EDGEWISE: BRINGING LONG-HORIZON AGENTS TO THE EDGE VIA SELECTIVE CLOUD DELEGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Complex and long-horizon agent tasks often rely on frontier models deployed in the cloud to achieve strong performance. However, using these models for every step of a long-horizon trajectory incurs substantial inference cost, even though not all steps may require the same level of model capability. In this paper, we find that allocating the LLM to only a small subset of key steps can achieve performance comparable to full LLM execution while significantly reducing inference cost. Motivated by this observation, we propose ***EdgeWise***, an edge-cloud collaboration framework that deploys an SLM on the local device as the primary solver and selectively delegates critical steps to a cloud LLM when stronger model capacity is needed. To enable autonomous delegation without additional models at inference time, we introduce a step-level RL method that jointly optimizes task solving and delegation, with fine-grained credit assignment for individual delegation decisions. We also introduce an external judge-based delegation strategy and use it to systematically investigate key design choices in edge-cloud collaboration, including when to delegate, how delegation decisions should be made, what context should be shared, and how frequently the cloud LLM should be invoked. Experiments show that EdgeWise matches or outperforms pure-LLM execution while reducing inference cost up to 62% on SWE-Rebench and 34% on Terminal-Bench 2.1. We further deploy EdgeWise on a local workstation, demonstrating comparable cost savings in a practical edge-cloud setting.

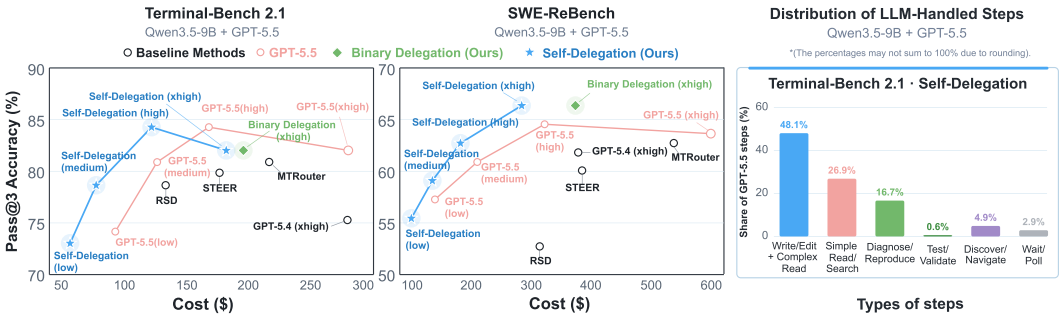

Figure 1: The two left figures show the trade-off between Pass@3 performance and inference cost for our method and the baselines across different settings and reasoning efforts, with GPT-5.5 as the comparison LLM. **Across all reasoning efforts on both benchmarks, our method achieves significantly lower costs than the vanilla model**. The right figure shows the proportions of different step types among the steps delegated to the LLM.

1 INTRODUCTION

LLM-based agents have achieved strong performance on increasingly complex long-horizon tasks (Team et al., 2026; Chandiramani et al., 2026; GPT, 2026). However, frontier LLMs require substantial computation and are typically served through cloud infrastructure, making it costly to rely on them throughout long-horizon task execution. Importantly, model capability requirements

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.

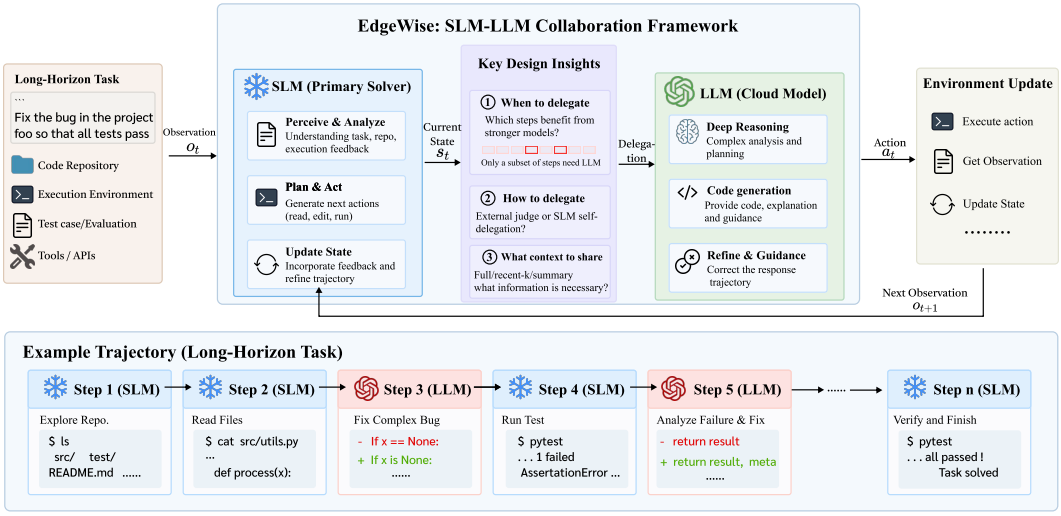

Figure 2: The upper panel illustrates the overall pipeline of EdgeWise, while the lower panel provides an example of how EdgeWise operates during task execution.

can vary across intermediate steps, and many simpler steps can be handled by smaller models deployed locally. Recent works have explored allocating queries of different difficulty levels to models with varying capabilities. (Panda et al., 2025; Maurya et al., 2025; Ramírez et al., 2024; Aggarwal et al., 2024). Beyond single-query routing, Router-R1 (Zhang et al., 2026a) and MERA (Yao et al., 2026) extend model selection to multi-step settings, dynamically determining which model should handle intermediate invocations as the task progresses. However, existing methods still have critical limitations. (1) **Limited Long-Horizon Evaluation**: As noted by TwinRouterBench (Yang et al., 2026), most existing methods are evaluated on single-query reasoning or multi-hop QA, leaving real-world long-horizon tasks such as coding and deep research largely unexplored. (2) **Model-Pool Dependency**: Existing approaches typically rely on additional decision models trained over predefined model pools, making their decisions dependent on specific model capabilities and potentially less reliable as the model pool evolves.

Motivated by these limitations, as well as recent advances in SLM capabilities and edge computing hardware that make it increasingly practical to deploy capable SLMs directly on local devices, such as the NVIDIA DGX Spark[1], we consider an edge–cloud setting where a locally deployed SLM serves as the primary solver for long-horizon tasks and selectively leverages a stronger cloud LLM when needed. Based on this setting, we propose **EdgeWise**, a step-level edge–cloud collaboration framework in which the local SLM continuously interacts with the environment and selectively delegates critical steps to the cloud LLM. We explore two delegation mechanisms: an external judge-based strategy that dynamically determines when the cloud LLM is needed, and a self-delegation strategy that integrates this capability directly into the SLM through step-level RL with fine-grained credit assignment. Beyond these mechanisms, we systematically analyze the key design factors of edge–cloud collaboration, including when to delegate, what context should be shared, and how delegation frequency should be controlled. Our analysis reveals that the benefit of the cloud LLM is concentrated in a small subset of critical steps, enabling EdgeWise to reduce cloud-model usage while preserving the performance of full LLM execution. Experimental results show that our methods, particularly RL-based self-delegation, maintain competitive performance while reducing inference cost by an average of 46% across models and benchmarks. We further validate EdgeWise in a practical edge–cloud deployment, where the locally deployed SLM achieves comparable performance and cost savings. Figure 1 shows the Pass@3–cost trade-off and demonstrates that our method selectively delegates harder steps to the LLM, with further analysis in Appendix F. In summary, our main contributions are as follows:

- We propose ***EdgeWise***, a step-level edge–cloud collaboration framework for long-horizon agents, where an SLM deployed on the local device serves as the primary solver and selectively leverages a stronger cloud LLM when needed.

