AGENT-AS-A-ROUTER: CLOSING THE FEEDBACK LOOP IN MODEL ROUTING FOR CODING TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Users of coding agents increasingly have access to several frontier Large Language Models (LLMs) that excel on different tasks, so routing each task to a suitable model matters for both quality and cost. Most prior open-source routers treat this as a one-shot classification problem whose information state is frozen after training. We argue that routing for agents is instead a *streaming decision problem with verifiable feedback* since gathering information matters for routing. A controlled diagnosis supports this view: supplying per-dimension performance statistics to an LLM router raises its score by 3–9 points over the same statistics with shuffled model labels, whereas scaling the router from 9B to 27B parameters or eliciting longer reasoning does not. We therefore formalize routing as a Context→Action→Feedback (C-A-F) loop evaluated by cumulative regret against a per-task oracle, instantiate it as **ACRouter** (an Orchestrator, a Verifier, and an execution-grounded Memory), and release **CodeRouterBench**, ∼10K coding tasks with per-task outcomes for 8 frontier LLMs plus a 176-task out-of-distribution (OOD) agentic-programming split. ACRouter attains the lowest cumulative regret on in-distribution streams, and component ablations show that closing the loop accounts for most of the gain over retrieval from a frozen memory. On OOD tasks, static routers trained offline fall to 24–39% resolve rate, at or below random routing. With an exact verifier, verification-driven escalation resolves 73.3% of OOD tasks, 9.1 points above the strongest single model; with deployable verifiers that do not read the grading tests, the gain falls within noise.

1 INTRODUCTION

Coding agents such as Claude Code (Anthropic, 2025) and Codex (OpenAI, 2025) turn LLMs into interactive systems for coding, debugging, and repository-level programming, yet they typically solve every task with one backbone model (Yue et al., 2025). This is natural from a *provider-centric* perspective (Cursor Research et al., 2026), but a user who subscribes to several providers, or even serves open models locally, cares about per-task quality and cost. Across 8 frontier models on 9 coding dimensions (Fig. 4), the best model changes from task to task, and always using the globally strongest model lags the per-task oracle by 13%. *Which model should handle each incoming task?*

Most prior open-source routers frame this as a static classification problem, employing language models as the routing policy (Ong et al., 2025; Liu et al., 2026; Varshney et al., 2026). However, our preliminary experiments reveal that a zero-shot LLM-as-a-Router, even when powered by a highly capable model like Claude Sonnet 4.6, still falls short of the per-task oracle by a wide margin (Table 1). This substantial performance gap suggests that the fundamental bottleneck in model routing extends beyond pure reasoning capabilities.

We ask *what limits such routers*. In a controlled study (§3.1), we vary only the information given to an LLM router. Correct per-dimension performance statistics improve the router by +3.1/+5.2/+5.0 points at 0.8B/9B/27B parameters. The same statistics with shuffled model labels do not help, and neither scaling from 9B to 27B nor longer reasoning closes the gap. Basic reasoning capacity is necessary, but beyond it the router is starved of *decision evidence*. A static router cannot acquire such evidence after deployment; a router embedded in the agent loop can, by verifying the outcome of each decision and remembering it.

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.

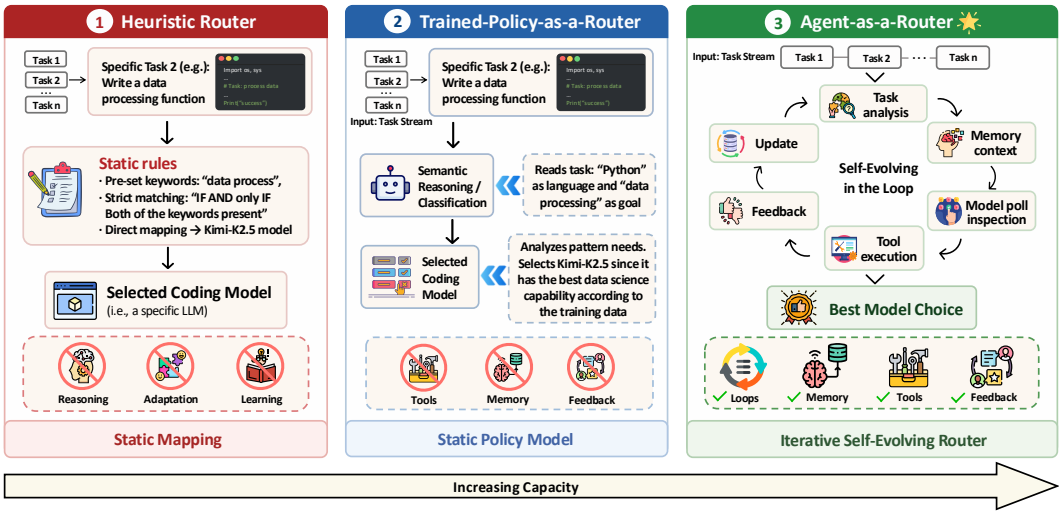

Figure 1: **Three routing strategies.** (1) Static heuristic routers dispatch via a lookup table (e.g., DimensionBest). (2) Static trained routers use a learned policy without memory. (3) Agent-as-a-Router verifies each decision and accumulates the outcome for later decisions.

We therefore propose **Agent-as-a-Router** (Fig. 1), formalizing routing as a Context-Action-Feedback (C-A-F) loop. The router observes a *Context* (prior plus accumulated experience), takes an *Action* (invoking a model), receives verified *Feedback* (score and cost), and writes it back into the Context for the next task. This C-A-F loop relates to a contextual bandit (Li et al., 2010), so *cumulative regret* against the per-task oracle becomes the natural evaluation metric in place of static accuracy.

We instantiate the framework as **ACRouter** (**A**gentic **C**oding **Router**), comprising three core modules (Orchestrator, Verifier, and Memory) and backed by a comprehensive toolkit that includes diverse routing policies, sandboxed execution environments and validation tools. We evaluate ACRouter on **CodeRouterBench**, which contains ∼10K tasks across 9 in-distribution (ID) coding dimensions and an out-of-distribution (OOD) agentic-programming testbed with per-task scores from 8 frontier LLMs to enable regret-based comparison on streaming tasks. ACRouter achieves the lowest cumulative regret in distribution (205.5 vs. 278.1 for the best static router). Under distribution shift (OOD Test), all offline-trained routers fall to or below random routing, while verification-driven escalation with an exact verifier exceeds the strongest single model by 9.1 points. Specifically, our contributions can be summarized as:

- **Framework**. We propose Agent-as-a-Router, a new problem formulation that treats routing as streaming decision making with regret-based evaluation, in which verified feedback is stored as task-level experience and reused both across tasks (memory) and within a task (escalation).
- **Artifacts.** We build ACRouter as the instantiation of C-A-F formulation, and present CodeRouterBench(∼10K tasks, 8 LLMs, 176 OOD agentic tasks with execution-verified outcomes) for regret-based router evaluation.
- **Findings**. Decision evidence rather than reasoning is a major routing bottleneck; ACRouter attains the lowest cumulative regret of all static, online, and single-model routers on in-distribution tasks and, with an exact verifier, on OOD tasks, while offline-trained static routers fall to or below random routing on OOD tasks; Component ablations show that *closing the loop* matters: an online-kNN loop without the finetuned policy already beats every static and bandit router.

2 Related Work

Static LLM routing. Routing among LLMs has been studied as cascades (Chen et al., 2024; Aggarwal et al., 2024), quality-aware query routing (Ding et al., 2024), preference-trained classifiers such

Table 1: **Diagnosis on the 2,919 in-distribution test tasks** (AvgPerf%). **(a)** Claude Sonnet 4.6 as a zero-shot router with increasing information. **(b)** Controlled study with Qwen3.5 routers: no statistics, the *same* statistics with shuffled model labels, or correct statistics. Statistics come from the separate probing set. 95% CIs are from 10,000 paired bootstrap resamples.

(a) Information ablation

Router	AvgPerf%
Oracle (per task)	57.00
DimensionBest (heuristic)	47.50
Vanilla LLM router	41.41
+Dimension description	41.18
+Perf. statistics	**47.74**

(b) Evidence vs. capacity

Router	None	Shuffled	Real	Real−Shuffled [CI]
0.8B	33.94	33.84	37.05	+3.21 [2.59, 3.88]
9B	42.25	38.01	47.45	+9.44 [8.17, 10.76]
27B	42.08	39.43	47.10	+7.68 [6.64, 8.76]
9B + extra reasoning: −2.11 (None), +0.27 (Real)				

as RouteLLM (Ong et al., 2025), and meta-models that predict per-model performance (Šakota et al., 2024; Shnitzer et al., 2024). Benchmarks such as RouterBench (Hu et al., 2024) and LLMRouterBench (Li et al., 2026) evaluate such routers offline on precomputed outputs, and Avengers (Zhang et al., 2025) routes via cost-adjusted cluster statistics. Most of these fit a selector once and evaluate it by single-shot accuracy; we evaluate routers on streams by cumulative regret, and we study agentic coding tasks with execution-verified outcomes.

Online routing and contextual bandits. Routing can be cast as a contextual bandit (Li et al., 2010; Agrawal & Goyal, 2013; Lattimore & Szepesvári, 2020; Yang et al., 2026): MetaLLM (Nguyen et al., 2024), MixLLM (Wang et al., 2025b), PILOT (Panda et al., 2025), and OrcaRouter (Bao et al., 2026) update a router from bandit feedback, and online cascade learning (Nie et al., 2024) learns a cascade over a query stream. Our formulation contains this family as a special case (§3.4), and we compare against LinUCB and LinTS directly. The difference lies in what the feedback carries and how it is reused: these methods compress feedback into per-arm or per-stage parameters, whereas ACRouter records task-level experience (task embedding, chosen model, verified score, cost, verification trace), retrieves it by similarity, and uses the verifier's verdict *within* a task to decide whether to escalate.

Coding agents and agent memory. Coding agents have moved from single-call generation (Austin et al., 2021; Chen et al., 2021) to harnesses that interleave planning, editing, execution, and self-debugging (Jimenez et al., 2024; Yang et al., 2024; Xia et al., 2024; Wang et al., 2025a), sometimes with multiple roles (Hong et al., 2024; Qian et al., 2024). Verbal feedback and episodic memory improve single agents over repeated trials (Shinn et al., 2023). These systems fix the backbone model; we use execution feedback and memory to choose *which* backbone to invoke.

3 Agent-as-a-Router

3.1 Diagnosis: What Limits an LLM Router?

A zero-shot LLM router (Claude Sonnet 4.6) scores 41.41 AvgPerf%, far from the 57.00 per-task oracle (Table 1a), and the best of all 8 candidate LLMs as a zero-shot router (Qwen3.5-Plus, 46.87) also stays below DimensionBest (Appendix D.1). Telling the router the task's dimension does not help, but exposing per-dimension performance statistics from a held-out probing set lifts it to 47.74 (+15.3% relative), slightly above DimensionBest, a heuristic that encodes the same statistics. To separate *information* from *reasoning*, Table 1b varies information and router size independently. At every size, correct statistics beat both no statistics and shuffled-label statistics of identical format and length, so the gain comes from correct decision evidence rather than from the prompt or the output parser. Capacity matters up to a point: from 0.8B to 9B the router gains 8–10 points with no or correct statistics; from 9B to 27B it gains nothing, and a prompt that elicits longer reasoning (16.8→113.0 output tokens) changes AvgPerf by −2.11 without statistics and +0.27 with them. With correct statistics, the 9B, 27B, and Sonnet routers all settle at the level of this evidence (47.10–47.74 vs. 47.50 for DimensionBest). We conclude that once a router has basic reasoning capacity, **missing decision evidence is a major bottleneck**. The remaining gap to the oracle (9–10 points) shows that coarse, dimension-level statistics are not enough, which motivates acquiring finer, task-level evidence during deployment.

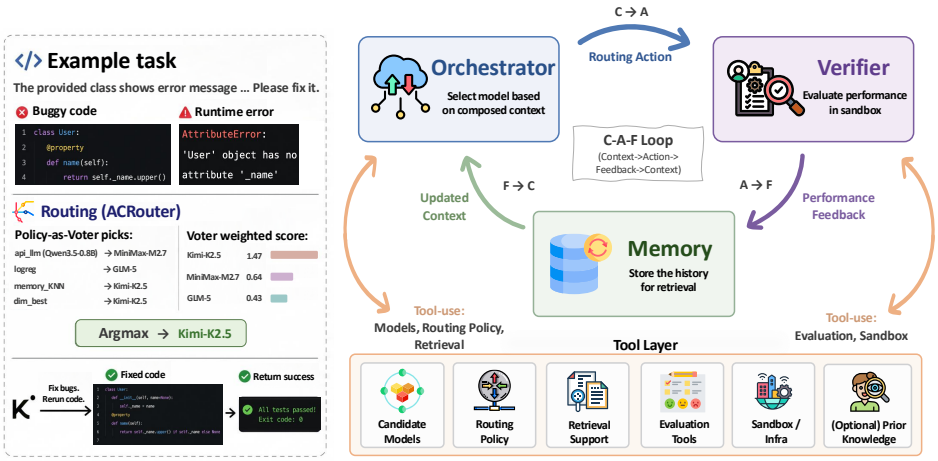

Figure 2: **ACRouter architecture.** The Orchestrator ensembles several voters to select a model, the Verifier scores the execution in a sandbox (and triggers escalation on agentic tasks), and Memory stores the verified outcome for future decisions.

Two design requirements follow: the router must (i) *acquire* execution-grounded evidence at each decision by checking the selected model's output rather than relying on priors or self-assessment, and (ii) *accumulate* this evidence across the task stream so later decisions can condition on it.

3.2 The C-A-F Loop

The router has a model pool $\mathcal{M} = \{m_1, \ldots, m_M\}$ and processes a stream of tasks $(t_1, \ldots, t_N)$ one at a time. At task t_i it observes Context c_i, selects Action $a_i \in [M]$, receives Feedback f_i, and writes the feedback into the next context:

$$c_i \xrightarrow{\text{Decide}} a_i \xrightarrow{\text{Execute}} f_i \xrightarrow{\text{Memorize}} c_{i+1}. \tag{1}$$

Here $c_i = (p_i, d_i, \mathcal{H}_{<i})$ holds the task prompt p_i, optional metadata d_i (dimension, difficulty, language), and the Memory state $\mathcal{H}_{<i}$ accumulated from earlier loops; $f_i = (\hat{s}_i, \hat{\kappa}_i)$ holds the verifier-observed score $\hat{s}_i \in [0, 1]$ and the monetary cost $\hat{\kappa}_i$ of the selected model.

Reward. With history $h_i = (c_1, a_1, f_1, \ldots, c_i)$, a policy $\pi(\cdot \mid h_i) \in \Delta^{M-1}$ is a contextual-bandit policy with side information c_i (Li et al., 2010; Lattimore & Szepesvári, 2020). The per-task reward trades off score and cost under user weights $\epsilon_1 > 0$, $\epsilon_2 \leq 0$:

$$r_i(a) = \epsilon_1 \, s_i(a) + \epsilon_2 \, \kappa_i(a), \tag{2}$$

where $s_i(a)$ and $\kappa_i(a)$ are the ground-truth score and cost of model m_a on t_i. When a router invokes several models on one task (escalation, §3.3), s_i is the score of the returned output and κ_i sums the costs of all invoked models. Only the ratio $\rho = -\epsilon_2/\epsilon_1$ decides which model is preferred: model j beats model k on t_i iff $s_i(j) - s_i(k) > \rho \left(\kappa_i(j) - \kappa_i(k)\right)$ (Lemma 2; formal statements and proofs are in Appendix B), so ρ is an exchange rate between score and dollars. We use $(\epsilon_1, \epsilon_2) = (1, -0.1)$ with cost in USD throughout: an extra dollar per task must buy at least 0.1 score (10 AvgPerf points).

