the representation, minimizing pooled squared error can regress toward a model-specific conditional mean. Such a predictor may fit average return while erasing the within-state differences that determine the routing action. In-state CE directly contrasts the candidate scores in each state, so every gradient is aligned with the ordering used by the router.

The online evidence spans all six benchmarks. Table 31 subtracts matched rows of Table 2. CE improves the mean over MSE in eleven of the twelve benchmark–encoder comparisons, including all three source domains and all three OOD domains for the Gemma encoder. The single exception is Qwen on τ-bench, where value regression is 0.029 higher. This $11/12$ ordering across encoders and domains is the primary evidence for the ranking objective.

Table 31: **Objective contrasts across benchmarks.** Each entry is the change in mean online performance at $\lambda = 0$ between matched rows of Table 2; positive values favor the first objective in the contrast.

encoder	objective contrast	SQL	WebShop	SWE-bench	ALFWorld	CTF	τ-bench
Gemma	CE − MSE	+.044	+.074	+.016	+.047	+.013	+.078
	Anchored CE − CE	.000	+.093	−.043	−.014	+.017	+.009
Qwen	CE − MSE	+.078	+.061	+.013	+.086	+.033	−.029
	Anchored CE − CE	+.045	+.028	+.110	+.020	+.017	+.032

The score-level behavior is consistent with this explanation. Figure 9 repeats the diagnostic on held-out pivotal states from all three source domains. Pointwise MSE produces a narrow marginal score distribution in every domain (standard deviation 0.057–0.081), whereas QDist spreads the scores substantially (0.145–0.187). Adding the MSE anchor moderates this spread (0.086–0.114) while retaining more variation than MSE alone. This probe illustrates the mechanism behind the online result: MSE suppresses model contrast, while CE restores the score variation required for state-dependent model choice.

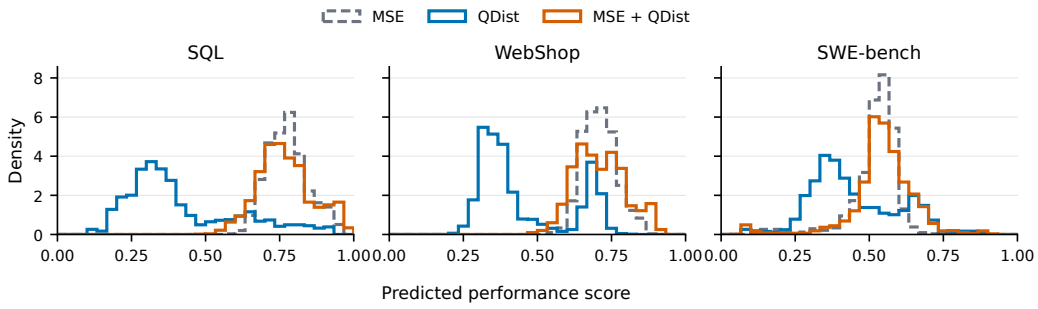

Figure 9: **Within-state supervision prevents narrow score collapse across source domains.** Each panel uses the same deterministic sample of 500 oracle-pivotal states from the held-out tree set and includes every model choice expanded at those states. Curves come from matched-seed Qwen checkpoints. QDist is pure within-state CE, while MSE+QDist is Anchored CE. Pure QDist produces the broadest predicted-score distribution; the MSE anchor restores an absolute value scale without returning to the narrow MSE distribution.

D.2 Effect of the performance anchor

Pure CE determines a distribution over models, not performance scores in backed-return units. Its common score offset is irrelevant to the argmax, but the numerical size of a performance gap relative to a predicted cost gap determines when λ changes the decision in Eq. 9. The MSE term in Eq. 4 ties the predicted performance scale to the backed values, while CE continues to enforce the within-state ordering. The two terms therefore serve different purposes: CE learns *which* model is better, and the anchor calibrates *how much* better it is for cost-aware routing.

At $\lambda = 0$, Anchored CE improves all six Qwen evaluations; with Gemma it improves WebShop, CTF, and τ-bench, ties SQL, and declines on SWE-bench and ALFWorld (Table 31). These comparisons evaluate the performance contribution of the anchor while keeping the cost coefficient fixed at zero.

