RIDGE: ROLE-ADAPTIVE ROUTING FOR LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems (MASs) can support complex iterative workflows. However, they commonly face a challenging trade-off between performance and cost. Existing systems typically assign a fixed model to each agent role, ignoring variations in subtask difficulty. Furthermore, current LLM routers are mainly designed for single-turn conversations and rely on paired model outputs that are difficult to obtain in multi-agent workflows. To address these issues, we propose RIDGE, a lightweight router that dynamically identifies model capability boundaries during task execution. It distills online Critic evaluations and forensic attribution into audit-calibrated, six-dimensional execution-risk states predicted by a role-conditioned hypernetwork for fast and strong models. An online MemoryBank further supports routing decisions. On SWE-bench Lite and WebArena-Verified, RIDGE matches the success rate of a static strong-model baseline while reducing strong-model token usage by up to 41.6%. It also routes 76.19% of failure-critical steps to the strong model, compared with 21.99% for OmniRouter and 15.06% for RouteLLM.

1 INTRODUCTION

LLM-based multi-agent systems (MASs) must balance reliability and cost: strong models are expensive, while fast-model errors can propagate through agent dependencies. Efficient routing therefore requires identifying steps beyond fast-model capability and selectively allocating strong-model resources. Figure 2 contrasts fixed role-level assignment with dynamic step-level routing.

Existing routers inadequately capture capability boundaries across roles and execution steps. Surface heuristics overlook semantic difficulty; fast-first cascades such as FrugalGPT Chen et al. (2024) risk propagating errors when intermediate outputs enter shared context before escalation. Preference routers such as RouteLLM Ong et al. (2025) rely primarily on single-turn pairwise feedback, whereas multi-round MAS trajectories complicate aligned supervision and step-level outcome attribution.

We propose RIDGE, a role-adaptive router that estimates role- and semantics-conditioned capability boundaries through audit-calibrated, six-dimensional risk predictions for fast and strong models. By comparing predicted risks, uncertainties, and capability gains, RIDGE distinguishes steps within the fast model's competence, near its boundary, or requiring the strong model.

Our main contributions are as follows:

(1) Boundary formulation: step-level routing as role- and semantics-conditioned boundary estimation with six-dimensional risks for both models; **(2) architecture:** a gated role-conditioned hypernetwork with separate fast and strong heads; **(3) supervision:** continuous Critic–forensic signals that distinguish causal failures from local imperfections; **(4) protocols:** role-structured SWE-bench Lite and WebArena-Verified workflows that preserve official tasks and evaluators while exposing assignments, costs, and failure-critical decisions Jimenez et al. (2024); hattami et al. (2025); and **(5) results:** strong-baseline performance with up to 41.6% fewer strong-model tokens and 76.19% failure-critical-step escalation, versus 21.99% for OmniRouter and 15.06% for RouteLLM.

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.

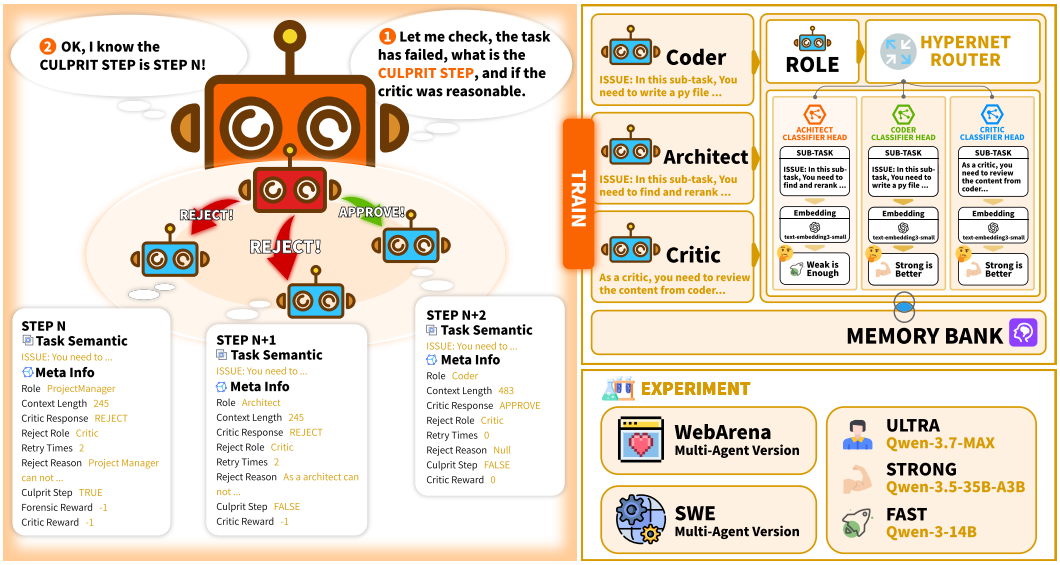

Figure 1: **End-to-end RIDGE workflow from training to inference.** During training, a state loss preserves the structured operational-risk profile, while a boundary loss distills continuous Critic–forensic supervision. During inference, role-specific heads estimate audit-calibrated execution-risk states for fast and strong models.

2 Related Work

2.1 Multi-Agent Systems

Reasoning and self-correction methods such as Chain-of-Thought, ReAct, and Reflexion underpin multi-agent systems that coordinate specialized agents for tasks including software engineering and code generation Wei et al. (2022); Yao et al. (2023); Shinn et al. (2023); He et al. (2025); Ishibashi & Nishimura (2024). Recent methods optimize workflow structure and resource allocation: MaAS samples query-dependent architectures from an agentic supernet; TALM, DyTopo, and AgentConductor adapt decomposition, topology, or interaction graphs; and other orchestration methods adjust communication and collaboration Zhang et al. (2025a); Shen & Joung (2026); Lu et al. (2026); Wang et al. (2026); Zhang et al. (2025b); Anonymous (2026).

2.2 Large Language Model Routing

Model routing balances capability and cost. FrugalGPT cascades from a cheaper to a stronger model but adds latency and may propagate errors before escalation Chen et al. (2024). RouteLLM and OmniRouter learn query-level preferences or utilities, while PILOT adapts routing from online bandit feedback under budget constraints; these methods remain role-agnostic and obscure model-specific step risk Ong et al. (2025); Mei et al. (2025); Panda et al. (2025). MasRouter jointly selects collaboration modes, roles, and LLM backbones, but mainly configures task-level systems before execution Yue et al. (2025).

Existing routers focus on isolated queries or pre-execution configuration, overlooking role-conditioned, model-specific step risk and multi-agent workflow signals. RIDGE addresses this gap through step-level capability-boundary estimation.

3 Method

We propose **RIDGE**, a role-adaptive router that predicts audit-calibrated execution-risk states for fast and strong models. By identifying requests near or beyond the fast model's capability boundary,

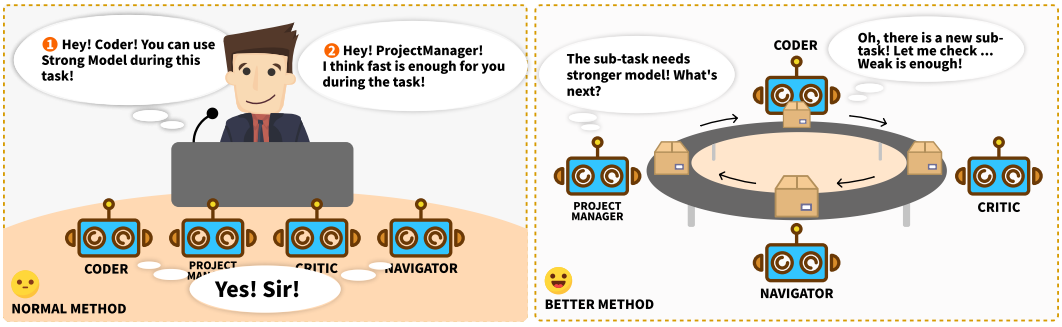

Figure 2: **Model allocation in multi-agent systems.** Top: fixed model assignment by role. Bottom: dynamic step-level routing based on each subtask.

it reduces unnecessary strong-model calls while preserving reliability. Figure 1 summarizes the end-to-end workflow.

RIDGE has four components: 1) semantic-core extraction and role-conditioned hypernetwork inputs; 2) role-specific heads predicting both models' six-dimensional risk states and uncertainties; 3) reliability enhancement via same-role, same-model semantic evidence and MemoryBank refinement; and 4) validation-calibrated routing.

