MERA: Model Evolution and Routing with Skill Adaptation for Agentic Systems at Scale

Yuhang Yao[6] Zeyu Wang[10] Tongyun Yang[3] Wanyi Chen[2] Yuhang Han[7]
Jie Xiao[1] Tianyu Shi[1,†]
[1]Gradient [2]Soochow University [3]Independent Researcher [6]Carnegie Mellon University [7]Shanghai Jiao Tong University [10]University of California, Los Angeles

Abstract

Language-model agents increasingly mix strong but expensive frontier models with cheaper models that are useful only on safe subsets of a workflow. The challenge is not only to choose a model once per user request, but to adapt many individual invocations inside a multi-step trace without silently degrading quality. We present MERA, a trace-driven framework that jointly evolves three tracks: SkillBook statistics for recurring prompt signatures, a learned invocation-level router, and a small-model adapter. The main empirical finding is that the joint schedule matters: in a code-generation setting with 590 executable code tasks and 3,328 weakly labelled router examples, the best four-cycle order is Skill → LLM → Router. This setting reaches 87.3% router accuracy with 4.4% fallback and reduces estimated serving cost to 51.8% of always using the large model. An eight-cycle run peaks at 87.8% router accuracy and then stabilizes in the 82–87% band. Component behavior is uneven but informative: SkillBook and router updates provide the largest cost-quality gains, while the 1.5B GRPO adapter gives a modest pass-rate improvement, reaching 47.5% on MBPP eval200 versus roughly 47.0% for the base setting, but does not yet show cumulative gains across cycles. MERA therefore frames agent self-evolution as a conservative systems loop over shared traces, where skill, model, and routing updates are admitted together through replay.[1]

Keywords

agentic systems, model routing, skill adaptation, model evolution

1 Introduction

Language-model agents execute heterogeneous chains of model calls. A single task may contain hard reasoning or policy-sensitive decisions, but it also contains many structured steps: extraction, formatting, tool-argument construction, post-processing, clarification, and templated summaries. Running every invocation on the strongest model is reliable but expensive. Conversely, routing only once at the user-task level is too coarse, because the easy and hard parts of a workflow often appear side by side. Recent work has improved agents through prompting, tool use, planning, search, and agent optimization [8, 9, 16, 20–22], but production systems also need mechanisms for adapting the models, routes, and reusable skills inside the workflow.

This creates a practical tension for deployed agent systems. The traces needed for adaptation are naturally produced online, but directly changing the serving policy from raw traces is risky: a cheaper model may fail silently, a learned adapter may regress on rare cases, and a reusable template may only be safe under a narrow prompt signature. A useful evolution loop therefore needs to separate observation from admission. It should collect evidence at the granularity of individual invocations, update several candidate components, and admit the resulting runtime state only when replay shows that the combined policy still satisfies verification constraints.

MERA addresses this problem by treating a single model invocation as the unit of adaptation. At runtime, an input-only router chooses among strong, cheap, and specialized models; a skill layer can dispatch stable templates for recurring local structure; and a verifier protects quality through fallback. This design keeps serving simple: the router does not depend on hidden agent state, and unsafe down-routing is corrected by verification. The heavier logic is moved offline, where traces are replayed to decide which prompts are easy, which failures should become training examples, and which recurrent patterns are stable enough to become skills.

Figure 1 shows the resulting feedback loop. Online traces drive update rounds in which SkillBook statistics, router training, and LLM adaptation share evidence. The tracks need not execute in a fixed order: SkillBook is CPU-bound, LLM adaptation is GPU-heavy, and router-last schedules can consume current-cycle traces. The combined state is admitted only after joint replay evaluation, so routing, skill promotion, and adapter updates are judged by their end-to-end effect rather than by isolated component metrics.

Our contribution is a systems framework and empirical study of jointly optimizing model evolution, routing, and skill adaptation in agentic pipelines. We instantiate MERA in code generation with an Evolver Dataset that separates executable code tasks from router-only weak supervision, and we evaluate four-cycle ordering, eight-cycle iteration, cost, fallback, and LLM pass rate. We find that the Skill → LLM → Router schedule is the strongest current operating point: skill evidence and learned routing produce the largest cost-quality improvement, and LLM adaptation adds a smaller pass-rate gain but has not yet accumulated across cycles. This makes MERA useful as both a deployment pattern and a diagnostic harness for deciding which update track is actually improving.