D.3 Effect of cost supervision

The two-head objective allows cost targets to update the shared encoder and trunk even when $\lambda = 0$. To isolate this auxiliary-supervision effect, we rerun the Qwen CE and Anchored-CE configurations with $\mathcal{L}_{\text{cost}}$ removed from the total loss. The cost output layer remains in the architecture for check-

point compatibility but receives no cost-target gradient. All evaluations use $\lambda = 0$, so its untrained output cannot affect model selection.

Table 32: **Cost supervision improves performance at $\lambda = 0$.** All rows use pivotal-only training and three seeds. "–" denotes the unrun SWE-bench ablation.

performance objective	cost loss	InterCode-SQL	WebShop	SWE-bench
CE	without	$0.324 \pm .029$	$0.524 \pm .036$	–
	with	$0.377 \pm .009$	$0.545 \pm .015$	$0.600 \pm .020$
Anchored CE	without	$0.348 \pm .017$	$0.499 \pm .011$	–
	with	$\mathbf{0.422 \pm .008}$	$\mathbf{0.573 \pm .036}$	$\mathbf{0.710 \pm .036}$

Cost supervision improves all four comparisons that were run. Under CE, the gains are 0.054 on SQL and 0.021 on WebShop; under Anchored CE, they are 0.074 on both benchmarks. Because the decision rule ignores predicted cost in every row, these gains come from joint representation learning rather than direct use of the cost estimate.

D.4 Training with non-pivotal states

The no-mechanism control retains every decision state while keeping the primary Anchored CE objective. Additional full-data alternatives incorporate $D(s)$ through loss weighting, soft loss gating, adaptive target temperature, or hard dispatch between objectives. They use the same split, architecture, one-epoch optimizer schedule, target network, and three-seed protocol as the primary router; only the training data and indicated mechanism change. Table 33 reports the complete comparison.

Table 33: **Pivotal-only training is at least as effective as the tested full-data recipes.** Every mechanism row trains on all decision states; only the first row restricts training to pivotal states.

mechanism	instantiation	InterCode-SQL	WebShop	SWE-bench
pivotal-only recipe	**Anchored CE**	$\mathbf{0.422 \pm 0.008}$	$\mathbf{0.573 \pm 0.036}$	$\mathbf{0.710 \pm 0.036}$
all decision states, no mechanism				
no mechanism	Anchored CE	0.373 ± 0.023	0.481 ± 0.024	0.653 ± 0.021
disagreement reweighting	$1.0 + D$	0.397 ± 0.009	0.525 ± 0.009	0.703 ± 0.006
	$\sigma(D)$	0.392 ± 0.017	0.566 ± 0.013	0.700 ± 0.010
	$\exp(3D)$	0.392 ± 0.037	0.532 ± 0.023	0.660 ± 0.050
	$0.05 + D^2$	0.382 ± 0.015	0.504 ± 0.045	0.647 ± 0.049
	$0.01 + D$	0.353 ± 0.000	0.515 ± 0.051	0.657 ± 0.059
	$\sigma((D - 0.75)/0.05)$	0.382 ± 0.015	0.501 ± 0.008	0.697 ± 0.047
soft gating on $D(s)$	center 0.5, $\tau = 0.15$	0.389 ± 0.029	0.506 ± 0.033	0.657 ± 0.021
	$\tau = 0.4$	0.382 ± 0.015	0.563 ± 0.004	0.633 ± 0.031
	center 0.85	0.397 ± 0.039	0.507 ± 0.034	0.627 ± 0.015
	$\tau = 0.05$	0.382 ± 0.000	0.519 ± 0.014	0.637 ± 0.021
adaptive CE temperature	gentle	0.382 ± 0.020	0.530 ± 0.025	0.683 ± 0.038
	very gentle	0.402 ± 0.022	0.473 ± 0.036	0.673 ± 0.031
	aggressive	0.397 ± 0.015	0.471 ± 0.026	0.623 ± 0.012
hard dispatch	non-pivotal weight 1.0	0.368 ± 0.000	0.479 ± 0.015	0.610 ± 0.026
	non-pivotal weight 0.1	0.358 ± 0.037	0.456 ± 0.027	0.687 ± 0.021