3.1 Problem Formulation and Model Capability Boundaries

Consider a multi-agent system in which multiple roles collaborate on a task. For the t-th routable model call, let x_t denote the current semantic request, $r_t \in \mathcal{R}$ the current role, and

$$\mathcal{M} = \{M_f, M_s\}, \tag{1}$$

the candidate set, where M_f and M_s denote the fast and strong models. RIDGE selects an LLM label $\ell_t \in \{f, s\}$ and executes M_{ℓ_t}. Deterministic scripts, components without LLM decisions, and system-specified fixed-model calls are excluded from routing.

Within this setting, we define the model capability boundary as the point beyond which the fast model becomes unreliable, requiring the strong model. It is not a fixed difficulty threshold shared across roles. RIDGE locates this boundary by jointly considering model-specific risks, uncertainties, semantic support, and role-level calibration. The corresponding prediction signals and calibration parameters are detailed in the following subsections.

3.2 Step-Level Training Samples and Execution-Risk State

Obtaining reliable step-level supervision is difficult in long, stochastic multi-agent workflows because final task outcomes do not clearly reveal the quality of each individual step Zhang et al. (2025c). Process reward models therefore evaluate intermediate agent decisions using progress-oriented signals Xi et al. (2026). RIDGE combines local evaluation from an online Critic with post-hoc forensic attribution of failed trajectories.

Local evaluation by the online Critic. The Critic acts as a fixed strong-model judge whose approve/reject decisions provide local evidence about the capability of the executed model branch Zheng et al. (2023). Its evaluations are incorporated into the hypernetwork through the step-level audit score and Critic-derived workflow signals.

Forensic auditing and attribution of failure-critical steps. For each failed task, an independent forensic Judge reviews a denoised trajectory, final environment state, objective, and verifiable evidence. Preprocessing retains role-visible inputs, outputs, and execution evidence while compressing repeated templates and loops, as detailed in the supplementary material. The Judge verifies the failure and attributes only directly causal steps, assigning each a severity score r_i^F and rationale; other steps are not penalized merely because the task failed. We then froze the protocol before large-scale data collection.

The forensic Judge provides outcome-level attribution, while the Critic supplies a local process signal r_i^C. Their weighted step-level supervision is

$$\rho_i = \mathrm{clip}\left(\lambda_F r_i^F + \lambda_C r_i^C, -1, 1\right), \tag{2}$$

where $\lambda_F + \lambda_C = 1$. We set $\lambda_F = 0.7$ and $\lambda_C = 0.3$, assigning greater weight to forensic attribution because it identifies globally failure-critical steps, while the Critic evaluates local execution quality, including non-critical imperfections. The Critic is fixed to the strong model only during data collection. At inference time, it is routed like any other role.

Execution-risk state. Separately from the scalar Critic–forensic supervision, RIDGE organizes decision-time request and workflow indicators into a six-dimensional execution-risk state:

$$\boldsymbol{s}_i = [s_i^{\text{len}}, s_i^{\text{err}}, s_i^{\text{reject}}, s_i^{\text{reviewer}}, s_i^{\text{severity}}, s_i^{\text{depth}}]. \tag{3}$$

The dimensions cover three execution-risk sources. Request length represents task load, and error history indicates prior execution errors. Four coordination dimensions capture recurring rejections, the review stage producing feedback, feedback severity estimated from text length, and collaboration depth from repeated attempts and revisions.

Together, these dimensions form a structured operational-risk profile rather than a direct quality measure. The state loss preserves this profile, while the boundary loss aligns its aggregate prediction with Critic–forensic evidence of model capability. Low fast-model scores support escalation, low strong-model scores discourage downgrading, and high strong-model scores do not imply fast-model success. The score also guides record filtering, validation calibration, and executed-branch MemoryBank updates.

Each resulting training sample is represented as

$$\mathcal{D}_i = (x_i, r_i, \ell_i, \boldsymbol{s}_i, \rho_i), \tag{4}$$

where x_i is the semantic core of the current request, r_i is the agent role, $\ell_i \in \{f, s\}$ identifies the LLM actually executed, $\boldsymbol{s}_i \in [0,1]^6$ is the decision-time operational-risk state, and ρ_i is the post-execution step-level audit supervision.

Semantic representation. RIDGE extracts the semantic core from the role-visible request, removes system templates, fixed-format instructions, and repeated generic prompts, and encodes the result as $\boldsymbol{e}_i$. Training and inference use the same preprocessing procedure and embedding space.

Role conditioning. The current role is encoded as $\boldsymbol{r}_i$ and provided to the hypernetwork to generate a role-adapted prediction head. Role conditioning constrains both parameter generation and semantic evidence retrieval. It does not prescribe a model tier for any role or substitute evidence from other roles.

3.3 Role-Conditioned Hypernetwork and Dual-Branch Prediction

RIDGE uses a role-conditioned hypernetwork to generate role-specific semantic prediction heads. Given a role vector $\boldsymbol{r}_i$, the hypernetwork H_θ produces the corresponding head parameters:

$$\boldsymbol{\phi}_{r_i} = H_\theta(\boldsymbol{r}_i). \tag{5}$$

Therefore, the role is not simply concatenated with the semantic embedding and passed to a global predictor; it directly determines the prediction-head parameters. Roles maintain distinct capability-boundary representations while sharing the hypernetwork's parameter-generation mechanism.

The semantic embedding is first transformed by a lightweight adapter A_ψ:

$$\tilde{\boldsymbol{e}}_i = A_\psi(\boldsymbol{e}_i). \tag{6}$$

The generated head applies a role-specific shared semantic layer, followed by separate fast- and strong-model branches. Each branch predicts a six-dimensional execution-risk state and its uncertainty. Both branches are computed in one forward pass but supervised only by observations from their corresponding models.

3.4 TRAINING OBJECTIVE

RIDGE uses complementary state and boundary losses. Both supervise only the executed branch; the unobserved branch receives no counterfactual target.

For sample i, ℓ_i identifies the LLM actually executed. Its branch predicts the mean execution-risk state $\boldsymbol{\mu}_i^{\ell_i}$ and the corresponding log-variance $\log \boldsymbol{v}_i^{\ell_i}$. The state loss is

$$\mathcal{L}_{\text{state}} = \frac{1}{N}\sum_{i=1}^{N} w_i \frac{1}{2D}\sum_{d=1}^{D}\Big[\exp(-\log v_{i,d}^{\ell_i})\cdot(s_{i,d}-\mu_{i,d}^{\ell_i})^2 + \log v_{i,d}^{\ell_i}\Big], \tag{7}$$

where $D = 6$, and w_i balances the sample counts of different role-model combinations. This objective jointly learns execution-risk means and predictive uncertainty, allowing RIDGE to distinguish confident high risk from uncertainty caused by limited evidence.

The boundary loss converts the continuous audit score into a capability-risk target,

$$q_i = \frac{1-\rho_i}{2}, \tag{8}$$

and aligns it with the executed branch's mean predicted risk:

$$\mathcal{L}_{\text{boundary}} = \frac{1}{\sum_i w_i}\sum_i w_i \,\text{SmoothL1}\left(\frac{1}{D}\sum_{d=1}^{D}\mu_{i,d}^{\ell_i}, q_i\right). \tag{9}$$

Therefore, the state loss preserves the six-dimensional operational-risk structure, while the boundary loss calibrates its aggregate risk with Critic–forensic supervision.

The complete objective is

$$\mathcal{L} = \mathcal{L}_{\text{state}} + \lambda_{\text{boundary}}\mathcal{L}_{\text{boundary}} + \lambda_{\text{mem}}\mathcal{L}_{\text{mem}} + \lambda_{\text{adapt}}\mathcal{L}_{\text{adapt}}. \tag{10}$$

We set $\lambda_{\text{boundary}} = 1.0$. Here, $\mathcal{L}_{\text{mem}}$ encourages diverse MemoryBank prototypes, and $\mathcal{L}_{\text{adapt}}$ limits embedding drift using mean-squared error and cosine distance. We train with AdamW, split data by task ID, and select checkpoints on validation data.

3.5 PREDICTION RELIABILITY ENHANCEMENT

RIDGE improves prediction reliability through two complementary mechanisms operating in different spaces.