2 Related Work

LLM routing. Prior work studies cost-quality dispatch across model pools, using prompt features, preference data, or explicit routing and cascading objectives [1–3, 7]. These methods are closest to MERA, but they usually route whole prompts or single-turn requests. MERA routes at the invocation level inside multi-step agent traces, where adjacent calls may have very different difficulty. It also treats routing as an evolving component: new router states are admitted only after verifier-backed replay.

[1]Code and results available at https://github.com/zeyuyuyu/router-skills-evolve

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

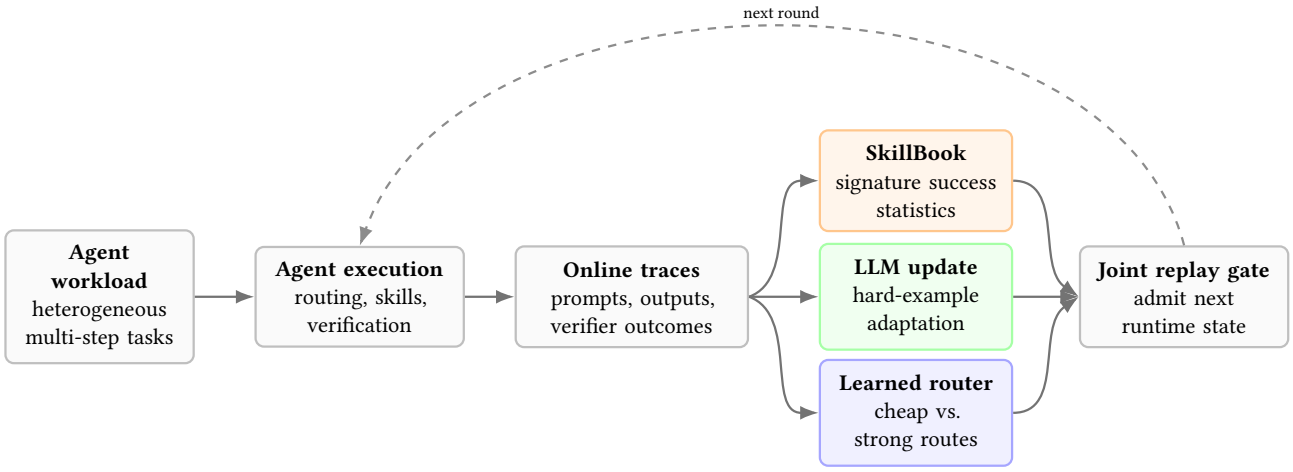

Figure 1: Overview of MERA. Online traces drive scheduled SkillBook, LLM-update, and router tracks; their combined state is admitted through joint replay evaluation.

Agent optimization. Agent research has emphasized tool use, planning, execution-time search, and optimizing agent graphs or functions [8, 9, 16, 20–22]. This line of work improves what an agent can do. MERA asks a complementary systems question: once an agent is running, how should its traces update the deployed model mix, routing policy, and reusable skills? The focus is therefore not on inventing a new planner, but on reducing the cost of repeated agent execution without removing fallback protection.

Model specialization and skills. Distillation, step-by-step training, reflective self-training, and skill-library construction adapt smaller models or externalize reusable behavior [4, 11, 18]. MERA uses similar ingredients but changes the admission criterion. A student model or skill is useful only on the slice where routing and verification show that it is safe. This makes specialization a deployment decision coupled to routing, not a standalone model-improvement claim. A fuller discussion appears in Appendix A.

3 Method

3.1 Runtime Routing

MERA separates the serving path from the update path. At runtime, the router observes only the serialized prompt for the current invocation and selects among a strong model, a cheap model, and optionally a specialized student. A skill selector may dispatch stable templates for recurring local structure, such as formatting, extraction, or single-tool argument construction. The selected model produces an output, and a verifier checks schema validity, tool-call legality, executable tests, or downstream success. If verification fails, the invocation falls back to a stronger model and the full event is logged.