No mechanism exceeds pivotal-only training on any of the three benchmarks. On SQL, very-gentle adaptive temperature reaches 0.402 ± 0.022 against 0.422 ± 0.008; on WebShop, sigmoid reweighting reaches 0.566 ± 0.013 against 0.573 ± 0.036; on SWE-bench, the best is $1 + D$ at 0.703 ± 0.006 against 0.710 ± 0.036, with the fifteen mechanism means spanning 0.610 to 0.703. The all-data no-mechanism control is instructive: at 0.653 ± 0.021 on SWE-bench it already matches or exceeds six of those fifteen rows, so much of the tested machinery does not improve on simply training on every decision state. Several mechanisms do exceed the ALL-STRONG SWE-bench result of 0.650 from Table 2, so the comparison is between strong recipes rather than against a weak field.

Three training seeds were necessary to reach this reading. Thresholded sigmoid reweighting spans 0.66 to 0.75 across its seeds and averages 0.697 ± 0.047, while $0.05 + D^2$ spans 0.59 to 0.68 and

averages 0.647 ± 0.049. Within-row spreads reach 0.11, wider than the gap between many rows, so these mechanisms cannot be separated on SWE-bench.

The no-mechanism row provides a matched-objective comparison of the data filter. Both rows use Anchored CE and the same training protocol; the control additionally retains non-pivotal states. Filtering to the pivotal subset raises the mean from 0.373 to 0.422 on SQL, from 0.481 to 0.573 on WebShop, and from 0.653 to 0.710 on SWE-bench while using 32.7% of the state–model training examples. The sweep therefore identifies the effective use of pivotality in this pipeline: apply it as a hard data-selection rule, rather than retaining every state and encoding disagreement through weights, gates, temperatures, or objective dispatch.

D.5 Representation probe

On a balanced probe that predicts whether a state–model edge has high backed return, a linear classifier over the frozen representation reaches 0.55 accuracy against 0.50 chance. Scaling Qwen3-Embedding from 0.6B to 4B, replacing it with EmbeddingGemma, and using a pairwise probe do not materially improve accuracy. Representation separability remains the limiting diagnostic, and encoder scale alone does not resolve it.

E Inference-Time Use of Predicted Pivotality

E.1 Setup

Section 3.5 uses predicted model differences to adjust the inference-time cost penalty. Here we compare that rule with alternative mappings and a fixed cost penalty. The pivotal labels used for training come from completed counterfactual trees; inference instead uses the router's predicted performance scores.

Let $q_m(s) = \hat{Q}^{\mathrm{perf}}(s, m)$ be the predicted performance of model m and let $q_{(1)}(s) \geq q_{(2)}(s)$ be the two highest scores. We use two disagreement statistics:

$$d_{\mathrm{range}}(s) = \max_m q_m(s) - \min_m q_m(s), \tag{20}$$

$$d_{\mathrm{gap}}(s) = q_{(1)}(s) - q_{(2)}(s). \tag{21}$$

Because modulation can also change the average cost penalty, we compare each resulting cost–performance frontier with the plain global-λ frontier at matched cost.

E.2 Three modulation rules

We compare three choices of effective cost coefficient $\lambda_{\mathrm{eff}}(s)$ in Fig. 10. The third is the coefficient used in Eq. 9. With reference scale r, sigmoid temperature τ, and $\sigma(x) = (1 + \exp(-x))^{-1}$, they are

$$\lambda_{\text{range-lin}}(s) = \lambda \left[1 - \min\{1, d_{\mathrm{range}}(s)/r\}\right], \tag{22}$$

$$\lambda_{\text{range-sig}}(s) = \lambda \left[1 - \sigma\left((d_{\mathrm{range}}(s) - r)/\tau\right)\right], \tag{23}$$

$$\lambda_{\text{gap-inv}}(s) = \lambda \left[1 - \frac{d_{\mathrm{gap}}(s)}{r + d_{\mathrm{gap}}(s)}\right]. \tag{24}$$

