TwinRouterBench: Fast Static and Live Dynamic Evaluation for Realistic Agentic LLM Routing

Pei Yang[1,*] Wanyi Chen[2,*] Tongyun Yang[3] Pengbin Feng[4] Jiarong Xing[5] Wentao Guo[1]
Yuhang Yao[6] Yuhang Han[7] Hanchen Li[8] Xu Wang[9] Zeyu Wang[10] Jie Xiao[1]
Anjie Yang[1] Liang Tian[1] Lynn Ai[1] Eric Yang[1] Tianyu Shi[1,†]
[1]Gradient [2]Soochow University [3]Independent Researcher [4]University of Southern California
[5]Rice University [6]Carnegie Mellon University [7]Shanghai Jiao Tong University [8]University of California, Berkeley
[9]University of the Chinese Academy of Sciences [10]University of California, Los Angeles

Abstract

Production LLM agents often spend most of their budget across many intermediate calls, yet existing router benchmarks mostly evaluate one-shot prompt routing without accounting for multi-step agentic tasks. We introduce **TwinRouterBench**, a step-level routing benchmark for choosing the cheapest sufficient model tier conditioned on the prefix visible before the next LLM call. The *static track* contains 970 router-visible prefixes from 520 instances across SWE-bench, BFCL, mtRAG, QMSum, and PinchBench, with execution verified tier labels and deterministic scoring over label correctness, trajectory membership, and token cost. The *dynamic track* runs routers end-to-end on SWE-bench Verified and reports official resolution and realized API spend. On a 100-case held-out SWE-bench split, a logistic router trained on the static labels achieves comparable resolution to unrouted Opus 4.6 while reducing API cost by 53.1%.[1]

Keywords

LLM routing, agentic benchmark, dynamic evaluation

1 Introduction

LLM routers choose which model should handle a request [7, 20]. In agentic systems, the request is not a complete one-shot prompt but the prefix before the next call: system and user messages, previous assistant outputs, tool observations, retrieval snippets, shell logs, and partial edits. A cheap model may be adequate for lookup or summarization but fail when the next call must write a patch or choose a tool. The routing question is therefore conditional and per-call: given the prefix now, what is the cheapest tier that keeps the task resolving?

Existing router benchmarks are valuable but mostly stop at query-level selection over complete prompts or precomputed outcome matrices [9, 15, 17]. Even when they include software tasks, the routable unit is usually the whole issue rather than the intermediate calls inside a trajectory. TRIM studies stepwise escalation for math reasoning [11], but it lacks tool outputs, retrieval state, executable feedback, and downstream verification.

TwinRouterBench targets this missing setting. As illustrated in Figure 1, it comprises two tracks. The static track exposes router-visible prefixes and verified tier labels from SWE-bench [10], BFCL [19], mtRAG [12], QMSum [21], and PinchBench [13]. Evaluation is deterministic arithmetic over labels, trajectory membership, and token costs, so router development is cheap and reproducible. The dynamic track then executes routers live on SWE-bench Verified, where success is official task resolution and cost is realized API spend. This separation lets offline router improvements be tested without conflating them with a live harness, while still requiring end-to-end evidence.

Our main contributions are: (1) a static step-level benchmark with 970 execution-verified labels under a fixed verification protocol; (2) a dynamic SWE-bench Verified routing harness with real API accounting; and (3) reference results showing that static labels can train a simple router that matches unrouted Opus 4.6 on held-out resolution while cutting API cost by 53.1%.

2 Positioning

Prior routing work studies quality-cost trade-offs for whole queries through cascades, learned routers, or difficulty predictors [1, 4, 6, 18]. Those systems ask whether an incoming request should be served by a cheap or expensive model, sometimes after a cheap first attempt. RouterBench, LLMRouterBench, and RouterArena then make this abstraction measurable at scale by collecting outcome matrices for complete prompts [9, 15, 17]. The abstraction is useful for standalone tasks, but it hides the intermediate states that dominate agentic execution: a router does not see a clean problem statement, but a partially completed trajectory shaped by previous model and tool outputs.

TwinRouterBench keeps that state in the benchmark input. The label is not only whether a model can answer a prompt in isolation, but whether replacing the next call preserves task success under the same harness. This also separates our setting from stepwise reasoning escalation in math, where traces are comparatively self-contained and do not need to replay a changing external environment [11]. For software agents, function calling, RAG, and summarization, the router-visible prefix is the object that must be benchmarked.

3 Problem and Benchmark

3.1 Conditional per-call routing

At step i of a multi-turn trajectory, the router observes only the prefix x_i available before the next LLM call. A step-level router is a conditional decision rule

$$\pi : x_i \mapsto t_i \in \mathcal{T} = \{\texttt{low}, \texttt{mid}, \texttt{mid\_high}, \texttt{high}\}. \quad (1)$$

[1]Code and data: https://github.com/CommonstackAI/TwinRouterBench. Website: https://commonstackai.github.io/TwinRouterBench/.

[*]Equal contribution.

[†]Corresponding author: tianyu@gradient.network.

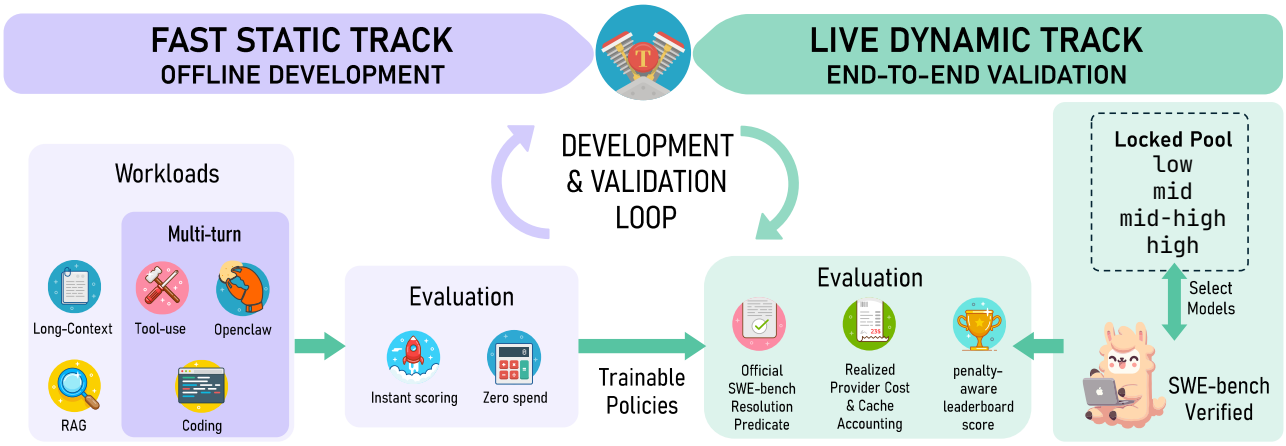

Figure 1: TwinRouterBench provides a fast static track for offline router development and a live dynamic track for end-to-end validation. The static track contains 970 step-level rows from 520 instances across five workloads; the dynamic track runs routers on SWE-bench Verified with realized API cost.