This input-only router is intentionally restrictive. It avoids coupling routing decisions to hidden agent state or implementation-specific tool traces, making the interface easier to deploy across agent harnesses. Reliability is instead protected after execution by the verifier and fallback path. The consequence is that MERA can start conservatively: early routers may prioritize low unsafe down-routing, while replay and SkillBook evidence gradually identify regions that can be served cheaply.

3.2 Trace Products

The update loop operates on complete traces collected from runtime execution or replay. MERA canonicalizes each trace into step slices containing the prompt, local context, tool schemas, generated output, verifier result, retry count, fallback metadata, and any skill assignment. These slices produce three evidence streams. SkillBook records success and failure statistics for recurring prompt signatures. The learned router trains on prompt text with cheap/strong labels derived from weak supervision or replay. The LLM adapter trains on selected hard examples where cheaper execution fails or where the router and SkillBook disagree.

The same slice can therefore support different updates without forcing the components to share the same supervision format. Router examples can remain input-only, while LLM examples preserve the context needed to execute the step. Skill examples are grouped by repeated local structure rather than by label alone.

3.3 Scheduled Updates

Algorithm 1 summarizes the loop. Each update round refreshes the three tracks over shared traces, but the order is a scheduling choice rather than a methodological constraint. SkillBook updates are CPU-side statistics and can run beside model training. LLM adaptation is GPU-heavy and uses the hard-example pool. Router training can be placed last when its labels should include current-cycle traces or adapter outputs. The ordering ablation in Section 4 changes only this schedule while holding chunks, objectives, and held-out evaluations fixed.

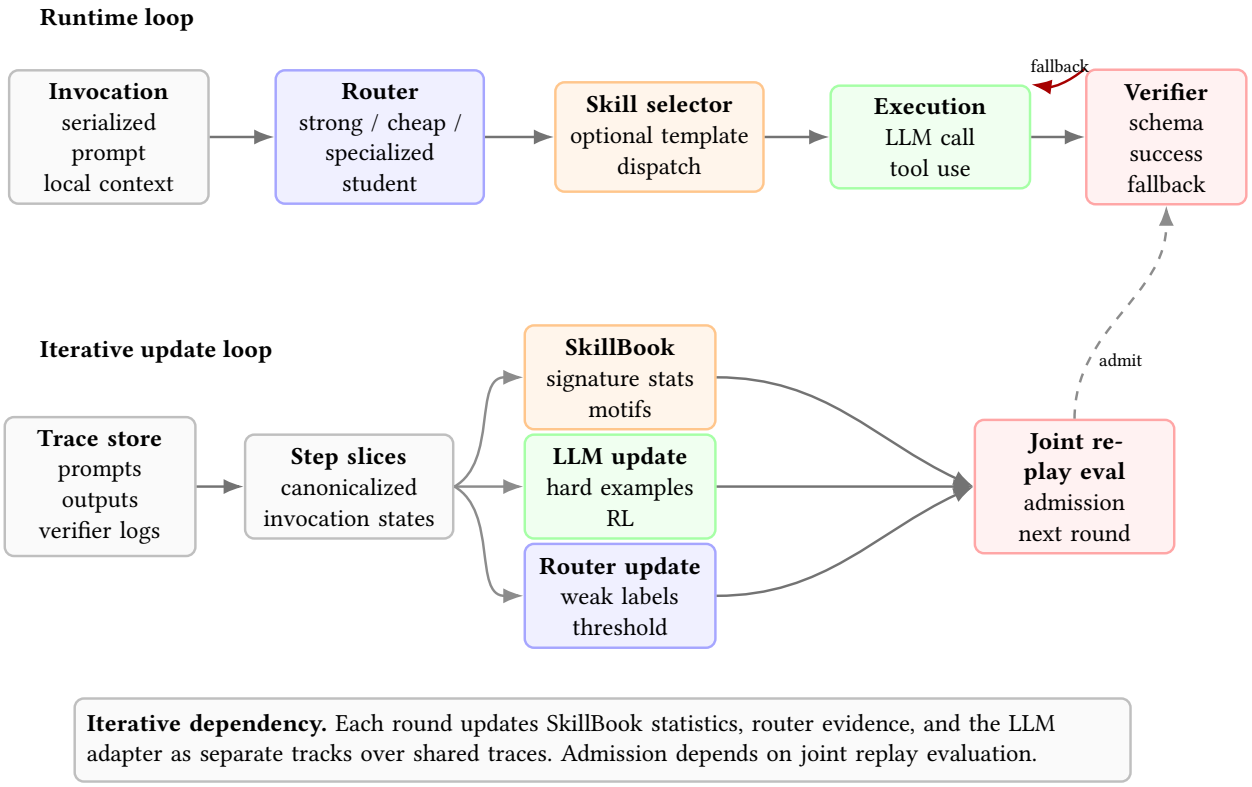

Figure 2: Detailed method view of MERA. Runtime routing remains input-only and verifier-protected; update tracks share traces and are admitted through joint replay.

