# RAOP: Step-Level Resource Orchestration for LLM Agents across Edge and Cloud

**Jinze Li**[1], **Xin Yang**[2], **Shuo Yang**[1], **Jinfeng Xu**[3], **Edith C.H. Ngai**[1*]
[1]The University of Hong Kong [2]University of Tokyo [3]The University of British Columbia
{lijinze-hku, shuo.yang}@connect.hku.hk xin.yang@weblab.t.u-tokyo.ac.jp
jinfeng.xu@ubc.ca, chngai@eee.hku.hk

## Abstract

LLM agents do not present a fixed inference job to an edge and cloud system. At each step, a placement decision chooses where to generate the next message, where to execute any resulting tool call, which prefix must be replayed, which cache state is updated, and whether the episode continues. As a result, the workload to be scheduled is partly produced by the scheduling decisions themselves. We formalize this problem as an *endogenous, sequentially revealed workload* (ESRW) controlled trajectory MDP. We then introduce **Resource Aware Orchestration Policy** (RAOP), a step level orchestration policy that first places language generation and then, after observing any tool call, places tool execution. RAOP predicts candidate action workloads from prefix replay, prefill, KV cache residency, queue, network, and tool payload features, and is trained with GRPO to optimize measured full trajectory success and latency rather than a one step latency surrogate. To make controlled evaluation auditable, the main simulator replays language and tool continuations produced by real model and tool execution while recomputing physical timing under the sampled placement sequence. As an analytical lens, we derive a Bellman regret decomposition showing that myopic placement policies incur error from workload prediction, success priors, and a controlled endogeneity term. On GSM8K, HotpotQA, WebShop, and a calibrated three tier deployment, RAOP matches the strong cloud only success reference within statistical uncertainty while reducing P99 latency by about 53% and communication by about 73%. Against the strongest learned step level baseline, RAOP improves average success by 2.5 points, indicating that full trajectory optimization is central for agent orchestration.

## 1 Introduction

LLM agents are becoming the default interface for tasks that require more than a single model response. They reason over a prompt, call tools, observe the environment, revise their plan, and eventually decide when to stop [Wei et al., 2022, Yao et al., 2023b, Schick et al., 2023, Shinn et al., 2023, Yao et al., 2022]. This execution pattern creates a systems problem that is especially important for phones, robots, vehicles, and other edge devices. Cloud models usually provide stronger reasoning, richer tool access, and higher aggregate throughput. Edge models avoid wide area network latency, reduce communication, and keep more context close to the user. A practical agent serving system must therefore decide not only which model to use, but where each unfinished step of a trajectory should run.

The common abstractions for this decision are not expressive enough for agent workloads. Classical offloading assumes that the job or computation graph is available at admission time, perhaps with uncertain but exogenous resource demands [Wang et al., 2020, Kang et al., 2017]. Query routing

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

and LLM cascades choose a model or backend for a request from prompt level features [Chen et al., 2023, Ong et al., 2024, Ding et al., 2024, Wang et al., 2025]. LLM serving systems improve batching, memory management, and scheduling inside a backend [Kwon et al., 2023, Agrawal et al., 2024, Jaillet et al., 2025]. These lines of work are essential, but they mostly treat the object being scheduled as fixed once the request arrives. Agent execution violates this assumption. The next message may be short or long, may call a tool or terminate, may change the future prompt, and may alter which backend owns a reusable prefix.

This paper studies the missing problem of controlled trajectory formation. At agent step $t$, a placement policy chooses a generation resource. The selected resource produces an agent message. If the message calls a tool, the policy then chooses a tool resource after the call is observed. These choices affect immediate latency through network transfer, queueing, prefix upload, prefill, decode, and tool execution. They also affect future workload through the generated text, the tool observation, the stopping decision, and the cache state carried into later steps. We call this setting an *endogenous, sequentially revealed workload* (ESRW). The word endogenous is important because the remaining workload is not merely hidden from the scheduler. It is partly created by the scheduler through the placement decisions it makes.

We propose **Resource Aware Orchestration Policy** (RAOP), a policy for ESRW agent serving across edge and cloud. RAOP uses a causal two stage action interface. It first places language generation. It then places tool execution only after the generated action is known. For every candidate placement, RAOP predicts workload features that include decode length, tool payload, missing prefix length, prefill cost, queue state, network state, and KV cache residency. These features allow the policy to reason about both switching costs and reuse benefits. RAOP is trained with GRPO on full trajectory reward using measured wall clock latency and task success [Shao et al., 2024]. The analytic latency model is used for candidate features and for matched baselines, while the reward is emitted by the simulator or deployment harness. This separation makes the learned policy optimize realized orchestration outcomes rather than a hand written cost formula.