Each tier denotes a pool of models with comparable cost and capability. Given a fixed pool $\mathcal{M}$, the ideal target is the cheapest tier containing at least one model that is sufficient for this call:

$$t_i^\star = \min\{t \in \mathcal{T} : \exists m \in \mathcal{M}_t,\ V_i(m; x_i) = 1\}. \tag{2}$$

For one-shot workloads, V_i is the task success predicate. For agentic workloads, it means replacing the current LLM call with model m under the fixed execution protocol still preserves downstream task success. Thus $t_i^\star$ is a conditional per-call quantity, not a globally optimal joint assignment over all future steps.

Operationally, the router cannot rewrite earlier outputs that produced x_i or choose future calls before their prefixes exist. A joint trajectory optimum may be useful for planning, but it is not what an online router observes. TwinRouterBench fixes the pool, verifier, continuation protocol, and cost model, then asks the local routing question at each observed prefix. If any ingredient changes, the target should be treated as a new benchmark version rather than a drop-in relabeling of the old rows.

3.2 Two-track benchmark

The released static label $\hat{t}_i$ estimates this target by execution verification: a candidate tier is accepted when at least one model in the tier pool resolves the mixed-model trajectory under the harness. Target tiers are therefore pool- and protocol-specific existential labels, not universal capability labels or model-independent lower bounds on cost. The release contains 970 rows from 520 instances across five workloads; each row stores the router-visible messages, benchmark id, instance id, step index, total steps, and target tier. Public rows expose only tier labels, not vendor model ids. Corpus and tier distributions are in Appendix Tables A2 and A3; the fixed 11-model pool is in Table A4.

The static track scores routers without online evaluator-side LLM calls. It combines row-level correctness, trajectory-level correctness, and token-cost savings. The dynamic track is separate: routers are executed end-to-end on SWE-bench Verified, choose a concrete model at each call, and are scored by official resolution plus realized API spend. Static and dynamic scores are not averaged because they answer different questions: offline routing quality across five workloads versus live software-agent performance on SWE-bench. In practice, the static track is for rapid iteration and training, while the dynamic track is the end-to-end check that the same routing policy survives live execution and provider accounting.

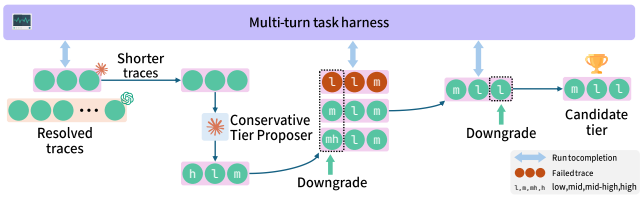

Figure 2: TwinRouterBench construction pipeline. Labels are produced by execution-verified downgrade search and tier-pool cascade verification, then checked by hardened judges or manual audit depending on the workload.

4 Dataset Construction

Figure 2 summarizes the construction pipeline: start from a successful trajectory, search cheaper tiers under causal prefixes, verify tiers as pools, harden open-ended judges, and audit multi-step labels.

Successful seed trajectories. We first collect trajectories that solve the original task under a strong-model setting. SWE-bench uses the mini-SWE-agent harness and official regression tests; BFCL, mtRAG, QMSum, and PinchBench use their native pass predicates. Failed seeds are discarded so labels do not conflate task difficulty with routing error.

Sequential downgrade search. For each surviving trajectory, an Opus 4.6 hint proposes which steps may be downgradeable and the cheapest tier worth trying first. The hint only prunes search.

We then run greedy sequential locking (Algorithm A1): verified steps stay locked, future steps remain strong, and the current step is tested from cheap to expensive tiers. A candidate is accepted only when the full trajectory still passes with the same number of steps. This reduces the naive $|\mathcal{T}|^N$ grid to $O(|\mathcal{T}|N)$ trials while preserving causal prefixes that include earlier downgraded outputs.

Tier-pool verification. A tier label should mean the pool can solve the call, not that one representative model happened to pass. For non-`low` labels we use a three-model cascade and accept a tier if any model in the pool passes under the mixed trace. In a SWE-bench last-step audit, this recovered 28 of 33 downgradeable labels versus 15 of 33 under single-model probing, showing that single-representative search is too brittle for supervision.

Open-ended judging and manual audit. SWE-bench and BFCL have executable or structural pass predicates. For mtRAG and QMSum, we replace weak reference-similarity judging with a stricter rubric that checks current-turn success, faithfulness, appropriateness, and completeness. The rubric hard-fails evidence conflicts and fails uncertain cases by default [14]. To check cross-step interaction risk, we also audit an independent 10% step sample from the multi-step workloads: 64 steps across BFCL, SWE-bench, and PinchBench. Reviewers marked 63 labels tight; one SWE-bench step was further downgradeable and was corrected before release. The full audit protocol and table are in Appendix A.13.

Release criterion. A row enters the static track only if the strong seed passed, a tier-pool model passed under the mixed trace, and open-ended workloads passed the hardened judge. Each released label is therefore an execution-verified estimate of the conditional per-call target under the fixed model pool, verifier, continuation protocol, and cost model.

The protocol intentionally favors conservative labels over aggressive savings. If a cheaper tier succeeds only under a brittle single-model probe, or if an open-ended judge cannot distinguish partial overlap from task success, the row does not receive that cheaper target. This makes the static labels less like a theoretical lower bound on cost and more like reusable supervision for routers: a policy trained on them should learn when downgrades are reliably safe under the released harness, not when a one-off run happened to pass.