Algorithm 1 MERA runtime and iterative update loop

Require: router R, model registry $\mathcal{M}$, SkillBook $\mathcal{K}$, verifier V

for each agent invocation x_t **do**
  choose model $m_t \leftarrow R(x_t)$ and optional skill $k_t \in \mathcal{K}$
  execute $y_t \leftarrow m_t(x_t, k_t)$
  if $V(x_t, y_t)$ fails **then**
    fallback to a stronger model
  end if
  log $(x_t, y_t, m_t, k_t, V(x_t, y_t))$
end for
for each update round **do**
  canonicalize traces into step slices
  update SkillBook, router, and LLM adapter under chosen schedule
  run joint replay; admit only if quality is preserved
end for

This design lets MERA distinguish scientific effects from systems effects. Figure 2 makes the dependency structure explicit: runtime execution produces trace slices, the three update tracks consume different evidence products from those slices, and replay admits only the combined state. If LLM pass rate is invariant across orderings, ordering is not changing what the adapter learns. If wall time improves under parallel schedules, the gain is a systems scheduling gain. If router accuracy changes, the change must be interpreted with seed sensitivity and replay labels in mind.

3.4 Joint Admission

MERA admits updates through joint replay rather than isolated component metrics. SkillBook, router, and verifier evidence define easy, hard, and uncertain regions. Easy regions are candidates for cheap serving or future student admission; hard examples feed the LLM update; uncertain regions remain protected by fallback. A new router, skill state, or adapter is promoted only if replay preserves quality while reducing cost or fallback risk.

This yields a conservative monotone deployment strategy. The runtime system can continue using strong-model fallback while update rounds search for cheaper safe regions. When an update does not improve the joint replay result, it remains an experimental artifact rather than entering the serving registry.

4 Experiments

4.1 Experimental Setup

We evaluate whether the full iterative stack improves after repeated updates, how sensitive the loop is to within-cycle ordering, and whether longer runs continue to improve. The Evolver Dataset contains 590 executable code tasks from HumanEval, MBPP, and hard traces, plus 3,328 weakly labelled router examples from UncommonRoute; split details are in Table 3. Each cycle updates SkillBook statistics, a BERT-tiny router checkpoint, and a Qwen2.5-Coder-1.5B LoRA adapter trained with local GRPO, then evaluates the combined state. Router metrics are measured on a fixed

Table 1: Four-cycle joint ordering ablation. Router metrics use a fixed 832-example held-out set; LLM pass is MBPP eval200.

Ordering	Router acc. (%)	Fallback (%)	Cost vs. large (%)	LLM pass (%)	Wall time
Skill → Router → LLM	81.6	3.9	57.7	47.5	99 min
Skill ‖ LLM → Router	82.7	4.1	56.4	47.5	99 min
Skill ‖ Router → LLM	76.0	3.8	63.2	47.0	100 min
Skill ‖ Router ‖ LLM	80.7	4.7	57.8	47.0	**95 min**
Skill → LLM → Router	**87.3**	4.4	**51.8**	46.5	99 min
LLM → Skill → Router	84.6	4.1	54.5	47.0	97 min

Table 2: Eight-cycle ablation for the Skill → LLM → Router schedule. This is one long-horizon execution, not eight independent repeated runs.

Cycle	Router acc. (%)	Fallback (%)	Cost vs. large (%)	LLM pass (%)
1	57.6	0.4	85.6	47.0
2	71.1	4.3	68.6	47.0
3	**87.8**	3.4	**52.8**	47.0
4	82.3	3.5	57.4	47.0
5	76.7	14.4	54.2	47.0
6	85.3	3.1	56.0	46.5
7	82.7	4.6	56.6	47.0
8	85.0	4.0	54.3	**47.5**

832-example held-out router set, and LLM pass rate is measured on MBPP eval200. The ordering scripts vary only the within-cycle schedule; data chunks, held-out sets, and objectives are held fixed.