Per-task oracle and regret. To compare routers under identical conditions, we pre-construct the full outcome matrix $O_{ij} = (s_{ij}, \kappa_{ij})$ and the reward matrix $R_{ij} = \epsilon_1 s_{ij} + \epsilon_2 \kappa_{ij}$. The per-task oracle picks $a_i^* = \arg\max_j R_{ij}$ independently for every task, which in general differs from the best single model. We report

$$\mathrm{CumReg}_N(\pi) = \sum_{i=1}^{N} \left(\max_j R_{ij} - r_i(a_i)\right). \tag{3}$$

Only the outcomes of invoked models are revealed to the router online; the full matrix is used only for scoring, with the same reward for every router. Because the reference changes from task to task,

CumReg differs from the usual bandit regret against the best fixed model. The two are linked by an exact decomposition (Proposition 3):

$$\mathrm{CumReg}_N(\pi) \;=\; \underbrace{\textstyle\sum_i \max_j R_{ij} - \max_j \sum_i R_{ij}}_{G_N\text{: complementarity gap}} \;+\; \underbrace{\textstyle\max_j \sum_i R_{ij} - \sum_i r_i(a_i)}_{\text{regret vs. best fixed model}}. \tag{4}$$

The gap $G_N \geq 0$ equals the CumReg of the best single model and measures how much routing can gain at all. On i.i.d. streams, a router that ignores the current task, such as a context-free bandit converging to the best model, cannot go below G_N in expectation, and $\mathbb{E}[G_N]$ grows linearly in N unless one model dominates (Corollary 4). Lower regret needs task-dependent decisions, and finer evidence can only lower the regret attainable with it (Proposition 5).

3.3 ACRouter Instantiation

ACRouter (Fig. 2) realizes the two requirements with three modules.

Orchestrator (integrating evidence). The Orchestrator is a weighted ensemble of voters. Each voter v proposes a model with a confidence $q_v(m)$: a Qwen3.5-0.8B policy LoRA-finetuned on the probing set, an online Memory-kNN voter whose confidence is the mean verified reward of the top-10 retrieved neighbors, and a static DimensionBest prior. The Orchestrator selects $a_i = \arg\max_m \sum_v \omega_v \, \gamma_v(p_i) \, \beta_v(d_i) \, q_v(m)$, where ω_v are voter weights calibrated on the probing split, γ_v adjusts for prompt length, and β_v boosts a voter on the dimensions where it is the calibrated specialist (Appendix A). To try models with little evidence, 5% of decisions explore, and Memory keeps an uncertainty-aware reward estimate per model (unseen models start at mean 0.5).

Verifier (producing evidence). The Verifier turns the selected model's output into a score $u_i = \sum_{k \in \mathcal{K}_{d(t_i)}} w_{d(t_i),k} \, \hat{s}_k(a_i, t_i) \in [0, 1]$ by aggregating task-type-specific tools $\mathcal{K}_d$ (AST parsing, sandbox execution, prompt-embedded tests, rule-based checks) with weights summing to one. For agentic tasks, where one attempt is expensive and outcomes are close to binary, the Verifier's verdict also drives *escalation* within the task. Attempts proceed through a fixed chain of cheaper models and stop at the first attempt the Verifier accepts. If none is accepted and at least k=2 attempts produced an applicable patch, ACRouter escalates to the premium model. §5.3 varies k, and Proposition 11 bounds such a chain by the Verifier's error rates and the gate.

Memory (accumulating evidence). Memory is an online vector store keyed by task embeddings (voyage-code-3 or BGE-large). Each record stores the chosen model, verified score, cost, and verification trace. Retrieval returns the top-10 cosine neighbors above similarity 0.5. The store is FIFO-bounded at 20K entries and updated after every loop, so recent failures of a model on similar tasks are visible to the next decision, at a granularity finer than dimension-level statistics.

3.4 Existing Routers as Partial C-A-F Loops

Removing components from the loop recovers the baseline families we compare against (Table 2). **Single-model** routers (*Always-m*) have no Orchestrator, Verifier, or Memory. **Static heuristics** read a frozen probing-set Memory: *DimensionBest* routes each task to its dimension's best model, and *kNN Retrieval* copies the best model of the nearest probing tasks. **Static trained policies** have a trained Orchestrator and no Memory or Verifier: LogReg on TF-IDF features, TF-IDF+MLP, RouteLLM (Ong et al., 2025) (matrix factorization and BERT), and LoRA-finetuned (Hu et al., 2022) Qwen3.5 (Qwen Team, 2026) (0.8B in the main table; 0.8B–27B in Appendix D.2). **Online bandits** keep per-arm parametric posteriors as Memory, use an $\arg\max$ Orchestrator, and observe only the scalar reward: LinUCB (Li et al., 2010) (α=λ=1) and LinTS (Agrawal & Goyal, 2013), with categorical or 64-dim projected-embedding contexts, warm-started on the probing set and updated online with the evaluation reward $(\epsilon_1, \epsilon_2) = (1, -0.1)$.

These families differ in how they estimate each model's reward on the current task: from the probing set (static), from per-arm posteriors (bandits), or from similarity-retrieved verified outcomes (ACRouter); a static router's regret slope stays constant along the stream (Proposition 7).

Table 2: **Routing methods as C-A-F configurations.** Single-model, heuristic, and trained routers leave Memory empty or frozen; online bandits update per-arm posteriors from the scalar reward; ACRouter activates all three core modules.

Method family	Orchestrator	Verifier	Memory
Single-model (*Always-m*)	Direct dispatch	—	—
Heuristic (DimensionBest, kNN)	Static lookup	—	Frozen probing prior
Trained (LogReg, MLP, RouteLLM, Qwen)	Trained model	—	—
Online bandit (LinUCB, LinTS)	arg max rule	Reward only	Per-arm posterior
ACRouter (loop-complete)	Voters + escalation	Sandbox-native	Online task-level kNN

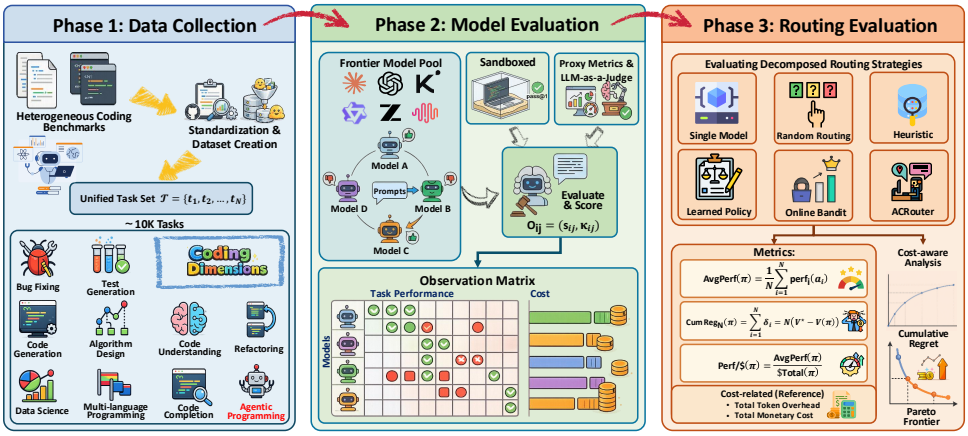

Figure 3: **CodeRouterBench construction and evaluation pipeline.** Phase 1: 15 benchmark sources are unified into ∼10K tasks across 10 coding dimensions. Phase 2: 8 frontier LLMs generate the observation matrix, scored by sandboxed execution and reference-based metrics. Phase 3: routers are replayed on task streams and compared on performance, regret, and cost.

4 CODEROUTERBENCH

Regret-based streaming evaluation requires per-task outcomes for *every* model, so any router can be replayed on any stream order. CodeRouterBench repurposes 15 public benchmarks (Chen et al., 2021; Austin et al., 2021; Zhuo et al., 2024; Jain et al., 2025; Lai et al., 2023; Gu et al., 2024; Jimenez et al., 2024; Zheng et al., 2023; Cassano et al., 2023; Tufano et al., 2019; Cassano et al., 2024; Lu et al., 2021) into 9 single-turn coding dimensions (1,111 tasks each). Five are scored by sandboxed execution and four by matching a reference output (Appendix Table 7). A deterministic split yields a **probing set** (7,080 tasks) for developing routers and an **in-distribution (ID) test** stream (2,919 tasks). The construction pipeline is shown in Fig. 3.

OOD agentic programming. The 10th dimension is held out entirely as an OOD test of 176 repository-level tasks from SWE-bench Verified (OpenAI, 2024) (112), FeatureBench (Zhou et al., 2026) (49), LongCLI-Bench (Feng et al., 2026) (14), and SWE-CI (Chen et al., 2026) (1), filtered for low similarity to the probing set. These tasks require multi-step planning, file navigation, and iterative debugging. Every model runs with mini-swe-agent[1] (40-step cap), and patches are graded in the Docker harness of Jimenez et al. (2024) (model versions and cost logging per source are in Appendix E.5). No router sees any OOD label during development.

Model pool. The pool contains Claude Opus 4.6 and Sonnet 4.6 (Anthropic, 2026a;b), GPT-5.4 (OpenAI, 2026), Qwen3-Max (Yang et al., 2025), Qwen3.5-Plus (Qwen Team, 2026), GLM-5 (Zeng et al., 2026), Kimi-K2.5 (Kimi Team et al., 2026), and MiniMax-M2.7 (MiniMax, 2026). **No model dominates** (Fig. 4): Opus 4.6 has the best average but loses algorithm design to GLM-5 (48 vs. 25 on the probing set), test generation to Qwen3-Max (83 vs. 40), and data science to Kimi-K2.5

[1] https://github.com/SWE-agent/mini-swe-agent

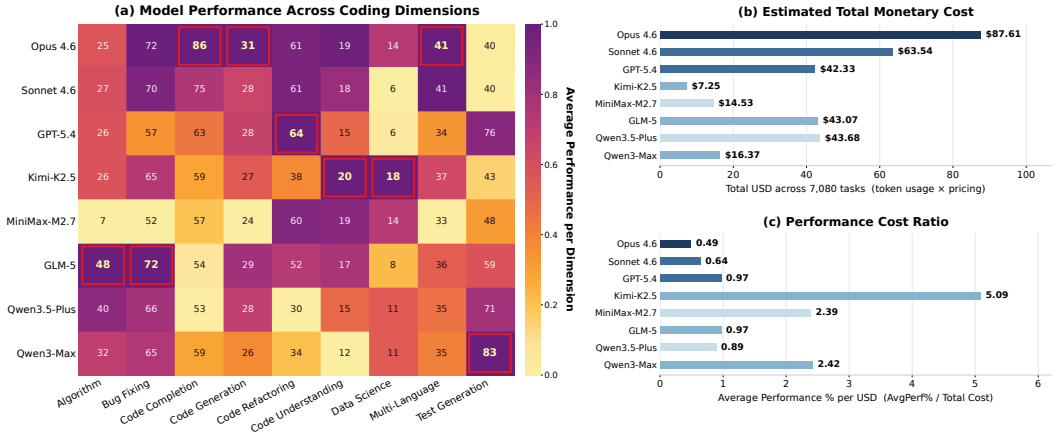

Figure 4: **Model complementarity on the probing set.** (a) Scores of 8 models on 9 dimensions. (b) Total cost over 7,080 tasks. (c) Performance per dollar.

(18 vs. 14). Five different models are dimension-best across the 9 dimensions, and costs differ by 12× between Opus and Kimi.

5 Experiments

Metrics. *AvgPerf*: mean score (OOD: harness-graded resolve rate). *CumReg*: Eq. 3. *Perf/$*: AvgPerf% per USD, where cost covers all backend calls at official prices (Appendix C.3) plus router-side tokens (self-hosted routers at a measured $0.054/M; Appendix C.3.2).

5.1 In-Distribution Streams

ACRouter attains the lowest regret. On the ID stream (Table 3, left), ACRouter reaches 49.98% AvgPerf and 205.5 CumReg. DimensionBest, which holds the full dimension-level prior, reaches 47.50 and 278.1, and the best trained and bandit routers are within 0.7 points of it. ACRouter costs about the same as DimensionBest ($13.21 vs. $12.89) and 2.6× less than Always-Opus ($34.02), which it outperforms by 6.2 points. Trained classifiers remain the most cost-efficient routers above 46% (Perf/$ 6.1–6.8). Fig. 5 shows that ACRouter's regret curve separates from the static routers early and keeps a lower slope.

Latency. Routing adds 0.35 s per task on one A100 GPU (6.6% of inference latency; Appendix E.6).

5.2 Which Components Close the Loop?

Closing the loop is the main ingredient. Table 4a removes one component at a time. The sharpest contrast is between two retrieval-only routers: the online-kNN loop, whose Memory fills from the stream, reaches regret 241.2, against 286.7 for kNN Retrieval on the frozen probing Memory, and already beats every static and bandit router in Table 3. The remaining components lower regret by a further 35.7. Among single removals that break the loop, removing the Verifier costs the most (+17.1 regret, against +9.6 for Memory). The finetuned policy voter matters most among all single removals (+26.6), as it supplies the prior that Memory refines.

Feedback quality matters more than timeliness. In Table 4b, delaying feedback by up to 100 tasks costs about one point, while flipping stored scores costs more, most at $p=\frac{1}{2}$, since flips shrink every expected score gap between models by a factor $1-2p$ (Proposition 13). Without feedback, the router falls to 47.32, near DimensionBest (47.50); its 1.42-point gap to the 48.74 baseline is the value of online feedback under a fixed policy.

Table 3: **Main results.** Left: ID stream (2,919 tasks, 9 dimensions). Right: OOD agentic programming (176 tasks). Stochastic rows: means over seeds (LinTS/LinUCB: 5 on ID; LinTS: 10 on OOD, s.d. 3.27; Random: 10). DimensionBest cannot be applied to the unseen OOD dimension. †On OOD, the Verifier's verdict is the harness outcome of each attempt (an exact verifier), so this row upper-bounds verification-driven escalation; Table 5 reports deployable verifiers.

	In-Distribution (n=2,919)			**OOD Agentic** (n=176)		
Router	**AvgPerf%↑**	**CumReg↓**	**Perf/$↑**	**AvgPerf%↑**	**CumReg↓**	**Perf/$↑**
Oracle (per task)	57.00	0	8.20	78.98	0	2.86
Agent-as-a-Router						
ACRouter (ours)†	**49.98**	**205.5**	3.78	**73.30**	**15.9**	0.85
Dynamic: Online Bandit						
LinTS	46.48	307.4	4.49	57.16	42.8	0.81
LinUCB	46.84	296.9	4.38	55.11	46.7	0.73
Static: Heuristic						
DimensionBest	47.50	278.1	3.69	—	—	—
kNN Retrieval	47.18	286.7	6.07	32.39	80.0	4.33
Static: Trained Policy						
LogReg	47.26	284.4	6.27	39.20	68.0	4.89
RouteLLM-BERT	47.22	285.5	6.22	31.82	80.8	5.25
TF-IDF+MLP	46.97	292.8	6.11	36.93	71.9	5.83
Qwen3.5-0.8B-FT	46.41	309.1	6.82	23.86	95.5	1.88
RouteLLM-MF	46.16	316.5	6.19	32.95	78.7	7.93
Single-Model						
Always-Opus 4.6	43.83	387.1	1.29	63.64	43.7	0.33
Always-GPT-5.4	42.21	432.9	2.55	64.20	25.1	3.51
Always-Kimi-K2.5	36.66	593.3	**12.62**	19.89	101.4	**13.60**
Always-Qwen3.5-Plus	37.16	580.2	2.05	27.27	88.9	4.39
Random	38.75	533.6	2.48	40.34	70.9	0.73

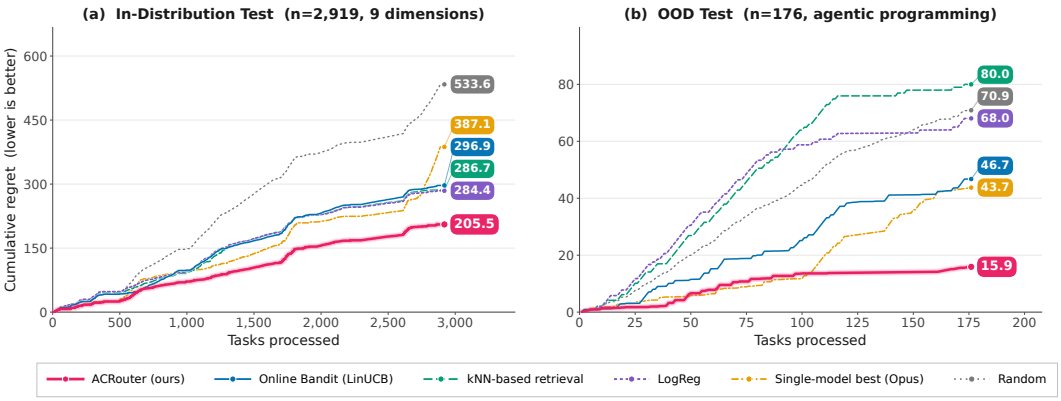

Figure 5: **Cumulative regret along the task streams.** Static routers accumulate regret faster on the ID stream (left) and much faster under distribution shift (right).