5 Evaluation, Baselines and Experiment Setup

5.1 Static scoring

For row i, let $\hat{t}_i$ be the released target tier and $\tilde{t}_i$ the router prediction. A row passes if $\tilde{t}_i \geq \hat{t}_i$ and is exact if $\tilde{t}_i = \hat{t}_i$. A trajectory passes only if every routed row in that trajectory passes. CostSave measures savings relative to an always-`high` policy, but credits savings only on passing trajectories; API cost spent on failed trajectories is charged as burned. The headline score averages RowPass, RowExact, TrajPass, and CostSave. Full equations, tier prices, cache accounting, and the cost ablation are in Appendix A.15.

The four components expose different failure modes. RowPass rewards safe predictions, RowExact rewards tight cost choices, TrajPass penalizes a single under-route that breaks a multi-step case, and CostSave asks whether the savings survive after failed trajectories are charged. Reporting all four prevents a router from looking good merely because it is always conservative, always cheap, or correct only on single-call workloads.

5.2 Dynamic SWE-bench scoring

The dynamic track executes SWE-bench Verified instances end-to-end using `mini-swe-agent` v2.2.8. At each LLM call, a router chooses a concrete model from the locked pool. We report official SWE-bench resolution, realized API cost from provider usage logs, and a leaderboard bill

$$\text{bill}_j = A_j + \left(1 - \text{resolved}_j\right)\gamma, \tag{3}$$

where A_j is realized API cost on instance j and γ=0.60 is the same unresolved-instance add-on for every policy. The aggregate leaderboard bill is

$$\text{Bill} = \sum_j A_j + \gamma \sum_j \left(1 - \text{resolved}_j\right). \tag{4}$$

For example, Table 2 has $25.66 + 0.60(100 - 75) = 40.66$ for the trained router and $54.73 + 0.60(100 - 74) = 70.33$ for unrouted Opus 4.6. This bill separates cheap failure from genuinely useful routing while leaving API spend visible as its own column.

5.3 Baselines

Static references include SR-KNN [16], an in-sample nearest-neighbor upper bound; ClawRouter [3], an open rule-based router; UncommonRoute [5], a public rule-based router; and an always-`high` Opus 4.6 envelope. We also train only the UncommonRoute routing policy. A frozen bge-small embedding is concatenated with routing-time metadata: message count, tool-call presence, tool-message count, request length, and code/question indicators. The trained router does not use target tier, verifier outcomes, future model outputs, or post-hoc trajectory information. Cross-validation selects L2 regularization. We never fine-tune language models.

6 Benchmark Results & Analysis

6.1 Static diagnostic results

Table 1 reports static scores on all 970 rows. SR-KNN is an in-sample upper-bound reference and ranks first among the data-driven and rule-based routers. The always-`high` row is a cost-envelope reference: it passes every row and trajectory, but has zero cost saving by construction.

The error patterns show why both row and trajectory terms are needed. ClawRouter is cheap but has only 11.75% RowExact and fails 38/40 SWE-bench trajectories; UncommonRoute is more exact but often jumps to `high`, reducing cost savings. SR-KNN keeps similar savings to ClawRouter while preserving many more trajectories. The failure-aware cost ablation in Appendix A.15 shows that looser cost formulas can mistakenly reward cheap calls on failed trajectories.

These static numbers should be read as diagnostics, not as final deployment claims. SR-KNN is an in-sample upper bound that shows the labels contain useful neighborhood structure. The rule-based routers reveal two opposite mistakes: upgrading too often passes rows but wastes cost, while routing too low looks cheap

Router	RowPass ↑	RowExact ↑	TrajPass ↑	CostSave ↑	**Combined** ↑
SR-KNN (in-sample ub.)	**91.86**	**78.76**	**84.74**	**56.18**	**77.89**
ClawRouter (rule-based)	80.82	11.75	64.64	53.82	52.76
UncommonRoute (rule-based)	88.35	61.13	62.37	15.99	56.96
Opus 4.6 (always-`high`)	100.00	17.53	100.00	0.00	54.38

Table 1: Static-track diagnostic results (percentages). Combined equally averages the four components.

Policy	Resolved ↑	Avg. API cost ↓	API cost ↓	Penalty cost ↓	Bill ↓
SR-KNN router	**75**/100	$0.56	$55.61	$15.00	$70.61
UncommonRoute (trained)	**75**/100	**$0.26**	**$25.66**	$15.00	**$40.66**
UncommonRoute (rule-based)	73/100	$1.73	$172.56	$16.20	$188.76
Opus 4.6 (no routing)	74/100	$0.55	$54.73	$15.60	$70.33

Table 2: Dynamic SWE-bench Verified results on 100 held-out instances. Bill is API cost plus the uniform unresolved-instance add-on.

only until failed trajectories are counted. This is why the benchmark reports row, trajectory, and cost terms side by side.

The tier distribution is also uneven: high-tier labels are concentrated in the SWE-bench slice, while BFCL, mtRAG, and QMSum are mostly low-tier saturated. We therefore treat static scores as diagnostics and use the held-out dynamic SWE-bench track to test whether routing remains useful within the hardest slice, not merely across workload identity.

Opus-as-router case study. With a single prompt describing the 11-model pool, Claude Opus 4.6 predicts `high` for only 7 of 147 verified-`high` SWE-bench steps and fails all 40 SWE trajectories. This is one diagnostic prompt, not a universal claim about frontier-model routers, but it motivates execution-verified supervision. Full breakdowns are in Appendix A.11.

6.2 Dynamic held-out SWE execution

Table 2 reports a shared 100-case SWE-bench Verified held-out split disjoint from static SWE supervision. Opus 4.6 and rule-based UncommonRoute use no offline routing labels; SR-KNN uses the upstream `semantic-router` defaults; only UncommonRoute (trained) is fit on static-track labels.

The trained router matches SR-KNN at 75/100 resolved but spends $25.66 rather than $55.61. Relative to unrouted Opus 4.6, it attains comparable resolution while reducing API cost by $1-25.66/54.73 = 53.1\%$; relative to rule-based UncommonRoute, it is 6.7× cheaper. We do not interpret the 75/100 versus 74/100 difference as a statistically significant resolution improvement. The main dynamic finding is that static-label supervision can preserve comparable resolution while substantially reducing realized API spend under the same held-out split and accounting protocol.