We report four quantities because they correspond to different deployment risks. Router accuracy measures whether the learned gate matches held-out cheap/strong labels. Fallback is the fraction of requests rejected by verification and rerun on the stronger path. Cost is normalized to always using the large model, so lower is better. LLM pass rate isolates the small-model adapter on executable MBPP tasks; it is not used as the sole admission criterion because the deployed system also depends on routing and verifier behavior.

4.2 Four-Cycle Joint Result

Table 1 is the main joint result. We use four cycles because it is long enough for each track to update repeatedly while still keeping the comparison across orderings controlled. The strongest operating point is **Skill → LLM → Router**, which reaches 87.3% router accuracy, 4.4% fallback, and 51.8% cost relative to always using the large model. This means the system preserves a low fallback rate while routing enough invocations away from the expensive model to reduce estimated serving cost by 48.2%. We therefore use this row as the current "Full" system setting.

The ordering result should be interpreted as a scheduling study, not as evidence that the LLM learns different content under different orders. LLM pass rate is nearly invariant across orderings, ranging only from 46.5% to 47.5%. This is expected because the LLM track trains on the same static MBPP chunks under each schedule. The router spread is larger, from 76.0% to 87.3%, but the BERT-tiny router is sensitive to random initialization; attributing this spread to ordering alone would require multi-seed averaging. The only clear wall-time gain comes from running the GPU-heavy router and LLM tracks in parallel, which reduces four-cycle time from about 99 minutes to 95 minutes. Because SkillBook is CPU-bound and sub-second relative to model training, parallelizing it changes scheduling cleanliness more than total time.

4.3 Long-Horizon Cycle Ablation

Table 2 asks whether the joint loop continues improving beyond the four-cycle setting. The answer is mixed. The router shows a two-phase pattern: it warms up over the first two cycles, peaks at 87.8% in cycle 3, and then remains mostly in the 82–87% band. The cycle-5 drop is consistent with a single router-seed outlier rather than a persistent failure mode. SkillBook coverage continues to expand, growing to 34 signatures and 120 observations by the end of the run. By contrast, the LLM track stays in a narrow 46.5–47.5% band across trained adapters. Additional cycles therefore improve coverage and test stability, but the current GRPO recipe does not yet produce cumulative LLM gains.

4.4 Discussion

The experiments support a conservative deployment strategy. Each cycle should refresh low-risk SkillBook and router evidence and evaluate any LLM adapter only through joint replay before admission. The exact within-cycle order is mostly a scheduling choice: SkillBook can run beside model training, router-last schedules are natural when router labels include current-cycle traces, and full parallelization mainly saves wall time. Model evolution should be admitted more cautiously: in the current joint-cycle setting, the GRPO adapter does not produce cumulative pass-rate gains. The next experimental step is to run multi-seed ordering studies, add KL or reference stabilization to the RL objective, and report larger held-out evaluations.

5 Conclusion

MERA treats execution traces as shared supervision for per-invocation routing, SkillBook adaptation, and small-model updates. The empirical result is deliberately conservative: the Skill → LLM → Router schedule is strongest, with learned routing reaching 87.3% accuracy, 4.4% fallback, and 51.8% estimated cost in the best four-cycle setting. Skill and router updates drive the largest gains, while the current 1.5B GRPO adapter gives a modest best-cycle pass-rate improvement but not yet cumulative growth. Future work should add multi-seed ordering studies, stronger RL stabilization, and larger online evaluations.