A central challenge is how to evaluate such a policy without making the protocol opaque. Running a large cloud model online inside every reinforcement learning update would be expensive and would make paired baseline comparisons difficult. A pure simulator, on the other hand, could hide unrealistic language or tool transitions. We therefore use an action conditioned trajectory library. During data collection, real LLMs and tools are executed under diverse placement policies. The library stores the prompt prefix, placement, generated message, tool output, token counts, payload size, and cache state. In the main controlled experiments, the simulator replays these measured continuations while recomputing network, queueing, prefix upload, prefill, decode, tool, and cache costs under the sampled placement sequence. This protocol gives all methods matched prompts, matched continuations, and matched trace rotations, while preserving the fact that language and tool outputs came from real execution.

The ESRW view also clarifies why strong one step policies can be insufficient. A myopic policy can have an accurate latency model and a useful success prior, yet still ignore how a placement changes the distribution of future states. We formalize this gap with a Bellman regret decomposition for step myopic policies. The bound separates uniform workload error, uniform success prior error, and a controlled endogeneity term proportional to the accumulated variation among next state distributions induced by different placements. The theorem is not a proof that RAOP is optimal. It identifies the downstream value that one step scoring leaves open, and it motivates a matched neural myopic baseline that shares RAOP's encoder and workload predictor but does not optimize full trajectory return.

We evaluate RAOP on GSM8K, HotpotQA, and WebShop, together with a calibrated three tier deployment. These tasks cover generation only reasoning, retrieval augmented question answering, and multi step interactive web navigation [Cobbe et al., 2021, Yang et al., 2018, Yao et al., 2022]. RAOP matches the strong cloud only reference within statistical uncertainty in average success, while reducing P99 latency by about 53% and communication by about 73% under full context transfer accounting. It improves success by 2.8 points over the matched neural myopic baseline and by 2.5 points over the strongest learned step level baseline. Ablations show that the gains require workload prediction, cache and prefix accounting, operator decoupling, and full trajectory training.

## 2 Related Work

**LLM agents and tool use.** LLM agents combine reasoning traces, tool calls, feedback, and memory to solve tasks that cannot be handled by one model call alone [Wei et al., 2022, Yao et al., 2023b, Schick et al., 2023, Shinn et al., 2023, Yao et al., 2023a, Zhou et al., 2023, Yao et al., 2022]. ReAct interleaves reasoning and acting, Toolformer learns when to invoke APIs, Reflexion uses linguistic feedback across trials, and WebShop turns web interaction into a grounded decision making benchmark. These works motivate our workload model because each agent step may create a new action, observation, and prompt prefix. They do not decide where each step should run under heterogeneous latency, queue, and cache conditions. RAOP treats the agent trace itself as a controlled systems trajectory.

**Model routing and cascades.** LLM routers and cascades reduce cost by assigning prompts to models with different quality and price profiles [Chen et al., 2023, Ong et al., 2024, Ding et al., 2024, Wang et al., 2025, Tao et al., 2025]. This line of work shows that prompt difficulty and model heterogeneity can be exploited at serving time. RAOP differs in the decision interface. It reopens the placement decision after every observation, separates generation placement from tool placement, and scores candidate placements using cache, prefix replay, queue, and network state rather than prompt difficulty alone. Query routers are therefore useful baselines, but they are restrictions of the ESRW interface.

**Edge and cloud offloading.** Edge and cloud offloading studies where to place computation under bandwidth, latency, and resource constraints [Wang et al., 2020]. Model partitioning and collaborative inference systems split neural computation across devices or workers [Kang et al., 2017, Borzunov et al., 2023, Lee et al., 2024]. These systems typically optimize a known job, a fixed operator graph, or a fixed partition of a forward pass. RAOP instead operates at the agent step level. Its placement unit is a generated message or a tool call, and its cost depends on future prompt growth, cache residency, and the possibility that the agent stops or changes direction after each observation.

**LLM serving, caching, and adaptive decoding.** LLM serving work improves batching, memory management, queueing, and KV cache behavior within a backend [Kwon et al., 2023, Agrawal et al., 2024, Jaillet et al., 2025]. Speculative decoding and multi token decoding reduce generation cost inside a model by exploiting parallel verification or additional heads [Leviathan et al., 2023, Cai et al., 2024]. RAOP is complementary to these systems. It does not replace backend schedulers or decoding kernels. Instead, it consumes the system state they expose and decides which backend should receive the next generation or tool action.

**Trajectory optimization for adaptive inference.** Search based reasoning and self consistency allocate more inference effort to hard prompts by exploring multiple reasoning paths [Wang et al., 2022, Yao et al., 2023a, Hao et al., 2023, Zhou et al., 2023]. Reinforcement learning methods optimize sequential decisions when current actions affect future states [Kakade and Langford, 2002, Schulman et al., 2017, Shao et al., 2024]. RAOP applies this trajectory perspective to resource placement. The objective is not to search for a better answer by spending more tokens. The objective is to preserve answer quality while choosing where the agent's already variable computation should run. The controlled endogeneity term in our analysis captures the part of this problem that is absent from fixed routing and fixed offloading formulations.