5.3 Out-of-Distribution Agentic Programming

Static and online routers on OOD. All offline-trained routers reach 23.9–39.2% resolve rate on the OOD split (Table 3, right), at or below Random (40.3%), even though they sit within 1.4 points of DimensionBest in distribution. Online bandits update from the new outcomes and reach 55–57%.

Closing the loop within a task. On OOD, the Verifier decides *inside* a task whether to accept a patch or escalate. With an exact verifier, ACRouter resolves 73.30% of tasks at CumReg 15.9, below every router in Table 3, exceeding Always-GPT-5.4 by +9.09 points and Always-Opus by +9.66 ([3.41,

Table 4: **What closes the loop** (ID stream). **(a)** Component ablations; Δ vs. the full ACRouter (49.98 AvgPerf, 205.5 CumReg). "Loop": verified feedback still reaches Memory. Projection: $\Delta\text{Reg} = N\,|\Delta\text{Perf}|/100$ at unchanged cost (Eq. 3). **(b)** Online-kNN loop with feedback delayed by d tasks or flipped ($s \mapsto 1-s$) w.p. p (5 seeds). *No feedback* never updates Memory; *Frozen kNN only* retrieves from the full probing set.

(a) Component ablations

Variant	Loop	ΔPerf	ΔReg
w/o Verifier	×	−0.58	+17.1
w/o Memory	×	−0.33	+9.6 Projection
w/o FT policy voter	✓	−0.91	+26.6 Projection
Rule-based Orchestrator	✓	−0.53	+15.4
Uniform voter weights	✓	−0.37	+10.8 Projection
w/o dimension prior	✓	−0.21	+6.1 Projection
Online kNN only	✓	−1.24	+35.7
Frozen kNN only	×	−2.80	+81.2

(b) Degraded feedback

Condition	AvgPerf%	Δ
Baseline (d=1, p=0)	48.74 ±0.18	—
Delay d=10	48.55	−0.19
Delay d=50	47.79	−0.95
Delay d=100	47.73	−1.01
Flip p=0.1	47.94	−0.80
Flip p=0.3	46.60	−2.14
Flip p=0.5	44.41	−4.33
No feedback	47.32 ±0.12	−1.42

Table 5: **Verification drives OOD performance.** Top: exact verifier, varying the escalation rule. Middle: k=2 with deployable verifiers that do not read the grading tests (pass-to-pass tests, an optimistic proxy; best of 9 test-blind Qwen judges; $\Delta \in [-4.55, +1.70]$ over 20 variants); they accept the first attempt on the 64 added tasks, which have no patches. Verifier compute is not charged. Δ: paired bootstrap 95% CI, exact McNemar p.

Escalation	AvgPerf%	CumReg	$Total	Δ vs. GPT-5.4 [CI], p
k=2 (default)	73.30	15.9	86.72	+9.09 [5.11, 13.64], 3×10^{-5}
None	71.02	15.8	46.07	—
k=1 / k=3	77.84 / 72.16	13.7 / 15.4	144.7 / 61.6	—
Pass-to-pass tests, k=2	64.77	25.4	31.37	+0.57 [−4.55, 5.68], 1.00
Test-blind judge, k=2	65.91	24.1	38.47	+1.70 [−3.41, 6.82], 0.65
Always-GPT-5.4 / Always-Opus	64.20 / 63.64	25.1 / 43.7	18.3 / 194.9	—

15.91], p=0.006) at 44% of Opus's cost, and k sets its cost (Table 5). Deployable verifiers reach only 59.7–65.9%, within noise of GPT-5.4.

6 Conclusion

Conclusion. A router with a frozen information state cannot acquire missing decision evidence after deployment. Agent-as-a-Router instead treats routing as a Context–Action–Feedback loop with cumulative regret as the metric, and CodeRouterBench makes this evaluation reproducible. Correct per-dimension statistics, not larger routers or longer reasoning, lift an LLM router to the DimensionBest level (§3.1). On the ID stream, an online-kNN loop fed by verified outcomes already beats every static and bandit router, and ACRouter lowers cumulative regret from 278.1 to 205.5 at about the cost of DimensionBest. On OOD tasks, offline-trained routers fall to or below random routing. The ID and OOD results show the value of acquiring verified evidence and reusing it across tasks through memory and within a task through verification-driven escalation, whose OOD gain depends on verifier quality. Closing the gap between deployable and exact verifiers is therefore the key open problem for routing agentic tasks.

Limitations and future work. Costs use public token prices and measured usage (imputed on the 64 added OOD tasks), cache hit rates are unobserved, OOD agents run with a 40-step budget, and OOD escalation assumes an exact verifier. Parametric memory, learned escalation policies, and learned verifiers are natural extensions.

AI USE STATEMENT

LLMs were used to review and edit the grammar and wording of the manuscript. All presentations in this manuscript were designed and verified by the authors.

ETHICS STATEMENT

This work uses public benchmarks and commercially available or open-weight models; no human subjects or personal data are involved. Routing can lower the cost of capable coding assistance, but escalation consumes extra model calls, whose cost we report alongside quality (verifier compute is not charged). Deployed routers that execute generated code should run it in isolated sandboxes, as ours do.

REPRODUCIBILITY STATEMENT

All ID and OOD numbers are offline replays of released per-task outcome matrices. Only the router's own decisions are recomputed, and only the invoked models' outcomes are revealed online. We will release the code, CodeRouterBench (tasks, per-model scores, costs, and splits), the OOD harness reports, the finetuned router adapter, and the evaluation scripts. Seeds, splits (identifier `coding-router-v1`), pricing, and router training hyperparameters are listed in the appendix. Confidence intervals are task-level paired bootstrap intervals with 10,000 resamples.

REFERENCES

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

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pp. 127–135, 2013.

Anthropic. Claude Code: An agentic coding tool. https://docs.anthropic.com/en/docs/agents-and-tools/claude-code/overview, 2025.

Anthropic. Introducing Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, 2026a.

Anthropic. Introducing Claude Sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, 2026b.

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

Zhenghua Bao, Fengya Tian, Chris Zhang, Zhenjun Chen, Xile Ma, and Yi Shi. OrcaRouter: A production-oriented LLM router with hybrid offline-online learning. *arXiv preprint arXiv:2605.30736*, 2026.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023.

Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. Can it edit? evaluating the ability of large language models to follow code editing instructions. In *First Conference on Language Modeling*, 2024.

Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. SWE-CI: Evaluating agent capabilities in maintaining codebases via continuous integration. *arXiv preprint arXiv:2603.03823*, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=cSimKw5p6R.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cursor Research, Aaron Chan, Ahmed Shalaby, Alexander Wettig, Aman Sanger, Andrew Zhai, Anurag Ajay, Ashvin Nair, Charlie Snell, Chen Lu, et al. Composer 2 technical report. *arXiv preprint arXiv:2603.24477*, 2026.

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

Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, et al. LongCLI-Bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. *arXiv preprint arXiv:2602.14337*, 2026.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. CRUXEval: a benchmark for code reasoning, understanding and execution. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 16568–16621, 2024.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

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? In *The Twelfth International Conference on Learning Representations*, 2024.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S.H. Cai, Yuan Cao, Ziwei Chai, Y. Charles, H.S. Che, Cheng Chen, Guanduo Chen, et al. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345, 2023.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, et al. LLMRouterBench: A massive benchmark and unified framework for llm routing. *arXiv preprint arXiv:2601.07206*, 2026.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 661–670, 2010.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. Adaptive vision-language model routing for computer use agents. *arXiv preprint arXiv:2603.12823*, 2026.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

MiniMax. MiniMax M2.7. https://www.minimax.io/models/text/m27, 2026.

Quang H. Nguyen, Thinh Dao, Duy C. Hoang, Juliette Decugis, Saurav Manchanda, Nitesh V. Chawla, and Khoa D. Doan. MetaLLM: A high-performant and cost-efficient dynamic framework for wrapping LLMs. *arXiv preprint arXiv:2407.10834*, 2024.

Lunyiu Nie, Zhimin Ding, Erdong Hu, Christopher Jermaine, and Swarat Chaudhuri. Online cascade learning for efficient inference over streams. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 38071–38090, 2024.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025.

OpenAI. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

OpenAI. Codex CLI: An agentic coding assistant. https://github.com/openai/codex, 2025.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4, 2026.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive LLM routing under budget constraints. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23934–23949, 2025. doi: 10.18653/v1/2025.findings-emnlp.1301.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026.

Marija Šakota, Maxime Peyrard, and Robert West. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 606–615, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Tal Shnitzer, Anthony Ou, Mirian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. In *Conference on Language Modeling*, 2024.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, and Maosong Sun. DebugBench: Evaluating debugging capability of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4173–4198, 2024.

Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology*, 28(4):1–29, 2019.

Tanay Varshney, Annie Surla, Michelle Xu, Gomathy Venkata Krishnan, Maximilian Jeblick, David Austin, Neal Vaidya, and Davide Onofrio. LLM Router: Rethinking routing with prefill activations. In *Conference on Language Modeling*, 2026. arXiv:2603.20895.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. MixLLM: Dynamic routing in mixed 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. 10912–10922, 2025b. doi: 10.18653/v1/2025.naacl-long.545.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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. *Advances in Neural Information Processing Systems*, 37:50528–50652, 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.

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

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Yiqun Zhang, Hao Li, Chenxu Wang, Linyao Chen, Qiaosheng Zhang, Peng Ye, Shi Feng, Daling Wang, Zhen Wang, Xinrun Wang, Jia Xu, Lei Bai, Wanli Ouyang, and Shuyue Hu. The Avengers: A simple recipe for uniting smaller language models to challenge proprietary giants. *arXiv preprint arXiv:2505.19797*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, et al. CodeGeeX: A pre-trained model for code generation with multilingual benchmarking on HumanEval-X. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 5673–5684, 2023.

Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, et al. FeatureBench: Benchmarking agentic coding for complex feature development. *arXiv preprint arXiv:2602.10975*, 2026.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

A ACROUTER SYSTEM ARCHITECTURE

ACRouter (§ 3.3) is composed of a two-layer modular architecture: a *decision layer* of three core modules that realize the C-A-F loop, and a *tool layer* of shared infrastructure that the core modules invoke (see Fig. 2). This appendix details the modules and the tool layer (§A.1); the C-A-F taxonomy of all reported routers is in Table 2 (§3.4), and the reward conventions and formal analysis are in Appendix B.

A.1 CORE MODULES AND TOOL LAYER

Core modules (decision layer).

- **Orchestrator**: the central coordinator that selects which candidate model to invoke for the current task t_i. It consults Memory state, ingests the DimensionBest prior, the top-10 historical neighbors retrieved from Memory by cosine kNN, and the task metadata, and combines three voter groups by weighted voting: a LoRA-finetuned Qwen3.5-0.8B policy voter, an online Memory-kNN voter, and static voters (DimensionBest prior). The score of model m is $\sum_v \omega_v \, \gamma_v(p_i) \, \beta_v(d_i) \, q_v(m)$, where $q_v(m)$ is the voter's confidence (policy probability, Memory mean verified reward, or prior score), γ_v discounts voters on prompts longer than their calibration range, and β_v is a multiplicative boost for the voter calibrated as the specialist of dimension d_i. Weights and specialist assignments are calibrated on the probing split only. With probability 0.05 the Orchestrator explores; Memory additionally keeps a per-model reward mean and uncertainty, initialized at mean 0.5 and standard deviation 1 for models without history, and samples from it so that new models are tried quickly.
- **Verifier**: a sandbox-native confidence estimator that examines each candidate's output and aggregates multiple signal tiers (AST parse, sandbox execution, prompt-embedded tests, and rule-based signals) into a unified score $u_i \in [0, 1]$ (§3.3). The Verifier produces the verdict that is written into Memory.
- **Memory**: an online vector store keyed by task embeddings (voyage-code-3 / BGE-large) whose value logs the chosen model, observed performance, monetary cost, and the Verifier's verification trace. Retrieval uses cosine kNN over embeddings (similarity threshold 0.5, k=10). The store is FIFO-bounded at 20K entries and is committed in place after each loop. Per-task granularity makes both past successes and recent failures of any candidate on similar tasks visible to the Orchestrator at the next decision.

Tool layer (execution infrastructure). Shared resources invoked by the core modules and required to instantiate any router under the C-A-F loop.

- **Candidate Models** (the model pool): the 8 LLMs available for routing (Claude Opus 4.6, Sonnet 4.6, GPT-5.4, Qwen3-Max, Qwen3.5-Plus, Kimi-K2.5, GLM-5, MiniMax-M2.7). The Orchestrator selects from these.
- **Routing Tools** (serve the Orchestrator): dimension-best lookup tables, trained classifiers, and LLM-based selectors, i.e. the mechanisms the Orchestrator uses to make its selection.
- **Evaluation Tools** (serve the Verifier): AST parser, sandbox runner, self-consistency checker (k-sample output agreement), and a prompt-test extractor that identifies in-prompt test cases. On ID tasks they produce quality signals without ground-truth tests, and the Verifier aggregates them into a confidence score; on OOD tasks the Verifier's verdict is the harness outcome of each attempt (an exact verifier; §5.3).
- **Embedding Encoder** (serves Memory): maps a task's prompt text into a dense vector for kNN retrieval. Implementations can range from a code-specialized API (e.g., voyage-code-3) to a local open-source model (e.g., BGE-large).
- **Infrastructure**: execution sandbox (Docker, timeouts), context parser (extracts dimension/difficulty/language from raw input), and an online updater for Memory statistics.

A.2 DECISION PROCEDURE

Algorithm 1 gives one C-A-F loop of ACRouter. The Orchestrator combines three voters: the finetuned Qwen3.5-0.8B policy, whose confidence $q_{\text{pol}}(m)$ is its probability for model m; the

Memory-kNN voter, whose confidence $q_{\text{mem}}(m)$ is the mean verified reward of m among the retrieved neighbors; and the DimensionBest prior, whose confidence $q_{\text{dim}}(m)$ is the probing-set mean score of m on dimension d_i. The voter weight ω_v is fitted on the probing split. The length factor $\gamma_v(p_i) \leq 1$ equals 1 when the prompt length lies inside the range on which voter v was calibrated and down-weights v otherwise. The specialist factor $\beta_v(d_i) \geq 1$ is above 1 only for the voter calibrated as the specialist of dimension d_i. Memory also keeps, for every model, a reward mean $\hat{\mu}_m$ and uncertainty $\hat{\sigma}_m$ computed from its verified rewards, with $(\hat{\mu}_m, \hat{\sigma}_m) = (0.5, 1)$ before its first observation; exploration samples from these estimates.

Algorithm 1 One C-A-F loop of ACRouter on task t_i

Require: pool $\mathcal{M}$; voters $v \in \{\text{pol}, \text{mem}, \text{dim}\}$ with $(\omega_v, \gamma_v, \beta_v)$; Memory $\mathcal{H}_{<i}$ (FIFO, 20K records); $\varepsilon = 0.05$; for agentic tasks, chain $(m_1, \ldots, m_{L-1})$, premium model m_L, gate k

Context: embed p_i; retrieve the top-10 records of $\mathcal{H}_{<i}$ with cosine similarity ≥ 0.5
if t_i is a single-turn task **then**
  $S(m) \leftarrow \sum_v \omega_v \gamma_v(p_i) \beta_v(d_i) q_v(m)$ for every $m \in \mathcal{M}$
  Action: with probability ε, $a_i \leftarrow \arg\max_m \tilde{\theta}_m$ with $\tilde{\theta}_m \sim \mathcal{N}(\hat{\mu}_m, \hat{\sigma}_m^2)$; otherwise $a_i \leftarrow \arg\max_m S(m)$
  run m_{a_i}; **Feedback:** Verifier score u_i and cost $\hat{\kappa}_i$
else
  for $\ell = 1, \ldots, L-1$ **do**
    run m_ℓ; **if** the Verifier accepts its patch **then** return it and stop
  end for
  if no patch was accepted and at least k attempts produced an applicable patch **then** run m_L and return its patch if the Verifier accepts it
  otherwise return the last attempt; $u_i \leftarrow$ Verifier verdict of the returned patch; $\hat{\kappa}_i \leftarrow$ total cost of all attempts
end if
Memorize: append $(\phi(p_i), a_i, u_i, \hat{\kappa}_i, \text{trace})$ to $\mathcal{H}$; update $(\hat{\mu}_{a_i}, \hat{\sigma}_{a_i})$

B Theoretical Analysis of the C-A-F Formulation

This section gives the formal statements behind §3.2–§3.4 and the findings of §5. §B.1 fixes notation and classifies routers by the information their decisions use. §B.2 states the reward conventions behind every number in the paper. §B.3 decomposes the cumulative regret and shows that, unless one model dominates, routers that ignore the task pay a cost linear in the stream length. §B.4 shows that finer evidence can only lower the attainable regret and characterizes static-router regret on a fixed distribution. §B.5, §B.6, and §B.7 analyze the three ingredients of ACRouter: similarity-retrieved Memory, verification-driven escalation, and robustness to corrupted feedback. §B.8 relates selected statements to the experiments.