The dynamic table also illustrates why the two-track design is necessary. Static scoring is cheap enough for development, but live execution is where cache behavior, model-family differences, tool formatting, and recovery from earlier choices become visible. The trained router is deliberately simple, so its dynamic performance should be viewed as an initial reference point rather than a ceiling for learned step-level routing.

The two tracks support router development at different stages. A router author can iterate on the static track, inspect whether failures come from upgrading too often, routing too low, or breaking trajectories, and then reserve live execution for candidates that improve the Pareto trade-off. This is important for agentic workloads because the cost of a call is not separable from its downstream state: a cheap but insufficient response can corrupt later tool inputs, while an always-`high` policy can hide whether cheaper sufficient steps exist at all. The benchmark therefore treats dynamic execution as the final check, but keeps the static track central as the scalable diagnostic layer that makes router debugging repeatable.

7 Conclusion and Limitations

TwinRouterBench turns LLM routing from one-shot query selection into conditional per-call evaluation for agentic workloads. Its static track gives fast, deterministic supervision over router-visible prefixes; its dynamic track tests whether a policy survives live SWE-bench execution with realized cost. The first results show that even a simple trained router can preserve comparable SWE-bench resolution while substantially reducing API spend.

Limitations. The 970-row static corpus is an initial release, not an exhaustive map of all agent domains. Labels depend on the fixed model pool, tier membership, harnesses, prices, and verification protocol; changing any of these creates a new benchmark version. Dynamic evaluation is currently reported only for SWE-bench Verified, high-tier labels are concentrated in that slice, and we do not claim statistically significant resolution gains from the 100-case dynamic split. Production deployment still requires rechecking prices, provider behavior, and harness assumptions in the target environment. Future versions should add more live domains, larger held-out dynamic splits, no-workload-id shortcut checks, and uncertainty estimates for steps near tier boundaries.

References

[1] Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. 2024. AutoMix: Automatically Mixing Language Models. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 131000–131034. doi:10.52202/079017-4164

[2] Emily M. Bender and Batya Friedman. 2018. Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science. *Transactions of the Association for Computational Linguistics* 6 (2018), 587–604. doi:10.1162/tacl_a_00041

[3] BlockRunAI. 2026. ClawRouter. https://github.com/BlockRunAI/ClawRouter. Version 0.12.149. Accessed: 2026-04-28.

[4] Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Transactions on Machine Learning Research* (2024). https://openreview.net/forum?id=cSimKw5p6R

[5] Commonstack AI. 2026. UncommonRoute: An open-source LLM routing library. https://github.com/CommonstackAI/UncommonRoute. Version 0.7.15. Accessed: 2026-04-28.

[6] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=02f3mUtqnM

[7] Tao Feng, Yanzhen Shen, and Jiaxuan You. 2024. GraphRouter: A Graph-based Router for LLM Selections. arXiv:2410.03834 [cs.CL] doi:10.48550/arXiv.2410.03834

[8] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for Datasets. *Commun. ACM* 64, 12 (2021), 86–92. doi:10.1145/3458723

[9] Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. RouterBench: A Benchmark for Multi-LLM Routing System. *CoRR* abs/2403.12031 (2024). arXiv:2403.12031 doi:10.48550/ARXIV.2403.12031

[10] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world GitHub Issues?. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=VTF8yNQM66

[11] Vansh Kapoor, Aman Gupta, Hao Chen, Anurag Beniwal, Jing Huang, and Aviral Kumar. 2026. TRIM: Hybrid Inference via Targeted Stepwise Routing in Multi-Step Reasoning Tasks. *CoRR* abs/2601.10245 (2026). arXiv:2601.10245 doi:10.48550/ARXIV.2601.10245

[12] Yannis Katsis, Sara Rosenthal, Kshitij Fadnis, Chulaka Gunasekara, Young-Suk Lee, Lucian Popa, Vraj Shah, Huaiyu Zhu, Danish Contractor, and Marina Danilevsky. 2025. mtRAG: A Multi-Turn Conversational Benchmark for Evaluating Retrieval-Augmented Generation Systems. *Trans. Assoc. Comput. Linguistics* 13 (2025), 784–808. doi:10.1162/TACL.A.19

[13] Kilo Code. 2026. PinchBench: Real-world benchmarks for AI coding agents. https://github.com/pinchbench/skill. Version 2.0.0. MIT license.

[14] Tzu-Lin Kuo, FengTing Liao, Mu-Wei Hsieh, Fu-Chieh Chang, Po-Chun Hsu, and Da-shan Shiu. 2025. RAD-Bench: Evaluating Large Language Models' Capabilities in Retrieval Augmented Dialogues. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 868–902. doi:10.18653/v1/2025.naacl-industry.66

[15] Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, et al. 2026. LLMRouterBench: A Massive Benchmark and Unified Framework for LLM Routing. arXiv preprint arXiv:2601.07206. arXiv:2601.07206 https://arxiv.org/abs/2601.07206

[16] Xunzhuo Liu, Huamin Chen, Samzong Lu, Yossi Ovadia, Guohong Wen, Hao Wu, Zhengda Tan, Jintao Zhang, Senan Zedan, Yehudit Kerido, Liav Weiss, Haichen Zhang, Bishen Yu, Asaad Balum, Noa Limoy, Abdallah Samara, Baofa Fan, Brent Salisbury, Ryan Cook, Zhijie Wang, Qiping Pan, Rehan Khan, Avishek Goswami, Houston H. Zhang, Shuyi Wang, Ziang Tang, Fang Han, Zohaib Hassan, Jianqiao Zheng, and Avinash Changrani. 2026. vLLM Semantic Router: Signal Driven Decision Routing for Mixture-of-Modality Models. arXiv:2603.04444 [cs.LG] doi:10.48550/arXiv.2603.04444 Version 0.3.0. Code: https://github.com/vllm-project/semantic-router.

[17] Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. 2025. RouterArena: An Open Platform for Comprehensive Comparison of LLM Routers. *CoRR* abs/2510.00202 (2025). arXiv:2510.00202 doi:10.48550/ARXIV.2510.00202

[18] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs from Preference Data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=8sSqNntaMr

[19] Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025 (Proceedings of Machine Learning Research, Vol. 267)*, Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (Eds.). PMLR / OpenReview.net, 48371–48392. https://proceedings.mlr.press/v267/patil25a.html

[20] Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. 2024. TensorOpera Router: A Multi-Model Router for Efficient LLM Inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina (Eds.). Association for Computational Linguistics, Miami, Florida, US, 452–462. doi:10.18653/v1/2024.emnlp-industry.34

[21] Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir R. Radev. 2021. QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 5905–5921. doi:10.18653/V1/2021.NAACL-MAIN.472

A Appendix

A.1 Dataset Card

Table A1: Dataset card for the TwinRouterBench static track. This follows the Bender and Friedman [2] and Gebru et al. [8] data statement conventions adapted for LLM benchmark releases.