[1] https://build.nvidia.com/spark

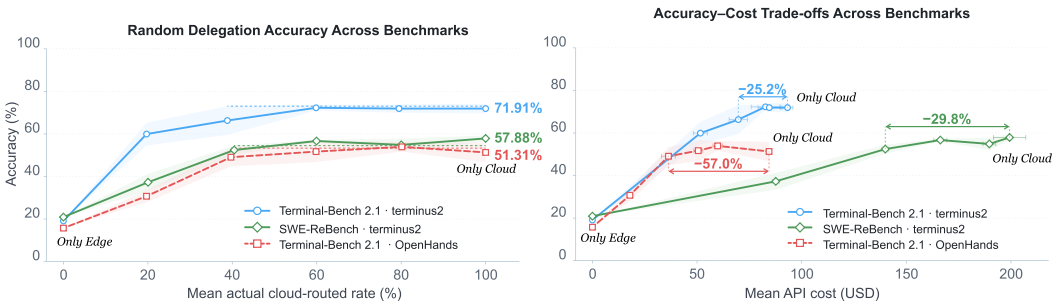

Figure 3: The leftmost and rightmost points represent Qwen3.5-9B-only and GPT-5.5-only, respectively, with four intermediate LLM usage ratios. The left figure shows accuracy as the LLM ratio increases, while the right reports the corresponding average cost over three runs. Each point is averaged over three runs, with shaded regions indicating standard deviation. This figure shows that, with a cloud usage rate of only around 40%, delegating the right steps to the cloud model achieves performance comparable to the vanilla model while reducing cost.

- We systematically investigate the **key design factors of EdgeWise**, including when the cloud LLM is needed, how delegation decisions should be made, what context should be shared between local and cloud models, and how frequently the cloud LLM should be invoked. Our analysis reveals that the benefit of the cloud LLM is concentrated in a small subset of critical steps.
- Within EdgeWise, we introduce two cloud delegation mechanisms: external-judge-based delegation and self-delegation. In particular, our **step-level RL method with fine-grained credit assignment** enables the local SLM to jointly learn task solving and when to leverage the cloud LLM, without an additional decision model at inference time.
- Extensive experiments on long-horizon coding-agent benchmarks demonstrate that our self-delegation method maintains competitive task performance while reducing inference cost by an average of 46% across different LLMs and benchmarks. We further conduct a real-world deployment evaluation, demonstrating the practical feasibility of our edge–cloud collaboration framework for long-horizon agents.

2 Observation: Delegating Only Key Steps Matches Large-Model Performance

Long-horizon coding tasks typically involve a sequence of heterogeneous steps, including problem understanding, root-cause localization, code editing, testing, and result inspection. These steps differ in complexity and may therefore require different levels of model capability. Nevertheless, existing agent harnesses typically rely on a large model throughout the entire trajectory, allocating the same level of model capability to every step. This raises a natural question: *Does every step in a long-horizon trajectory require a large model?* To investigate this question, we first conduct a random delegation experiment. We use the SLM as the primary solver and randomly delegate a subset of steps to the LLM. Before each step, a delegation decision is independently sampled according to a predefined probability. If selected, the LLM executes the current step, after which control returns to the SLM for subsequent execution; otherwise, the SLM handles the step locally. This setup allows us to examine how task performance changes when the LLM participates in only a subset of steps, and whether the placement of these LLM calls affects the overall performance.

In our experiments, we use Qwen3.5-9B (Qwen Team, 2026) as the local model and GPT-5.5 (OpenAI, 2026) with xhigh reasoning effort as the cloud model, with a maximum of 80 turns per trajectory. We follow the default settings of each benchmark, including the original execution-time constraints, and evaluate on Terminal-Bench 2.1 and SWE-Rebench using two agent harnesses, Terminus2 and OpenHands (OpenHands, 2026). To investigate whether the allocation of cloud intelligence matters, we randomly select a fixed proportion of steps to be handled by the cloud model. As shown in Figure 3, when the cloud-model ratio reaches 40%, performance exhibits substantial variation across different random allocations. Notably, some allocations achieve performance comparable to using the cloud model throughout the entire trajectory, while reducing inference cost by at least 25%. This suggests that **a relatively small number of well-placed cloud-model calls can be**

sufficient for strong task performance, with the local model effectively handling the remaining steps. Overall, these results indicate that effective edge SLM–cloud LLM collaboration depends not only on *how much* stronger model capacity is used, but more importantly on *where* it is allocated along the trajectory.

3 EDGEWISE: A FRAMEWORK FOR SLM–LLM COLLABORATION IN PROBLEM SOLVING

Based on this finding, we introduce **EdgeWise**, an edge SLM and cloud LLM collaboration framework that uses the SLM as the primary solver and selectively delegates challenging steps to the LLM when stronger capability is needed. In this section, we systematically investigate the key design choices of such collaboration, including when and how to delegate, with the goal of maintaining strong task performance while reducing LLM usage and inference cost.

3.1 WHEN SHOULD THE SLM DELEGATE TO THE LLM?

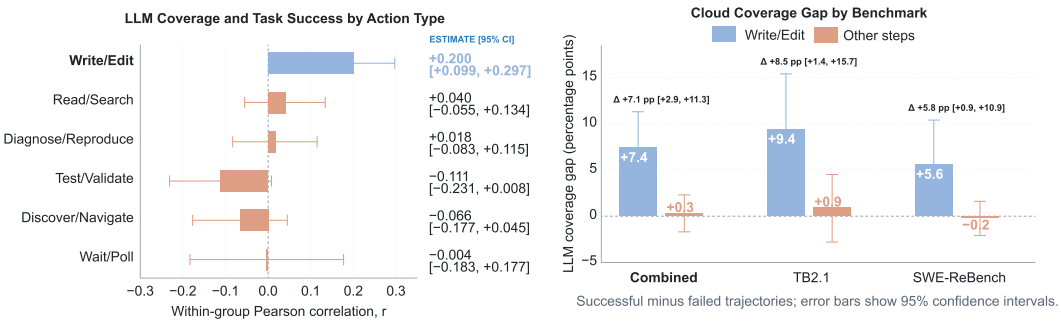

Figure 4: Left: Correlation between cloud coverage and task success across action types. Right: Cloud coverage differences between successful and failed trajectories. Error bars denote 95% confidence intervals. Higher coverage of Write/Edit steps is consistently associated with task success.

In Section 2, we observe that delegating only a subset of steps to the LLM can substantially reduce inference cost, with some random allocations achieving performance comparable to full LLM execution. However, at a 40% delegation ratio, performance varies considerably across random allocations, suggesting that **where to delegate** is critical under a limited budget. This motivates the first key question in designing EdgeWise: **Which steps along the trajectory should be delegated to the large model?** To investigate this question, we analyze 2.4k coding-agent trajectories and categorize each step into six action types: *Write/Edit*, *Read/Search*, *Diagnose/Reproduce*, *Test/Validate*, *Discover/Navigate*, and *Wait/Poll*. For each type, we measure its cloud coverage, defined as the fraction of steps delegated to the cloud model. To control for task difficulty and delegation budget, we group trajectories by benchmark, task, and random-selection budget, and center both cloud coverage and binary reward within each group. We then pool the centered observations and compute the Pearson correlation:$r_a = \frac{\sum_i \widetilde{C}_{a,i}\widetilde{R}_i}{\sqrt{\sum_i \widetilde{C}_{a,i}^2}\sqrt{\sum_i \widetilde{R}_i^2}}$, where $\widetilde{C}_{a,i}$ and $\widetilde{R}_i$ denote the centered cloud coverage of action type a and task reward, respectively. This measures whether greater cloud coverage of a particular action type is associated with higher task success under the same task and delegation budget. We estimate 95% confidence intervals using cluster bootstrap over task-and-budget groups.