References

[1] 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

[2] Jasper Dekoninck, Maximilian Baader, and Martin Vechev. 2025. A Unified Approach to Routing and Cascading for LLMs. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, 12987–13010. https://proceedings.mlr.press/v267/dekoninck25a.html

[3] Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. 2025. BEST-Route: Adaptive LLM Routing with Test-Time Optimal Compute. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, 13870–13884. https://proceedings.mlr.press/v267/ding25d.html

[4] Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, Toronto, Canada, 8003–8017. doi:10.18653/v1/2023.findings-acl.507

[5] 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)*. https://openreview.net/forum?id=VTF8yNQM66

[6] Injae Na, Keonwoong Noh, and Woohwan Jung. 2025. Automatic Transmission for LLM Tiers: Optimizing Cost and Accuracy in Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, Vienna, Austria, 16987–17004. doi:10.18653/v1/2025.findings-acl.873

[7] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2024. RouteLLM: Learning to Route LLMs with Preference Data. *arXiv preprint arXiv:2406.18665* (2024). doi:10.48550/arXiv.2406.18665

[8] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*. https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html

[9] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. https://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[10] Claudio Spiess, Mandana Vaziri, Louis Mandel, and Martin Hirzel. 2025. AutoPDL: Automatic Prompt Optimization for LLM Agents. In *Proceedings of the Fourth International Conference on Automated Machine Learning (Proceedings of Machine Learning Research, Vol. 293)*. PMLR, 13/1–20. https://proceedings.mlr.press/v293/spiess25a.html

[11] Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. 2026. SkillX: Automatically Constructing Skill Knowledge Bases for Agents. *arXiv preprint arXiv:2604.04804* (2026). doi:10.48550/arXiv.2604.04804

[12] Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Jiaqi Liu, Honglin Guo, Yajie Yang, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. 2026. AgentGym-RL: An Open-Source Framework to Train LLM Agents for Long-Horizon Decision Making via Multi-Turn RL. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=ZgCCDwcGwn

[13] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024) Datasets and Benchmarks Track*. https://neurips.cc/virtual/2024/poster/97468

[14] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. *arXiv preprint arXiv:2207.01206* (2022). doi:10.48550/arXiv.2207.01206

[15] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-Bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=roNSXZpUDN

[16] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*. https://openreview.net/forum?id=WE_vluYUL-X

[17] Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. 2025. ToolHop: A Query-Driven Benchmark for Evaluating Large Language Models in Multi-Hop Tool Use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vienna, Austria, 2995–3021. doi:10.18653/v1/2025.acl-long.150

[18] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-R: Training Language Model Agents to Reflect via Iterative Self-Training. *arXiv preprint arXiv:2501.11425* (2025). doi:10.48550/arXiv.2501.11425

[19] Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025. Router-R1: Teaching LLMs Multi-Round Routing and Aggregation via Reinforcement Learning. https://openreview.net/forum?id=DWf4vroKWJ NeurIPS 2025 poster.

[20] Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. 2024. Offline Training of Language Model Agents with Functions as Learnable Weights. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 60315–60335. https://proceedings.mlr.press/v235/zhang24cd.html

[21] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 62138–62160. https://proceedings.mlr.press/v235/zhou24r.html

[22] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language Agents as Optimizable Graphs. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 62743–62767. https://proceedings.mlr.press/v235/zhuge24a.html

A Detailed Related Work

Our work lies at the intersection of agent systems, LLM routing, model specialization, and skill adaptation. Prior work on *agent systems* has largely emphasized planning, tool use, execution-time search, automated agent configuration, and agent post-training [8–10, 12, 16, 20–22]. This line of research clarifies how to make agents more capable, but it typically assumes that model choice is fixed or handled by simple heuristics. In contrast, MERA treats model allocation and reusable skill structure as first-class systems problems and uses execution traces as the shared substrate for routing, specialization, and skill updates.

The most closely related direction is *LLM routing*. Existing routers generally predict which model should answer a query based on prompt features, expected quality, or cost-quality trade-offs [1–3, 7]. Recent work has also studied progressively upgraded model tiers and multi-round routing and aggregation rather than one-shot dispatch [6, 19]. These approaches are effective for single-turn tasks or pooled-model inference, but agent settings introduce a different granularity. The relevant decision is often not "which model should handle this user request," but "which model should handle this specific invocation within a multi-step trace." MERA adopts invocation-level routing because it is fine-grained enough to capture heterogeneous step difficulty while remaining simple enough to deploy in production.