Field	Value
Dataset name	TwinRouterBench v1.0 (static track)
Task type	Step-level LLM routing (4-class classification over tier IDs 0–3)
Size	970 rows from 520 trajectory instances
Source workloads	SWE-bench (Apache-2.0), BFCL (CC BY 4.0), mtRAG (IBM Research; see original license), QMSum (MIT), PinchBench [13] (MIT; the 48 derived rows from 12 of its 53 tasks are included in this release and covered by the package-level Apache-2.0 license below)
Provenance	Message prefixes extracted from successful agent trajectories under strong-model execution; tier labels produced by greedy sequential-locking downgrade search and execution verification
Label semantics	Per-step cheapest-sufficient-tier estimate: the lowest tier whose pool contains a model that correctly handles the current step, verified by end-to-end trajectory pass with fixed step count under the 11-model pool, Opus-initialized sequential-locking downgrade search, and cascade protocol of §4, cross-checked by manual audit
License	Apache-2.0
Intended uses	Training and evaluating step-level LLM routers; benchmarking agent deployment cost reduction
Out-of-scope uses	Claims of absolute routing optimality; deployment to model pools substantially different from the released tier map without re-labeling; use as a general SWE-bench or BFCL capability benchmark
Potential biases	Over-represents agentic code-repair (SWE-bench) at the `high` tier; BFCL/mtRAG/QMSum are nearly saturated at `low`; English-only content; labels are pool- and harness-specific
Maintenance plan	New tier-pool manifests and score protocol versions will be tracked via GitHub releases; tier map and pricing are versioned; current release is v1.0
Human oversight	Final human audit of a ∼10% random step sample from BFCL, SWE-bench, and PinchBench (64 steps total); one SWE-bench step was flagged as further-downgradeable and its verified tier was lowered by one tier before release; no other still-downgradeable label was found in the audited sample
Release URL	Public code and data package: https://github.com/CommonstackAI/TwinRouterBench

License metadata follows upstream documentation as of the release date; users should verify upstream terms before redistributing derived artifacts.

A.2 Corpus overview

Benchmark	Instances	Steps	Steps/trace (median)	Prefix tokens (median)	Scenario
SWE-bench	40	336	9.0	∼5,300	GitHub issue code repair (multi-step agent)
BFCL	130	248	1.0	∼1,600	Function calling (single- + multi-turn)
mtRAG	193	193	1	∼1,900	Multi-turn retrieval-augmented QA
QMSum	145	145	1	∼3,000	Query-focused meeting summarization
PinchBench	12	48	4	∼10,500	23-task general agent suite
Total	**520**	**970**			

Table A2: Corpus overview. Prefix-token medians are measured on the router-visible `messages` field using the Anthropic tokenizer.

A.3 Target-tier distribution and sequential-locking search

For space, we defer the workload target-tier counts (referenced from §3) and the sequential-locking downgrade search pseudocode (referenced from §4) to this appendix.

A.4 Artifact Structure

The reviewer package contains the static JSONL, locked manifests, scoring code, dynamic split, and compact dynamic summaries needed to inspect the benchmark and recompute the reported tables. The expected layout is:

Benchmark	low	mid	mid_high	high
BFCL	239	8	1	0
mtRAG	183	8	1	1
QMSum	132	10	3	0
PinchBench	41	3	3	1
SWE-bench	94	33	41	168
Total	689	62	49	170

Table A3: Target-tier distribution by workload.

Algorithm A1 Sequential-locking downgrade search (per trajectory)

Require: trajectory τ, hints $h_1, \ldots, h_N$, harness $\mathcal{H}$, tiers $T_0 < T_1 < T_2 < T_3$
locked$[i] \leftarrow T_3$ for all i
for $i = 1$ to N **do**
  if h_i = not-downgradeable **then**
    continue
  end if
  for t from h_i **upto** T_2 **do**
    run $\mathcal{H}$ with steps $< i$ locked, step i at t, steps $> i$ at T_3
    if trajectory passes **then**
      locked$[i] \leftarrow t$; **break**
    end if
  end for
end for
return locked

```
data/static/question_bank.jsonl
data/static/manifest.json
data/dynamic/model_pool.json
data/dynamic/model_pricing.json
data/dynamic/tier_to_model.json
data/dynamic/ttl_policy.json
main/eval/
main/metrics.py
README.md
```

The static file contains 970 rows and exposes only tier labels, not vendor model IDs in individual records. Dynamic scoring uses the locked pool, pricing, tier map, and TTL policy files under `data/dynamic/`; held-out SWE-bench IDs and aggregate outputs are provided with the release artifacts. The README gives installation and smoke-test commands for the static grader and dynamic scoring CLI. For E&D-track submission, the release package also includes `metadata/croissant.json` with Croissant core and Responsible AI fields, license and source-workload metadata, and executable smoke-test scripts for the static grader and dynamic-summary scorer.

A.5 Static Model Pool

The static track uses a fixed pool of eleven concrete models grouped into four capability tiers (Table A4). Target tier labels are defined existentially over this pool: during the downgrade-and-cascade procedure of §4, we accept a row as belonging to tier t as soon as any one model in that tier's pool correctly handles the current step, as inferred from the full trajectory still passing with the same number of steps. Public static rows store only the resulting tier label rather than any specific model ID, so routers trained on this data remain portable when the underlying pool is updated.

Table A4: Static model pool (eleven models, four tiers). Accepted aliases for the same logical model are shown in parentheses. Dynamic-track pricing for representative models is given separately in Table A6.