B.1 Setup and Information States

A stream consists of tasks $t_1, \ldots, t_N$ and a model pool $\mathcal{M} = \{m_1, \ldots, m_M\}$. Model j on task i has score $s_{ij} \in [0, 1]$, cost $\kappa_{ij} \in [0, \bar{\kappa}]$, and reward $R_{ij} = \epsilon_1 s_{ij} + \epsilon_2 \kappa_{ij}$ with $\epsilon_1 > 0 \geq \epsilon_2$, so every reward lies in an interval of width at most $B = \epsilon_1 + |\epsilon_2|\,\bar{\kappa}$. We write $R_{i\cdot} = (R_{i1}, \ldots, R_{iM})$ for the reward row of task i. At task i the router observes $c_i = (p_i, d_i, \mathcal{H}_{<i})$; a single-model decision a_i earns $r_i = R_{i,a_i}$. Whenever we take expectations over the stream, the pairs $(t_i, R_{i\cdot})$ are i.i.d. draws from a task distribution $\mathcal{D}$ (for the replayed benchmark, $R_{i\cdot}$ is a fixed function of t_i), and $\mu_j = \mathbb{E}_{\mathcal{D}}[R_j]$. The replayed streams are uniformly random orders of a fixed task set. Proposition 3 holds for every order, and Proposition 7 holds exactly with $\mathcal{D}$ the empirical task distribution, because each position of a random order is uniform over the task set; the other expectation statements treat the stream as i.i.d.

Definition 1 (Information states)**.** A router is *context-free* if its single-model decision a_i depends only on $\mathcal{H}_{<i}$ and internal randomness, not on (p_i, d_i); *static* if $a_i = g(p_i, d_i, \xi_i)$ for a map g fixed before deployment and internal randomness ξ_i, so that a_i does not depend on $\mathcal{H}_{<i}$; and *closed-loop* if a_i may depend on both (p_i, d_i) and the verified outcomes in $\mathcal{H}_{<i}$.

Single models and non-contextual bandits are context-free, the heuristic and trained routers of Table 2 are static, and the contextual bandits and ACRouter are closed-loop.

B.2 Reward Weights and the Per-Task Oracle

AvgPerf reports the raw mean score. CumReg uses the canonical evaluation reward $(\epsilon_1, \epsilon_2) = (1, -0.1)$ with κ in USD under Table 8, i.e. $r_i(a) = s_i(a) - 0.1\,\kappa_i(a)$, and \$Total and Perf/\$ report cost separately. The online bandits (LinUCB, LinTS) use the same weights for their posterior updates, and every router, whether bandit, classifier, or ACRouter, is scored under this single evaluation reward, so the CumReg column is comparable across method families. When a router invokes a set A_i of models on task i and returns the output of $\hat{a}_i \in A_i$, its reward is $r_i = \epsilon_1 s_{i\hat{a}_i} + \epsilon_2 \sum_{a \in A_i} \kappa_{ia} \le R_{i\hat{a}_i}$.

The oracle in Eq. 3 is recomputed for each task, $r_i^* = \max_{j \in [M]} R_{ij}$, so that

$$\mathrm{CumReg}_N(\pi) \;=\; \sum_{i=1}^{N} \big(r_i^* - r_i\big) \;=\; \sum_{i=1}^{N} \Big[\max_{j \in [M]} R_{ij} - r_i\Big].$$

Since escalation only adds cost, $r_i \le R_{i\hat{a}_i} \le r_i^*$ and every summand is nonnegative. CumReg thus sums task-level reward gaps to a reference that may pick a different model on every task; it is neither the gap to the best fixed model nor the gap in mean score alone (Proposition 3 relates it to the former).

Lemma 2 (Scale invariance and exchange rate)**.** *Let $\rho = -\epsilon_2/\epsilon_1 \ge 0$. (i) Replacing (ϵ_1, ϵ_2) by $(\lambda\epsilon_1, \lambda\epsilon_2)$ with $\lambda > 0$ leaves every oracle decision a_i^* unchanged and multiplies the CumReg of any fixed sequence of decisions by λ. (ii) Model j earns a higher reward than model k on task i iff $s_{ij} - s_{ik} > \rho\,(\kappa_{ij} - \kappa_{ik})$.*

Proof. $R_{ij} = \epsilon_1(s_{ij} - \rho\,\kappa_{ij})$. Rescaling multiplies every entry of R and every realized reward by λ, which preserves each $\arg\max_j R_{ij}$ and scales each difference $r_i^* - r_i$ by λ. For (ii), $R_{ij} > R_{ik}$ iff $s_{ij} - \rho\kappa_{ij} > s_{ik} - \rho\kappa_{ik}$ because $\epsilon_1 > 0$. □

With $\rho = 0.1$, a model that costs one dollar more per task must score 0.1 higher to be preferred. ID calls cost about one cent or less on average (e.g., Always-Opus spends \$34.02 on 2,919 tasks), so the cost term shifts ID rewards by about 10^{-3} and mainly separates models with near-equal scores. One premium OOD attempt costs about \$1 (Always-Opus spends \$194.91 on 176 tasks), which shifts the reward by about 0.1, comparable to the score differences between models.

B.3 Regret Decomposition

Proposition 3 (Complementarity gap)**.** *Let $j^\star \in \arg\max_j \sum_{i=1}^N R_{ij}$ be the best fixed model on the stream and $G_N = \sum_{i=1}^N \max_j R_{ij} - \sum_{i=1}^N R_{ij^\star}$. For every router π,*

$$\mathrm{CumReg}_N(\pi) \;=\; G_N + \Big(\sum_{i=1}^{N} R_{ij^\star} - \sum_{i=1}^{N} r_i\Big).$$

Moreover $G_N = \min_j \mathrm{CumReg}_N(\textit{Always-}m_j) \ge 0$, and $G_N = 0$ iff a single model attains $\max_j R_{ij}$ on every task of the stream.

Proof. The identity follows by adding and subtracting $\sum_i R_{ij^\star}$ in Eq. 3. The CumReg of Always-m_j is $\sum_i \max_k R_{ik} - \sum_i R_{ij}$, which is minimized at $j = j^\star$ with value G_N; each of its summands $\max_k R_{ik} - R_{ij^\star}$ is nonnegative. If some model k is optimal on every task, then $\sum_i R_{ik} = \sum_i \max_j R_{ij} \ge \sum_i R_{ij^\star} \ge \sum_i R_{ik}$, so $G_N = 0$; conversely, $G_N = 0$ forces every summand to vanish, so $j^\star$ is optimal on every task. □

Corollary 4 (Routers that ignore the task)**.** *Let $\Delta_{\mathcal{D}} = \mathbb{E}_{\mathcal{D}}[\max_j R_j] - \max_j \mu_j \ge 0$. On an i.i.d. stream, every context-free router satisfies*

$$\mathbb{E}\big[\mathrm{CumReg}_N(\pi)\big] \;\ge\; N\,\Delta_{\mathcal{D}} \;\ge\; \mathbb{E}[G_N] \;\ge\; N\,\Delta_{\mathcal{D}} - B\sqrt{N \log M/2}.$$

Hence $\mathbb{E}[G_N]/N \to \Delta_{\mathcal{D}}$, and G_N grows linearly in N in expectation whenever $\Delta_{\mathcal{D}} > 0$. Moreover, $\Delta_{\mathcal{D}} = 0$ iff some model is reward-optimal on $\mathcal{D}$-almost every task. Whenever no model dominates in this sense, every context-free router, including a bandit whose regret against the best fixed model is sublinear, has expected CumReg linear in N.

Proof. For a context-free router, a_i is a function of $\mathcal{H}_{<i}$ and internal randomness, both independent of $(t_i, R_{i\cdot})$. Hence $\mathbb{E}[r_i \mid \mathcal{H}_{<i}] = \sum_j \Pr(a_i = j \mid \mathcal{H}_{<i})\,\mu_j \le \max_j \mu_j$, so $\mathbb{E}[\max_j R_{ij} - r_i] \ge \Delta_{\mathcal{D}}$; summing over i gives the first inequality. For the second, $\mathbb{E}[\max_j \sum_i R_{ij}] \ge \max_j \mathbb{E}[\sum_i R_{ij}] = N \max_j \mu_j$, so $\mathbb{E}[G_N] \le N\,\mathbb{E}[\max_j R_j] - N \max_j \mu_j = N\Delta_{\mathcal{D}}$. Conversely, each $X_j = \sum_i (R_{ij} - \mu_j)$ is a sum of N independent terms in an interval of width B, hence $(B\sqrt{N}/2)$-sub-Gaussian, so $\mathbb{E}[\max_j X_j] \le B\sqrt{N \log M/2}$ and $\mathbb{E}[\max_j \sum_i R_{ij}] \le N \max_j \mu_j + B\sqrt{N \log M/2}$, which gives the last inequality. If $\Delta_{\mathcal{D}} = 0$, then for $j^\star \in \arg\max_j \mu_j$ the nonnegative variable $\max_j R_j - R_{j^\star}$ has mean zero and vanishes almost surely; the converse is immediate. □

B.4 Value of Evidence and Static-Router Regret

Let $\mathcal{G}$ be a σ-algebra that describes what a router knows about the current task when it decides, for instance $\sigma(d(t))$ for the dimension or $\sigma(\phi(t))$ for a task embedding. Define the *information-limited regret*

$$\Delta(\mathcal{G}) \;=\; \mathbb{E}\Big[\max_j R_j - \max_j \mathbb{E}[R_j \mid \mathcal{G}]\Big] \;\ge\; 0,$$

so that $\Delta(\{\emptyset, \Omega\}) = \Delta_{\mathcal{D}}$ for the trivial σ-algebra (Corollary 4) and $\Delta(\sigma(R_{\cdot})) = 0$ when the router observes the whole reward row.

Proposition 5 (Finer evidence lowers the attainable regret)**.** *On an i.i.d. stream, suppose a_i depends on the current task only through $\mathcal{G}$, and otherwise only on $\mathcal{H}_{<i}$ and internal randomness. Then $\mathbb{E}[\mathrm{CumReg}_N(\pi)] \ge N\,\Delta(\mathcal{G})$, with equality for the router that plays $\arg\max_j \mathbb{E}[R_j \mid \mathcal{G}]$ on every task. If $\mathcal{G} \subseteq \mathcal{G}'$, then $\Delta(\mathcal{G}') \le \Delta(\mathcal{G})$.*

Proof. The history and the internal randomness are independent of the current task, so conditioning on them does not change $\mathbb{E}[R_{ij} \mid \mathcal{G}]$, and given $(\mathcal{G}, \mathcal{H}_{<i})$ the decision depends only on randomness independent of $R_{i\cdot}$. Therefore $\mathbb{E}[r_i \mid \mathcal{G}, \mathcal{H}_{<i}] = \sum_j \Pr(a_i = j \mid \mathcal{G}, \mathcal{H}_{<i})\,\mathbb{E}[R_{ij} \mid \mathcal{G}] \le \max_j \mathbb{E}[R_{ij} \mid \mathcal{G}]$. Taking expectations and summing over i gives the bound, and the stated router attains it with equality. For the refinement, the tower property gives $\mathbb{E}[R_j \mid \mathcal{G}] = \mathbb{E}\big[\mathbb{E}[R_j \mid \mathcal{G}'] \mid \mathcal{G}\big] \le \mathbb{E}\big[\max_k \mathbb{E}[R_k \mid \mathcal{G}'] \mid \mathcal{G}\big]$ for every j. Taking the maximum over j and then expectations yields $\mathbb{E}[\max_j \mathbb{E}[R_j \mid \mathcal{G}]] \le \mathbb{E}[\max_k \mathbb{E}[R_k \mid \mathcal{G}']]$, i.e. $\Delta(\mathcal{G}') \le \Delta(\mathcal{G})$. □

Remark 6 (An information hierarchy)**.** The families of Table 2 know nested amounts about the current task: nothing (context-free), the dimension $\sigma(d)$ (DimensionBest), and prompt features or embeddings (kNN, trained routers, and contextual bandits, which refine $\sigma(d)$ whenever the features determine the dimension). Proposition 5 shows that each refinement can only lower the information-limited regret. Whether a router reaches $N\Delta(\mathcal{G})$ depends on how well it estimates $\mathbb{E}[R_j \mid \mathcal{G}]$ from the probing set or from $\mathcal{H}_{<i}$, and this is where static and closed-loop routers differ. The Verifier's verdict under escalation is bought by running a model, so it lies outside this hierarchy and is analyzed in §B.6. The diagnosis of §3.1 is an estimation effect in these terms: correct per-dimension statistics hand the router good estimates of the dimension-level means, and it moves to the DimensionBest level (47.74 vs. 47.50), while shuffled statistics keep $\mathcal{G}$ and the format but attach the values to the wrong models.

Proposition 7 (Static-router regret on a fixed distribution)**.** *Let g be a static router and, for a task distribution Q, let $L_Q(g) = \mathbb{E}_Q[\max_j R_j - R_{g(t)}]$ be its expected per-task regret, with the expectation also taken over the internal randomness of g. On an i.i.d. stream from Q, $\mathbb{E}[\mathrm{CumReg}_N(g)] = N\,L_Q(g)$, so the expected slope of its regret curve does not change with N.*

Proof. Each task is drawn from Q and g is fixed, so each per-task regret $\ell_i = \max_j R_{ij} - R_{i,g(t_i)}$ has expectation $L_Q(g)$. Summing these expectations gives the identity. □

B.5 Memory as a Nonparametric Reward Estimator

Let $\phi(t)$ be the unit-norm task embedding and $\mu_j(x) = \mathbb{E}[R_j \mid \phi(t) = x]$ the mean reward of model j at embedding x. For unit vectors, cosine similarity $\langle x, x' \rangle \ge \tau$ is equivalent to $\|x - x'\|_2 \le \sqrt{2(1-\tau)}$, so Memory's retrieval threshold $\tau = 0.5$ restricts neighbors to Euclidean radius 1. Memory holds records $r = 1, \ldots, n$ in the order they were written, with embedding x_r, chosen

model a_r, and stored reward y_r. Because the loop is closed, a_r may depend on all earlier stored rewards. Let $\mathcal{F}_r$ be generated by the embeddings of all records and by $(a_1, y_1, \ldots, a_r, y_r)$.

Assumption 8. (i) *Smoothness*: each μ_j is L_μ-Lipschitz on the unit sphere. (ii) *Noise*: $y_r = \mu_{a_r}(x_r) + \beta_r + \eta_r$, where the bias β_r is determined by $(\mathcal{F}_{r-1}, a_r)$ and satisfies $|\beta_r| \le b_{a_r}$, and η_r is zero-mean and σ-sub-Gaussian conditional on $(\mathcal{F}_{r-1}, a_r)$.

Stored rewards lie in an interval of width B, so by the conditional Hoeffding lemma $y_r - \mathbb{E}[y_r \mid \mathcal{F}_{r-1}, a_r]$ is $(B/2)$-sub-Gaussian, and (ii) holds with $\sigma = B/2$ once β_r absorbs the gap between this conditional mean and $\mu_{a_r}(x_r)$. An exact verifier has $b_j = 0$ when a_r depends on the current task only through x_r; if a_r also reads the prompt or dimension of task r, β_r also absorbs the resulting selection bias.