3.5.1 SEMANTIC EVIDENCE SUPPORT

Execution-risk-state means alone may produce overconfident predictions in sparsely observed semantic regions. RIDGE therefore estimates training support separately for the fast and strong branches. Each branch retrieves only samples matching both the current role and model type. Evidence from another role or the other model type cannot substitute for this support.

Support uncertainty increases with the nearest semantic distance and decreases with the number of effective same-role, same-model neighbors. The support index is built exclusively from training tasks; validation and test tasks are excluded.

3.5.2 EXECUTION-STATE MEMORYBANK

RIDGE maintains a shared MemoryBank of representative execution-risk-state prototypes. Each branch retrieves prototypes similar to its predicted state and refines the prediction through weighted aggregation. Although prototypes are shared, retrieval and refinement remain branch-specific, preventing cross-model transfer of labels or semantic evidence. The bank is not a history input: routing uses only the current role and semantics, while the prototypes provide learned operational-risk patterns.

Online, task-level audit scores update only the prototypes attended by the executed-model branch; the other branch remains unchanged. The MemoryBank is stored separately from offline model weights and semantic-support indices.

3.6 Capability-Boundary Estimation and Routing

RIDGE combines weighted, role-aware state risk with aleatoric uncertainty and same-role, same-model semantic support; predicted hazards or weak evidence increase conservative branch risk.

Critic–forensic scores calibrate its direction: low fast-model scores favor escalation, low strong-model scores discourage downgrade, and high strong-model scores establish only strong-model competence. Validation uses role, semantics, both branch predictions, evidence reliability, and failure severity to set a role-specific acceptable-risk threshold and critical-risk threshold, which partition the fast-model risk into acceptable, boundary, and critical regions:

(1) Acceptable risk: retain the fast model unless a supported strong branch exceeds the decisive-upgrade margin; **(2) critical risk:** select the strong model if its evidence is unavailable or its risk is no higher, retaining the fast model only if the supported strong branch is worse; **(3) boundary risk:** select the strong model only if supported risk reduction exceeds the upgrade margin; **(4) strong-only evidence:** select the fast model only when strong-model risk is below the downgrade threshold, otherwise retaining the strong model; and **(5) no effective evidence:** fail closed to the strong model and record an insufficient-evidence fallback.

Thus, routing is neither category- nor role-fixed: RIDGE jointly estimates both models' states and confidence under the current role and semantics, then selects the model sufficient for the request.

4 Experiment

4.1 Experimental Setup

We evaluate on SWE-bench Lite Jimenez et al. (2024); Yang et al. (2024) and WebArena-Verified hattami et al. (2025), constructing role-based multi-agent workflows while retaining the original tasks, environments, official evaluators, and success criteria. Training uses 114 stratified WebArena-Verified and 1,100 SWE-bench Full tasks, identifier-disjoint from evaluation and excluding all SWE-bench Lite test instances. One run per task per model yields approximately 10,000 step records. Experiments use three 64-GB Sugon DCUs, Qwen3-14B as the fast model, and Qwen3.5-35B-A3B-AWQ as the strong model. To limit online overhead, semantic embeddings are cached for reuse; Qwen3.7-Max forensic judgments are restricted to offline data construction and excluded from inference-time usage. Trainable routers use shared trajectory sources and task splits with method-specific supervision. All methods use identical workflows, prompts, tools, environments, evaluators, and runtime limits. Workflow construction, solvability validation, and implementation details appear in the appendix.

4.2 Baseline Methods

We compare RIDGE with the following model configurations and routing methods:

(1) Force-Strong: uses the strong model for every routable step, providing an upper-bound reference for task performance and strong-model resource consumption; **(2) Force-Fast:** uses the fast model for every routable step, measuring performance with minimal dependence on the strong model; **(3) FrugalGPT:** Chen et al. (2024) invokes the fast model first and cascades to the strong model when the result is inadequate; **(4) RouteLLM:** Ong et al. (2025) predicts strong-versus-fast model preferences from input semantics; and **(5) OmniRouter:** Mei et al. (2025) estimates a unified utility across multiple candidate models.

All routers share training and validation splits, trajectory sources, and semantic embeddings, with test tasks excluded from training, semantic indexing, and calibration. RouteLLM and OmniRouter use pairwise preferences and per-model success signals, respectively, with validation-tuned preference thresholds or quality–cost weights. RIDGE additionally models role-conditioned execution states.

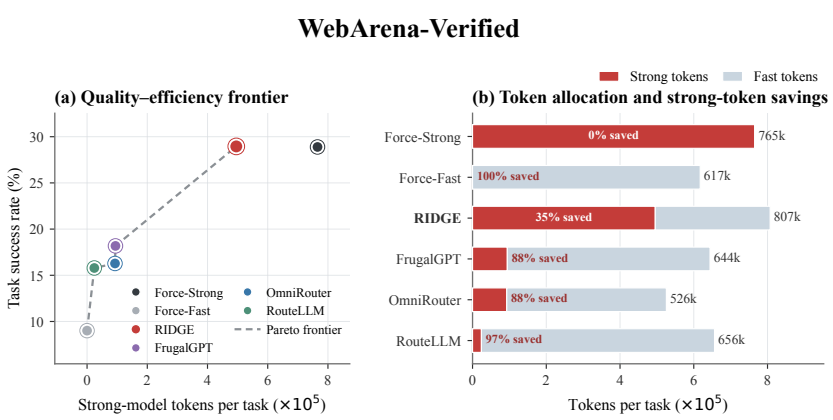

Figure 3: **Success–token Pareto frontier on WebArena-Verified.** The figure compares task success rates and token usage across model-routing strategies.

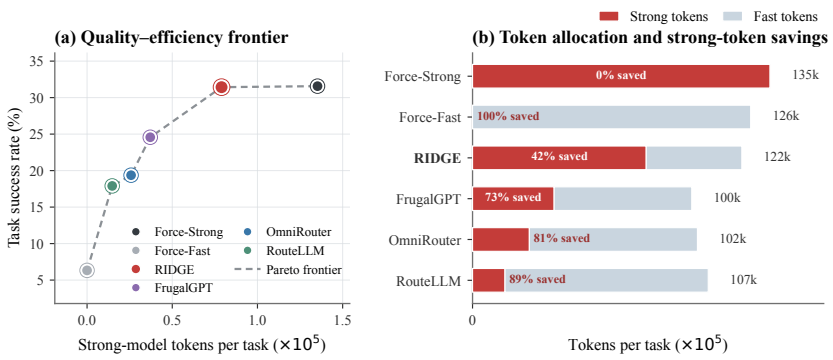

Figure 4: **Success–token Pareto frontier on SWE-bench Lite.** The figure compares task success rates and token usage across model-routing strategies.

5 Results & Analysis

5.1 Task Success and Token Efficiency

We evaluate task success and strong-model token savings on SWE-bench Lite and WebArena-Verified. Because local models lack consistent API pricing, we report per-model input and output tokens and reductions relative to Force-Strong.

RIDGE matches Force-Strong while substantially reducing strong-model usage (Figures 3 and 4). On WebArena-Verified, it achieves 28.95% success versus 28.89%, with 35.3% fewer strong-model tokens. On SWE-bench Lite, it achieves 31.43% versus 31.57%, reducing strong-model tokens from 135,247 to 78,935 (41.6%). It also exceeds FrugalGPT, OmniRouter, and RouteLLM by 6.86, 12.06, and 13.53 percentage points, respectively. Thus, RIDGE reserves the strong model for capability-demanding steps while routing non-failure-critical calls to the fast model.

OmniRouter and RouteLLM remain less effective after step-level adaptation. Their query-level objectives require supervision or aligned strong–fast outputs that are scarce in multi-agent trajectories. Binary step outcomes further omit model states, role differences, and error severity, limiting failure-critical-step detection.

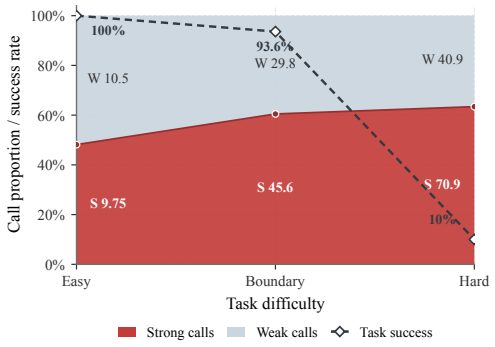

Figure 5: **RIDGE routing across WebArena-Verified difficulty groups.** Model usage is reported as the average number of calls per task.