Tier	Models
high	`anthropic/claude-opus-4.6` (alias: `anthropic/claude-opus-4-6`); `openai/gpt-5.4` (alias: `openai/gpt-5.4-2026-03-05`)
mid_high	`anthropic/claude-haiku-4-5`; `google/gemini-3-flash-preview` (alias: `gemini/gemini-3-flash-preview`); `qwen/qwen3.5-397b-a17b`
mid	`minimax/minimax-m2.5`; `qwen/qwen3.5-27b`; `qwen/qwen3-coder`
low	`deepseek/deepseek-v3.2`; `z-ai/glm-4.5-air`; `qwen/qwen3.5-9b`

tier	input ($/1M)	cache_read ($/1M)	cache_write ($/1M)	output ($/1M)
low	0.26	0.13	0.26	0.5
mid	0.30	0.059	0.30	2.0
mid_high	0.50	0.05	0.083	5.0
high	5.0	0.50	6.25	25.0

Table A5: Per-tier accounting constants (USD per million tokens) for the static scoring protocol. These rates are not the prices of any single model; they are representative values derived from the pricing ranges of multiple models in each tier. The `high` tier is 10–50× more expensive than the lower three, which are within 2–5× of each other. Dynamic-pool model prices are in Table A6. This asymmetry drives the ablation in Appendix A.15.

Table A6: Concrete representative models and live OpenRouter unit prices for the TwinRouterBench dynamic pool. These model prices are distinct from the static tier-accounting constants in Table A5. Prices are USD per million tokens; cache-write prices fall back to input prices where the provider does not publish a separate cache-write rate. These are the concrete p values substituted into $c_i(t)$ when scoring the dynamic track; the released scorer ranks the resulting leaderboard bill including the flat unresolved penalty cost (§5.2). The `mid` representative `minimax-m2.7` was selected because it ranks highest among models at a comparable price point on the SWE-bench leaderboard and falls within the same tier as `minimax-m2.5` (the mid-tier representative used during static label construction); the static pool in Table A4 is frozen at the version used when the tier labels were constructed.

Tier	Representative model	Context	Max out	Input	Output	Cache read	Cache write
high	anthropic/claude-opus-4.6	1M	128K	5.00	25.00	0.50	6.25
mid_high	google/gemini-3-flash-preview	1.05M	65.5K	0.50	3.00	0.05	0.0833
mid	minimax/minimax-m2.7	196.6K	196.6K	0.30	1.20	0.059	0.30
low	deepseek/deepseek-v3.2	163.8K	163.8K	0.252	0.378	0.0252	0.252

A.6 Static tier accounting rates

A.7 Dynamic Pool Representative Models and Pricing

A.8 Rule-based UncommonRoute Configuration

Rule-based UncommonRoute is run with `MetadataSignal` and `EmbeddingSignal` in a two-signal ensemble, using the upstream defaults `risk_tolerance`= 0.5, `ensemble_weights`= (0.55, 0.45), and `low_escalation_threshold`= 0.55. Because the benchmark-specific seed set, classifier, and Platt scaler are absent in this rule-based setting, the embedding signal abstains and only the metadata signal contributes in practice.

A.9 Step-Level Cost Case Study: `sympy__sympy-12096`

Table A7 shows a 13-step SWE-bench trajectory. Plan A is all-high execution with prompt caching; Plan B is verified step-level routing (the tier assignments in the "Tier" column). The step-level routing achieves a 39.6% cost reduction relative to all-high while resolving the issue. Last-step output tokens are estimated (∗).

Table A7: Step-level cost case study for `sympy__sympy-12096`. Plan A = all-`high` with prompt caching. Plan B = verified step-level routing. Token counts are in thousands; costs in USD.

Step	Tier	Plan A in	Plan A out	Plan A cost	Plan B in	Plan B out	Plan B cost
1	mid	1,321	112	$0.0111	1,284	108	$0.0005
2	mid	1,744	71	$0.0051	1,699	68	$0.0003
3	mid_high	1,846	69	$0.0032	1,647	66	$0.0003
4	mid_high	2,502	67	$0.0067	2,343	65	$0.0003
5	low	3,287	89	$0.0084	3,729	87	$0.0010
6	mid_high	4,103	256	$0.0131	3,900	254	$0.0010
7	low	4,512	55	$0.0060	5,215	55	$0.0009
8	mid_high	4,612	168	$0.0071	4,403	168	$0.0007
9	high	4,921	241	$0.0103	4,921	241	$0.0368
10	high	5,149	85	$0.0060	5,149	85	$0.0060
11	high	5,591	63	$0.0069	5,591	63	$0.0069
12	low	5,653	56	$0.0046	6,789	59	$0.0018
13	low	5,864	111*	$0.0069	7,077	109*	$0.0010
Total	–	–	–	**$0.0953**	–	–	**$0.0576**

A.10 mtRAG/QMSum Judge Hardening Details

Table A8 documents the changes from the legacy judge rubric to the hardened version used in the final dataset. Each row identifies a specific false-pass failure mode in the legacy judge and the corresponding hardened rule applied in construction.

Table A8: mtRAG/QMSum judge hardening. The changes convert weak reference-similarity judging into a conservative task-success predicate aligned with the current-turn/query requirement and RadBench-style faithfulness criteria [14].

Risk in legacy judge	Hardened rule	Purpose
No primary criterion	First decide whether the answer completely and correctly resolves the current turn/query	Prevents answer/reference surface resemblance from substituting for task success
Reference treated as answer string	Treat reference as coverage hints; evidence/transcript is authoritative	Allows valid paraphrases and rejects reference-following errors
Unsupported facts	Hard fail for unsupported crucial numbers, dates, amounts, ratios, votes, or claims	Suppresses hallucinated but plausible-looking answers
Evidence contradiction	Hard fail when candidate contradicts retrieved passages or meeting transcript	Enforces faithfulness
Unanswerable/no-context cases	mtRAG pre-filter: must be answerable, have context passages, and positive human relevance feedback	Removes structurally unreliable samples before routing search
All tiers fail	Mark discarded; do not emit case JSON or supervision row	Prevents false passes from becoming verified labels

A.11 Frontier LLM-as-Router Diagnostic Summary

Table A9 shows the Opus 4.6 predictions on verified-`high` SWE-bench steps; Table A10 provides additional diagnostics. These numbers support the claim that a strong model's unexecuted routing judgment is not a reliable substitute for execution-verified step-level supervision (§6.1).

Verified tier	$\tilde{t}$ = `low`	$\tilde{t}$ = `mid`	$\tilde{t}$ = `mid_high`	$\tilde{t}$ = `high`	Total
$\hat{t}$ = `high`	34	43	63	**7**	147

Table A9: Opus 4.6 predictions on verified-`high` SWE-bench steps. 140 of 147 are routed below `high`.