Proposition 9 (Memory estimation error)**.** *Under Assumption 8, fix a query embedding x. For each model j, let $h_j(x)$ be a retrieval radius, possibly chosen from the data (e.g. the distance to the k-th nearest record of model j), and let $\hat{\mu}_j(x)$ be the mean stored reward over the $k_j \ge 1$ records with $a_r = j$ and $\|x_r - x\| \le h_j(x)$. With probability at least $1 - \delta$, simultaneously for all $j \in [M]$,*

$$\big|\hat{\mu}_j(x) - \mu_j(x)\big| \;\le\; L_\mu\, h_j(x) + b_j + \sigma\sqrt{\frac{2\log(2Mn^2/\delta)}{k_j}} \;=:\; e_j(x).$$

Consequently, on this event, if the best model $j^*(x)$ at x beats every other model j by more than $e_{j^*}(x) + e_j(x)$, the Memory decision $\arg\max_j \hat{\mu}_j(x)$ is optimal given $\phi(t) = x$.

Proof. Write $\hat{\mu}_j(x) - \mu_j(x) = \frac{1}{k_j}\sum_r \big(\mu_j(x_r) - \mu_j(x) + \beta_r\big) + \frac{1}{k_j}\sum_r \eta_r$, where the sums run over the selected records. The first term is at most $L_\mu h_j(x) + b_j$ in absolute value by (i) and (ii). For the second, condition on all embeddings; the distances $\|x_r - x\|$ then take at most n values, so it suffices to consider n radii h. For a fixed h and model j, the indicators $w_r = \mathbf{1}\{a_r = j,\ \|x_r - x\| \le h\}$ are determined by $(\mathcal{F}_{r-1}, a_r)$. By (ii) and optional skipping, the sum of η_r over the first m records with $w_r = 1$ is $\sigma\sqrt{m}$-sub-Gaussian and exceeds $\sigma\sqrt{2m\log(2Mn^2/\delta)}$ in absolute value with probability at most $\delta/(Mn^2)$. The selected records are those with $w_r = 1$ at $h = h_j(x)$, and there are k_j of them, so a union bound over $m \in [n]$, the n radii, and the M models gives the first claim. On this event, $\hat{\mu}_{j^*}(x) \ge \mu_{j^*}(x) - e_{j^*}(x) > \mu_j(x) + e_j(x) \ge \hat{\mu}_j(x)$ for every $j \ne j^*$. □

Remark 10 (Task-level versus dimension-level evidence)**.** Dimension-level statistics estimate $\mathbb{E}[R_j \mid d]$, whose gap to $\mu_j(x)$ can be as large as the within-dimension variation $\sup_{x,x' \in d} |\mu_j(x) - \mu_j(x')|$ and does not shrink as data accumulate. With the fixed retrieval size $k = 10$, Memory's variance term stays of order $\sigma k_j^{-1/2}$, while its bias $L_\mu h_j(x)$ shrinks as the stream fills the neighborhood of x with records of model j. The FIFO bound on the store (20K records) limits how long outdated outcomes persist under drift, and the bias term b_j makes the Verifier's accuracy part of Memory's error.

B.6 Verification-Driven Escalation

Consider binary outcomes, as on the OOD split. For one task, let $Y_\ell \in \{0, 1\}$ indicate that model ℓ's attempt is correct, $V_\ell \in \{0, 1\}$ that the Verifier accepts it, and κ_ℓ its cost; in the replay these are defined for every model whether or not a chain runs it. An *accept-or-continue chain* over models $1, \ldots, L$ runs them in order, returns the first accepted attempt among $1, \ldots, L-1$, and returns attempt L if none of them is accepted.

Proposition 11 (Chain versus the per-task oracle over the chain)**.** *Let $Y^{\max} = \max_{\ell \le L} Y_\ell$, and let Y^{ch} and κ^{ch} be the chain's outcome and total cost. Then*

$$\Pr\big(Y^{\mathrm{ch}} = 1\big) \;\ge\; \Pr\big(Y^{\max} = 1\big) - \sum_{\ell=1}^{L-1} \Pr\big(V_\ell \ne Y_\ell\big),$$

$$\mathbb{E}\big[\kappa^{\mathrm{ch}}\big] \;=\; \sum_{\ell=1}^{L} \mathbb{E}\big[\kappa_\ell\, \mathbf{1}\{V_1 = \cdots = V_{\ell-1} = 0\}\big].$$

In particular, with an exact verifier ($V_\ell = Y_\ell$ for all $\ell < L$), $Y^{\mathrm{ch}} = Y^{\max}$ on every task: the chain resolves exactly the tasks that some model in the chain resolves, in any order, and the order

affects only the cost. If model L runs only on a gate event E in addition to the rejection of attempts $1, \dots, L-1$ (for ACRouter, at least k of them produced an applicable patch), the first bound holds after also subtracting $\Pr(E^c,\, Y_1 = \cdots = Y_{L-1} = 0,\, Y_L = 1)$; with an exact verifier this probability is the whole gap $\Pr(Y^{\max} = 1) - \Pr(Y^{\mathrm{ch}} = 1)$.

Proof. Suppose $Y^{\max} = 1$, let ℓ^* be the first index with $Y_{\ell^*} = 1$, and suppose $V_\ell = Y_\ell$ for all $\ell \le \min(\ell^*, L-1)$. Then every attempt before ℓ^* is correctly rejected. If $\ell^* < L$, attempt ℓ^* is accepted and returned; if $\ell^* = L$, the chain reaches and returns attempt L. Either way $Y^{\mathrm{ch}} = 1$. Hence $\{Y^{\max} = 1,\, Y^{\mathrm{ch}} = 0\} \subseteq \bigcup_{\ell<L}\{V_\ell \neq Y_\ell\}$, and a union bound gives $\Pr(Y^{\mathrm{ch}} = 1) \ge \Pr(Y^{\max} = 1) - \sum_{\ell<L}\Pr(V_\ell \neq Y_\ell)$. The chain returns one of its own attempts, so $Y^{\mathrm{ch}} \le Y^{\max}$, which gives equality for an exact verifier. Model ℓ is run iff attempts $1, \dots, \ell-1$ are all rejected, which gives the cost identity. With a gate, the argument above fails additionally only when $\ell^* = L$ and E does not occur, i.e. on $\{E^c,\, Y_{<L} = 0,\, Y_L = 1\}$, and adding this event to the union gives the gated bound. With an exact verifier and $Y^{\max} = 1$, the chain fails exactly on this event, since every attempt before ℓ^* is then correctly rejected. □

Proposition 12 (When escalation beats the premium model)**.** *For a two-model chain (a cheap model c, then a premium model p) with binary outcomes, the expected reward difference between the chain and Always-p is*

$$\begin{aligned}\mathbb{E}\big[r^{\mathrm{ch}}\big] - \mathbb{E}[r_p] \;=\; &\epsilon_1 \Pr(V_c = 1)\big[\Pr(Y_c = 1 \mid V_c = 1) - \Pr(Y_p = 1 \mid V_c = 1)\big] \\ &+ |\epsilon_2|\big(\mathbb{E}\big[\kappa_p \mathbf{1}\{V_c = 1\}\big] - \mathbb{E}[\kappa_c]\big).\end{aligned}$$

Proof. The chain returns c's attempt when $V_c = 1$ and p's otherwise, so its score is $V_cY_c + (1-V_c)Y_p$ and its cost is $\kappa_c + (1-V_c)\kappa_p$. Subtracting the score Y_p and cost κ_p of Always-p gives a score difference $V_c(Y_c - Y_p)$ and a cost difference $\kappa_c - V_c\kappa_p$. Taking expectations with $\epsilon_2 = -|\epsilon_2|$ gives the identity. □

When $\Pr(V_c = 1) > 0$, the first term rewards precision: on the tasks where the cheap attempt is accepted, the chain gains score iff the Verifier's precision $\Pr(Y_c = 1 \mid V_c = 1)$ exceeds the premium model's success rate on those tasks, and with an exact verifier the bracket equals $\Pr(Y_p = 0 \mid V_c = 1) \ge 0$. The second term is the premium spend avoided on accepted tasks minus the cheap attempt's cost. For ACRouter's gate $E = \{A \ge k\}$, with A the number of applicable patches among attempts $1, \dots, L-1$, the event E^c grows with k, so with an exact verifier the resolve rate cannot increase with k, while the premium model runs less often and the cost falls. Table 5 shows both trends (k=1: 77.84%, \$144.7; k=2: 73.30%, \$86.72; k=3: 72.16%, \$61.6).

B.7 Corrupted Feedback

In the degraded-feedback test (Table 4b), each stored score is replaced by $\tilde{s} = 1 - s$ with probability p, independently of everything else; the flip acts on the score only.

Proposition 13 (Flip noise)**.** *For $s \in [0,1]$, $\mathbb{E}[\tilde{s} \mid s] = (1-2p)\, s + p$. Hence, for any two models, $\mathbb{E}[\tilde{s}_j] - \mathbb{E}[\tilde{s}_k] = (1-2p)\big(\mathbb{E}[s_j] - \mathbb{E}[s_k]\big)$, and the corrupted reward $\tilde{r} = \epsilon_1\tilde{s} + \epsilon_2\kappa$ satisfies*

$$\mathbb{E}[\tilde{r}_j] - \mathbb{E}[\tilde{r}_k] \;=\; \epsilon_1(1-2p)\big(\mathbb{E}[s_j] - \mathbb{E}[s_k]\big) + \epsilon_2\big(\mathbb{E}[\kappa_j] - \mathbb{E}[\kappa_k]\big).$$

Consequently: (i) for $p < \frac{1}{2}$ the expected-score ranking of models is preserved, and at $p = \frac{1}{2}$ it is erased; (ii) a Hoeffding test that separates two models at a fixed confidence needs $(1-2p)^{-2}$ times as many observations as without noise; (iii) for $p < \frac{1}{2}$ the corrupted reward ranks models as the clean reward would with exchange rate $\rho/(1-2p)$ in place of ρ.

Proof. $\mathbb{E}[\tilde{s} \mid s] = (1-p)\, s + p\,(1-s)$, and the differences follow by linearity. Corrupted scores stay in $[0,1]$, so a Hoeffding interval of half-width u needs a number of samples proportional to u^{-2}; the gap to be resolved is scaled by $1-2p$, which gives (ii). For (iii), divide the reward difference by $\epsilon_1(1-2p) > 0$ and compare with Lemma 2(ii). □

In Table 4b the loss grows with p (-0.80, -2.14, -4.33 points at $p = 0.1, 0.3, 0.5$), as the shrinking score gaps of part (ii) suggest. At $p = \frac{1}{2}$ the stored scores no longer indicate which model scores higher, and the loop falls below the no-feedback condition (44.41 vs. 47.32), consistent with Memory and the dimension posterior then absorbing noise instead of evidence. Part (iii) also implies that score noise tilts a cost-aware router toward cheaper models.

B.8 Summary: Statements and Their Empirical Counterparts

Table 6: Selected formal statements and their empirical counterparts.

Statement	**Empirical counterpart**
Lemma 2 (exchange rate)	The cost term is a tie-breaker on ID and decisive on OOD.
Prop. 3, Cor. 4 (complementarity gap)	On ID, Always-Opus has the highest AvgPerf among single models (Table 3) and a cost term of only 3.4, so it is the best fixed model and $G_N = 387.1$; by Eq. 4, ACRouter's CumReg of 205.5 means it beats the best fixed model by 181.6.
Prop. 5 (value of evidence)	Correct vs. shuffled statistics (Table 1) keep $\mathcal{G}$ fixed and change the estimate of $\mathbb{E}[R_j \mid \mathcal{G}]$.
Prop. 9 (Memory error)	Online-kNN loop with vs. without feedback (48.74 vs. 47.32, Table 4b).
Props. 11–12 (escalation)	The escalation rows of Table 5: resolve rate and cost fall together as the gate threshold k grows.
Prop. 13 (flip noise)	Loss that grows with the flip rate p in Table 4b.

C Benchmark and Setup Details

This section consolidates the construction details of CodeRouterBench: the coding dimensions and their sources (§C.1), the deterministic 70/30 split protocol (§C.2), the API pricing and self-hosted serving rate used for every dollar number reported (§C.3), and the prompt templates used by the LLM-as-a-Router baselines and the +Perf stats ablation (§C.4).

C.1 Coding Dimensions

Table 7 lists the ten dimensions, their sources, and how each is scored.

Table 7: Coding dimensions in CodeRouterBench. The 9 single-turn dimensions each contain 1,111 tasks (70/30 deterministic split into probing and in-distribution test, see Section C.2). The 10th dimension, agentic programming, is held out as the OOD test (176 tasks). Eval: Exec = sandboxed execution against tests (pass@1, or the fraction of tests passed); Match = match against the reference output or fix, with partial credit; BLEU = BLEU against the reference summary; Sandbox = Docker-based patch application and test execution.

Dimension	**Description**	**Source(s)**	**Eval**
Code Generation	Function-level synthesis	HumanEval+, MBPP+ (Liu et al., 2023), BigCodeBench	Exec
Algorithm Design	Competitive programming	LiveCodeBench, BigCodeBench	Exec
Bug Fixing	Locate and repair defects	DebugBench (Tian et al., 2024), SWE-bench Lite	Match
Code Completion	Output prediction, prefix completion	CRUXEval; HumanEval+, MBPP+, BigCodeBench variants	Match / Exec
Code Refactoring	Rewrite code to a reference edit	Bugs2Fix, CanItEdit	Match / Exec
Data Science	Data analysis pipelines	DS-1000, BigCodeBench	Exec
Multi-Language	Cross-language tasks	HumanEval-X, MultiPL-E	Exec
Code Understanding	Explain & summarize code	CodeXGLUE Summarization	BLEU
Test Generation	Generate test suites	LiveCodeBench, HumanEval+, MBPP+ variants	Exec
Agentic Programming (OOD)	Long-horizon, multi-file repository tasks	SWE-bench Verified, LongCLI-Bench, FeatureBench, SWE-CI	Sandbox

C.2 Data Split and Evaluation Levels

The 9 single-turn dimensions are split deterministically (seed `coding-router-v1`) into three roles that mirror a typical router development workflow:

- **Probing set** (train 60% + val 10% = 7,080 tasks across 9 single-turn dimensions): Used to *develop* routers — profile model strengths per dimension, train classifiers, calibrate

DimensionBest, and warm-start ACRouter's Memory module (200 val tasks). This is the dev split for building custom routers.

- **In-distribution test** (test 30% = 2,919 tasks across 9 single-turn dimensions): Held-out evaluation in a controlled environment with per-task per-model scores. All in-distribution (ID) numbers in Table 3 (left) use this split. The ID per-dim task counts (306–347 per dimension) sum to 2,919.
- **OOD test** (Real-world agentic programming, 176 tasks): Held out as the agentic-programming dimension. No router has access to any ground-truth data from this dimension. Tests whether a router developed on the probing set generalizes to fundamentally different task types. Tasks are extracted from SWE-bench Verified (OpenAI, 2024), LongCLI-Bench (Feng et al., 2026), FeatureBench (Zhou et al., 2026), and SWE-CI (Chen et al., 2026), filtering out tasks with high prompt similarity to the probing set. Evaluation uses the SWE-Bench Docker harness (Jimenez et al., 2024) with mini-swe-agent, capped at 40 steps per task.

C.3 Model Pricing

Backend tokens are priced at official API rates (§C.3.1); self-hosted router-side tokens use a measured H100-amortised rate of $0.054/M (§C.3.2).

C.3.1 API Pricing

Table 8 gives the per-model API prices used for all in-distribution costs. OOD agent runs are costed at the providers' CNY prices converted to USD for the five models served by Chinese providers (released with the artifact). Every backend cost is computed from the logged input and output tokens, except on the 64 added OOD tasks (Appendix E.5). Memory's voyage-code-3 embeddings ($0.18/M tokens) add about $0.10 on the ID test stream and $0.07 on OOD.

Table 8: Per-model API pricing in USD per million tokens (input/output asymmetric), used for all in-distribution costs. The 8 models are partitioned into tiers (premium / high / mid / low) ordered by their geometric mean of input and output prices.