Table 3: **Ablation results on failure-critical step routing.** Each variant removes one component of RIDGE.

Ablation	Failure-Critical Steps Escalated
w/o Role Conditioning	43.04%
w/o Hypernetwork	35.44%
w/o Six-Dimensional State	34.18%

Table 2: **Strong-model escalation rates on 250 human-selected failure-critical steps from historical WebArena-Verified trajectories.**

Method	Strong-Model Selections	Strong-Model Escalation Rate
RIDGE	**189/250**	**75.60%**
OmniRouter	54/250	21.6%
RouteLLM	41/250	16.4%

The FrugalGPT-style cascade calls the fast model first and escalates on low confidence, stalled execution, or reviewer rejection. This adds tokens and latency, while accepted errors can propagate through shared context.

5.2 Failure-Critical Step Routing

Beyond task completion, we test whether routers selectively escalate genuinely failure-critical steps. From WebArena-Verified trajectories disjoint from training, the forensic Judge selects 250 failure-critical steps, 250 non-failure-critical Critic-rejected steps, and 250 successful steps. To guard against the role-conditioned hypernetwork overfitting to the Judge's selection criteria, human annotators independently select an additional 250 failure-critical steps that are disjoint from the Judge-selected evaluation set. We compare RIDGE, OmniRouter, and RouteLLM on both evaluations.

RIDGE achieves the highest failure-critical-step escalation rate on both the Judge-selected and independently human-selected sets (Tables 1 and 2), substantially outperforming OmniRouter and RouteLLM. Its consistent advantage across both evaluation protocols indicates that RIDGE captures failure-critical risk rather than overfitting the Judge's selection preferences.

Table 1: **Strong-model escalation rates on WebArena-Verified.** Failure-critical steps are identified by the forensic Judge; other rejected steps are rejected by the online Critic but not attributed as causally decisive. Successful steps are those receiving no penalty.

Method	Failure-Critical Steps Escalated	Other Rejected Steps Escalated	Successful Steps Escalated
RIDGE	**76.19%**	**73.81%**	27.6%
OmniRouter	21.99%	11.03%	12.50%
RouteLLM	15.06%	9.70%	4.50%

5.3 Routing Across Task Difficulty

We evaluate whether RIDGE adapts allocation to task difficulty by grouping WebArena-Verified tasks according to static-model outcomes: *easy* tasks are solved by Force-Fast, *medium* tasks only by Force-Strong, and *hard* tasks by neither. For each group, Figure 5 reports RIDGE's average fast- and strong-model calls and task success rate.

Easy tasks require the fewest calls, with the fast model used slightly more often. Medium tasks, near the fast model's capability boundary, require substantially more calls and favor the strong model; RIDGE solves 93.6%, covering most tasks solvable by Force-Strong. Hard tasks use the strong model most, yet RIDGE solves 10.0% that fail in the Force-Strong reference runs.

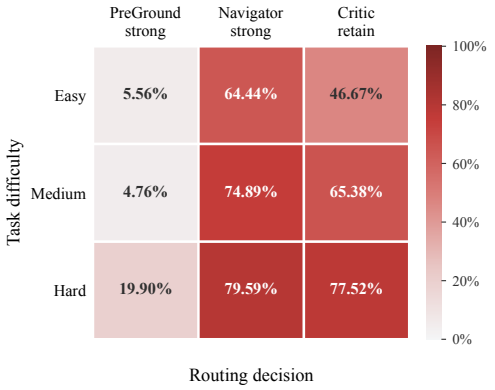

Figure 6: **Role-specific routing rates across WebArena-Verified difficulty groups.** PreGround and Navigator columns report strong-model escalation rates; the Critic column reports its retention rate.

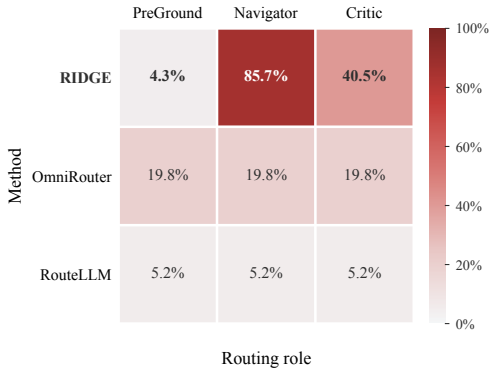

Figure 7: **Strong-model call rates under different role labels for identical semantics.** Only the role label is changed across columns.

5.4 Role-Specific Routing Across Difficulty

Figure 6 analyzes RIDGE's role-specific routing across task difficulty groups.

PreGround is rarely escalated, consistent with its simpler element-filtering role and limited need for strong-model intervention. Navigator receives substantially more strong-model calls because it initiates consequential browser actions, especially as task difficulty increases. Critic retention also rises on harder tasks, reflecting greater verification demands. Thus, task difficulty affects roles unevenly.

Holding semantics fixed, changing only the role from PreGround to Navigator raises RIDGE's strong-model call rate from 4.31% to 85.69%, whereas OmniRouter and RouteLLM remain unchanged (Figure 7), demonstrating role-sensitive routing.

5.5 Ablation Study

We ablate three central components of RIDGE and evaluate their effect on the routing rate for failure-critical steps. As shown in Table 3, full RIDGE achieves 76.19% on this metric.

All ablations reduce failure-critical-step escalation, confirming the value of role conditioning, hypernetwork generation, and structured multidimensional risk. The scalar variant further shows that complementary risk dimensions add information beyond a single audit score.

6 Conclusion

RIDGE addresses the reliability–cost trade-off in LLM-based multi-agent systems through role- and semantics-conditioned capability-boundary estimation. It jointly predicts execution risks and uncertainties for fast and strong models to guide step-level model selection.

On SWE-bench Lite and WebArena-Verified, RIDGE achieves task success rates comparable to the static strong-model baseline while reducing strong-model token usage by up to 41.6%. It routes 76.19% of Judge-selected and 75.60% of 250 human-selected failure-critical steps to the strong model, substantially exceeding both baselines. These results support role-aware risk estimation for selective strong-model allocation while preserving task success.

Limitations and Future Work. RIDGE is limited by conservative strong-to-fast downgrades, its two-model formulation, and dependence on sufficient role–model coverage. Future work will investigate larger model pools, routing under explicit budget constraints, and reliable capability-boundary estimation with sparse role–model samples.

AI USE STATEMENT

We used generative AI tools to assist with language editing and improve the readability of the manuscript. LLMs also constitute part of the experimental framework: they generate agent execution trajectories, Critic feedback, and offline forensic annotations used for router supervision and evaluation, as described in the paper. These automated annotations are distinguished from human validation. The authors reviewed the AI-assisted manuscript revisions for technical accuracy and consistency with the reported experiments. We take responsibility for the final content of this work, including all claims, results, and artifacts produced with the assistance of generative AI.

REFERENCES

Anonymous. Internalizing self-consistency in language models: Multi-agent consensus alignment, 2026. URL https://openreview.net/forum?id=Ma0LTdFVeL.

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. ISSN 2835-8856. URL https://openreview.net/forum?id=cSimKw5p6R. Featured Certification.

Amine El hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. Webarena verified: Reliable evaluation for web agents. In *Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=94tlGxmqkN.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.

Yoichi Ishibashi and Yoshimasa Nishimura. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Yuxing Lu, Yucheng Hu, Xukai Zhao, and Jiuxin Cao. Dytopo: Dynamic topology routing for multi-agent reasoning via semantic matching, 2026. URL https://arxiv.org/abs/2602.06039.

Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing. *ACM SIGKDD Explorations Newsletter*, 27(2): 107–116, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive LLM routing under budget constraints. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23934–23949, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1301. URL https://aclanthology.org/2025.findings-emnlp.1301/.

Ming-Tung Shen and Yuh-Jzer Joung. Talm: Dynamic tree-structured multi-agent framework with long-term memory for scalable code generation. In *Proceedings of the 2026 International Workshop on Agentic Engineering*, AGENT '26, pp. 144–151, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723995. doi: 10.1145/3786167.3788424. URL https://doi.org/10.1145/3786167.3788424.

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.

Siyu Wang, Ruotian Lu, Zhihao Yang, Yuchao Wang, Yanzhou Zhang, Lei Xu, Qimin Xu, Guojun Yin, Cailian Chen, and Xinping Guan. Agentconductor: Topology evolution for multi-agent competition-level code generation, 2026. URL https://arxiv.org/abs/2602.17100.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, WWW '26, pp. 4184–4195, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723070. doi: 10.1145/3774904.3792551. URL https://doi.org/10.1145/3774904.3792551.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

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 Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=imcyVlzpXh.