Table A10: Frontier LLM-as-router (Opus 4.6) diagnostic summary on SWE-bench. The key finding is severe under-prediction of `high`-tier steps: only 5% of verified-`high` steps are predicted `high`, causing every SWE-bench trajectory to fail under this routing.

Diagnostic	Value	Scope	Interpretation
SWE-bench valid rows	300	336 rows total, 36 malformed	Valid subset for confusion-matrix analysis
Verified-high predicted high	7/147 (4.8%)	SWE-bench valid rows	Severe under-prediction of high-tier steps
Verified-high under-predicted	140/147 (95.2%)	SWE-bench valid rows	Most high steps routed below high
SWE-bench trajectory pass	0/40	Legacy Opus routing run	Every trajectory has ≥ 1 under-routed step
Middle-third under-prediction	73%	Step-position diagnostic	Core editing/test-running steps most under-predicted
First-third under-prediction	46%	Step-position diagnostic	Lower but still substantial under-routing
Last-third under-prediction	53%	Step-position diagnostic	Recovery/finalization steps remain difficult

A.12 Label-Validation Audit Trail

Table A11 documents the validation mechanisms applied during dataset construction. Together they ensure that no label enters the released JSONL unless the strong baseline passed, a tier-pool model actually passes under the mixed-model trace, the judge does not mark the answer as pseudo-passing, and the final human audit sees no surviving issue.

A.13 Manual Audit Protocol and Summary

The manual audit is the final quality gate of the static track: it runs *after* the Opus-initialized downgrade search, tier-pool cascade check, open-ended judge hardening, and prefix reconstruction of §4 are all complete. The audit is deliberately a human-judgment pass rather than a second harness execution. Its purpose is to cross-check that the released tier label is still defensible on the same router-visible context that a deployed router would see, not to recompute any pass predicate.

Table A11: Label-validation audit trail. Each component addresses a specific failure mode in the construction pipeline.

Validation component	Scope	Failure mode addressed	Release action
Successful seed traces	All released instances	Conflating task failure with routing failure	Only passing seed traces enter degradation search
Sequential-locking downgrade search	Multi-step traces	Missing the last-passing tier; hypothetical overlays on the strongest trace	Push one tier below the Opus proposal under the causal mixed-model prefix until the harness fails, then lock the last passing tier
Tier cascade (3 models/tier)	All non-`low` tier labels	Single-model false negatives within a tier	Lock a cheaper tier only after an actual passing run
Strict mtRAG/QMSum judge	Open-ended RAG and summarization	False passes from weak judging	Use hard-fail rubric; discard all-tier failures
Final human audit	∼10% random step samples from each of BFCL, SWE-bench, and PinchBench	Still-downgradeable tier labels that the upstream search did not tighten to the conditional optimum	Human review under the router-visible prefix decides whether the released tier can be lowered by one more step; one SWE-bench step was flagged as further-downgradeable and its tier was lowered accordingly

Sampling. We follow the GT tier-optimality review protocol included with the release artifacts. The audit unit is a single routed step so that single-step and multi-step cases are treated on the same granularity. Within each of the three workloads that contain multi-step trajectories (BFCL, SWE-bench, PinchBench) we draw an independent random sample of roughly 10% of the workload's routed steps. mtRAG and QMSum are single-turn tasks that lack multi-turn compounding complexity; their labels are already constrained by the hardened judge whose pseudo-pass patterns were eliminated through iterative calibration (§4), and are not resampled in this audit.

Review procedure. For each sampled step, the router-visible `messages` prefix is held fixed at its released form. An optional large-model pre-label may be attached as a reference, but the final decision is always taken by a human reviewer, who asks whether lowering the verified tier by one more level would still produce a response that correctly handles the current step; the step is then labelled as *tight*, *uncertain*, or *further-downgradeable*. A *tight* verdict means the released tier is already at the conditional optimum and cannot be lowered one more step without breaking task resolution; a *further-downgradeable* verdict means the reviewer believes the released tier is still too high and the label can be tightened by one more tier. The audit inspected 64 steps in total; one SWE-bench step was flagged as further-downgradeable and its verified tier was lowered by one tier before release, while the remaining 63 steps were judged tight. This internal audit is intended to catch residual labels that the upstream search did not tighten to the conditional optimum, and to clarify label semantics; it is not presented as an external annotation study or as a confidence interval for the full corpus.

Table A12: Manual audit summary. A sampled step is counted as *Downgradeable* when the reviewer believes its verified tier can still be lowered by one more tier without breaking task resolution; *Tight %* is the complementary share whose tier label is already at the conditional optimum. Sampling is done at the step granularity across the three workloads that contain multi-step trajectories; mtRAG and QMSum are single-turn tasks already guarded by the hardened judge and are not resampled here. The audit is a targeted internal quality check.

Slice	Total steps	Sampled steps	Sampling %	Downgradeable	Tight %
BFCL	248	25	10.1	0	100.0
SWE-bench	336	34	10.1	1	97.1
PinchBench	48	5	10.4	0	100.0
Total	632	64	10.1	1	98.4

A.14 Static-Track Breakdown

Per-workload cost savings. Table A13 shows that SWE-bench is the hardest static slice. It carries 34.64% of the row-weighted CostSave score, and every router has negative cost savings there because one under-routed step fails the whole trajectory and triggers the failure-aware bill. On the other four workloads, SR-KNN and ClawRouter (rule-based) both exceed 85% cost savings on average; the global ranking depends on whether a router preserves code-repair trajectories.

Router	BFCL	mtRAG	QMSum	Pinch	SWE-bench	Overall
SR-KNN	**90.49**	**92.35**	90.24	35.36	**-1.64**	**56.18**
ClawRouter (rule-based)	89.87	83.45	**92.12**	**55.20**	-6.55	53.82
UncommonRoute (rule-based)	47.14	91.93	90.24	39.80	-86.08	15.99
Row weight	25.57	19.90	14.95	4.95	34.64	100.00

Table A13: Per-workload CostSave under the static scoring protocol. The SWE-bench slice is the main stress test: failed trajectories trigger the failure-aware numerator in §5.1.

Router failure modes. The three baselines fail in different ways (Table A14). On the 800 rows whose verified tier is below `high`, SR-KNN is mostly exact; ClawRouter (rule-based) almost always routes one tier above the label; UncommonRoute (rule-based) is exact more often than ClawRouter (rule-based) but jumps to `high` 217 times.