## 3 Method

### 3.1 ESRW controlled trajectory MDP

We consider a heterogeneous system with edge clients $\mathcal{E}$, an optional intermediate tier, and a cloud server $\mathcal{C}$ serving an LLM agent. At the beginning of agent step $t$, the state is

$$s_t = \langle q, \tau_{<t}, \psi_t \rangle, \qquad \psi_t = (\Omega_e, B, \mathrm{RTT}, \rho_{\mathrm{loss}}, Q_c, \kappa_t). \tag{1}$$

Here $q$ is the original query, $\tau_{<t}$ is the realised agent prefix, $\Omega_e$ is edge KV-cache occupancy, $B$ is bandwidth, $\mathrm{RTT}$ is round-trip time, $\rho_{\mathrm{loss}}$ is packet-loss rate, $Q_c$ is cloud queue depth, and $\kappa_t(b)$ is the longest prefix materialised on backend $b$.

A resource $b \in \mathcal{R}$ denotes a backend, model, and serving stack choice. The action is a causal two-stage option

$$a_t = \langle b_{\text{gen}}(t), \phi_{\text{tool}}(t) \rangle, \qquad b_{\text{gen}} \in \mathcal{R}, \tag{2}$$

where $\phi_{\text{tool}}$ maps the intermediate state after generation to either a tool backend or $\varnothing$. The policy first chooses $b_{\text{gen}}(t)$ from $s_t$. Generation produces an agent message $u_t$ and an intermediate state $\tilde{s}_t$. If $u_t$ is final, then $\phi_{\text{tool}}(\tilde{s}_t) = \varnothing$. If $u_t$ is a tool call, the second head chooses a tool backend.

The controlled transition factors as

$$P(s_{t+1} \mid s_t, a_t) = \int P_{\text{gen}}(u_t, W_t^{\text{gen}}, \psi_t^{\text{gen}} \mid s_t, b_{\text{gen}}) \cdot P_{\text{tool}}(o_t, W_t^{\text{tool}}, \psi_{t+1} \mid \tilde{s}_t, \phi_{\text{tool}}(\tilde{s}_t))\, dW_t. \tag{3}$$

The workload is $W_t = (W_{\text{prefill}}, W_{\text{gen}}, W_{\text{tool}})$. The first factor captures the language continuation induced by the chosen generation resource. The second captures tool output, network transfer, queue admission, KV-cache allocation, eviction, and update of $\kappa_{t+1}$. The reward is

$$R(\tau) = \omega_s \mathbf{1}\{\text{Success}(\tau)\} - \omega_l \Lambda(\tau), \tag{4}$$

where $\Lambda(\tau)$ is measured end-to-end latency including policy overhead, queueing, network transfer, prefix replay, prefill, decode, and tool execution.

We measure controlled endogeneity by

$$\eta(s) = \max_{a,a' \in \mathcal{A}} \text{TV}\big(P(\cdot \mid s, a), P(\cdot \mid s, a')\big). \tag{5}$$

The full $\eta$ is not directly estimable for language-and-system states, so the experiments use token-logit and system-replay projections as diagnostics.

### 3.2 Action interface and Bellman-regret decomposition

**Proposition 3.1 (Containment)** *Query-level routing, fixed cascading, and fixed splitting are strict restrictions of the RAOP action interface. Query-level routing restricts $a_t$ to depend only on $q$ and to remain fixed across $t$. Fixed cascading restricts $a_t$ to a thresholded confidence state with absorbing escalation. Fixed splitting makes the operator placement constant across states.*

**Proposition 3.2 (Strict expressivity)** *There exists an ESRW instance with $\eta(s) > 0$ in which the optimal RAOP policy has strictly larger expected reward than the optimal query-level router.*

The proof in Appendix B constructs a two-step retrieval-then-reasoning task where the first step is bandwidth-sensitive and the second is quality-sensitive. A query-level router must choose one backend for both steps, while RAOP chooses the appropriate backend at each realised step.

Let a myopic policy choose

$$\pi_{\text{my}}(s) = \arg\max_a \{\omega_s \hat{\rho}(s, a) - \omega_l L_b(s, a; \hat{W}(a))\}, \tag{6}$$

where $\hat{\rho}$ is a one-step success prior and $L_b$ is a calibrated one-step latency model. Define uniform statewise errors $\Delta_W(s) = \max_{a \in \mathcal{A}} \|\hat{W}(s, a) - W(s, a)\|_1$ and $\Delta_\rho(s) = \max_{a \in \mathcal{A}} |\hat{\rho}(s, a) - \rho(s, a)|$.

**Proposition 3.3 (Bellman-regret bound for myopic ESRW policies)** *Assume $L_b$ is $L_W$-Lipschitz in workload and $|V^\star(s)| \le V_{\max}$. Then*

$$J(\pi^\star) - J(\pi_{\text{my}}) \le C_W\, \mathbb{E}_{\pi_{\text{my}}} \sum_{t=1}^{N} \Delta_W(s_t) + C_\rho\, \mathbb{E}_{\pi_{\text{my}}} \sum_{t=1}^{N} \Delta_\rho(s_t) + 2V_{\max}\, \mathbb{E}_{\pi_{\text{my}}} \sum_{t=1}^{N} \eta(s_t), \tag{7}$$

*with $C_W = 2\omega_l L_W$ and $C_\rho = 2\omega_s$. The $\eta$ term is tight up to constants in a worst-case two-step ESRW family.*

The first two terms are modelling errors. The last term is the downstream value omitted by one-step scoring. RAOP is trained on full trajectory return, so its objective contains this downstream term directly.