Model specialization and skill construction are also closely related. Prior work studies distillation, small-model adaptation, reflective self-training, and explicit skill libraries [4, 11, 18]. The standard objective in this area is to train a smaller model to imitate a larger one, to improve agents through iterative self-improvement, or to externalize reusable procedures into portable skill libraries. Our setting differs in two ways. First, the target is not a generic downstream benchmark, but recurring step types extracted from real agent traces. Second, student deployment is conditional: the student is introduced only for steps that the router and verifier deem safe. This turns student training into a systems component rather than a standalone model-compression exercise. Our treatment of skills is likewise more operational than program-synthesis-style skill discovery: skills are promoted only when they expose stable, verifiable structure that improves deployment efficiency.

Evaluation and benchmarks provide the final piece. Recent benchmarks cover grounded web interaction, real-world tool use, multi-hop tool use, tool-agent-user interaction, multimodal computer control, and software engineering [5, 8, 13–15, 17]. These settings matter because correctness can often be externally checked. MERA builds on this observation by using a shared verifier stack for both online protection and periodic model and skill updates. The verifier is not an auxiliary metric; it is the mechanism that connects replay, routing supervision, student selection, skill promotion, and safe production rollout. Across these benchmarks, performance degrades as tool ecosystems become larger, action spaces become more compositional, workflows require longer dependency chains, and execution environments become less scripted. MERA takes this as a design constraint: instead of attempting full agent replacement from the start, it isolates easy, reusable, and automatically checkable steps, then expands the role of specialized students and explicit skills only after routing and verification are stable.

B Additional Experiment Details

Table 3: Evolver Dataset summary. Code tasks are executable; router tasks are weakly labelled prompts for route supervision.

Split family	Train	Dev	Test
Code tasks	472	59	59
Router tasks	2662	332	334

Table 4: Router behavior within the cumulative-slice joint-cycle setting. Lower fallback and lower cost are better.

Joint-cycle variant	Acc. (%) ↑	Fallback (%) ↓	Cost (%) ↓
4-cycle `skill_first`, cycle 4	**87.3**	4.4	**51.8**
4-cycle `xiaojie`, cycle 4	82.7	4.1	56.4
4-cycle `llm_first`, cycle 4	84.6	4.1	54.5
8-cycle iterated, cycle 3 peak	**87.8**	**3.4**	52.8
8-cycle iterated, cycles 4–8 mean	84.3	5.3	55.7

C Limitations

MERA is designed around replay, verification, and conservative staged deployment, and these choices introduce corresponding limitations. First, the quality of both routing labels and student admission decisions depends on verifier coverage. If the verifier fails to capture an important semantic failure mode, replay may overestimate the safety of down-routing or skill promotion. Second, our simple-step-first strategy is intentionally biased toward narrow and highly checkable slices. This makes early deployment safer, but it also means that the framework may realize its gains gradually and may leave a substantial fraction of difficult long-horizon reasoning on the strongest model for a long time.

Third, the current evaluation protocol is trace-centric rather than fully online. Replay is attractive because it enables controlled counterfactual evaluation of routing, student models, and skills under a shared verifier, but replay cannot perfectly capture distribution shift induced by changing the runtime policy itself. Fourth, skill promotion assumes that repeated local subgraphs of agent behavior can be identified and canonicalized into stable templates. Some tasks may remain too heterogeneous for this representation to pay off. Finally, the

empirical benefits of MERA depend on workload repetition: if the trace distribution is highly non-stationary or contains few reusable step types, the gains from specialization and skill adaptation may be limited.

D Broader Impact

MERA aims to reduce the cost of agentic systems without giving up end-to-end reliability. A positive consequence is that stronger agent workflows may become deployable at lower serving cost, which could make high-quality automation more accessible. The same trace-centric design may also improve operational safety by requiring replay, verification, and fallback before new routing or specialization decisions reach production.

The same capabilities also carry risks. More efficient agents may accelerate large-scale automation in settings where reliability, privacy, or oversight matter. If verification is incomplete, down-routing or skill promotion could create subtle failures that are cheap to execute but costly to detect. The framework could also be used to lower the operating cost of agents in sensitive domains without adequate human review. These risks suggest that deployment should remain conservative, with explicit verifier coverage, staged admission, audit logs, and scope restrictions on production use.

E LLM Usage

Large language models are a core methodological component of this work. They are used both as runtime policies within the agent and as the objects being routed, specialized, and evaluated. The method further relies on replay across multiple candidate models to generate routing labels and admission decisions. Our use of LLMs is therefore part of the scientific contribution itself rather than a writing-only aid.