Prediction pattern on $\hat{t} <$ `high`	SR-KNN	ClawRouter (rule-based)	UncommonRoute (rule-based)
Exact tier	**627**	107	488
Up one tier	10	**649**	47
Jump to `high`	117	21	**217**
Fail: predicted below $\hat{t}$	46	23	**48**

Table A14: Error patterns on the 800 non-`high` rows.

SWE-bench exposes why row-level behavior is insufficient (Table A15). UncommonRoute (rule-based) recognizes many verified-`high` steps, but missing even one step in an 8–13 call trajectory fails the instance.

Router	Predicted `high` on $\hat{t} =$ `high`	Hit rate	Failed traj.
SR-KNN	137/168	**81.5%**	**10/40**
ClawRouter (rule-based)	7/168	4.2%	38/40
UncommonRoute (rule-based)	105/168	62.5%	39/40

Table A15: SWE-bench high-tier decisions. One under-routed critical step fails the whole trajectory.

A.15 Cost-Savings Accounting Ablation

This section provides the full derivation and per-variant numbers referenced in §6.1.

Pricing asymmetry drives the ablation. Table A16 reports, for three representative step profiles, the per-step saving against the always-`high` baseline under each target tier. The ratio `save_mid / save_low` is within 3% of 1 across all three step sizes, which means a router that mis-routes from $\hat{t} =$ `low` to $\tilde{t} =$ `mid` loses almost no money in absolute terms.

Table A16: Savings vs. always-`high` for different target tiers on three representative step profiles. Mis-routing from `low` to `mid` is nearly cost-free.

Step profile	save(high−low)	save(high−mid)	save(high−mid_high)	save_mid / save_low
4k cold step	3.621¢	3.530¢	3.467¢	0.975
20k warm step	1.965¢	2.032¢	1.900¢	1.034
100k cold step	61.13¢	60.65¢	62.67¢	0.992

Five scoring variants. We compare five N/D accounting schemes on the same predictions:

- **A** (legacy row-level denominator): $D = \sum_i (\text{cost}(3; i) - \text{cost}(\hat{t}_i; i))$ over passing rows with $\hat{t}_i \neq$ `high`; N ignores failures.
- **B** (legacy all-row denominator): D as in A but over all rows; failing trajectories subtract $\sum_i \text{cost}(\tilde{t}_i; i)$ at trajectory level.
- **C** (intermediate high-row exclusion): exclude $\hat{t}_i =$ `high` rows from D.
- **D** ($D = \sum_i \text{cost}(3; i)$, failed-step penalty at step level, no trajectory-level penalty).
- **E** (final failure-aware formula): same D as D, but failed trajectories charge $-\sum_i \text{cost}(\tilde{t}_i; i)$ at trajectory level.

Why variant E recovers the consistent ranking. Variant E interprets a failing trajectory as "the router burned its own (cheap) budget *and* the user paid the full `high` bill to recompute." Formally, the user-side bill on a failing trajectory is $\sum_i \text{cost}(\tilde{t}_i; i) + \sum_i \text{cost}(3; i)$, so savings against always-`high` equal $-\sum_i \text{cost}(\tilde{t}_i; i)$, which is exactly what N encodes. The $\sum_i \text{cost}(3; i)$ retry term is absorbed by D rather than double-counted, keeping CostSave $\in [-\infty, 100]$ with 0 representing always-`high` routing. On the full benchmark, ClawRouter (rule-based) has

Table A17: Ablation of CostSave accounting. Variants A–D all rank ClawRouter (rule-based) above SR-KNN; only variant E (the final failure-aware formula) aligns with RowPass, RowExact, and TrajPass.

Variant	SR-KNN cost%	ClawRouter (rule-based) cost%	Ranking
A (legacy row-level denominator)	84.99	**89.71**	Claw wins
B (legacy all-row denominator)	84.45	**84.59**	Claw wins
C (intermediate high-row exclusion)	79.11	**91.31**	Claw wins
D ($D = \sum \text{cost}(3)$, step-level fail)	61.81	**67.67**	Claw wins
E (final failure-aware formula)	**56.18**	53.82	SR wins

649 up-one errors (near-free in absolute terms) that can no longer compensate for its 38 failing SWE-bench trajectories, and its CostSave aligns with its RowPass/RowExact/TrajPass figures.

A.16 Opus 4.6 Confusion Matrix

For reference, the Opus 4.6 routing run recorded under an earlier three-metric accounting protocol reports an overall of 84.05, with row pass =85.77, row exact =82.88, and cost savings =83.48. This is not directly comparable to the static scores in Table 1 because the earlier accounting does not apply the failure-aware trajectory penalty in §5.1 and does not report a trajectory-pass metric; we therefore treat it as a historical data point rather than a ranked baseline.

Table A18 shows the full confusion matrix summarized in §6.1.

Table A18: Opus 4.6 routing predictions vs. verified tiers on the 300 valid SWE-bench rows (36 malformed outputs excluded). Diagonal cells are highlighted. Opus rarely commits to `high`, even when the verified tier is `high`.

	$\tilde{t}$ = low	$\tilde{t}$ = mid	$\tilde{t}$ = mid_high	$\tilde{t}$ = high	row total
$\hat{t}$ = low	**52 (61%)**	15	17	1	85
$\hat{t}$ = mid	13	**7 (23%)**	10	0	30
$\hat{t}$ = mid_high	8	13	**16 (42%)**	1	38
$\hat{t}$ = high	34	43	63	**7 (5%)**	147

A.17 mtRAG Row Example

A second row format, complementing the SWE-bench example in §3, illustrates how `low`-tier labels arise: after several retrieval turns, a short follow-up query can be answered by the cheapest tier in the pool.

```
{
  "id": "mtrag_..._turn_3_step_1",
  "benchmark": "mtrag",
  "step_index": 1, "total_steps": 1,
  "messages": [
    {"role": "system", "content": "Answer based on the retrieved passages..."},
    {"role": "user",   "content": "[Passage] Major.minor update... [Passage] ..."},
    {"role": "assistant", "content": "..."},
    ...multi-turn history...
    {"role": "user",   "content": "Major.minor update."}
  ],
  "target_tier": "low", "target_tier_id": 0
}
```