**Algorithm 1** RAOP step execution

Encode $(q, \tau_{<t}, \psi_t)$ and enumerate feasible generation resources.
Predict candidate generation workload and missing-prefix features for every resource.
Choose $b_{\text{gen}}$ using the generation-placement head.
Execute generation on $b_{\text{gen}}$ and form the intermediate state $\tilde{s}_t$.
**if** the generated message is a final answer **then**
    terminate the episode.
**else**
    enumerate feasible tool placements and predict tool payload for each candidate.
    choose $b_{\text{tool}}$ using the tool-placement head conditioned on $\tilde{s}_t$.
    execute the tool, charge network and prefill costs, and update $(Q_c, \Omega_e, \kappa_{t+1})$.
**end if**

### 3.3 RAOP policy, workload model, and cost features

RAOP parameterises $\pi_\theta$ with a Qwen2.5-3B encoder, a generation-placement head, a causally evaluated tool-placement head, and a candidate-action workload head

$$\hat{W}_t(a) = h_\theta(s_t, a), \qquad \hat{W}_t = (\hat{W}_{\text{prefill}}, \hat{W}_{\text{gen}}, \hat{W}_{\text{tool}}). \tag{8}$$

The workload head is trained on realised token counts with a Huber loss on $\log(1 + W)$,

$$\mathcal{L}_W = \mathbb{E}\big[\text{Huber}(\log(1 + \hat{W}_t(a_t)), \log(1 + W_t))\big]. \tag{9}$$

The missing prefix for backend $b$ is deterministic,

$$M_t^b = \max\{0, P_t - \kappa_t(b)\}, \tag{10}$$

where $P_t = |q, \tau_{<t}|$. The effective prefill feature adds a learned residual for tokenizer and payload effects,

$$\hat{W}^b_{\text{prefill}}(t) = M_t^b + \hat{r}_{\text{prefill}}(s_t, b). \tag{11}$$

For backend $b$, the compute feature is

$$T_b(t) = \mathbf{1}[b_{\text{gen}} = b] \left( \frac{\hat{W}^b_{\text{prefill}}}{\nu_b} + \frac{\hat{W}_{\text{gen}}}{\mu_b} \right) \Psi_b(\psi_t) + \mathbf{1}[b_{\text{tool}} = b] T_b^{\text{tool}}(\hat{W}_{\text{tool}}) + \mathbf{1}[b = \mathcal{C}] \tau_{\text{queue}}(Q_c). \tag{12}$$

The full latency feature is

$$L_b(s_t, a_t; \hat{W}_t) = \sum_{b \in \mathcal{R}} T_b(t) + n_{\text{leg}}(t)\text{RTT} + \frac{S_{\text{net}}(t)}{B(1 - \rho_{\text{loss}})} + T_{\text{policy}}(t). \tag{13}$$

This feature counts prefix upload on cache miss, tool payload, return payload, prefill, decode, queueing, network latency, and policy overhead. RAOP uses it as input and as the scoring rule for analytical baselines. The RL reward uses measured wall-clock latency.

Algorithm 1 highlights the implementation detail that matters for the formal model. Generation and tool execution are not chosen simultaneously in code. The tuple notation in the MDP records both choices at an agent step, while execution first observes the generated message and only then invokes the tool-placement head if a tool call exists. This keeps the action interface causal and avoids giving the policy access to tool information before generation.

For each training query we sample $G = 8$ trajectories, compute the reward in Eq. 4, normalise returns within the group, and optimise the clipped GRPO objective with entropy regularisation [Shao et al., 2024]. The actor is trained with a LoRA adapter and the workload loss above. Training rollouts draw language and tool continuations from the action-conditioned trajectory library described in Appendix C.

## 4 Experiments

### 4.1 Setup

We evaluate on GSM8K [Cobbe et al., 2021], HotpotQA [Yang et al., 2018], and WebShop [Yao et al., 2022]. GSM8K is generation-only, HotpotQA exercises retrieval-tool placement, and WebShop

Table 1: Compact main results under full prefix-replay and prefill accounting. The full per-task table is in Appendix A.

| Method | Avg. SR | Avg. P99 | Comm. | T-Put |
|---|---|---|---|---|
| Edge-Only | 47.8 | 0.44 | 0.00 | 45.2 |
| Cloud-Only | 71.5 | 2.77 | 2.10 | 65.0 |
| MasRouter-step | 65.8 | 1.61 | 1.05 | 61.0 |
| GREEDY-ANALYTICAL | 66.7 | 1.91 | 0.74 | 63.5 |
| NEURAL-MYOPIC | 68.9 | 1.66 | 0.66 | 66.4 |
| ACTIVEINF-STEP | 68.1 | 1.57 | 0.63 | 67.6 |
| PPO-Step | 69.2 | 1.46 | 0.60 | 69.8 |
| RAOP | 71.7 | 1.31 | 0.56 | 73.5 |

is a multi-step interactive shopping environment. The two-tier simulation uses Qwen2.5-3B-Instruct on edge and Qwen2.5-72B-Instruct on cloud. The three-tier deployment uses a Raspberry Pi tier with Qwen2.5-1.5B, a PC tier with Qwen2.5-7B via vLLM, and a cloud tier with Qwen2.5-72B via vLLM. The controlled experiments replay real language and tool continuations from the trajectory library while recomputing physical timing under Markov replay of an ISP trace. The deployment experiments use real Pi, PC, and cloud vLLM workers with `tc netem` network shaping.