Jusheng Zhang, Yijia Fan, Kaitong Cai, Xiaofei Sun, and Keze Wang. OSC: Cognitive orchestration through dynamic knowledge alignment in multi-agent LLM collaboration. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 6320–6337, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.335. URL https://aclanthology.org/2025.findings-emnlp.335/.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025c. URL https://openreview.net/forum?id=GazlTYxZss.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

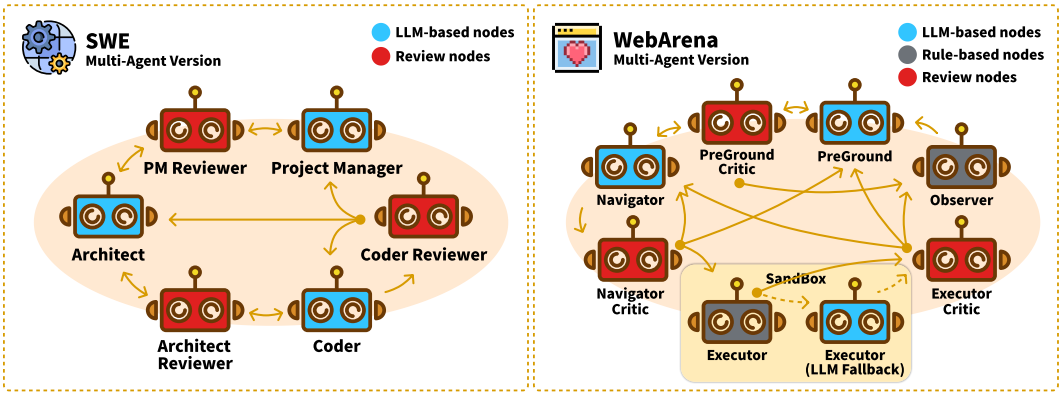

Figure 8: **Schematic overview of the two multi-agent benchmarks.** Left: the multi-agent version of SWE-bench Lite. Right: the multi-agent version of WebArena-Verified.

A Multi-Agent Benchmark Construction

To evaluate dynamic model routing in complex workflows with real execution feedback, we construct two role-based multi-agent benchmarks from SWE-bench Lite Jimenez et al. (2024) and WebArena-Verified hattami et al. (2025). The construction follows three principles: preserving the original task objectives, execution environments, and success criteria; decomposing only the solution process into agents with explicit responsibilities; and retaining the executable official evaluators as the sole source of final outcomes.

To verify that the transformed workflows preserve task solvability, we execute all routable roles with DeepSeek-V4-Flash-Preview. This configuration achieves official success rates of 53.4% on SWE-bench Lite and 69.7% on WebArena-Verified, confirming that role decomposition does not prevent the workflows from solving the original tasks.

A.1 SWE Multi-Agent Benchmark

A.1.1 Original Tasks and Environment

The SWE multi-agent benchmark is built on the 300 test instances of SWE-bench Lite. Each instance retains the original repository, issue description, base commit, test patch, and `FAIL_TO_PASS` and `PASS_TO_PASS` test targets. We use the SWE-bench Lite variant augmented with BM25 code-retrieval results, which provide issue-relevant candidate snippets as localization hints but do not replace inspection of the actual repository.

For each task, the target repository is restored to its specified `base_commit` in an isolated working directory. Agents can access only the issue description, retrieval context, and current repository; they cannot access the reference patch or test answers. After completion, the generated patch is applied in the instance-specific isolated container and evaluated using the official test patch and test targets. The patch is also stored in the standard SWE-bench submission format with `instance_id` and `model_patch` fields.

A.1.2 Roles and Collaborative Workflow

We decompose single-agent software repair into three execution roles and a corresponding review process:

1. **Project Manager**: analyzes the issue, infers likely root causes, develops a high-level repair strategy, and identifies files, classes, or functions requiring further inspection.
2. **Architect**: searches and reads the actual repository to verify upstream hypotheses and determine the minimal modification scope.

3. **Coder**: generates a patch from verified code context and may not modify files that the Architect has not inspected.
4. **Role Reviewer**: independently reviews the project plan, localization evidence, and final patch, and returns insufficiently supported outputs to the responsible role.

The Reviewer controls the workflow through `APPROVE`, `REJECT`, and an explicit rollback target. Rejected outputs do not advance to the next stage but return to the Project Manager, Architect, or Coder for revision. A task may therefore involve multiple planning, localization, coding, and review cycles, approximating collaborative software development.

A.1.3 Patch Validation and Official Outcomes

Coder-generated patches first undergo static validation, including checks that target files exist, edited content matches uniquely, no uninspected files are modified, and the patch format is valid. A valid patch is then applied to the isolated instance environment and evaluated against the task-specific tests. A task succeeds only if the patch resolves the target failing tests without breaking previously passing tests.

The multi-agent transformation does not alter the issue description, repository version, test patch, or success criteria. Agent discussions, rejections, and retries affect only patch generation and do not contribute to the final score. Patches produced by different routing methods therefore remain directly comparable under identical SWE-bench tasks and evaluation conditions.

A.2 WebArena Multi-Agent Benchmark

A.2.1 Task and Environment Selection

WebArena-Verified contains 812 official task configurations spanning information retrieval, navigation, and state-changing operations. We preserve each task's original intent, start page, site information, reference answer, and evaluation conditions, and execute tasks in self-hosted Reddit, Shopping, Shopping Admin, and GitLab environments. Cross-site tasks are retained to evaluate information transfer and sequential interaction across websites.

Because the local Map site could not be reproduced reliably, we exclude its tasks and obtain 700 candidate tasks. We select 114 representative tasks covering 29 task subtypes and divide them into task-disjoint training and validation sets using the 80/20 split reported in Table 4. From the remaining 586 tasks, we sample 150 test tasks while balancing sites, official evaluator types, and task templates with a fixed random seed. Four Wikipedia tasks lack reproducible local containers, yielding a final test set of 146 executable tasks. Training, validation, and test tasks are strictly separated by task ID.

A.2.2 Roles and Closed-Loop Execution

We decompose each web task into the following functional stages:

1. **Observer**: collects the current page, URL, and accessibility tree without invoking a language model.
2. **PreGround**: filters and ranks task-relevant interactive elements on the current page.
3. **PreGroundCritic**: verifies that the candidate elements sufficiently cover the required operations and requests re-filtering when necessary.
4. **Navigator**: plans the next action from the task objective, page state, and candidate elements, or returns the final answer and stops when the task is complete.
5. **Critic**: reviews the action plan before execution, verifies the result afterward, and returns erroneous decisions to the appropriate stage.
6. **Executor**: converts an approved plan into executable Playwright actions.
7. **Sandbox**: executes actions in a real browser and returns the updated page state, execution result, and network trace.

The Critic performs plan review, execution-error analysis, outcome verification, and final stopping checks. Rejected actions return to the Navigator, PreGround, or Observer; successful actions trigger a new observation and the next interaction round. Each WebArena task therefore naturally produces multiple rounds of observation, candidate selection, planning, execution, and feedback, with role-level subtasks of varying difficulty.

A.2.3 Environment Isolation and Official Evaluation

To prevent state contamination across tasks, the corresponding site containers are recreated from fixed images before each task, their writable layers are discarded, and login states are re-established. Parallel experiments use independent site stacks, ports, and containers so that tasks do not share database state or browser sessions.

Complete HTTP Archive (HAR) network traces are recorded during browser execution. At task completion, the final answer is converted to the standard WebArena-Verified response format, and a `RETRIEVE`, `NAVIGATE`, or `MUTATE` result is constructed according to the official task type. The official `AgentResponseEvaluator` checks final answers, while the `NetworkEventEvaluator` validates interactions and state changes from the HAR trace. Differences between local and container addresses are handled only through URL normalization. Confirmed endpoint changes between site versions are aligned only when the request method and key parameters match exactly; task objectives and expected results are never modified. Final statistics include only task records containing official evaluator outputs.

A.3 Reproducibility and Release

We will release the SWE and WebArena multi-agent workflows, including role definitions, graph structures, task lists, data splits, environment configurations, execution scripts, and official evaluation interfaces. Restricted benchmark data will not be redistributed; instead, we provide the code and manifests required to construct the multi-agent versions from official task configurations. This release enables researchers to reproduce results for different model configurations and routing strategies under identical task IDs, environment states, and evaluation conditions.