Model	$/M input	$/M output	Tier
Claude Opus 4.6	$5.00	$25.00	premium
Claude Sonnet 4.6	$3.00	$15.00	high
GPT-5.4	$2.50	$15.00	high
Qwen3-Max	$1.20	$6.00	mid
GLM-5	$0.88	$3.22	mid
Kimi-K2.5	$0.60	$3.07	mid
Qwen3.5-Plus	$0.40	$2.40	low
MiniMax-M2.7	$0.30	$1.20	low

C.3.2 Local Serving Token Cost Calculation

Self-hosted routers (the orchestrator LLM in ACRouter and the LoRA-finetuned Qwen3.5 routers) do not incur a per-token API charge. We instead amortise GPU rental cost over the measured serving throughput, yielding a single combined-throughput price in USD per million tokens. All numbers below are reproducible from the artifact released alongside the paper (`local_pricing_benchmark/`).

Hardware and pricing. Throughput is measured on a single NVIDIA H100 80GB HBM3. We adopt a representative on-demand H100 rental rate of $6.88/GPU-hour as the cost basis (CoreWeave[2] / Lambda Cloud[3] public pricing at the time of writing).

Workload. We replay the 2,919-task in-distribution test split as router queries: each input is the actual zero-shot router prompt (system message + 8-model description + the task's dimension and

[2]https://www.coreweave.com/pricing
[3]https://lambda.ai/pricing

prompt, truncated to 2,000 characters), and the model is asked to emit a JSON model-pick. This is the routing call that the finetuned Qwen routers make in our experiments, so the measured throughput is their router-side load.

Configuration. We use the offline engine of SGLang Zheng et al. (2024) (no HTTP overhead) running Qwen3.5-0.8B in bfloat16 with flashInfer attention. Decoding is greedy (T=0) with `max_new_tokens`= 96 and a stop sequence on three consecutive newlines, matching the JSON-structured output expected by the router parser. Requests are issued in batches of 64 and looped over the 2,919 prompts in a fixed seed-42 shuffle. We discard the first 32 warm-up prompts (kernel compilation, KV-cache pre-allocation) and then time exactly 300.5 seconds of steady-state generation.

Measured throughput. Over the 300.5-second window the engine processed 22,848 requests, consuming 8,393,247 input tokens and producing 2,152,601 output tokens, for a combined sustained throughput of:

$$\mathrm{TPS}_{\text{in+out}} = \frac{8{,}393{,}247 + 2{,}152{,}601}{300.5\,\text{s}} \approx 35{,}094 \text{ tokens/s}. \tag{5}$$

Splitting the throughput by direction yields 27,931 input-tokens/s and 7,163 output-tokens/s; the routing prompt is input-heavy (4:1 input:output) because generation is capped at 96 tokens.

Derived per-token cost. Multiplying by 3,600 yields $\sim 1.263 \times 10^8$ tokens per GPU-hour, so the rental cost per million combined tokens is:

$$\frac{\$6.88}{1\text{ hour}} \cdot \frac{1\text{ hour}}{1.263 \times 10^8 \text{ tokens}} \cdot 10^6 = \boxed{\$0.054 \text{ per 1M tokens}}. \tag{6}$$

This is the rate used for self-hosted router-side tokens of ACRouter and the Finetuned Qwen3.5-0.8B router throughout the paper. Backend model tokens are always priced at API rates (Table 8); only the orchestrator/router LLM that we run ourselves is amortized this way.

Reproducibility. The exact benchmark script, prompt-construction code, and resulting JSON (including per-15-second progress logs) will be released.

C.4 Router Prompt Templates

Three prompt variants are explored for the supplementary experiments in LLM routers: a zero-shot Vanilla template (§C.4.1, used in the main experiments for trained policy), a 3-shot in-context variant (§C.4.2), and a +Perf stats ablation that injects probing-set per-dimension scores (§C.4.3).

C.4.1 Zero-Shot Router Prompt

The zero-shot LLM router (Vanilla in Table 1) receives the following system prompt:

```
You are a coding task router. Your objective is to maximize the
performance-cost trade-off: choose the model that achieves the
best quality for its cost on this task. [8 models listed with
capability descriptions, sorted by cost tier]. Prefer cheaper models
when quality is comparable. Respond with JSON: {"model": "...",
"reasoning": "..."}.
```

Figure 6: Prompt template for zero-shot router.

C.4.2 Few-Shot Router Prompt

The 3-shot LLM router uses the same system prompt as the zero-shot variant, but prepends three demonstration examples before the target task. The prompt structure is:

Example selection strategy. Examples are drawn from the probing split with the following design choices:

```
## Examples
### Example 1
- Dimension: bug_fixing
- Difficulty: medium
- Language: python
- Prompt: <first 300 chars of task prompt>...
- Best model: glm-5
- Scores: claude-opus-4-6: 0.72, claude-sonnet-4-6: 0.70, glm-5:
0.73, gpt-5.4: 0.68, kimi-k2.5: 0.65, ...
### Example 2
[...similar format...]
### Example 3
[...similar format...]
--
Now route the following task:
## Task to Route
**Dimension**: code_completion
**Difficulty**: hard
**Language**: python
**Prompt**: <full task prompt>
```

Figure 7: Few-shot prompt template for model routing.

1. **Same-dimension priority**: all 3 examples are sampled from tasks sharing the same dimension as the target task. If fewer than 3 same-dimension examples are available, the remaining slots are filled from other dimensions.
2. **Non-trivial examples only**: tasks where all 8 models achieve identical scores are excluded, since they carry no routing signal.
3. **Oracle labels**: each example shows the oracle-best model (the model with the highest score for that task, with ties broken by cost ascending then alphabetical) along with all 8 models' scores, giving the LLM both the answer and the full score distribution.
4. **Prompt truncation**: task prompts in examples are truncated to 300 characters to control input token cost; the target task's prompt is included in full.
5. **Fixed seed**: examples are sampled with a fixed random seed (42) for reproducibility.

Response format. The LLM is instructed to respond with a JSON object: `{"model": "<name>", "reasoning": "<explanation>"}`. Responses are parsed with a multi-strategy fallback: (1) direct JSON parse, (2) regex extraction from markdown code blocks, (3) model-name string matching, (4) fallback to the default model (`claude-sonnet-4-6`).

C.4.3 Performance-Statistics Ablation Prompt

In the +Perf stats ablation (Table 1), model descriptions are replaced with tabular performance data drawn from the probing set:

```
Model capabilities (average scores per dimension): Claude Opus
4.6: code_gen=0.315, algo=0.254, bug_fix=0.717, completion=0.860,
refac=0.607, ds=0.142, multi=0.408, underst=0.193, test_gen=0.392.
[... similar for all 8 models]
```

Figure 8: Supplied tabular performance information in +Perf setting of ablation study.

The Anonymized variant additionally rewrites every model identifier to Model-A through Model-H to remove any prior the LLM may carry about specific provider strengths.

Table 9: Router-model comparison on the 2,919-task in-distribution test split. Dollar columns use the pricing in Table 8; CumReg uses the per-task cost-aware oracle with $r = s - 0.1\,\kappa_{\text{USD}}$. **Panel A**: LLM-as-a-Router with all 8 models in 0-shot and 3-shot settings. **Panel B**: Agent online — each agent starts with empty memory, warms up on 200 val tasks, then routes the test stream while accumulating cosine-kNN experience under verifier feedback (seed=42). Rows ordered by AvgPerf within each panel.

Mode	Router LLM	AvgPerf%	CumReg↓	$Total	Perf/$↑
Panel A: LLM-as-a-Router (0-shot)					
0-shot	Qwen3.5-Plus	46.87	296.3	$13.06	3.59
0-shot	Qwen3-Max	46.60	304.1	$11.82	3.94
0-shot	Kimi-K2.5	46.29	313.3	$14.11	3.28
0-shot	GPT-5.4	45.91	324.0	$10.26	4.48
0-shot	GLM-5	45.42	338.7	$14.56	3.12
0-shot	MiniMax-M2.7	42.11	435.6	$16.90	2.49
0-shot	Claude Sonnet 4.6	41.41	456.4	$21.02	1.97
0-shot	Claude Opus 4.6	39.27	518.9	$21.20	1.85
Panel A (cont.): LLM-as-a-Router (3-shot)					
3-shot	GLM-5	46.03	320.8	$12.22	3.77
3-shot	Qwen3-Max	45.80	327.7	$14.63	3.13
3-shot	GPT-5.4	45.66	331.3	$9.80	4.66
3-shot	Claude Sonnet 4.6	45.53	335.3	$12.01	3.79
3-shot	Qwen3.5-Plus	45.39	339.5	$12.64	3.59
3-shot	MiniMax-M2.7	45.29	342.5	$13.88	3.26
3-shot	Kimi-K2.5	45.10	348.5	$17.98	2.51
3-shot	Claude Opus 4.6	41.17	463.5	$21.34	1.93
Panel B: Agent online (200-task warm-up, seed=42)					
Agent	MiniMax-M2.7	45.27	343.2	$15.30	2.96
Agent	GLM-5	45.21	345.0	$15.84	2.85
Agent	Qwen3-Max	44.92	353.7	$18.00	2.50
Agent	Qwen3.5-Plus	43.98	381.0	$16.17	2.72
Agent	Claude Sonnet 4.6	43.65	390.7	$16.84	2.59
Agent	Kimi-K2.5	42.70	417.9	$12.45	3.43
Agent	GPT-5.4	40.95	468.9	$11.24	3.64
Reference	*Random*	*38.75*	*533.6*	*$15.64*	*2.48*

D Supplementary Experiments

This section reports two experimental studies that complement the main results: an LLM-as-a-Router sweep over all 8 candidate models in 0-shot, 3-shot, and online-agent modes (§D.1); and a Qwen-router parameter-scaling sweep that isolates the contribution of router capacity (§D.2).

D.1 LLM-as-a-Router Across All 8 Models

D.1.1 Setup

To test whether the information-deficit diagnosis in Table 1 extends beyond Claude Sonnet 4.6, we evaluate every one of the candidate models in the role of an LLM-as-a-Router on the in-distribution test split. Each model is given the same Vanilla zero-shot or 3-shot prompt template (Section C.4) and instructed to pick a backend model for each task. Costs use the pricing in Table 8 for backend tokens; the LLM router's own tokens are priced at the same API rates as its backend role (since these are API-served, not self-hosted). For **Panel B** the 0-shot router is augmented with a cosine-kNN Memory of 200 warm-up tasks and online-updated as the test stream proceeds (seed 42).

D.1.2 Results and Findings

Three findings stand out from Table 9. First, *coding ability ≠ routing ability*: all 8 LLM routers remain below DimensionBest, and Claude Opus 4.6 ranks last in both 0-shot and 3-shot despite being the strongest individual coder. Second, few-shot prompting does not reliably solve routing: it slightly improves some router LLMs but still leaves substantial regret to the per-task oracle. Third, agent-mode online Memory is mixed across router LLMs; it helps MiniMax-M2.7 but does not uniformly improve stronger prompt-following routers. Placing retrieved experience in an LLM prompt is therefore not enough by itself, which is why ACRouter aggregates verified rewards numerically in its Memory voter (§3.3) instead. Random is reported as a reference.

D.2 Qwen Router Scaling Sweep

D.2.1 Setup

We sweep five Qwen3.5 sizes (0.8B / 2B / 4B / 9B / 27B) under unified LoRA finetuning (**FT v4**: attention+MLP, r=16, α=32, dropout 0.05, identical across sizes for a controlled scaling curve), evaluated on the canonical n=2,919 in-distribution test split. Qwen3.5-0.8B-Finetuned is the variant reported in the Trained-Policy block of Table 3; the larger sizes measure the effect of router capacity.

D.2.2 Results and Findings

Table 10: Qwen router scaling sweep on the canonical n=2,919 in-distribution test split (FT v4 LoRA, attention+MLP, r=16, α=32). **Gap%**: relative gap to the per-task oracle, $(57.00 - \text{AvgPerf})/57.00$. **Toks**: average input+output tokens per task (router side). **CumReg**: per-task cost-aware regret; it is reported only where archived per-task decisions are available. The 0.8B row is the Qwen3.5-0.8B-Finetuned entry in Table 3.

Size	AvgPerf%↑	Gap%↓	Toks/task	CumReg↓
0.8B	46.41	18.6	500	309.1
2B	46.69	18.1	501	–
4B	46.21	18.9	492	–
9B	46.56	18.3	496	–
27B	**46.74**	**18.0**	495	–

Findings.

- **Scale yields diminishing returns under FT v4.** Across a $\sim 30\times$ parameter range, AvgPerf moves only ~ 0.5 points (46.21−46.74). The routing signal saturates well below 27B; beyond basic capacity, scaling does not substitute for decision evidence (Table 1b).
- **0.8B is the cost-efficient choice.** Because larger sizes leave AvgPerf essentially unchanged, we report 0.8B in Table 3 as the most cost-efficient finetuned policy, and treat the larger sizes as scaling controls.

E Supplementary Analyses

This section unpacks the in-distribution and OOD numbers in Table 3 along several diagnostic axes: per-model capability profiles (§E.1), the bias of a vanilla LLM router (§E.2), the dollar-cost regime decomposition (§E.3), and the per-method breakdowns of both the in-distribution result (§E.4) and the OOD result (§E.5), followed by the robustness of the streaming evaluation (§E.6). All numbers come from the released results bundle and match Table 3.

E.1 Model Capability Profiles

To visualize how model strengths differ across the 9 single-turn coding dimensions, Figure 9 presents radar charts for representative models. Each axis represents one dimension; the radial distance is

the model's test-split score on that dimension, min–max scaled per axis. The shapes are distinctly non-circular: Claude Opus excels on code completion and bug fixing but underperforms on algorithm design, where Qwen3.5-Plus and Qwen3-Max lead. Qwen3-Max dominates test generation but is average elsewhere. These complementary profiles are precisely what routing exploits: no single model is best everywhere, and a dimension-aware router can select the right specialist for each task type.

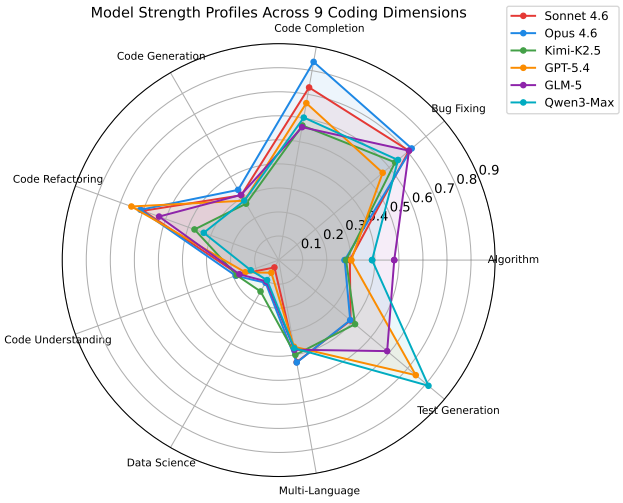

Figure 9: Radar chart of representative models across 9 coding dimensions. Each model has a distinct strength profile, confirming the model complementarity that routing exploits. Axes are scaled per-dimension (0 = worst model, 1 = best model on that dimension).

E.2 Router Bias Distribution

Figure 10 compares how often each model is selected by the per-task Oracle, DimensionBest, and the 0-shot (Vanilla in Table 1) and 3-shot LLM routers. The 0-shot router picks Sonnet 4.6 for about 31% of tasks (Oracle: about 9%) and almost never picks Qwen3-Max, GLM-5, or MiniMax-M2.7, which together are the Oracle's choice on about 27% of tasks; 3-shot demonstrations spread its choices but still over-select Sonnet 4.6. The router thus follows a prior about the models rather than the dimensional structure that DimensionBest exploits. This matches the +Perf stats result (Table 1a): providing the per-dimension performance prior lets the LLM router match (and slightly exceed) DimensionBest with no architectural change, which points to missing decision evidence as a major bottleneck for the evaluated routers (Table 1b).

E.3 Cost Comparison

This subsection decomposes the dollar-cost columns of Table 3 into three deployment-cost regimes. Costs are computed using the per-model asymmetric input/output pricing in Table 8 and include router-side overhead where applicable (priced at \$0.054/M for self-hosted policies, see §C.3.2).