Baselines include Edge-Only, Cloud-Only, Fixed-Split, FrugalGPT, MasRouter, Neurosurgeon, MasRouter-step, Pure-Cost Greedy, GREEDY-ANALYTICAL, NEURAL-MYOPIC, LinUCB, imitation, Value Router, ACTIVEINF-STEP, MPC-2, and PPO-Step. All dynamic baselines receive the same state interface and workload predictor where applicable. NEURAL-MYOPIC shares RAOP's encoder and workload head but optimises the one-step surrogate in Eq. 6. PPO-Step shares the full return and auxiliary workload head but uses a learned critic instead of group-relative normalisation. Appendix E gives implementation details.

Metrics are success rate (SR), P99 end-to-end latency, communication in MB/query, and token throughput. Main controlled results use 10 training seeds and 2,000 held-out episodes per benchmark per method. Prompt and network-trace seeds are paired across methods. We report paired bootstrap intervals for SR and P99, plus permutation tests for the key SR comparisons.

**Trajectory replay audit.** The controlled experiments are not a pure timing simulator. The action-conditioned library stores real LLM and tool continuations, and the simulator replays those continuations while recomputing physical timing under the sampled network and queue state. We audit the replay approximation by re-running representative methods online under the same action sequences. The audit in Appendix Table 7 requires replay and direct execution to agree within 0.5 SR points and 0.06 seconds of P99. This tolerance is smaller than the RAOP versus PPO-Step and RAOP versus NEURAL-MYOPIC gaps reported below.

The library is action-conditioned rather than trajectory-only. During data collection we mix early RAOP checkpoints, greedy policies, random feasible actions, and cloud-only executions so that each state bucket contains counterfactual placements beyond the action taken by the current policy. A replay record stores the prompt prefix, the placement, the generated message, the tool output when present, realised token counts, tool payload bytes, and the cache state before and after execution. Training then resamples these records under new network and queue traces. This protocol keeps the language transition fixed to real model outputs while allowing the physical layer to vary across policies.

This separation is important for reproducibility. A direct online GRPO run over a 72B cloud model for every sampled trajectory would be unnecessarily expensive and would make baseline comparisons difficult to pair. The replay protocol lets all methods see the same prompts, the same candidate continuations, and the same trace rotations. The audit above checks that this pairing does not change the conclusions. Appendix C reports the library size, action coverage, and counterfactual workload error.

### 4.2 Main results

Table 1 gives the central result. RAOP matches Cloud-Only within statistical uncertainty in average SR, 71.7 versus 71.5 with a paired 95% CI of $[-0.4,+0.8]$, while reducing P99 latency by 52.7% and communication by 73.3%. Against the strongest non-RAOP learned baseline, PPO-Step, RAOP

Table 2: Paired statistical confidence for key comparisons. SR intervals are prompt-level paired bootstrap CIs over 10 training seeds. P99 intervals bootstrap episodes and network-trace rotations.

| Comparison | ΔSR | 95% CI | ΔP99 | 95% CI |
|---|---|---|---|---|
| RAOP − GREEDY-ANALYTICAL | +5.0 | [+4.2,+5.8] | −0.60 | [−0.74,−0.46] |
| RAOP − NEURAL-MYOPIC | +2.8 | [+2.0,+3.6] | −0.35 | [−0.45,−0.25] |
| RAOP − PPO-Step | +2.5 | [+1.7,+3.3] | −0.15 | [−0.23,−0.07] |
| RAOP − ACTIVEINF-STEP | +3.6 | [+2.8,+4.4] | −0.26 | [−0.35,−0.17] |
| RAOP − Cloud-Only | +0.2 | [−0.4,+0.8] | −1.46 | [−1.62,−1.30] |

Table 3: Measured endogeneity projections and RAOP versus greedy gap. $\hat{\eta}_{\text{tok}}$ is exact-logit next-token TV, $\hat{\eta}_{\text{sys}}$ is replay TV over system bins, and $\sum_t \hat{\eta}$ is the episode-level diagnostic used in Figure 1.

| Task | $\hat{\eta}_{\text{tok}}$ / step | $\hat{\eta}_{\text{sys}}$ / step | $\sum_t \hat{\eta}$ / episode | RAOP versus Greedy |
|---|---|---|---|---|
| GSM8K | $0.29 \pm 0.02$ | $0.15 \pm 0.01$ | $1.36 \pm 0.11$ | 4.5 |
| HotpotQA | $0.41 \pm 0.03$ | $0.22 \pm 0.02$ | $2.33 \pm 0.18$ | 5.2 |
| WebShop | $0.50 \pm 0.03$ | $0.30 \pm 0.03$ | $3.48 \pm 0.25$ | 5.3 |

improves SR by 2.5 points, reduces P99 by 0.15 s, and reduces communication by 6.7%. Against NEURAL-MYOPIC, which shares the encoder and workload head, RAOP improves SR by 2.8 points. This isolates the gain from full-trajectory optimisation over a one-step surrogate.