As shown in the left panel of Figure 4, *Write/Edit* is the action type whose cloud coverage is most strongly associated with task success, with a correlation of 0.200 and a 95% confidence interval of $[0.099, 0.297]$. *Read/Search* shows a weaker positive association, while the remaining action types have smaller correlations with confidence intervals overlapping zero. The right panel further supports this finding: successful trajectories allocate more cloud coverage to *Write/Edit*, with gaps of 9.4 percentage points on Terminal-Bench 2.1 and 5.6 on SWE-ReBench, while differences for other action types remain small. Together, these results identify **Write/Edit** as the action type most strongly associated with task success, suggesting that under a limited delegation budget, performance depends more on delegating critical steps than on uniformly increasing cloud-model usage.

3.2 WHO SHOULD DECIDE WHETHER TO DELEGATE?

Our analysis suggests that different steps benefit differently from stronger model allocation, motivating a more selective delegation strategy. This raises a practical question: *how should the delegation decision be made at each step?* We explore two mechanisms. The first uses an external judge to dynamically determine whether the current step should be delegated, while the second integrates this capability directly into the SLM, enabling it to jointly learn task solving and delegation.

3.2.1 KEY-STEP-AWARE DELEGATION WITH AN EXTERNAL JUDGE

Building on the key-step analysis in Section 3.1, we design a key step-aware delegation mechanism that selectively delegates critical steps to the cloud LLM. Specifically, based on the patterns observed in Figure Figure 4, we use Write/Edit and complex Read/Search operations as guidance for identifying key steps in our delegation mechanisms, and translate these patterns into delegation criteria. We then introduce an external judge to dynamically identify such key steps and determine whether the current step should be delegated to the cloud LLM throughout the trajectory. Given the trajectory history h_t at step t, the judge determines whether the next step exhibits the characteristics of a key step and makes a binary delegation decision: $a_t = J(h_t), a_t \in \{\text{Local}, \text{Cloud}\}$. If the current state is identified as requiring stronger model, the next step is delegated to the cloud model; otherwise, the local model continues execution. After each step, the judge reevaluates the updated trajectory, allowing key-step identification and model allocation to adapt dynamically as the task progresses. We instantiate GPT-5.4-mini as the external judge and incorporate the key-step patterns identified in Figure 4 into its delegation criteria. As shown in Figure 5, this key step aware delegation strategy preserves most of the performance of the pure cloud-model baseline while reducing cloud-model usage. These results validate that the identified key-step patterns can effectively guide delegation, further supporting our finding in Section 3.1 that delegating Write/Edit and complex Read/Search steps to the LLM benefits SLM task solving. We therefore use this mechanism as the training-free variant of EdgeWise and further investigate its design choices, including LLM context (Section 3.3), delegation context (Section 3.3), and delegation frequency (Section 3.4).

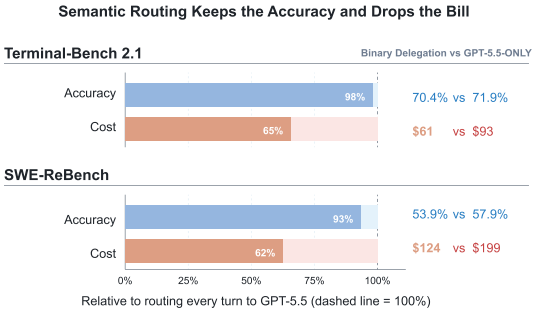

Figure 5: Performance and API cost of our training-free key-step delegation method compared with the pure large-model baseline. All metrics are averaged over three independent runs.

3.2.2 TRAINING THE SLM FOR SELF-DELEGATION

In the previous section, we use an external judge to determine whether each step should be handled by the SLM or delegated to the LLM. To eliminate the need for an additional model, we formulate self-delegation as a joint task-solving and delegation problem, where the SLM learns to solve the task while selectively invoking the cloud LLM when additional model capability is useful. We adopt a two-stage training pipeline: SFT first teaches the SLM the delegation format, followed by RL that encourages efficient task solving while learning to delegate only critical steps to the LLM.

Stage1: SFT training on based model. Specifically, we use GPT-5.5 to generate task-solving trajectories. At each step, GPT-5.5 additionally considers whether the step would require delegation if executed by a smaller model, and generates the corresponding delegation label. Importantly, these labels are not intended to indicate that a step *must* be delegated to the LLM. Instead, they serve as format supervision, enabling the SLM to learn delegation as a valid action. Given the constructed dataset $\mathcal{D}_{\text{SFT}} = \{(x_i, y_i)\}_{i=1}^{N}$, we optimize the standard language modeling objective: $\mathcal{L}_{\text{SFT}} = -\sum_{(x,y)\in\mathcal{D}_{\text{SFT}}} \log p_\theta(y \mid x)$, where y contains both the task-solving trajectory and the corresponding delegation actions.

Stage 2: RL Training for Self-Delegation. After the first-stage training, we observe that although the SLM learns how to delegate to the LLM, it does not yet learn how to allocate cloud-model usage effectively across different steps, leading to suboptimal delegation behavior Section D.3 and

degraded task performance Section C.1. To address this issue, we introduce an RL-based training method that jointly encourages successful task completion and selective allocation of LLM calls to critical steps, as identified in Section 3.1. During training, we follow the output format introduced in the SFT stage. At each step, the SLM first decides whether to delegate to the LLM. For local execution, the SLM generates the analysis and action itself; otherwise, the LLM generates the subsequent output. Since the delegated output is generated by the LLM, these tokens are excluded from the policy loss. Our trajectory-level reward combines an outcome reward for task correctness with a cloud-usage regularization term:

$$R_i = V_i + \mathbb{I}(V_i = 1)\frac{1}{1 + \lambda C_i}, \tag{1}$$

where V_i indicates whether the final answer is correct and C_i is the number of LLM calls. λ is a hyperparameter determined by the typical number of cloud calls per trajectory. A larger number of cloud calls requires a smaller λ, and vice versa, to maintain an appropriate scale for the cost reward. In our experiments, we set $\lambda = 0.1$ based on the cloud call statistics of the training data. The second term favors successful trajectories that use fewer cloud invocations, serving as a regularizer on overall cloud usage rather than directly optimizing monetary inference cost. Importantly, this trajectory-level signal only controls the overall amount of delegation and does not specify which steps should receive the limited cloud-model capacity. To explicitly learn *where* to allocate LLM calls, we introduce step-level credit assignment for individual delegation decisions. In standard GRPO, tokens within a sampled response typically share the same trajectory-level advantage, making it difficult to distinguish critical delegation decisions from routine ones. Based on the criteria derived in Section 3.1, we use GPT-5.4-mini as a lightweight critic model. Given the full rollout trajectory, the critic model evaluates each delegation decision according to the key-step criteria defined in Section 3.1. Decisions consistent with the critic model's assessment receive a positive advantage, while unnecessary delegation or missed delegation receives a negative advantage. No additional advantage is assigned when local execution is sufficient. In this way, the step-level signal does not simply encourage fewer delegations; instead, it encourages SLM to concentrate cloud-model usage on steps where delegation is most valuable. For delegation tokens at step i, its final advantage is

$$A_t = A_{\text{traj}} + \beta \frac{N_{\text{trajectory}}}{N_{\text{route}}} A_i^{\text{del}}, \tag{2}$$

where A_{traj} is the trajectory-level GRPO advantage, $N_{\text{trajectory}}$ is the number of training tokens in the trajectory, A_i^{del} is the token level advantage on delegation tokens and N_{route} is the number of delegation tokens. The scaling factor prevents the step-level signal from being diluted when delegation tokens are sparse, while β controls its relative contribution to the policy loss. We evaluate the effectiveness of this step-level credit assignment in Section C.2 and further compare the SFT and RL models in Section C.1. We can see our method not only reduces the number of delegations but also allocates cloud model capacity more selectively to critical steps. We show the agreement between SLM and critic model in Appendix E.