Three regimes emerge consistent with Table 3. The *cheap* tier (trained classifiers \$6.8–7.7: LogReg \$7.54, RouteLLM-MF \$7.46, TF-IDF+MLP \$7.69, RouteLLM-BERT \$7.59, Qwen3.5-0.8B-Finetuned \$6.81) attains near-DimensionBest performance at minimal cost: these methods add little router token overhead and tend to route a substantial share of tasks to mid-tier models (Kimi-K2.5, Qwen3-Max). The *moderate* tier (DimensionBest \$12.89, ACRouter \$13.21, online bandits \$10–11) pays for an informed mix of expensive and cheap backend models in exchange for higher per-task quality; ACRouter additionally pays for its kNN Memory and Verifier overhead. The *costly-default* tier (Always-Opus \$34.02, Always-Qwen3.5+ \$18.14, Random \$15.64) commits to a costly default. Always-Opus is not cost-efficient on ID: it spends \$34.02 for 43.83% AvgPerf, while ACRouter achieves 49.98% AvgPerf for \$13.21. The cheapest single-model option, Always-Kimi-K2.5 at \$2.90,

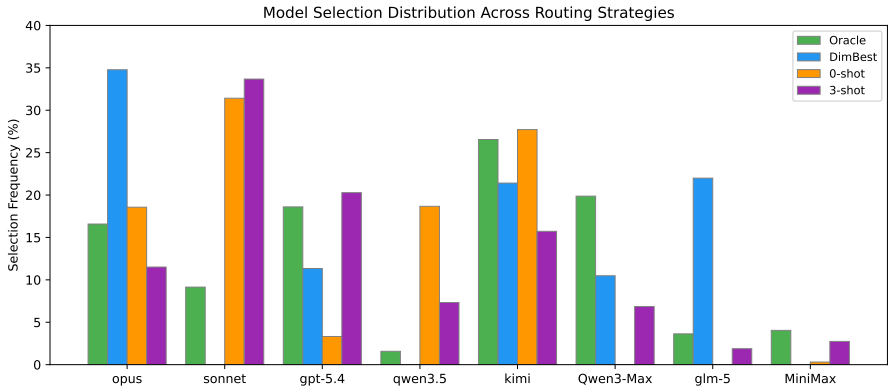

Figure 10: Router bias analysis: selection frequency of each model over the 2,919 ID test tasks for the per-task Oracle, DimensionBest, and the 0-shot (Vanilla) and 3-shot LLM routers. The 0-shot router over-selects Sonnet 4.6 and almost never selects Qwen3-Max, GLM-5, or MiniMax-M2.7.

sets the cost floor with 36.66% AvgPerf (Perf/$ 12.62, the highest in Table 3), at a substantial AvgPerf cost compared with ACRouter.

E.4 In-distribution Per-Method Breakdown

Table 11 reports the per-method breakdown on the in-distribution test (n=2,919 across 9 single-turn coding dimensions) corresponding to the left-side block of Table 3, with the dollar-cost column made explicit. AvgPerf% is the average performance of execution-graded scores across dimensions, and CumReg uses the per-task cost-aware reward $r = s - 0.1\,\kappa_{\text{USD}}$ against the per-task oracle.

Table 11: Per-method breakdown on the in-distribution test (2,919 tasks across 9 single-turn coding dimensions). Costs include both backend and router-side tokens (self-hosted at $0.054/M, API at Table 8 rates). CumReg uses the cost-aware reward $r = s - 0.1\,\kappa_{\text{USD}}$.

Family	Router	AvgPerf%↑	CumReg↓	$Total	Perf/$↑
Bound	Oracle (cost-aware)	57.00	0	$6.95	8.20
Agentic	ACRouter (ours)	**49.98**	**205.5**	$13.21	3.78
Bandit	LinTS (5-seed avg)	46.48	307.4	$10.35	4.49
Bandit	LinUCB (5-seed avg)	46.84	296.9	$10.69	4.38
Heuristic	DimensionBest	47.50	278.1	$12.89	3.69
Heuristic	kNN Retrieval	47.18	286.7	$7.77	6.07
Trained	LogReg	47.26	284.4	$7.54	6.27
Trained	RouteLLM-BERT	47.22	285.5	$7.59	6.22
Trained	TF-IDF+MLP	46.97	292.8	$7.69	6.11
Trained	Qwen3.5-0.8B-Finetuned	46.41	309.1	$6.81	6.82
Trained	RouteLLM-MF	46.16	316.5	$7.46	6.19
Single-Model	Always-Opus 4.6	43.83	387.1	$34.02	1.29
Single-Model	Always-Kimi-K2.5	36.66	593.3	$2.90	**12.62**
Single-Model	Always-Qwen3.5-Plus	37.16	580.2	$18.14	2.05
Single-Model	Random	38.75	533.6	$15.64	2.48

Three patterns stand out. ACRouter's 49.98% AvgPerf and 205.5 CumReg are the best of every non-oracle family on this in-distribution split, beating DimensionBest's full dimension-level prior by 2.48 AvgPerf points at $13.21 vs. $12.89. Trained classifiers cluster near DimensionBest in AvgPerf (46.16−47.26%) but at roughly half the cost ($6.81−$7.69), giving them the strongest Perf/$ among non-oracle methods that exceed 46% AvgPerf. Always-Opus pays the most ($34.02)

yet trails ACRouter by 6.15 AvgPerf points and accumulates 387.1 regret, illustrating that committing to the strongest single model is not cost-efficient even on the controlled split.

E.5 OOD Per-Method Breakdown

OOD set composition. The OOD split has 176 tasks: 112 from SWE-bench Verified (the original release) and 64 added after the first release from FeatureBench (49), LongCLI-Bench (14), and SWE-CI (1). All models run under mini-swe-agent with a 40-step cap; patches are graded in Docker. On the 112 SWE-bench tasks, runs that did not start (406 of the 896 task–model cells, mostly for the five cheaper models) count as unresolved at zero cost. On the 64 added tasks, four columns come from newer provider versions (GPT-5.4 at medium reasoning effort, Kimi-K2.6, MiniMax-M2.5, and Qwen3.6-Plus in the Qwen3-Max column), and because their token logs were not retained, each model's cost per task is set to its mean on the 112 SWE-bench tasks. Table 12 reports each model on both parts. The two parts favor different models: Opus 4.6 is best on the 112 SWE-bench tasks (66.07%), and GPT-5.4 is best on the 64 added tasks (82.81%). ACRouter reaches 66.96% on the 112 tasks at 51% of Opus's cost and 84.38% on the 64 tasks.

Table 12: Per-model results on the 176-task OOD split (harness-graded resolve rate, % of tasks whose patch at least applies, mean agent calls per task, and total backend cost). The per-task oracle resolves 78.98% (75.89% on the 112 SWE-bench tasks, 84.38% on the 64 added tasks).

Model	Resolved% (176)	112 / 64	Apply_ok%	Calls/task	$Total
Claude Opus 4.6	63.64	66.07 / 59.38	90.34	17.1	$194.91
Claude Sonnet 4.6	43.18	57.14 / 18.75	86.93	19.0	$140.42
GPT-5.4	**64.20**	53.57 / 82.81	98.86	5.2	$18.27
GLM-5	44.89	28.57 / 73.44	62.50	15.2	$38.24
Kimi-K2.5	19.89	18.75 / 21.88	53.41	12.2	$1.46
MiniMax-M2.7	35.23	14.29 / 71.88	47.73	7.1	$18.43
Qwen3-Max	32.95	8.93 / 75.00	48.86	6.0	$4.21
Qwen3.5-Plus	27.27	2.68 / 70.31	39.77	15.3	$6.21

Static and online routers. The probing dimensions are dominated by short single-file tasks, while OOD tasks require navigating repositories. Static routers' OOD decisions are replayed from the saved routers on the real task prompts. Bandits update per-arm posteriors from the stream (LinTS $57.16_{\pm 3.27}$ over 10 seeds; LinUCB 55.11).

Escalation chain. The OOD instantiation runs the cheaper models in a fixed order (MiniMax-M2.7, Kimi-K2.5, GPT-5.4, GLM-5). It stops at the first attempt the Verifier accepts and escalates to Opus 4.6 when no attempt is accepted and at least k attempts produced an applicable patch. The chain and $k{=}2$ are fixed for all OOD tasks. The chosen model's harness outcome is the reported score, and the cost of every attempted model is charged.

Resolved vs. applied. The OOD AvgPerf column is the harness-graded resolve rate, not the looser `apply_ok` signal, which only checks that a patch applies without running tests.

E.6 Robustness of the Streaming Evaluation

Delayed, noisy, and missing feedback. We stress-test the online-kNN loop (the retrieval component of ACRouter); results are in Table 4b. A delay of d tasks means feedback from task t reaches Memory and the dimension posterior only before task $t{+}d$. Noise p flips each feedback score $s \mapsto 1{-}s$ with probability p before the update, while the task's true score is unchanged. Without feedback, Memory is never updated. Delays of up to 100 tasks cost about one point. Noise matters more, yet even with half of the feedback flipped the loop stays above Random (38.75). Without feedback, the router falls to 47.32, near DimensionBest (47.50); the paired 95% CI of its -1.42-point gap to the 48.74 baseline is $[-1.65, -1.20]$. Proposition 13 relates the growing loss under flips to the shrinking score gaps.

Adding a new model mid-stream. To test exploration, we add DeepSeek-V4-Flash, released after the benchmark was built, as a 9th model on the 112 SWE-bench OOD tasks (it resolves 79/112 with

the same harness). Over 5,000 paired task orders, adding it from task 1 improves AvgPerf by $+3.79$ [3.64, 3.94] over the 8-model pool, with a median of 5 tasks before its first selection. Adding it at task 57 improves AvgPerf by $+2.46$ [2.37, 2.56], with a median of 1 task before its first selection. This experiment uses the uncertainty-driven Memory estimate without the LLM policy voter.

Latency. On one A100 80GB GPU over the ID stream, ACRouter adds 0.352 s of routing latency per task (6.60% of the selected model's 5.51 s inference latency), compared with 0.331 s for the 0.8B policy alone and 0.002 s for kNN retrieval. Ensemble aggregation, Memory retrieval and update, and the ID Verifier together add about 0.02 s.

F Discussion

F.1 Why Routing Matters

Model complementarity recurs with every model generation, each of which introduces new strengths (GLM-5 on algorithms, Qwen3-Max on test generation, Kimi-K2.5 on data science). Cost differentials persist: the most expensive model in our pool (Claude Opus 4.6 output at \$25/M) is over $20\times$ the cheapest (MiniMax-M2.7 output at \$1.20/M, Table 8). As the ecosystem expands with open-source, domain-specialized, and reasoning-specialized models, the value of routing grows, which makes routing a standing component of agent systems.

F.2 Practitioner's Guide: Building Your Own Agent Router

The C-A-F decomposition doubles as a construction kit:

1. **Set up the tool layer**: Plug in your candidate models and configure the execution sandbox.
2. **Profile via probing set**: Use CodeRouterBench (or your own tasks via C-A-F) to build a dimension$\times$model performance matrix.
3. **Start with DimensionBest**: A static Memory + lookup achieves about 83% of oracle AvgPerf at near-zero overhead. This is your baseline.
4. **Add a classifier**: Swap the routing tool to a trained classifier (LogReg or RouteLLM) for cheaper deployment with comparable AvgPerf to DimensionBest. Trained classifiers achieve Perf/\$ of 6.11–6.82 in Table 3.
5. **Complete the C-A-F loop (ACRouter)**: When deploying on new distributions, activate all three modules to close the feedback loop. Initialize Memory with whatever priors you have; the loop then refines them from verified outcomes.
6. **Customize tools**: Swap evaluation tools in the Verifier (e.g., domain-specific tests), add custom routing tools.
7. **Extend the benchmark**: Add tasks via C-A-F. New models need responses + scoring; new dimensions need a task set + scoring function.

G Task Examples Across All 10 Dimensions

Figures 11–20 present two task examples from each of the 10 dimensions, showing the input prompt, the reference, and the outputs and scores of three of the 8 models. The decision line gives the choice of DimensionBest, of the 3-shot Claude Sonnet 4.6 router (ACRouter on the OOD dimension, where DimensionBest does not apply), and of the per-task oracle over all 8 models under $r = s - 0.1\,\kappa$; «ORACLE marks a shown model that is this oracle. Agentic-programming scores are the harness resolve outcome. These examples illustrate the diversity of coding tasks in CodeRouterBench.

Code Generation

```
Task: code_generation_1  [python]
================================================================================
PROMPT:
 def pairs_sum_to_zero(l):     """     pairs_sum_to_zero takes a list
of integers as an input.     it returns True if there are two distinct
elements in the list that sum to zero, and False otherwise.     >>>
pairs_sum_to_zero([1, 3, 5, 0])     False     >>> pairs_sum_to_zer
...
--------------------------------------------------------------------------------
GROUND TRUTH:
 for i in range(len(l)):         for j in range(len(l)):             if i != j
and l[i] + l[j] == 0:                 return True     return False...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]
    ```python def pairs_sum_to_zero(l):     for i in
    range(len(l)):         for j in range(i + 1, len(
  GPT-5.4  [score=1.00]
    ```python def pairs_sum_to_zero(l):     """
    pairs_sum_to_zero takes a list of integers as an inp
  Qwen3.5  [score=1.00]
    ```python def pairs_sum_to_zero(l):     """
    pairs_sum_to_zero takes a list of integers as an inp
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Opus    |  LLM 3-shot -> Qwen3.5    |  Oracle -> Kimi-K2.5
```

```
Task: code_generation_2  [python]
================================================================================
PROMPT:
Write a python function to find the element that appears only once in a
sorted array.
--------------------------------------------------------------------------------
GROUND TRUTH:
(execution-based evaluation)
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    ```python def find_single_element(arr):     """     Find the
    element that appears only once in a s
  GPT-5.4  [score=0.00]
    ```python def single_non_duplicate(nums):     """     Return
    the element that appears only once in a
  Qwen3.5  [score=0.00]
    ```python def find_single_element(nums):     low, high = 0,
    len(nums) - 1     while low < high:
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Opus    |  LLM 3-shot -> Opus    |  Oracle -> Qwen3-Max
```

Figure 11: Code Generation examples.

Algorithm Design

```
Task: algorithm_1  [python]
================================================================================
PROMPT:
You are given a 0-indexed integer array nums, and an integer k. In one
operation, you can remove one occurrence of the smallest element of
nums. Return the minimum number of operations needed so that all
elements of the array are greater than or equal to k. Example 1:
Input: n
...
--------------------------------------------------------------------------------
GROUND TRUTH:
(execution-based evaluation)
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    ```python class Solution: def minOperations(self, nums:
    List[int], k: int) -> int:    retur
  GPT-5.4  [score=0.00]
    ```python from typing import List class Solution: def
    minOperations(self, nums: List[int], k: i
  Qwen3.5  [score=0.00]
    ```python from typing import List class Solution: def
    minOperations(self, nums: List[int], k: i
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> GLM-5    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Kimi-K2.5
```

```
Task: algorithm_2  [python]
================================================================================
PROMPT:
import random import math def task_func(LETTERS=[chr(i) for i in
range(97, 123)]):     """     Create a dictionary where keys are letters
from a predefined list LETTERS and values are lists of random integers.
Then, calculates the population standard deviation for each list
...
--------------------------------------------------------------------------------
GROUND TRUTH:
random_dict = {k: [random.randint(0, 100) for _ in
range(random.randint(1, 10))] for k in LETTERS}     sd_dict = { k:
math.sqrt(sum((i - s...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]
    ```python import random import math def
    task_func(LETTERS=[chr(i) for i in range(97, 123)]):
  GPT-5.4  [score=1.00]
    ```python import random import math def
    task_func(LETTERS=[chr(i) for i in range(97, 123)]):     re
  Qwen3.5  [score=1.00]
    ```python import random import math def
    task_func(LETTERS=[chr(i) for i in range(97, 123)]):     ""
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> GLM-5    |  LLM 3-shot -> Qwen3.5    |  Oracle -> Qwen3-Max
```

Figure 12: Algorithm Design examples.