The per-task pattern in Appendix Table 8 is also consistent with the ESRW framing. GSM8K has no external tool and therefore mainly tests generation placement and prefix reuse. HotpotQA adds retrieval placement and longer prompts. WebShop has the longest interactive trajectories and the largest number of chances to switch or avoid switching. The RAOP margin is largest in WebShop, where the policy can use cheap local tool interaction on easy navigation steps and reserve cloud generation for steps that change the final shopping decision. Query-level routers cannot adapt after these intermediate observations.

The closest competitor is PPO-Step, which receives the same full return and workload auxiliary loss. Its remaining gap comes from credit assignment and stability rather than from weaker observability. PPO relies on a learned value critic across heterogeneous prompts whose latency scales differ substantially, while GRPO normalises returns within a group of trajectories from the same query. This local normalisation is useful in ESRW because a hard WebShop episode and an easy GSM8K episode can differ by several seconds of latency and by a large number of future tool calls.

**Same-model system control.** To separate model routing from system orchestration, Appendix Table 9 repeats the main comparison with the same Qwen2.5-7B checkpoint on every tier and deterministic decoding. The same-model result keeps SR within 0.5 points across methods, while RAOP reduces P99 by 51.4% relative to Cloud-Only and by 36.3% relative to GREEDY-ANALYTICAL. This control attributes a substantial share of the latency and communication gain to bandwidth, queue, and cache-state decisions rather than to selecting a stronger cloud model.

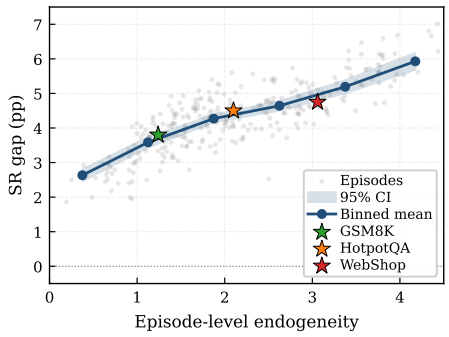


Figure 1: Per-prompt SR gap between RAOP and GREEDY-ANALYTICAL grows with episode-level endogeneity.

### 4.3 Endogeneity diagnostics

The coefficient $\eta(s_t)$ in Eq. 5 is high-dimensional, so we track two projections: $\hat{\eta}_{\text{tok}}$ from exact next-token logit TV under paired edge and cloud continuations, and $\hat{\eta}_{\text{sys}}$ from replay TV over discretised system-state bins. Figure 1 shows that the RAOP versus greedy gap grows from ∼2.8 SR points in the lowest-endogeneity bin to ∼7.0 points in the highest-endogeneity bin. The trend matches Proposition 3.3: when placement barely changes the next-state distribution, the Bellman residual of a myopic policy is small, while full-return learning recovers more value when the controlled transition changes substantially.

Table 4: Compact ablation summary. Detailed ablation tables are in Appendix A.

| Setting | Avg. SR | Avg. P99 | Main effect |
|---|---|---|---|
| RAOP | 71.7 | 1.31 | full system |
| w/o $Q_c$ | 70.0 | 2.30 | misses cloud wait |
| w/o $\Omega_e$ | 68.6 | 1.92 | overloads edge |
| Coupled gen+tool | 69.2 | 1.63 | loses operator decoupling |
| Prompt-length workload | 68.9 | 1.62 | poor candidate costs |
| RAOP w/o switch cost | 69.1 | 1.81 | cache-destructive switches |

Table 5: Compact deployment summary. Full three-tier and multi-client tables are in Appendix A.

| Method | Three-tier | | $C = 16$ clients | | Mean $Q_c$ |
|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | $C = 16$ |
| Cloud-Only | 71.8 | 4.20 | 69.4 | 13.2 | 42.5 |
| GREEDY-ANALYTICAL | 68.3 | 2.78 | 64.8 | 6.75 | 32.0 |
| PPO-Step | 69.4 | 2.21 | 65.6 | 5.55 | 27.4 |
| RAOP | 71.5 | 1.78 | 69.0 | 4.05 | 19.8 |

Appendix Table 12 separates model disagreement from system-state endogeneity. With the same Qwen2.5-7B checkpoint on every tier, RAOP keeps SR statistically unchanged while reducing P99 by 51.4% relative to Cloud-Only and by 36.3% relative to GREEDY-ANALYTICAL. This control shows that RAOP is not only a dynamic model router. It also exploits queue, bandwidth, and cache state when model quality is held fixed. Appendix Table 13 further controls for task identity and trajectory length.