3.3 What Context Should Be Provided for Collaboration?

Effective SLM–LLM collaboration depends not only on when cloud model is introduced, but also on what information is available to different components when making their decisions. In EdgeWise, context serves two distinct purposes. The LLM requires sufficient information to understand the current task state and effectively perform a delegated step, while the judge requires appropriate information to determine whether delegation is necessary in the first place. We therefore investigate the context provided to these two components separately and we empirically compare these configurations in Section 4.3.

What Information Should Be Passed to the LLM? When a step is delegated, the LLM requires sufficient context to understand the current task state, while longer context also increases inference cost. We therefore investigate five context configurations: *(1) Full History*, which provides the complete preceding trajectory; *(2) Previous Step*, which provides only the analysis, action, and observation from the preceding step; *(3) Recent History*, which provides the most recent k turns; *(4) No SLM Reasoning*, which removes the SLM's analysis while retaining executed commands and observations; and *(5) Relevant Steps*, where the judge selects previous steps most relevant to the upcoming delegated step. These configurations examine the trade-off between context completeness and inference cost.

What Information Should Be Passed to the Judge? Unlike the LLM, the judge only determines whether the next step should be delegated and may therefore require different contextual information. We consider three configurations for step n: *(1) Full History*, containing steps 1 to $n-1$; *(2) Full History + Current Analysis*, which additionally includes the SLM's analysis for step n; and *(3) Current Analysis Only*, which provides only the SLM's current analysis.

3.4 How Frequently Should the LLM Be Invoked?

Beyond determining which steps should be delegated, another important design choice is how frequently the LLM should be invoked. More frequent delegation provides greater access to stronger model capabilities but also increases inference cost, whereas overly conservative delegation may prevent the LLM from intervening at critical steps. Therefore, EdgeWise requires a mechanism to explicitly control LLM involvement and balance task performance against inference cost.

In Section 3.2.1, the external judge makes a binary decision at each step, determining whether the next step should be handled by the SLM or delegated to the LLM. While effective for identifying potential delegation opportunities, binary decisions provide limited control over the overall delegation frequency. To address this limitation, we extend the judge to produce a continuous *delegation score*. Given the trajectory history h_t at step t, the judge produces $s_t = J(h_t)$, where s_t represents the estimated necessity of invoking the LLM for the next step. The delegation decision is then determined by a predefined threshold τ:$a_t = \begin{cases} \text{Cloud}, & s_t > \tau, \\ \text{Local}, & s_t \leq \tau. \end{cases}$ After each step, the judge reevaluates the updated trajectory and produces a new delegation score, allowing the decision to adapt as task execution progresses. By adjusting τ, we can directly control the level of LLM involvement: a lower threshold results in more frequent delegation, while a higher threshold keeps more steps on the local SLM. This mechanism enables EdgeWise to operate under different cloud budgets and provides a flexible way to explore the trade-off between task performance and inference cost.Appendix G shows the relationship between the delegation threshold and the resulting cloud-call budget.

4 Experiments

4.1 Experimental Setup

For training, we use SWE-Smith with Qwen3.5-9B as the SLM and Kimi-K3 as the LLM. We use 2K examples for SFT, followed by RL training on 40 NVIDIA A100 80GB GPUs for 175 steps, with a batch size of 8 and 16 rollouts per example. We evaluate on Terminal-Bench 2.1 and SWE-ReBench-03 using the Terminus2 harness, limiting each trajectory to 80 turns while retaining the default benchmark settings. Learning curve and cloud usage are shown in Appendix E and Section D.3, respectively.

4.2 Main Results of Our Framework

We evaluate our delegation strategies against several multi-turn baselines, including LLM-only, random delegation, RSD (Liao et al., 2025), STEER (Lee et al., 2026), and MTRouter (Zhang et al., 2026b). We use Qwen3.5-9B as the SLM and GPT-5.5 (xhigh) or Claude Opus 4.8 via Bedrock[2] as the cloud LLM, and evaluate on Terminal-Bench 2.1 and SWE-ReBench with a maximum of 80 turns. We report Pass@1, Pass@3, and inference cost in Table 1. Cost is computed from cloud LLM API usage based on input and output token pricing; local SLM computation is excluded, while judge API cost is included for binary delegation. Our methods consistently achieve better performance–cost trade-offs than the baselines. In particular, self-delegation reduces Pass@3 cost by 53%/63% on SWE-ReBench and 35%/33% on Terminal-Bench 2.1 with GPT-5.5/Opus 4.8, respectively, while maintaining performance comparable to LLM-only. The advantage is more pronounced under Pass@3: with Opus 4.8 on Terminal-Bench 2.1, self-delegation trails LLM-only in Pass@1 but matches it in Pass@3. We hypothesize that SLM–LLM collaboration mitigates overthinking and increases exploration diversity across trajectories. We further study generalization in Appendix D across different harnesses, SLMs and judge models.

[2]We observe slightly lower performance for Opus 4.8 via Bedrock due to configuration differences.

Table 1: Performance and cost across benchmarks and baselines. Binary and Threshold Delegation costs include the external judge. **Bold** and underlined denote the best and second-best results excluding vanilla LLM. The last rows report Self-Delegation changes relative to vanilla LLM.

Method	Qwen3.5-9B + GPT-5.5(xhigh)				Qwen3.5-9B + Opus4.8(xhigh)			
	Pass@1		*Pass@3*		*Pass@1*		*Pass@3*	
	Acc. (%) ↑	Cost ($) ↓	Acc. (%) ↑	Cost ($) ↓	Acc. (%) ↑	Cost ($) ↓	Acc. (%) ↑	Cost ($) ↓
Terminal-Bench2.1								
Vanilla LLM	74.16	90.31	82.02	279.73	67.42	64.78	75.28	187.72
Random Delegation	66.29	74.34	79.78	209.13	58.42	54.00	67.42	158.18
RSD(Liao et al., 2025)	62.92	**45.61**	78.65	**133.25**	53.93	**28.09**	61.80	**82.39**
STEER(Lee et al., 2026)	66.29	61.81	79.78	176.50	53.93	41.74	62.92	130.10
MTRouter(Zhang et al., 2026b)	68.54	73.55	80.90	216.30	60.67	51.47	71.91	160.13
Delegation Strategies (Ours)								
Binary Delegation	**71.91**	59.16	**82.02**	195.63	**66.29**	58.16	**79.78**	168.19
Self-Delegation	**71.91**	56.96	**82.02**	181.88	64.04	42.98	75.28	126.53
Δ vs. Vanilla LLM	*-3.03%*	*-36.93%*	*+0.00%*	*–34.98%*	*-5.01%*	*–33.65%*	*+0.00%*	*–32.60%*
SWE-ReBench								
Vanilla LLM	56.36	200.50	63.64	598.08	60.00	199.06	69.09	584.70
Random Delegation	50.00	138.38	62.73	455.54	50.90	132.17	60.00	398.70
RSD(Liao et al., 2025)	39.03	98.49	52.73	313.09	46.36	155.26	64.55	465.72
STEER(Lee et al., 2026)	48.18	125.90	60.00	383.63	51.82	141.01	**66.36**	431.38
MTRouter(Zhang et al., 2026b)	48.18	180.11	62.73	536.45	49.09	175.20	60.00	540.90
Delegation Strategies (Ours)								
Binary Delegation	55.45	127.20	**66.36**	372.60	47.27	127.63	60.91	357.07
Self-Delegation	**58.18**	**97.80**	**66.36**	**283.02**	**57.27**	**74.28**	**66.36**	**221.39**
Δ vs. Vanilla LLM	*+3.23%*	*-51.22%*	*+4.27%*	*–52.68%*	*-4.55%*	*–62.68%*	*-3.95%*	*–62.14%*