We set $r = 0.2$ for the range rules with $\tau = 0.1$ for the sigmoid, and use $r = 0.05$ for inverse+top-two-gap. The linear range rule is the direct baseline: it removes more cost pressure as predicted cross-model disagreement grows, but saturates once the range reaches r. The sigmoid retains the same statistic while replacing this hard transition with a smooth one. The inverse rule changes both components. Its top-two gap is not inflated by a single weak model, and its mapping is smooth, monotone, and has no hard saturation point. Algebraically, it reduces to $\lambda_{\text{gap-inv}}(s) = \lambda r/(r + d_{\mathrm{gap}}(s))$.

E.3 Results

Range+linear gives the cleanest comparison with cost control by the global λ alone. Its frontier remains above the matched-cost plain frontier in all nine benchmark–λ comparisons, so success

falls more gradually as the realized cost is reduced. This establishes the first inference-time effect of pivotality: selectively relaxing the price at predicted high-contrast states gives a smoother quality–cost tradeoff than applying the same price at every state.

The lower-cost inverse-gap sweep establishes the second effect. Its operating points match or exceed ACRouter at lower cost on SQL and WebShop, and reach comparable SWE-bench performance with 62% less cost (Fig. 3). Range+sigmoid and inverse+top-two-gap are less uniform in the full matched-cost grid, showing that the mapping determines the exact frontier, but the range-linear result and the low-cost baseline comparisons demonstrate the two practical benefits of pivotality-aware inference.