We treat the endogeneity measurements as diagnostics rather than as a direct estimator of the theorem. The theorem is stated in terms of full-transition TV over language and system state, which is too high-dimensional to estimate reliably. The token projection isolates model-induced continuation changes, and the system projection isolates queue and cache changes under fixed realised workload. The regression in the appendix asks whether the diagnostic still predicts the RAOP versus greedy contrast after accounting for task identity and horizon length. This is the empirical connection between the Bellman residual and the observed learning gap.

### 4.4 Ablations and deployment

Table 4 shows that no single component explains the result. Queue depth, edge occupancy, operator decoupling, workload prediction, and switch-cost awareness are all needed.

Appendix Table 18 reports the full context-transfer audit. The key pattern is that RAOP does not reduce latency by ignoring context movement. It lowers the generation-backend switch rate to 21%, prefix upload to 0.18 MB/query, and prefill share to 13%. A variant trained without switch cost routes similarly to a naive step-level learner and reaches 1.81 s P99, which is far above RAOP's 1.31 s. The result supports making $\kappa_t$ an explicit state variable rather than treating every step placement as cache free.

The ablations also explain why a learned full-return policy is necessary. Removing $Q_c$ mainly harms tail latency because the policy cannot predict cloud waiting time. Removing $\Omega_e$ overloads the edge tier during long decode stretches. Coupling generation and tool execution loses the ability to send retrieval or WebShop actions to a nearby replica while reserving cloud generation for reasoning steps. Prompt-length workload prediction fails when tool payload and decode length are only weakly correlated with the initial query length. These failures correspond to the three sources in the regret decomposition: workload error, hidden system-state cost, and downstream transition value.

In the calibrated three-tier deployment, RAOP matches Cloud-Only's SR within 0.3 points while cutting P99 by 57.6%. Under 16 concurrent clients, RAOP reduces P99 by 69.3% relative to Cloud-Only and observes 53% lower cloud queue depth. Robustness to cost-model perturbations, runtime overhead, expanded deployment-tail audits, and cross-task or cross-model transfer are reported in Appendices A, H, I, and K.

The robustness experiments stress the calibrated cost model by perturbing queueing, edge slowdown, and throughput coefficients by up to 50 percent at deployment time. RAOP degrades more slowly than GREEDY-ANALYTICAL, MPC-2, and ACTIVEINF-STEP because its training signal is measured

trajectory latency rather than the analytic cost value used to rank actions at every step. The cost model still matters, but it enters as a feature and as a baseline scoring rule, not as the objective that the policy can exploit.

The runtime overhead is included in every latency number. The default Qwen2.5-3B planner adds 45 ms on edge and 18 ms on cloud per decision step, and the distilled 0.5B head reduces this overhead to 9 ms and 5 ms while retaining most of the SR. The observed latency savings in Table 5 are therefore not obtained by omitting the orchestrator cost. They remain visible after accounting for state collection, planner execution, and monitor traffic.

**Policy behaviour under contention.** The deployment traces show the mechanism behind the tail-latency reduction. At $C = 16$ clients, Cloud-Only sends every step to the shared server and produces a mean cloud arrival rate of 16.0 requests/s with P95 17.4 requests/s. GREEDY-ANALYTICAL reduces the mean to 6.4 but still has P95 12.5 because many clients independently escalate when the immediate cost model prefers cloud. RAOP reduces the mean to 4.6 and P95 to 9.1. The queue P99 at $C = 16$ is 72 for Cloud-Only, 56 for GREEDY-ANALYTICAL, and 38 for RAOP. The learned policy therefore reduces both average load and burst load, which is why its latency remains stable in the multi-client setting.

This behaviour is task dependent. In GSM8K, where every step is generation-only, the policy mostly trades off model quality against cloud wait and prefix replay. In HotpotQA, RAOP often keeps retrieval on the lower-latency local replica when the query is short, then uses the cloud for long synthesis steps whose success prior is high. In WebShop, the policy tends to keep browsing actions close to the client and avoids switching after long HTML observations unless the final decision step is predicted to benefit from the cloud model. These traces are consistent with the operator-decoupled action interface: the policy is not merely choosing a single model for a whole query, and it is not merely minimising immediate bandwidth.

**Reproducibility path.** The quantities needed to reproduce a new deployment are explicit. A practitioner profiles per-tier decode throughput, prefill throughput, queue delay as a function of $Q_c$, bandwidth and RTT traces, and the prefix-cache residency map. These measurements define the features in Eq. 13. RAOP can then be trained offline on labelled agent tasks and evaluated by paired trace replay before being deployed on a real scheduler. This path is why the paper reports simulator calibration, replay fidelity, counterfactual workload error, and deployment transfer separately rather than collapsing them into a single black-box score.