4.3 Effect of Context Passing on SLM/LLM Collaboration

Context Provided to LLM. In this section, we evaluate the context-passing strategies from the SLM to the LLM introduced in Section 3.3. For the recent-k-steps strategy, we set $k = 3$. As shown in the results, passing the full context achieves the best performance but incurs the highest inference cost. We attribute this to long-horizon tasks, where critical steps delegated to the LLM often require more complete historical information. Full context minimizes information loss during the SLM-to-LLM transition. This is further supported by the *no SLM reasoning* strategy, which preserves nearly the full interaction history while excluding the SLM's reasoning, achieving the second-best performance at a lower cost. These results suggest that effective SLM–LLM collaboration requires not only deciding *when* to switch models, but also *what* information to transfer. Preserving sufficient context during model transitions is critical, and context should therefore not be compressed indiscriminately.

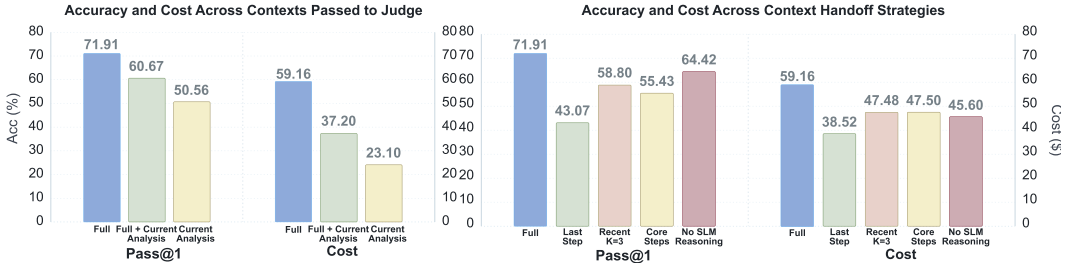

Figure 6: The left panel shows the accuracy and inference cost under three different context configurations provided to the judge. The right panel shows the accuracy and inference cost under five different context configurations passed from the SLM to the LLM.

Context Provided to the Judge. In this section, we evaluate the three judge context configurations introduced in second part of Section 3.3. The results are shown in the left panel of Figure 6. Among the three configurations, Full History achieves the best performance. We attribute this advantage to the complete trajectory information available to the judge, which provides a more comprehensive view of previous actions, observations, and task progress. In contrast, incorporating the SLM's current analysis may introduce additional bias when its interpretation is inaccurate, potentially causing the judge to miss critical steps that require LLM intervention. Although this reduces LLM usage

and inference cost, it also leads to lower task performance. These results suggest that the complete trajectory provides a more reliable context for delegation decisions.

4.4 Practical Edge–Cloud Collaboration

Budget-Aware Cloud Delegation. Following Section 3.4, we introduce a threshold into the external-judge-based method to explicitly control the delegation budget. We first split a validation set from SWE-Smith and evaluate thresholds of $0.2, 0.4, 0.6, 0.8$. As shown in Figure 7(a), thresholds around 0.2–0.4 achieve better performance–cost trade-offs on the validation set. Based on this observation, we further evaluate the method on Terminal-Bench 2.1 and SWE-ReBench with different LLMs, as shown in Figures (b) and (c). The results exhibit a similar trend, where thresholds around 0.2–0.4 maintain strong task performance while substantially reducing inference cost compared with the LLM-only baseline. We further analyze the relationship between the threshold and delegation budget in Appendix G.

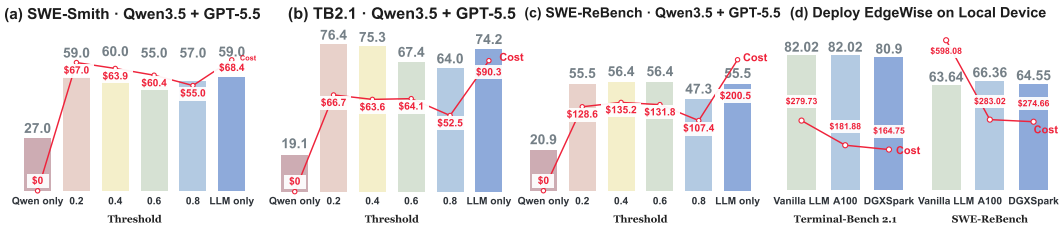

Figure 7: The left 3 figures shows the accuracy and inference cost under three different context configurations provided to the judge. The right figure shows the accuracy and inference cost under five different context configurations passed from the SLM to the LLM.

Real-World Edge Deployment. In this section, we deploy EdgeWise on a real-world edge device to evaluate its practical feasibility. Specifically, our self-delegation model is deployed on an NVIDIA DGX Spark as the primary solver, selectively delegating steps to a cloud LLM via API when stronger capability is needed. We follow the same experimental configuration as in the previous evaluation. As shown in Figure 7(d), the edge deployment achieves Pass@3 performance comparable to our A100-based evaluation while reducing inference cost by approximately 50% compared with the vanilla LLM. These results demonstrate that EdgeWise preserves its performance-cost benefits in a practical edge–cloud setting. Additional details are provided in Section H.2.

5 Related Work

Recent studies have explored model routing to reduce inference cost while maintaining performance, ranging from query-level model selection (Feng et al., 2025; Ong et al., 2025; Zhao et al., 2024) to cascade and fine-grained routing that dynamically invokes stronger models during generation (Aggarwal et al., 2024; Zheng et al., 2025; Fu et al., 2026). More recent work extends routing to multi-step and agentic settings, dynamically selecting models or collaboration strategies across intermediate steps (Zhang et al., 2026a; Yue et al., 2025; Yao et al., 2026). However, existing approaches often rely on additional routers tied to a predefined model pool and remain underexplored for complex long-horizon agent tasks such as coding and deep research. We provide a more comprehensive discussion of related routing methods in the Appendix A.

6 Conclusion

In this work, we present EdgeWise, an efficient agent framework that enables collaboration between local SLMs and stronger cloud models for long-horizon tasks. Our analysis shows that stronger intelligence is primarily needed at a small number of critical steps. Based on this observation, EdgeWise selectively delegates these steps to the cloud model while retaining the remaining execution locally. Across agentic coding benchmarks, EdgeWise substantially reduces cloud-model usage while maintaining competitive task performance, demonstrating the potential of step-level model collaboration for building efficient and practical agent systems.

AI USE STATEMENT

In this paper, we used AI tools solely to refine grammar and improve the clarity and readability of the text.

ETHICS STATEMENT

All datasets used in this work are publicly available and contain no sensitive information. Our method enhances LLM reasoning without introducing new data collection or human interaction. While stronger reasoning ability may be misused, we emphasize that this work is intended for beneficial research and responsible applications.

REPRODUCIBILITY STATEMENT

We will release our code and data once the paper is published. The appendix includes detailed experimental setups and hyperparameters so that others can reproduce our results. We also encourage the community to follow good research practices when using our code and data, to help maintain the reliability and transparency of future work.

REFERENCES

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

Aakshita Chandiramani, Aaron Blakeman, Abdullahi Olaoye, Abhibha Gupta, Abhilash Somasamudramath, Abhinav Khattar, Adeola Adesoba, Adi Renduchintala, Adil Asif, Aditya Agrawal, et al. Nemotron 3 super: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2604.12374*, 2026.