B Design Principles for Forensic Audit Prompts

To ensure consistent and reliable forensic supervision across SWE and WebArena, we follow a principle of *unified decision protocols with domain-specific evidence*. Both benchmarks share the same failure categories, causal-attribution criteria, severity scale, and structured output schema, while adapting the input evidence and role responsibilities to each domain. SWE audits primarily use the issue description, code patch, and sandbox test results, whereas WebArena audits additionally incorporate official evaluation conditions, the final page state, and HTTP Archive (HAR) network traces.

B.1 Forensic-Visible Trajectories and Evidence Denoising

The forensic Judge does not read lengthy raw logs. Instead, it receives an audit package comprising the task objective, a structured execution trajectory, and final objective evidence. In execution order, the trajectory records each step's identifier, role, invoked model, role-visible input, model output, and local result. It also distinguishes attributable routed steps from deterministic steps included only for context.

To reduce irrelevant context, we remove repeated role descriptions and formatting templates, replace recurring task objectives and upstream outputs with step references, and truncate overly long fields. Deterministic programs extract objective signals such as tool results, action status, and review verdicts, reducing the risk that the Judge is misled by agents' self-reports.

For consecutive chains with similar role–semantic content, we retain the first and last executions and compress intermediate repetitions into a non-attributable loop summary. The summary records the original step range, repetition count, participating roles, and representative behavior. This preserves

evidence of loops and their possible root causes without overwhelming the Judge with repeated text. Compression affects only the forensic input; the original trajectory remains unchanged.

The Judge first uses official evaluation outputs and final objective evidence to verify that the task genuinely failed, and only then performs step-level causal attribution. Before large-scale annotation, we evaluated the protocol on 20 randomly sampled complex failures from each benchmark and obtained 95% agreement with human review. We then froze the protocol for data collection. We retain the complete audit input and the Judge's raw response to support human review and reproducibility.

The forensic auditor first verifies that the task genuinely failed and distinguishes trainable model-capability failures from evaluator errors, environment failures, underspecified tasks, and failures in non-routable nodes. A failure is considered routable, and step-level attribution is produced, only when objective evidence identifies a routable LLM step that directly affected the task outcome. Rule-based nodes, deterministic components, and fixed-model nodes may provide causal context but cannot be assigned responsibility, nor is their responsibility automatically transferred to downstream agents.

For failure chains spanning multiple consecutive steps, the auditor applies a counterfactual causal test: holding other conditions fixed, would the final failure still have occurred if the current step had been handled correctly? Only steps with direct and verifiable causal contributions are retained. Attribution severity uses a shared continuous scale: decisive root causes receive scores from -1.0 to -0.85, major direct causes from -0.85 to -0.65, and substantial contributing causes from -0.65 to -0.4. Minor, indirect, or insufficiently supported effects are excluded, preventing failure penalties from being indiscriminately propagated across the trajectory.

All forensic outputs are constrained to `JSON` and contain a mutually exclusive task-level verdict, evidence-grounded justification, and, when applicable, the attributed step identifier, role, severity, and rationale. This design reduces ambiguity from free-form parsing and allows audit results to be used directly for step-level capability-boundary supervision and subsequent experimental auditing.

C Experimental Details

C.1 Training-Data Collection and Splits

We perform stratified sampling from WebArena-Verified and SWE-bench Full to cover diverse websites, interaction types, repositories, and issue categories. The collection set contains 114 WebArena-Verified tasks and 1,100 SWE-bench Full tasks after excluding every SWE-bench Lite test instance. Each task is executed through two complete multi-agent trajectories, one using the fast model and one using the strong model. We record the agent role, step semantics, invoked model, execution-risk state, online Critic evaluation, and forensic attribution for failed tasks, producing approximately 10,000 step-level records.

RIDGE and all trainable baselines share the same raw trajectories, task-level training and validation splits, and model outputs. Each method reads only the supervision required by its original objective: preference routers use relative strong–fast outcomes, capability predictors use execution outcomes, and RIDGE additionally uses role-conditioned states, Critic evaluations, and forensic step attribution. No method receives exclusive tasks or model outputs. Evaluation uses the remaining WebArena-Verified tasks and SWE-bench Lite, with training, validation, and test sets strictly separated by task identifier.

C.2 Model and Runtime Configuration

All experiments run on three Sugon DCU accelerators with 64 GB of memory each. Qwen3-14B serves as the fast model, while Qwen3.5-35B-A3B-AWQ serves as the strong model. Both use the same inference interface, context-processing procedure, and decoding configuration. The online Critic is fixed to the strong model during data collection. For failed tasks, Qwen3.7-Max independently audits the full trajectory, final environment state, and official evaluation output to identify steps with direct causal responsibility for failure.

Critic and forensic scores are combined with weights of 0.3 and 0.7, respectively, to form step-level capability-boundary supervision. The forensic Judge is used only for post-task offline annotation;

```
System Prompt:
You are an independent forensic auditor. Use only the provided evidence and output strict JSON.

User Prompt:
Role: Independent Forensic Auditor for Multi-Agent Systems.
Task: Verify a failed software-repair run and identify only the routable LLM steps that directly and materially
caused the failure. Do not produce role-level scores.

[EVIDENCE PRIORITY]
Official issue and executable tests > sandbox outcome > submitted patch > role-visible trajectory > agent self-
narration.

[UNIFIED VERDICT PROTOCOL]
Select exactly one mutually exclusive verdict:
routable_failure: Objective evidence identifies at least one routable LLM step that directly and materially
caused the genuine task failure.
actually_completed: Objective evidence proves that the task was completed and the evaluator produced a false
negative.
environment_failure: Infrastructure failed before agent behavior could be meaningfully evaluated.
task_underspecified: The task or evaluation target is genuinely indeterminate.
non_routable_failure: A deterministic, rule-based, or fixed-model component is the root cause. Never transfer
its responsibility to a routable step.
insufficient_evidence: The task genuinely failed, but the available evidence cannot reliably isolate a routable
cause.
Only routable_failure may contain culprit_steps. Every other verdict must return an empty culprit_steps list.
Non-routable steps must never appear in culprit_steps.
A downstream routable step is attributable only if it had sufficient visible evidence to detect or recover from
an upstream defect and independently made an avoidable, materially harmful decision.

[UNIFIED STEP-SEVERITY RUBRIC]
Assign a continuous severity in [-1.0, -0.4] only to direct, evidenced causes:
[-1.0, -0.85]: Dominant necessary cause. Without this step, the failure would very likely not have occurred.
(-0.85, -0.65]: Major direct cause. The step substantially determined the failure, possibly together with
another major cause.
(-0.65, -0.4]: Material contributing cause. The step was not dominant but made a necessary and clearly
evidenced contribution.
Minor or uncertain contributions must be omitted. Apply the but-for test and cite the decisive evidence in each
reason.

[FAILURE VALIDITY CHECK]
Return environment_failure only when objective evidence shows that Docker, the sandbox, repository checkout,
dependency installation, the test harness, permissions, or the filesystem failed before the patch was
meaningfully evaluated.
Ordinary assertion failures, syntax errors introduced by the patch, incorrect edits, and failing tests are
agent failures.
Return task_underspecified only when the issue is genuinely insufficient to determine a reasonable repair
target, even after repository exploration, or when the target tests are logically unrelated to the issue.
Return actually_completed only when the submitted patch objectively satisfies the issue and the reported
failure is unrelated or caused by an evaluator error. Trace the complete failure chain. If the traceback
originates from user-edited code, treat the failure as agent-caused.

--- ISSUE ---
{{ISSUE_TEXT}}
--- STRUCTURED ROLE-VISIBLE TRAJECTORY ---
{{TRAJECTORY}}
--- SUBMITTED PATCH ---
{{PATCH}}
--- SANDBOX AND TEST EVIDENCE ---
{{SANDBOX_RESULT}}

[DOMAIN RESPONSIBILITIES]
ProjectManager: Incorrect root-cause strategy, fabricated paths, or a plan that directly directs downstream
agents toward the wrong repair.
Architect: Failure to inspect relevant source code, selection of incorrect files despite sufficient evidence,
or failure to provide necessary context to Coder.
Coder: Malformed patches, hallucinated code, placeholders, syntax errors, or incorrect logic despite sufficient
verified context.
Reviewer: Provides process evidence for evaluating adjacent routable steps. A Reviewer marked non-scorable or
fixed-model is context only and cannot be a culprit.
These responsibilities guide step-level causal attribution only. Do not score roles globally.

[CAUSAL ANALYSIS]
Start from the sandbox error and determine exactly what failed.
Trace the causal chain upstream through Coder, Architect, and ProjectManager.
Apply the but-for test: would the failure still have occurred if the examined step had been correct?
Distinguish an independently harmful downstream decision from an unavoidable consequence of incorrect upstream
information.
If a non-routable component is the root cause, return non_routable_failure.
If the task failed but no routable cause can be reliably isolated, return insufficient_evidence.
Each trajectory row is marked as either a scorable routing decision or a non-scorable contextual step. Never
attribute a non-scorable step or transfer its responsibility to another role.

[OUTPUT FORMAT]
Output JSON only. No Markdown, code fence, or commentary.
{
  "verdict": "routable_failure",
  "reason": "Short task-level conclusion grounded in objective evidence",
  "culprit_steps": [
    {
      "step": 4,
      "role": "Coder",
      "severity": -0.8,
      "reason": "Direct causal contribution and supporting evidence"
    }
  ]
}
```