Bug Fixing

```
Task: bug_fixing_1  [python3]
================================================================================
PROMPT:
Fix the following buggy code: ```python3 class Solution:     def
minSteps(self, s: str, t: str) -> int:         cnt1=Counter(s)
cnt2=Counter(t)         sm=0         cnt=cnt1-cnt2+(cnt2-cnt1
for i in cnt.values():             sm+=i         return sm ``` Bug des
...
--------------------------------------------------------------------------------
GROUND TRUTH:
class Solution:     def minSteps(self, s: str, t: str) -> int:
cnt1=Counter(s)         cnt2=Counter(t)         sm=0
cnt=cnt1-cnt2+(cnt...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.89]
    ```python3 from collections import Counter class Solution:
    def minSteps(self, s: str, t: str) -
  GPT-5.4  [score=0.89]
    ```python from collections import Counter class Solution:
    def minSteps(self, s: str, t: str) ->
  Qwen3.5  [score=0.92]   <<ORACLE
    ```python3 from collections import Counter class Solution:
    def minSteps(self, s: str, t: str) -
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> GLM-5    |  LLM 3-shot -> Qwen3.5    |  Oracle -> Qwen3.5
```

```
Task: bug_fixing_2  [java]
================================================================================
PROMPT:
Fix the following buggy code: ```java class Solution {     public int
numJewelsInStones(String jewels, /*String stones) {         int count =
0;         for(int i=0; i<stones.length(); i++)             char c =
stones.charAt(i);         String undefinedObject = null;
...
--------------------------------------------------------------------------------
GROUND TRUTH:
class Solution {     public int numJewelsInStones(String jewels, String
stones) {         int count = 0;         for(int i=0;
i<stones.length(); i++)...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]   <<ORACLE
    ```java class Solution {     public int
    numJewelsInStones(String jewels, String stones) {
  GPT-5.4  [score=0.82]
    ```java class Solution {     public int
    numJewelsInStones(String jewels, String stones) {        in
  Qwen3.5  [score=0.93]
    ```java class Solution {     public int
    numJewelsInStones(String jewels, String stones) {        in
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> GLM-5    |  LLM 3-shot -> Sonnet    |  Oracle -> Opus
```

Figure 13: Bug Fixing examples.

Code Completion

```
Task: code_completion_1  [python]
================================================================================
PROMPT:
Given the following function: ```python def f(title):     return
title.lower() ``` Predict the output of: ' Rock Paper SCISSORS '
--------------------------------------------------------------------------------
GROUND TRUTH:
' rock paper scissors '
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]
    The function `f` converts the input string to lowercase using
    the `.lower()` method. Given the inpu
  GPT-5.4  [score=1.00]
    ```python " rock paper scissors " ```
  Qwen3.5  [score=0.00]
    ```python def f(title):     return title.lower() print(f('
    Rock Paper SCISSORS ')) ```
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Opus    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Kimi-K2.5
```

```
Task: code_completion_2  [python]
================================================================================
PROMPT:
Given the following function: ```python def f(text):     new_text =
text     while len(text) > 1 and text[0] == text[-1]:         new_text =
text = text[1:-1]     return new_text ``` Predict the output of: ')'
--------------------------------------------------------------------------------
GROUND TRUTH:
')'
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]
    Let me trace through the function with input `')'`. 1. `text
    = ')'`, `new_text = ')'` 2. Check whil
  GPT-5.4  [score=1.00]
    ```python ')' ```
  Qwen3.5  [score=1.00]
    ```python ')' ```
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Opus    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Qwen3-Max
```

Figure 14: Code Completion examples.

Code Refactoring

```
Task: code_refactoring_1  [java]
================================================================================
PROMPT:
Refactor/fix the following Java code: ```java public TYPE_1 METHOD_1 (
TYPE_2 VAR_1 ) { TYPE_1 response = new TYPE_1 ( ) ; VAR_2 . info (
STRING_1 ) ; try { VAR_2 . info ( ( STRING_2 + ( VAR_3 ) ) ) ; response
= ( ( TYPE_1 ) ( VAR_3 . METHOD_2 ( VAR_1 ) ) ) ; VAR_2 . info ( ( ST
...
--------------------------------------------------------------------------------
GROUND TRUTH:
public TYPE_1 METHOD_1 ( TYPE_2 VAR_1 ) { TYPE_1 response = new TYPE_1 (
) ; VAR_2 . info ( STRING_1 ) ; try { VAR_2 . info ( ( STRING_2 + (
VAR_3 ) )...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.70]
    ```java public TYPE_1 METHOD_1(TYPE_2 VAR_1) {     TYPE_1
    response = new TYPE_1();     VAR_2.info(
  GPT-5.4  [score=0.72]
    ```java public TYPE_1 METHOD_1(TYPE_2 VAR_1) {     TYPE_1
    response = new TYPE_1();     VAR_2.info(ST
  Qwen3.5  [score=0.72]
    ```java public TYPE_1 METHOD_1(TYPE_2 VAR_1) {     TYPE_1
    response = new TYPE_1();     VAR_2.info(ST
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> GPT-5.4    |  LLM 3-shot -> Sonnet    |  Oracle -> Qwen3-Max
```

```
Task: code_refactoring_2  [java]
================================================================================
PROMPT:
Refactor/fix the following Java code: ```java public void METHOD_1 (
java.lang.Object obj , java.lang.String VAR_1 ) throws TYPE_1 { assert
obj != null ; if ( VAR_2 . METHOD_2 ( ) ) VAR_2 . METHOD_3 ( ( STRING_1
+ obj ) ) ; obj = METHOD_4 ( obj ) ; METHOD_5 ( obj , VAR_3 , null
...
--------------------------------------------------------------------------------
GROUND TRUTH:
public void METHOD_1 ( java.lang.Object obj , java.lang.String VAR_1 )
throws TYPE_1 { assert obj != null ; if ( VAR_2 . METHOD_2 ( ) ) VAR_2 .
METHOD...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.75]   <<ORACLE
    ```java public void METHOD_1(java.lang.Object obj,
    java.lang.String VAR_1) throws TYPE_1 {     ass
  GPT-5.4  [score=0.61]
    ```java public void METHOD_1(final Object obj, final String
    VAR_1) throws TYPE_1 {     assert obj !=
  Qwen3.5  [score=0.14]
    ```java public void processObject(Object obj, String
    paramName) throws ExceptionType {     // Valida
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> GPT-5.4    |  LLM 3-shot -> MiniMax-M2.7    |  Oracle -> Opus
```

Figure 15: Code Refactoring examples.

Data Science

```
Task: data_science_1  [python]
================================================================================
PROMPT:
import pandas as pd import matplotlib.pyplot as plt values = [[1, 2],
[3, 4]] df = pd.DataFrame(values, columns=["Type A", "Type B"],
index=["Index 1", "Index 2"]) # Plot values in df with line chart #
label the x axis and y axis in this plot as "X" and "Y" # SOLUTION START
--------------------------------------------------------------------------------
GROUND TRUTH:
df.plot() plt.xlabel("X") plt.ylabel("Y")
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    df.plot.line() plt.xlabel("X") plt.ylabel("Y") plt.show()
  GPT-5.4  [score=0.00]
    ```python ax = df.plot(kind="line") ax.set_xlabel("X")
    ax.set_ylabel("Y") plt.show() ```
  Qwen3.5  [score=0.00]
    ```python import pandas as pd import matplotlib.pyplot as plt
    values = [[1, 2], [3, 4]] df = pd.Dat
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Kimi-K2.5    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Kimi-K2.5
```

```
Task: data_science_2  [python]
================================================================================
PROMPT:
Draw and return the daily turnover line chart from a pandas DataFrame.
The function should raise the exception for: ValueError: If 'df' is not
a DataFrame or lacks 'Date' or 'Sales' columns, or has no data to plot.
The function should output with:     Axes: Matplotlib Axes object
...
--------------------------------------------------------------------------------
GROUND TRUTH:
 if not isinstance(df, pd.DataFrame) or not all(col in df.columns for
col in ['Date', 'Sales']):         raise ValueError("Invalid 'df': must
be a...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    ```python import pandas as pd import matplotlib.pyplot as
    plt def task_func(df):     """     Dra
  GPT-5.4  [score=0.00]
    ```python import pandas as pd import matplotlib.pyplot as plt
    def task_func(df):     if not isinsta
  Qwen3.5  [score=0.00]
    ```python import pandas as pd import matplotlib.pyplot as plt
    def task_func(df):     if not isinsta
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Kimi-K2.5    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Qwen3-Max
```

Figure 16: Data Science examples.
```

Multi-Language

```
Task: multi_language_1  [javascript]
================================================================================
PROMPT:
/* Concatenate list of strings into a single string >>>
concatenate([])  '' >>> concatenate(['a', 'b', 'c'])  'abc' */
const concatenate = (strings) => {
--------------------------------------------------------------------------------
GROUND TRUTH:
 return strings.join(''); }
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.90]
    ```javascript const concatenate = (strings) => {     return
    strings.join(''); }; ```
  GPT-5.4  [score=0.90]
    ```javascript const concatenate = (strings) => {     return
    strings.join(''); }; ```
  Qwen3.5  [score=0.90]
    ```javascript const concatenate = (strings) => {     return
    strings.join(''); }; ```
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Opus    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Qwen3-Max
```

```
Task: multi_language_2  [ts]
================================================================================
PROMPT:
//This function takes an array l and returns an array l' such that // l'
is identical to l in the odd indicies, while its values at the even
indicies are equal // to the values of the even indicies of l, but
sorted. // >>> sort_even([1, 2, 3]) // [1, 2, 3] // >>> sort_even([5, 6,
...
--------------------------------------------------------------------------------
GROUND TRUTH:
(execution-based evaluation)
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    ```ts function sort_even(l: number[]): number[] {     const
    evenValues = l.filter((_, i) => i % 2 ==
  GPT-5.4  [score=0.00]
    ```ts function sort_even(l: number[]): number[] {     const
    evens = l.filter((_, i) => i % 2 === 0).so
  Qwen3.5  [score=0.00]
    ```typescript function sort_even(l: number[]): number[] {
    const evens = l.filter((_, i) => i % 2
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Opus    |  LLM 3-shot -> Qwen3.5    |  Oracle -> Qwen3-Max
```

Figure 17: Multi-Language examples.

Code Understanding

```
Task: code_understanding_1  [python]
================================================================================
PROMPT:
Summarize the following Python function in natural language: ```python
def generate_form(args):     """Generate form."""     form_name =
args.get('<form>')     logger.info('Start generating form.')
_generate_form(form_name)     logger.info('Finish generating form.') ```
--------------------------------------------------------------------------------
GROUND TRUTH:
Generate form.
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.10]   <<ORACLE
    This function retrieves a form name from the provided
    arguments dictionary (using the key `'<form>'`
  GPT-5.4  [score=0.05]
    This function generates a form based on a form name provided
    in `args`. In plain language, it: 1.
  Qwen3.5  [score=0.05]
    This function orchestrates the process of generating a form.
    It performs the following steps: 1. *
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Kimi-K2.5    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Opus
```

```
Task: code_understanding_2  [python]
================================================================================
PROMPT:
Summarize the following Python function in natural language: ```python
def set_completer_frame(self, frame=None):     """Set the frame of
the completer."""     if frame:         self.Completer.namespace
= frame.f_locals         self.Completer.global_namespace = f
...
--------------------------------------------------------------------------------
GROUND TRUTH:
Set the frame of the completer.
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.18]
    This function configures the completer's namespace context. If
    a stack frame is provided, it sets th
  GPT-5.4  [score=0.12]
    This function sets which variable scopes the completer should
    use for tab completion. - If a `frame
  Qwen3.5  [score=0.13]
    This method configures the namespace context used by a code
    completer. It accepts an optional execut
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Kimi-K2.5    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Kimi-K2.5
```

Figure 18: Code Understanding examples.

Test Generation

```
Task: test_generation_1  [python]
================================================================================
PROMPT:
Generate comprehensive test cases for the following function: ```python
def string_sequence(n: int) -> str:     """     Return a string containing
space-delimited numbers starting from 0 upto n inclusive.     >>>
string_sequence(0)     '0'     >>> string_sequence(5)     '0 1 2 3 4
...
--------------------------------------------------------------------------------
GROUND TRUTH:
[EvalPlus reference test, helpers omitted] def ref_func(n: int) -> str: return
 " ".join(map(str, range(n + 1))) ... inputs = [[0], [3], [10], [1], ...] (135
 inputs) ... assertion(candidate(*inp), ref_func(*inp), 0)
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    ```python import pytest from typing import Any # Assuming
    the function is defined in a module, w
  GPT-5.4  [score=0.00]
    ```python import pytest from solution import string_sequence
    @pytest.mark.parametrize(     "n, ex
  Qwen3.5  [score=1.00]   <<ORACLE
    ```python import unittest from typing import Callable #
    Assuming the function is imported from the
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Qwen3-Max    |  LLM 3-shot -> Kimi-K2.5    |  Oracle -> Qwen3.5
```

```
Task: test_generation_2  [python]
================================================================================
PROMPT:
Generate comprehensive test cases for the following function: ```python
def frequency_lists(list1):     list1 = [item for sublist in list1 for
item in sublist]     return {x: list1.count(x) for x in list1} ```
Task description: Write a function to find frequency of each eleme
...
--------------------------------------------------------------------------------
GROUND TRUTH:
[EvalPlus reference test, helpers omitted] inputs = [[[[1, 2, 3, 2], [4, 5, 6,
 2], [7, 8, 9, 5]]], ...] results = [{1: 1, 2: 3, 3: 1, 4: 1, 5: 2, 6: 1, 7:
1, 8: 1, 9: 1}, ...] (106 pairs) ... assertion(frequency_lists(*inp), exp, 0)
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=0.00]
    ```python import pytest from collections import Counter
    def frequency_lists(list1):     list1 =
  GPT-5.4  [score=0.00]
    ```python import pytest from solution import frequency_lists
    def test_empty_outer_list_returns_em
  Qwen3.5  [score=0.00]
    ```python import unittest # Function under test def
    frequency_lists(list1):     list1 = [item for s
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> Qwen3-Max    |  LLM 3-shot -> Qwen3-Max    |  Oracle -> Qwen3-Max
```

Figure 19: Test Generation examples.

Agentic Programming

```
Task: agentic_programming_1  [python]
================================================================================
PROMPT:
ManifestStaticFilesStorage crashes with max_post_process_passes = 0.
Description To reproduce: Derive a custom class from
ManifestStaticFilesStorage and set max_post_process_passes to 0: ...
UnboundLocalError: local variable 'substitutions' referenced before assignment
 ...
--------------------------------------------------------------------------------
GROUND TRUTH:
+++ b/django/contrib/staticfiles/storage.py @@ -260,6 +260,7 @@ ... +
substitutions = False for i in range(self.max_post_process_passes):
substitutions = False
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]
    @@ -261,6 +261,7 @@ ... + substitutions = False for i in
    range(self.max_post_process_passes): substitutions = False ...
  GPT-5.4  [score=0.00]
    @@ -261,8 +261,8 @@ ... + substitutions = False for i in
    range(self.max_post_process_passes): - substitutions = False ...
  Qwen3.5  [score=1.00]
    @@ -261,6 +261,7 @@ ... + substitutions = False for i in
    range(self.max_post_process_passes): substitutions = False ...
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> n/a    |  ACRouter -> Kimi-K2.5    |  Oracle -> Kimi-K2.5
```

```
Task: agentic_programming_2  [python]
================================================================================
PROMPT:
Consider removing auto-transform of structured column into NdarrayMixin ...
Proposal: - Add a FutureWarning here telling the user to wrap `data` in
`Column` and that in the future (5.2) the structured array will be added as a
`Column`. - Change the behavior in 5.2 by removing this clause. ...
--------------------------------------------------------------------------------
GROUND TRUTH:
+++ b/astropy/table/table.py @@ -1239,13 +1239,6 @@ ... - # Structured ndarray
 gets viewed as a mixin unless already a valid - # mixin class - if (not
isinstance(data, Column) and not data_is_mixin ... - data =
data.view(NdarrayMixin) - data_is_mixin = True ...
--------------------------------------------------------------------------------
MODEL OUTPUTS:
  Opus     [score=1.00]   <<ORACLE
    @@ -1239,13 +1239,6 @@ ... - # Structured ndarray gets viewed as a
    mixin unless already a valid ... - data = data.view(NdarrayMixin) -
    data_is_mixin = True ...
  GPT-5.4  [score=0.00]
    @@ -1239,12 +1239,18 @@ ... - data = data.view(NdarrayMixin) -
    data_is_mixin = True + warnings.warn( ... + FutureWarning, ...
  Qwen3.5  [score=0.00]
    (empty model_patch in the released predictions)
--------------------------------------------------------------------------------
ROUTER DECISIONS:
  DimensionBest -> n/a    |  ACRouter -> Opus    |  Oracle -> Opus
```

Figure 20: Agentic Programming examples (10th dimension, OOD).