Yifan Dou, Shikan Fang, and Shibo Li. Conformal cascade: Distribution-free accuracy guarantees for multi-tier llm inference. *arXiv preprint arXiv:2607.25018*, 2026.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, volume 2025, pp. 26186–26203, 2025.

Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang Yuan, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. R2r: Efficiently navigating divergent reasoning paths with small-large model token routing. *Advances in Neural Information Processing Systems*, 38:124108–124145, 2026.

GPT. Gpt5.6, 2026. URL https://openai.com/index/previewing-gpt-5-6-sol/.

Shihong Huang, Shengjie Wang, Hong Ma, and Zhou Xu. Rlcascaderouter: Quality-estimator-free cascade routing via reinforcement learning. *arXiv preprint arXiv:2608.15817*, 2026.

Sangmook Lee, Dohyung Kim, Hyukhun Koh, Nakyeong Yang, and Kyomin Jung. Confidence-guided stepwise model routing for cost-efficient reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31483–31491, 2026.

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. Reward-guided speculative decoding for efficient llm reasoning. *arXiv preprint arXiv:2501.19324*, 2025.

Kaushal Kumar Maurya, KV Aditya Srivatsa, and Ekaterina Kochmar. Selectllm: Query-aware efficient selection algorithm for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20847–20863, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

OpenAI. GPT-5.5 Instant: smarter, clearer, and more personalized, 2026. URL https://openai.com/zh-Hans-CN/index/introducing-gpt-5-5/.

OpenHands. The open platform for agentic software development, 2026. URL https://www.openhands.dev/.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive llm routing under budget constraints. In *EMNLP (Findings)*, pp. 23934–23949, 2025.

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

Guillem Ramírez, Alexandra Birch, and Ivan Titov. Optimising calls to large language models with uncertainty-based two-tier selection. *arXiv preprint arXiv:2405.02134*, 2024.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-router: A new paradigm for llm routing with reasoning. *arXiv preprint arXiv:2602.02823*, 2026.

Pei Yang, Wanyi Chen, Tongyun Yang, Pengbin Feng, Jiarong Xing, Wentao Guo, Yuhang Yao, Yuhang Han, Hanchen Li, Xu Wang, et al. Twinrouterbench: Fast static and live dynamic evaluation for realistic agentic llm routing. *arXiv preprint arXiv:2605.18859*, 2026.

Yuhang Yao, Zeyu Wang, Wanyi Chen, Tongyun Yang, Yuhang Han, Jie Xiao, Chengke Bao, Tianyi Zhao, Lynn Ai, Eric Yang, et al. Mera: Model evolution and routing with skill adaptation for agentic systems at scale. *arXiv preprint arXiv:2608.10333*, 2026.

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

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 141233–141265, 2026a.

Yiqun Zhang, Hao Li, Zihan Wang, Shi Feng, Xiaocui Yang, Daling Wang, Bo Zhang, Lei Bai, and Shuyue Hu. Mtrouter: Cost-aware multi-turn llm routing with history–model joint embeddings. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44206–44226, 2026b.

Zesen Zhao, Shuowei Jin, and Z Morley Mao. Eagle: Efficient training-free router for multi-llm inference. *arXiv preprint arXiv:2409.15518*, 2024.

Hang Zheng, Hongshen Xu, Yongkai Lin, Shuai Fan, Lu Chen, and Kai Yu. Disrouter: Distributed self-routing for llm selections. In *International Conference on Learning Representations*, volume 2026, pp. 142781–142810, 2026.

Wenhao Zheng, Yixiao Chen, Weitong Zhang, Souvik Kundu, Yun Li, Zhengzhong Liu, Eric P Xing, Hongyi Wang, and Huaxiu Yao. Citer: Collaborative inference for efficient large language model decoding with token-level routing. *arXiv preprint arXiv:2502.01976*, 2025.

A Related Works

A.1 Adaptive Model Collaboration

With the increasing inference cost of large language models, recent studies have explored model routing to reduce computational cost while maintaining model performance. Early methods mainly focus on query-level routing, where the router selects a model before solving the query (Feng et al.,

2025; Ong et al., 2025; Zhao et al., 2024). DiSRouter (Zheng et al., 2026) further allows candidate LLMs to estimate whether they are suitable for solving the current query. Beyond deciding which model to use, R2-Router (Xue et al., 2026) jointly determines the model and its output-length budget. However, query-level routing makes the routing decision before solving the problem, while the initial query alone may not accurately reflect the actual difficulty encountered during problem solving. This motivates cascade routing, which dynamically determines whether a stronger model is needed based on the output of a smaller model (Aggarwal et al., 2024; Ramírez et al., 2024). Conformal Cascade (Dou et al., 2026) uses conformal prediction to estimate the reliability of the small model's answer and escalates to a larger model when the answer is considered unreliable. RLCascadeRouter (Huang et al., 2026) formulates model invocation as an MDP and uses reinforcement learning to optimize the performance-cost trade-off.

Beyond response-level cascading, other studies explore fine-grained routing during generation. CITER (Zheng et al., 2025) dynamically determines whether the small or large model should generate the next tokens. R2R (Fu et al., 2026) uses a router to identify reasoning-divergent positions and invokes the large model only at these critical positions. While the above methods mainly focus on reasoning generation, routing becomes more challenging in agentic and multi-step tasks, where models need to handle new observations at each step. Router-R1 (Zhang et al., 2026a) trains a router to dynamically invoke different models from a model pool for multi-hop QA. MasRouter (Yue et al., 2025) trains a router to determine the collaboration mode among multiple agents and allocate an LLM to each agent. MERA (Yao et al., 2026) assigns easier model invocations to smaller models and falls back to larger models for more difficult ones, while further using successful large-model invocations to train the smaller model.

Despite this progress, existing approaches still have several limitations. First, long-horizon agent tasks, such as coding and deep-research tasks, can involve complex and lengthy trajectories, and existing routing approaches may not scale effectively to such settings. Second, existing methods typically rely on an additional router that selects models from a predefined model pool. Such routers are therefore tied to the specific candidate models considered during routing, making it difficult to generalize when the underlying models are updated or replaced.

B Limitations

In this work, we primarily focus on long-horizon coding tasks and do not evaluate our framework on other types of agent tasks, such as deep research or computer-use agents (CUA). While we expect similar opportunities for selective edge–cloud collaboration to exist in these settings, further experiments are needed to validate this hypothesis. We leave the extension to broader long-horizon agent tasks for future work.

C Ablation Study

C.1 Comparison between SFT model and RL model

In this section, we compare our final self-delegation model with the model obtained after the first-stage SFT. As shown in Table 2, the SFT-only model achieves substantially lower Pass@1 on Terminal-Bench 2.1. This suggests that SFT primarily teaches the model how to perform delegation and reduce reliance on the LLM, but does not provide sufficient supervision for identifying *when* delegation is necessary. In particular, the SFT labels serve mainly as format supervision and do not explicitly distinguish critical steps from routine ones. As a result, the model may reduce cloud usage without consistently allocating the LLM to the steps where stronger intelligence is most beneficial. In contrast, our subsequent RL stage introduces fine-grained credit assignment for individual delegation decisions, enabling the model to better identify critical steps while maintaining efficient LLM usage. This comparison highlights the importance of the RL stage for learning an effective self-delegation policy rather than simply learning the delegation action.

Table 2: The pass@1 and cost of SFT-model and Self-delegation model.