Figure 9: **Forensic audit prompt for SWE-bench Lite.** The prompt is used to audit failures and produce step-level attribution when collecting training data with the fast model.

```
System Prompt:
You are an independent forensic auditor. Use only the provided evidence and output strict JSON.

User Prompt:
Role: Independent Forensic Auditor for Multi-Agent Systems.
Task: Verify a failed web-interaction run and identify only the routable LLM steps that directly and materially
caused the failure. Do not produce role-level scores.

[EVIDENCE PRIORITY]
Official evaluation criteria and executable evidence > final environment state > HAR network evidence > role-
visible trajectory > agent self-narration.

[UNIFIED VERDICT PROTOCOL]
Select exactly one mutually exclusive verdict:
routable_failure: Objective evidence identifies at least one routable LLM step that directly and materially
caused the genuine task failure.
actually_completed: Objective evidence proves that the task was completed and the evaluator produced a false
negative.
environment_failure: Infrastructure failed before agent behavior could be meaningfully evaluated.
task_underspecified: The task or evaluation target is genuinely indeterminate.
non_routable_failure: A deterministic, rule-based, or fixed-model component is the root cause. Never transfer
its responsibility to a routable step.
insufficient_evidence: The task genuinely failed, but the available evidence cannot reliably isolate a routable
cause.
Only routable_failure may contain culprit_steps. Every other verdict must return an empty culprit_steps list.
Non-routable steps must never appear in culprit_steps.
A downstream routable step is attributable only if it had sufficient visible evidence to recover from an
upstream defect and independently made an avoidable, materially harmful decision.

[UNIFIED STEP-SEVERITY RUBRIC]
Assign a continuous severity in [-1.0, -0.4] only to direct, evidenced causes:
[-1.0, -0.85]: Dominant necessary cause.
(-0.85, -0.65]: Major direct cause.
(-0.65, -0.4]: Material contributing cause.
Minor or uncertain contributions must be omitted. Apply the but-for test and cite decisive evidence.

[EXACT-MATCH COMPLETION AUDIT]
The official evaluator has marked the task as failed. Before attributing failure, verify every evaluation
criterion:
Every MUST_INCLUDE or required_contents item must appear in the final content in substance.
Required URLs must match the specified target.
Required network events must be present in HAR with the expected method, path, and status.
When both page content and a network event are required, both must be satisfied.
Cross-site copying tasks must preserve the required source content. A plausible paraphrase is insufficient.
Return actually_completed only when objective DOM, final-state, or HAR evidence proves completion. An agent's
narration, success claim, or STOP action is insufficient.
Return environment_failure only with objective evidence such as 5xx responses, connection failures, browser
crashes, or a page that never loaded. Loops, stuck controls, and ineffective actions are agent failures.
Return task_underspecified only when no correct initial action could reasonably be determined from the goal and
starting state.

--- USER GOAL ---
{{USER_GOAL}}
--- OFFICIAL EVALUATION CRITERIA ---
{{ORACLE_EVALUATION_CRITERIA}}
--- OBJECTIVE HAR EVIDENCE ---
{{HAR_EVIDENCE}}
--- STRUCTURED ROLE-VISIBLE TRAJECTORY ---
{{TRAJECTORY}}
--- FINAL ENVIRONMENT STATE AND ERRORS ---
{{FINAL_STATE_AND_ERRORS}}

[DOMAIN RESPONSIBILITIES]
Observer: A deterministic DOM extractor. It is non-routable and may only be used as causal context.
PreGround: Ranks and filters Observer elements into the candidate set available to Navigator. It is responsible
when a required element exists in Observer output but is missing or unusably buried in Navigator's candidate
set.
Navigator: Selects and plans actions. Failures include loops, incorrect navigation, or hallucinated elements
despite sufficient visible evidence.
Executor: Translates approved actions into browser operations. Failures include invalid selectors, syntax
errors, or operations inconsistent with the approved action.
Critic: Validates action outcomes. Failures include accepting an objectively failed action or rejecting an
objectively successful action.

[CAUSAL ANALYSIS]
Before blaming Navigator for missing an element, verify that the element appeared in Navigator's candidate
input. If Observer captured it but PreGround removed it, attribute the failure to the corresponding PreGround
step.
If Executor produces an objectively visible execution error and Critic approves it, evaluate Critic's
independent contribution using the unified severity rubric.
Non-routable nodes must never appear in culprit_steps. If a non-routable node is the root cause, return
non_routable_failure.

[OUTPUT FORMAT]
Output JSON only. No Markdown, code fence, or commentary.
{
  "verdict": "routable_failure",
  "reason": "Short task-level conclusion grounded in objective evidence",
  "culprit_steps": [
    {
      "step": 15,
      "role": "Navigator",
      "severity": -0.8,
      "reason": "Direct causal contribution and supporting evidence"
    }
  ]
}
```

Figure 10: **Forensic audit prompt for WebArena-Verified.** The prompt incorporates browser states, official evaluation conditions, and network traces to attribute failures when collecting training data with the fast model.

Table 4: **RIDGE hyperparameter settings.** All main experiments use the same configuration and a single random seed.

Component	Hyperparameter	Setting
Reproducibility	Random seed; train/validation split	42; task-disjoint 80/20 split
Semantic encoder	Encoder; embedding dimension	`text-embedding-3-small`; 1,536
Embedding adapter	Architecture; activation	Residual $1536 \to 384 \to 1536$; GELU
Hypernetwork	Hidden layers; normalization and activation	32 and 64; layer normalization followed by ReLU
Prediction head	Hidden dimension; outputs	64; independent fast/strong means and log-variances for a six-dimensional execution-risk state
Predictive variance	Log-variance clipping	$[-6, 3]$
MemoryBank	Number and dimension of entries	16 learnable six-dimensional entries
Optimization	Optimizer; learning rate; weight decay; batch size	AdamW; 3×10^{-5}; 3×10^{-2}; 64
Training schedule	Maximum epochs; early-stopping patience; gradient clipping	100; 40 epochs by validation loss; maximum gradient norm 1.0
Sample weighting	Reweighting rule	Inverse role–model group frequency
Training objective	State; boundary; MemoryBank; adapter weights	1.0; 1.0; 0.01; 0.005
Adapter loss	Preservation terms	Equal combination of mean-squared error and cosine distance
Audit target	Score-to-risk mapping	$(1 - \rho)/2$
Calibration weighting	Audit-based sample weight; coefficient	$w_i^{\text{audit}} = 1 + \lambda_{\text{audit-w}}\lvert\rho_i\rvert$; $\lambda_{\text{audit-w}} = 1.0$
Semantic support	Retrieval pool; neighbors; similarity threshold	Same role and model branch; 8 nearest neighbors; cosine similarity ≥ 0.8
Uncertainty	Aleatoric/support weights; uncertainty penalty	$0.5/0.5$; 0.2
Calibration	Decisive escalation margin; remaining thresholds	0.71 success quantile; selected on the task-disjoint validation set

its calls are excluded from inference-time model-call and token statistics. All methods share identical workflows, system prompts, tool interfaces, environments, evaluators, maximum action counts, runtime limits, and rules for failures, timeouts, and environment errors. Only the model-selection strategy differs.