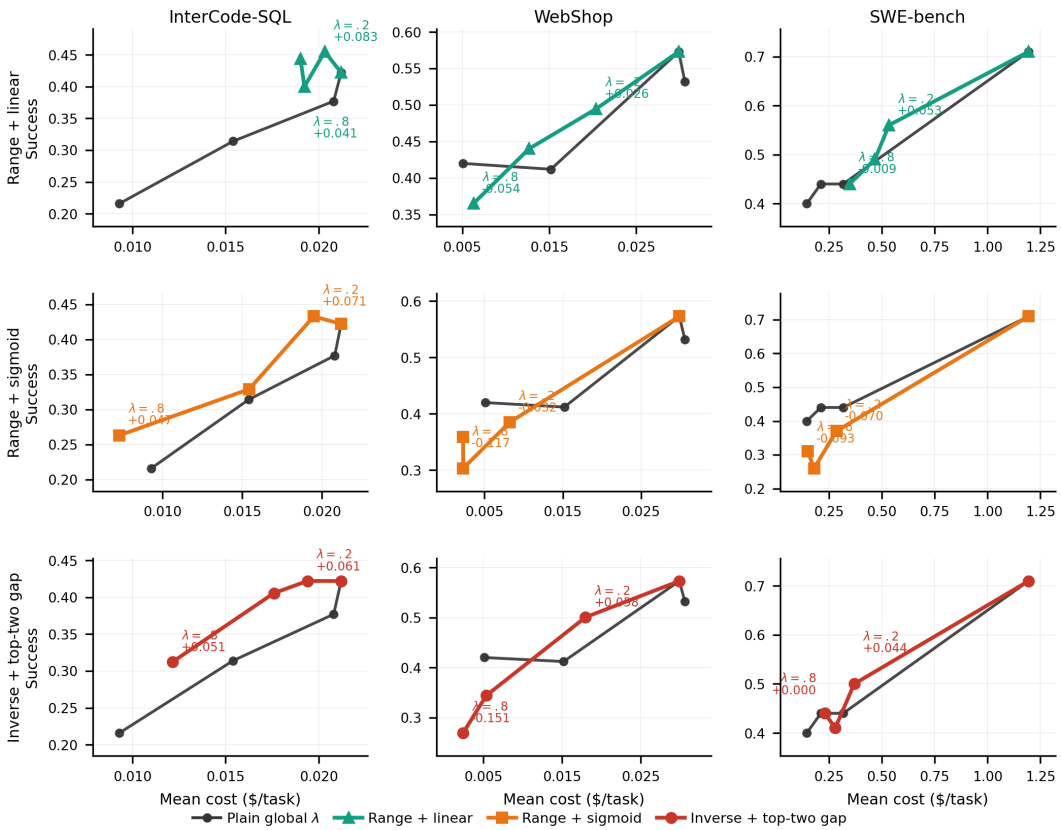

Figure 10: **Pivotality-aware range modulation improves the plain global-λ frontier.** The top row uses range+linear modulation and lies above the black plain-λ frontier at all nine matched-cost points. The lower rows test range+sigmoid and inverse+top-two-gap alternatives; columns show InterCode-SQL, WebShop, and SWE-bench. Among the nonzero colored points, labels mark $\lambda = .2$ and .8, and the unlabeled intervening marker is $\lambda = .4$. The signed value beneath each label reports the success difference from the black frontier at matched cost; labels on the plain frontier and shared $\lambda = 0$ anchor are omitted for clarity.

E.4 Inverse+top-two-gap hyperparameters

The inverse rule has two deployment-time scales. The global coefficient λ determines the overall price assigned to predicted cost, while r determines how quickly a top-two gap relaxes that price. Equivalently,

$$\frac{\lambda_{\text{gap-inv}}(s)}{\lambda} = \frac{r}{r + d_{\text{gap}}(s)}. \tag{25}$$

Thus $r \ll d_{\text{gap}}(s)$ strongly protects a high-gap state from cost pressure, whereas $r \gg d_{\text{gap}}(s)$ approaches the plain global-λ rule. Figure 3 varies λ at the comparison setting $r = 0.05$. We additionally examine $r \in \{0.1, 0.3, 0.5\}$ jointly with $\lambda \in \{0.2, 0.4, 0.8\}$ below.

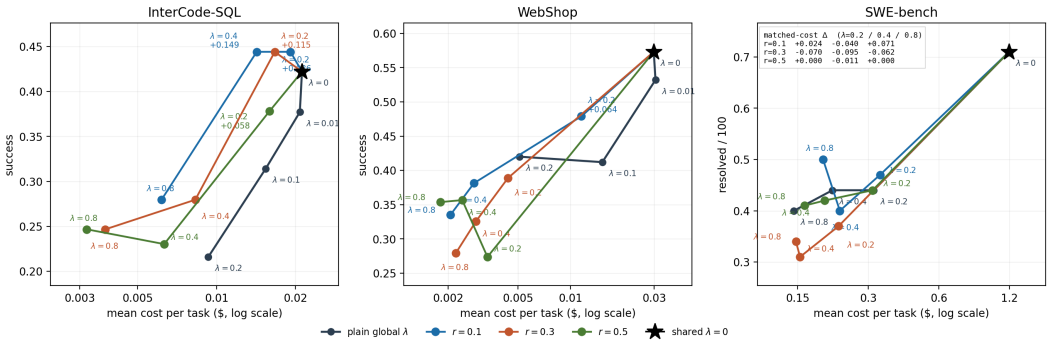

Figure 11: **Sensitivity of inverse+top-two-gap modulation to the reference scale r.** The reference scale and global coefficient are varied jointly on InterCode-SQL, WebShop, and SWE-bench. Signed annotations show the success difference from the plain frontier at matched cost where the measured cost ranges overlap.

Increasing λ moves the router toward cheaper models, but the quality response remains benchmark dependent: the inverse rule improves the SQL frontier, while aggressive cost pressure eventually removes its advantage on WebShop and produces a mixed response on SWE-bench. The reference scale mainly determines how far this movement proceeds. Smaller r lowers $r/(r + d_{\text{gap}})$ and therefore relaxes the cost penalty more strongly, generally moving the operating point toward higher cost and higher quality; larger r moves the policy back toward the plain rule.

The ordering is not monotone at matched cost across benchmarks. In particular, changing r alters both the mean effective penalty and its variation across states, so an apparent improvement cannot be attributed solely to allocating capacity at pivotal states. We therefore tune (λ, r) on the validation frontier as a coupled pair rather than treating r as a benchmark-independent pivotality threshold.