	Pass@1	Cost
SFT-Model	57.30%	$ 53.22
Self-Delegation	71.91%	$56.96

C.2 The Effectiveness of Token Advantage

To demonstrate the effectiveness of our step-level RL algorithm, we conduct an ablation study on the proposed token-level advantage. Specifically, we use the same reward function as described in Section 3.2.2, while removing the token-level advantage and training the model solely with the standard GRPO advantage. We evaluate both variants on Terminal-Bench 2.1 using GPT-5.5 xhigh as the LLM and report the Pass@1 performance and delegation statistics in Table 3.

The results show that incorporating the token-level advantage not only improves task performance but also leads to more effective delegation behavior. With token-level advantage, 43% of delegated steps correspond to *write/edit* operations, which we consider more critical steps that typically require stronger reasoning capability, compared with only 29% without token-level advantage. In contrast, the variant without token-level advantage delegates substantially more *read/search* operations to the LLM. This suggests that fine-grained credit assignment encourages a more selective allocation of cloud-model usage across different types of steps. Moreover, the model trained with token-level advantage achieves better performance with a noticeably lower average number of delegations, indicating that fine-grained credit assignment enables the model to allocate expensive LLM calls more selectively to critical steps and this critical step may harder so need more reasoning tokens which lead higher cost.

Table 3: Ablation study of the token advantage during the RL training stage.

	w/ token advantage	w/o token advantage
Acc.	71.91%	65.17%
Cost	$56.96	$55.05
Write / Edit Rate	42.82%	29.20%
Read / Search Rate	32.11%	40.73%
Avg Delegation number/task	5.80	8.97

C.3 Effect of SLM Capability on Self-Delegation

In this section, we investigate whether the effectiveness of self-delegation can be attributed simply to improved SLM capability rather than our delegation-specific training. As a control, we first train Qwen3.5-9B with standard GRPO on the same training data to improve its task-solving capability. We then apply the same SFT stage to teach the strengthened model the delegation format and action, but without introducing our step-level delegation reward. The resulting model therefore has improved task-solving capability and can perform delegation, but does not receive explicit supervision on *when* to delegate. As shown in Table 4, this stronger SLM with delegation still underperforms our self-delegation method. This suggests that stronger task-solving capability alone, even when combined with the ability to delegate, is insufficient to learn an effective delegation policy. Instead, our step-level credit assignment plays an important role in teaching the model where cloud assistance is most beneficial This observation is further supported by the random-delegation experiment in Figure 3. Using the original Qwen3.5-9B without additional training, different placements of cloud-model calls lead to substantial performance variation, with some allocations achieving performance comparable to the vanilla LLM while using substantially fewer cloud calls. Together, these results suggest that effective SLM–LLM collaboration depends not only on the capability of the local SLM, but critically on learning *when* to delegate.

Table 4: Comparison of using GRPO train Qwen3.5-9B and self-delegation model.

Model	**Cloud LLM**	**Pass@1 (%)**
Vanilla Qwen3.5-9B	✗	19.10
Vanilla + GRPO + SFT	✓	32.58
Self-Delegation	✓	71.91

D More Experiments

D.1 Generalization Across Different SLMs

In this section, we evaluate our framework with a different SLM to examine its generalizability across local models. Specifically, we use TMax-9B as the SLM and GPT-5.5 xhigh as the LLM, and evaluate Pass@1 on Terminal-Bench. As shown in Table 6, our framework exhibits a similar trend to that observed with Qwen3.5-9B. With TMax-9B as the local model, our method achieves the same Pass@1 as GPT-5.5 while reducing inference cost by 30.6%.

Table 5: The pass@1 and cost of using Tmax-9B as SLM of our binary delegation method in the framework.

	Pass@1	Jugde Cost	LLM Cost
Qwen3.5-9B	50.56%	$ 0.00	$ 62.11
Qwen3.8-27B	70.79%	$0.00	$75.74
GPT-5.4-mini	71.91%	$5.77	$53.39

D.2 The alteration of Judge Model

In this section, we investigate the impact of using different judge models in the binary delegation method. We consider three judges with different model capacities: Qwen3.5-9B (the SLM itself), Qwen3.8-27B, and GPT-5.4-mini. We keep Qwen3.5-9B as the SLM and GPT-5.5 (xhigh) as the LLM, and evaluate Pass1 on Terminal-Bench 2.1. The results are shown in Table 5. We observe that using the SLM itself as the judge leads to relatively poor performance. However, as the capability of the judge model increases, Qwen3.8-27B and GPT-5.4-mini achieve comparable Pass1 performance. Although using GPT-5.4-mini incurs a lower inference cost than Qwen3.8-27B, the similar performance achieved by the two models demonstrates that our framework is not tied to a specific judge model. In particular, an open-source model such as Qwen3.8-27B can serve as an effective alternative to a proprietary judge while maintaining comparable task performance.

Table 6: The pass@1 and cost of using Tmax-9B as SLM of our binary delegation method in the framework.

	Pass@1	Cost
Tmax-9B	28.09%	$ 0
Binary Delegation	74.16%	$62.64
GPT-5.5(xhigh)	74.16%	$90.31

D.3 Cloud rate in the final answer trajectory

In this section, we examine the LLM delegation rate across different training checkpoints to investigate how the model's tendency to delegate to the cloud evolves during training. The results are shown in Figure 11. At the beginning of training, immediately after SFT, the model has learned the basic delegation behavior but has not yet learned when delegation is appropriate. Consequently,

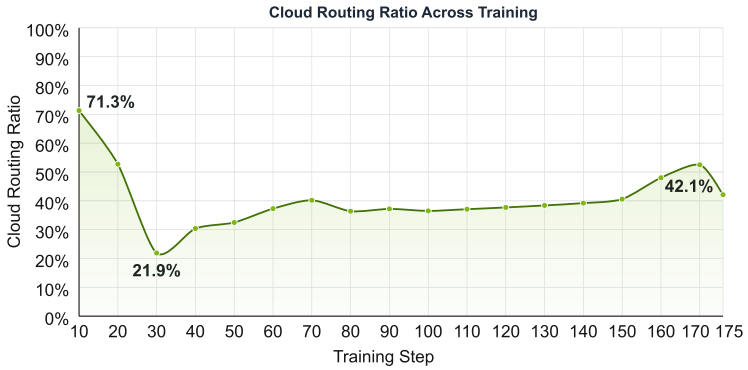

Figure 8: LLM delegation rate in answer trajectories throughout the training process.

a large proportion of steps are delegated to the cloud LLM. As training progresses, the delegation rate decreases substantially, indicating that the penalty on LLM delegation in our reward function effectively encourages the model to reduce unnecessary cloud calls. After reaching its minimum, the delegation rate gradually increases again. We attribute this trend to our token-level advantage, which provides fine-grained supervision and enables the model to progressively learn when delegation to the LLM is beneficial, rather than simply minimizing the number of delegations. Eventually, the delegation rate stabilizes and fluctuates around 40%, at which point we select the final checkpoint for evaluation.

D.4 Generalization of Self-Delegation

In this section, we evaluate the generalization of the checkpoint trained with our self-delegation method. We use GPT-5.5 xhigh as the LLM and evaluate on Terminal-Bench 2.1 using the OpenHands harness. As shown in Table 7, binary delegation generalizes less effectively, likely because the judge prompt is specifically designed for the Terminus-2 harness. In contrast, our trained self-delegation model achieves a Pass@1 comparable to GPT-5.5 xhigh while reducing inference cost by 29.41%. These results demonstrate that the learned self-delegation capability does not overfit to a specific agent harness and can generalize to a different harness without additional training.

Table 7: Pass@1 and inference cost on Terminal-Bench 2.1 using the OpenHands harness.