C.3 RIDGE Hyperparameter Settings

Table 4 summarizes the fixed architecture, training, support-retrieval, and calibration settings used in all RIDGE experiments. Unless specified below, role-specific risk thresholds and routing margins are selected exclusively on the task-disjoint validation set.

The audit-based coefficient is used only during validation-set calibration. Samples with clearer audit evidence receive greater influence when learning role-specific risk weights and routing thresholds: $w_i^{\text{audit}} = 1 + \lambda_{\text{audit-w}}|\rho_i|$. Thus, with $\lambda_{\text{audit-w}} = 1.0$, samples with $|\rho_i| = 0$, 0.5, and 1 receive weights of 1.0, 1.5, and 2.0, respectively. This coefficient is neither a training-loss weight nor part of the forensic–Critic mixture.

We further examine sensitivity to the forensic-supervision mixture and auxiliary-loss weights. As shown in Table 5, each experiment varies one weight while holding the remaining architecture, training, and routing settings fixed, allowing us to isolate how supervision and auxiliary-loss weighting affect failure-critical and successful-step escalation.

Each variant increases failure-critical-step escalation but also escalates substantially more successful steps. The default configuration therefore achieves the highest balance score of 74.30%, retaining a 76.19% failure-critical-step escalation rate while producing the lowest successful-step escalation rate of 27.60%, and is used in all main experiments.

Table 5: **Sensitivity to supervision and regularization weights.** Each variant changes one weight from the default configuration. Forensic weights of 0.5 and 0.9 use complementary Critic weights of 0.5 and 0.1, respectively. The balance score is the mean of the failure-critical escalation rate and one minus the successful-step escalation rate.

Configuration	Failure-Critical Steps Escalated	Successful Steps Escalated	Balance Score
Default	76.19%	27.60%	**74.30%**
Forensic weight 0.5	81.14%	39.80%	70.67%
Forensic weight 0.9	77.34%	35.71%	70.82%
Boundary weight 0.5	81.14%	37.76%	71.69%
Boundary weight 2.0	79.34%	35.71%	71.82%
Memory weight 0.005	81.14%	39.80%	70.67%
Memory weight 0.02	79.34%	35.71%	71.82%
Adapter weight 0.0025	81.14%	37.76%	71.69%
Adapter weight 0.01	81.14%	35.71%	72.72%

D Detailed Training and Inference Procedures

RIDGE uses a shared MemoryBank $\boldsymbol{M} \in \mathbb{R}^{K\times 6}$ for both model branches. For each branch, RIDGE normalizes the predicted mean state, retrieves similar row-normalized memory prototypes using temperature-scaled attention, and uses their gated weighted combination to produce a refined state within $[0,1]^6$. Consistent with the main paper, the offline state and boundary losses supervise the unrefined branch mean $\boldsymbol{\mu}^\ell$, while the routing policy uses the refined state $\hat{\boldsymbol{s}}^\ell$ for $\ell \in \{f, s\}$.

Algorithm 1: RIDGE Dual-Arm Offline Training

Require: Training records $\mathcal{D} = \{(x_i, r_i, \ell_i, \boldsymbol{s}_i, \rho_i)\}$, where $\ell_i \in \{f, s\}$ identifies the actually executed model M_{ℓ_i}
Require: Semantic encoder E, embedding adapter A, role-conditioned hypernetwork H, and MemoryBank $\boldsymbol{M}$
Ensure: Trained router, semantic-support index $\mathcal{I}$, and role-specific calibration $\mathcal{C}$
Divide $\mathcal{D}$ into task-disjoint training and validation sets
Initialize A, H, the dual prediction head, and $\boldsymbol{M}$
for each training minibatch $\mathcal{B}$ **do**
  for each $(x_i, r_i, \ell_i, \boldsymbol{s}_i, \rho_i) \in \mathcal{B}$ **do**
    Encode and adapt the semantic request
    Generate the role-conditioned dual prediction head
    Jointly predict the fast and strong mean states and clipped log-variances in one forward pass
    Supervise only the executed branch ℓ_i
    Compute $\mathcal{L}_{\text{state}}$
    Compute $\mathcal{L}_{\text{boundary}}$
  end for
  Compute $\mathcal{L}_{\text{mem}}$
  Compute $\mathcal{L}_{\text{adapt}}$
  Combine the losses into $\mathcal{L}$
  Update A, H, the dual prediction head, and $\boldsymbol{M}$ using AdamW
end for
Restore the checkpoint with the lowest validation loss
Build $\mathcal{I}$ from adapted training embeddings, grouped strictly by the same role and model branch
Weight validation samples by $w_i^{\text{audit}} = 1 + \lambda_{\text{audit-w}}|\rho_i|$
Calibrate role-specific risk weights, confidence thresholds, acceptable and critical risk thresholds, downgrade thresholds, and upgrade margins using the weighted validation tasks
return trained router, $\boldsymbol{M}$, $\mathcal{I}$, and $\mathcal{C}$

Algorithm 2: RIDGE Online Inference and MemoryBank Update

Require: Current request x_t, role r_t, trained router, MemoryBank $\boldsymbol{M}$, semantic-support index $\mathcal{I}$, and calibration $\mathcal{C}_{r_t}$
Ensure: Selected model label $\ell_t \in \{f, s\}$
if the router artifacts or calibration for r_t are unavailable **then**
  return strong ▷ Fail-closed fallback
end if
Extract, encode, and adapt the semantic core of x_t
Generate the role-conditioned dual prediction head
Jointly predict the fast and strong mean states and predictive uncertainties in one forward pass
for $\ell \in \{f, s\}$ **do**
  $(\hat{\boldsymbol{s}}_t^\ell, \mathcal{T}_t^\ell) \leftarrow \text{MemoryRead}(\boldsymbol{\mu}_t^\ell, \boldsymbol{M})$ ▷ Refine the state and retain its memory trace
  Retrieve the eight nearest training samples from $\mathcal{I}$ with the same (r_t, ℓ)
  Estimate semantic-support uncertainty $U_t^{\text{sup},\ell}$ from the nearest semantic distance and the number of effective same-role, same-branch neighbors
  $n_t^\ell \leftarrow$ the corresponding effective-neighbor count
  Compute total uncertainty U_t^ℓ from predictive and semantic-support uncertainty
  Compute conservative branch risk R_t^ℓ from the role-weighted refined state and uncertainty penalty
  Convert U_t^ℓ into confidence score C_t^ℓ
  $\text{confident}_t^\ell \leftarrow [U_t^\ell \le \tau_{r_t}^{\text{conf}}]$
  $\text{valid}_t^\ell \leftarrow \text{confident}_t^\ell \vee [n_t^\ell > 0]$
end for
if valid_t^f **then**
  if $R_t^f \le \tau_{r_t}^{\text{acc}}$ **then**
    if valid_t^s and $R_t^f - R_t^s \ge m_{r_t}^{\text{dec}}$ **then**

$\ell_t \leftarrow s$
else
$\ell_t \leftarrow f$
end if
else if $R_t^f \geq \tau_{r_t}^{\text{crit}}$ **then**
if $\neg\,\text{valid}_t^s$ or $R_t^s \leq R_t^f$ **then**
$\ell_t \leftarrow s$ ▷ Critical fast risk or safe fallback
else
$\ell_t \leftarrow f$ ▷ The supported strong branch is worse
end if
else
if valid_t^s and $R_t^f - R_t^s \geq m_{r_t}^{\text{up}}$ **then**
$\ell_t \leftarrow s$
else
$\ell_t \leftarrow f$
end if
end if
else if valid_t^s **then**
if $R_t^s \leq \tau_{r_t}^{\text{down}}$ **then**
$\ell_t \leftarrow f$ ▷ Strong prediction indicates excess capability
else
$\ell_t \leftarrow s$
end if
else
$\ell_t \leftarrow s$ ▷ No effective same-role, same-model evidence
end if
Execute role r_t using model M_{ℓ_t}
if delayed Critic–forensic supervision ρ_t becomes available after post-task offline audit **then**
if online MemoryBank updating is enabled **then**
Update and row-normalize the shared MemoryBank using the audit-weighted trace $\mathcal{T}_t^{\ell_t}$ only
Persist the updated MemoryBank separately from offline weights and the semantic-support index
end if
end if
return ℓ_t, (R_t^f, R_t^s), (C_t^f, C_t^s), and the routing reason