	Pass@1	Cost
Qwen3.5-9B	15.73%	$ 0.00
GPT5.5(xhigh)	49.43%	$85.18
Binary Delegation	35.96%	$20.73
Self Delegation	48.31%	$60.76

E Learning Curve



Figure 9: Learning curve during the training. From left to right, the three panels illustrate entropy loss, reward and the advantage of judge tokens.

In this section, we show the learning curve of some crucial metrics during the training. The details in Figure 9. We use a running average to smooth the curves. In addition to the standard training dynamics of entropy and reward, we focus on the rightmost panel, which shows the evolution of the advantage assigned to delegation tokens. At the early stage of training, the average advantage is negative, indicating that many delegation decisions are inconsistent with the critic's feedback and therefore receive corrective updates. As training progresses, the advantage increases and eventually converges toward zero, suggesting that the model gradually adapts its delegation behavior and fewer delegation tokens require strong corrective updates. This trend provides evidence that our step-level credit assignment effectively guides the model toward more appropriate delegation decisions.

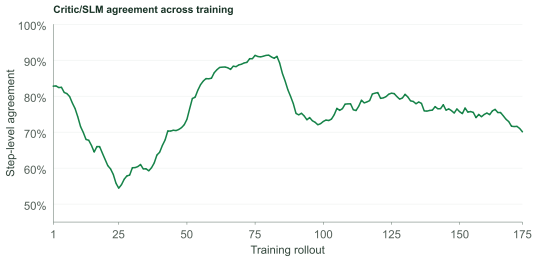

Figure 10: The critic and SLM agreement with the training steps.

We give futher analyse about the agreement of critic and SLM, the results in Figure 10. At the beginning of RL training, the critic agreement first decreases. This is expected because the SFT stage primarily teaches the model the delegation format rather than *when* to delegate. Once RL optimization begins, the policy moves away from its SFT initialization under the trajectory-level task and cost rewards, while its delegation behavior has not yet been well aligned with the step-level supervision. Consequently, many delegation decisions receive negative advantages, as reflected by the strongly negative delegation-token advantage at the beginning of training. As training progresses, our step-level credit assignment begins to correct these decisions: inappropriate delegation decisions receive negative feedback, while appropriate ones are reinforced. Correspondingly, the delegation-token advantage increases and the critic agreement recovers substantially, reaching over 90% at its peak. These consistent trends provide evidence that the step-level reward effectively teaches the model *when* to delegate, rather than merely reducing the overall number of LLM calls. In later training, the agreement stabilizes below its peak and the delegation advantage approaches zero, as the policy jointly balances task success, inference cost, and critic guidance instead of simply imitating the critic.

F Distribution of LLM-Handle Steps

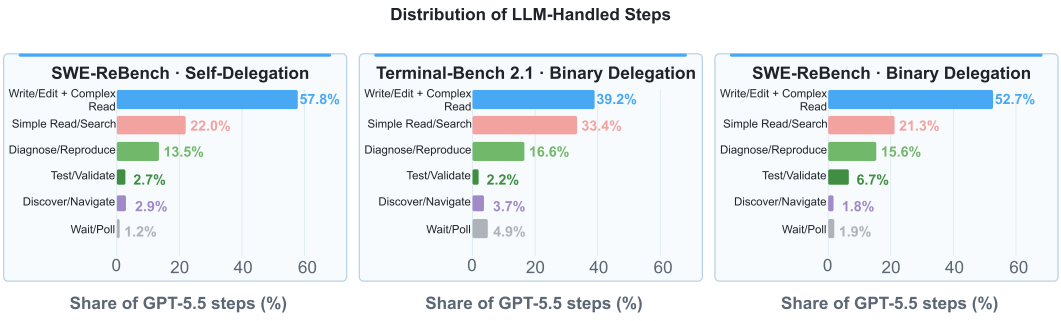

Figure 11: Distribution of step types handled by the LLM on Terminal-Bench 2.1 and SWE-ReBench.

In this section, we further analyze the types of steps handled by GPT-5.5 under both self-delegation and binary delegation across the two benchmarks. The results reveal a consistent delegation pattern across different tasks and strategies. In both settings, the LLM is primarily invoked for more complex operations, particularly write/edit steps and complex read/search steps. Write/edit operations

often require understanding the existing codebase, identifying the correct modification location, and generating changes that remain consistent with the surrounding implementation, making them more challenging than routine execution steps. Similarly, read/search operations can involve cross-file exploration and dependency tracing, which require reasoning over broader repository context. This pattern is consistent with our earlier analysis of critical steps: stronger model capabilities provide greater benefits on steps that require more complex reasoning and cross-file understanding. Importantly, the LLM is not invoked uniformly throughout the trajectory; instead, both delegation mechanisms concentrate cloud-model usage on these more challenging operations while leaving simpler steps to the local SLM. This further suggests that our methods can effectively identify where stronger intelligence is most valuable and selectively allocate the LLM to those steps.

G The Details of Cloud Budget in the Budget-Aware Cloud Delegation

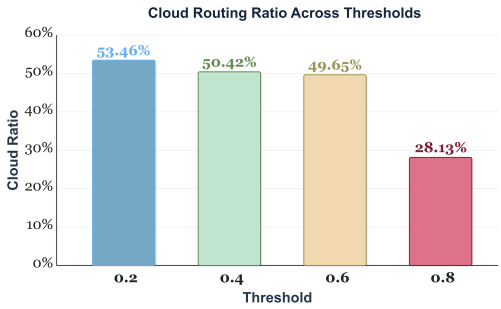

Figure 12: The cloud rate under different threshold.

In this section, we further investigate how the delegation threshold affects the actual cloud-model usage. We use Qwen3.5-9B as the local SLM and GPT-5.5 (xhigh) as the cloud LLM, and evaluate the delegation behavior under different threshold values. For each threshold, we measure the cloud rate, defined as the proportion of agent steps that are delegated to the cloud LLM during task execution. As shown in Figure 12, the cloud rate consistently decreases as the delegation threshold increases. This is expected because a higher threshold requires a higher delegation score before invoking the cloud LLM, causing more steps to remain on the local SLM. These results establish an empirical relationship between the delegation threshold and actual cloud-model usage, showing that the threshold can serve as an effective control parameter for adjusting the level of cloud involvement. We note that the threshold itself does not directly specify a fixed delegation budget; rather, it indirectly controls cloud usage by changing the criterion for delegation. Therefore, different threshold values correspond to different empirical levels of cloud usage, with larger thresholds generally resulting in fewer cloud-model invocations.

H Details of Implementation

H.1 Details of Token-level Advantage

Here, we provide the implementation details of the delegation-token advantage. When both the SLM and the critic determine that the current step should be delegated to the LLM, we assign an advantage of $+1$; when their decisions disagree, we assign an advantage of -1. Here, d_i denotes the delegation decision made by the SLM, while $\hat{d}_i$ denotes the decision made by the critic model.

$$A_i^{\text{del}} = \begin{cases} +1.0, & d_i = \hat{d}_i \\ -1.0, & d_i \neq \hat{d}_i \end{cases} \tag{3}$$

H.2 Details of Deploying on DGXSpark

We deploy Qwen3.5-9B on a single NVIDIA DGX Spark using vLLM 0.19.1 with BF16 precision and single-device inference, setting the GPU memory utilization to 0.65. The deployment reserves approximately 78.7 GB of unified memory at runtime, including 16.8 GiB for model weights and approximately 57.1 GiB available for the KV cache. During active serving, vLLM achieves an average generation throughput of 11.19 tokens/s. On Terminal-Bench 2.1, the average and median end-to-end task completion times are 11.72 and 8.30 minutes, respectively. These measurements provide the deployment and runtime characteristics of EdgeWise on a practical edge device.