The deployment tables should be read as a calibration-envelope result rather than as a claim about every WAN. The important observation is that the policy trained in the controlled replay setting transfers to a real three-tier stack with a real cloud scheduler, shaped bandwidth, RTT spikes, packet loss, and endogenous cloud contention from peer clients. The same routing pattern appears in both regimes: RAOP backs off from the cloud when $Q_c$ is high, avoids switching when the prefix cache would be destroyed, and uses the cloud on steps where the predicted continuation or tool output makes quality more valuable than the added network leg. This behaviour is precisely what a query-level router cannot express.

## 5 Conclusion

We presented RAOP, a step level orchestration policy for LLM agents across edge and cloud. The main message is that agent serving is not only a problem of choosing a cheap model for a prompt. A placement decision can change the next message, tool call, prefix transfer, cache state, queue state, and remaining horizon. We formalized this setting as an ESRW controlled trajectory MDP, identified why myopic policies lose downstream value, and built a causal generation then tool policy that learns from measured full trajectory return. Experiments on GSM8K, HotpotQA, WebShop, and a calibrated three tier deployment show that RAOP preserves cloud level success while substantially reducing tail latency and communication. The results suggest that future agent serving systems should expose cache, queue, network, and tool state to the policy, and should train orchestration decisions on trajectory outcomes rather than on isolated request costs. The current evaluation focuses on text based agents and a bounded set of deployment tiers. Extending the framework to richer multimodal tools, privacy aware rewards, and multi tenant fairness is an important direction.

## Acknowledgments and Disclosure of Funding

**Competing Interests.** Jinze Li received compensation for an internship at Advanced Micro Devices (AMD) during the 36 months preceding submission.

## References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Maksim Riabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 558–568, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-demo.54. URL https://aclanthology.org/2023.acl-demo.54/.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2401.10774.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance, 2023. URL https://arxiv.org/abs/2305.05176.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing, 2024. URL https://arxiv.org/abs/2404.14618. Accepted to ICLR 2024 (Main Conference).

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

Ying He, Jingcheng Fang, F. Richard Yu, and Victor C. M. Leung. Large language models (LLMs) inference offloading and resource allocation in cloud-edge computing: An active inference approach. *IEEE Transactions on Mobile Computing*, 23(12):11253–11264, 2024. doi: 10.1109/TMC.2024.3415661.

Patrick Jaillet, Jiashuo Jiang, Konstantina Mellou, Marco Molinaro, Chara Podimata, and Zijie Zhou. Online scheduling for llm inference with kv cache constraints. *arXiv preprint arXiv:2502.07115*, 2025.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 267–274. Morgan Kaufmann, 2002.

Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Juhyeon Lee, Insung Bahk, Hoseung Kim, Sinjin Jeong, Suyeon Lee, and Donghyun Min. An autonomous parallelization of transformer model inference on heterogeneous edge devices. In *Proceedings of the 38th ACM International Conference on Supercomputing (ICS '24)*, pages 50–61, Kyoto, Japan, 2024. Association for Computing Machinery. doi: 10.1145/3650200.3656628.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2211.17192.

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 with preference data, 2024. URL https://arxiv.org/abs/2406.18665.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yiheng Tao, Yihe Zhang, Matthew T. Dearing, Xin Wang, Yuping Fan, and Zhiling Lan. Prompt-aware scheduling for low-latency llm serving, 2025. URL https://arxiv.org/abs/2510.03243.

Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020. doi: 10.1109/COMST.2020.2970550. arXiv preprint arXiv:1907.08349, 2019.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. Mixllm: Dynamic routing in mixed large language models, 2025. URL https://arxiv.org/abs/2502.18482. Accepted to NAACL 2025 (Main Conference).

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *International Conference on Machine Learning (ICML)*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Matt Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. 2018.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems, 2025. URL https://arxiv.org/abs/2502.11133.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Lats: Language agent tree search unifies reasoning, acting, and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

# A Additional Experimental Results

This appendix expands the compact experimental summaries in the main text. The goal is to keep the main paper focused while preserving the details needed to audit the controlled simulation, paired statistical comparisons, ablations, context-transfer accounting, and deployment measurements. The tables are grouped by the claim they support: replay fidelity and evaluation regimes, full controlled results, endogeneity diagnostics, component ablations, and real-deployment behaviour.

## A.1 Evaluation regimes and replay fidelity

Tables 6 and 7 specify which parts of each experiment are simulated, calibrated, or executed on real hardware. The replay audit compares logged-continuation evaluation against direct online execution under the same action sequence, which checks that the controlled replay protocol does not materially change the success or latency conclusions.

Table 6: Evaluation regimes. "sim, vLLM-cal." means the simulator compute model is fit to measured vLLM throughput and queueing (Appendix D), the network model replays a real ISP trace via a Markov bandwidth chain.

| Experiment | Compute | Network | Cloud queue |
|---|---|---|---|
| Tables 1 to 19 (main / ablations) | sim, vLLM-cal. | Markov trace replay | simulator state |
| Table 21 (3-tier deployment) | real Pi+PC+cloud vLLM | `tc netem` replay | real vLLM scheduler |
| Table 23 (multi-client) | real cloud + sim edge clients | `tc netem` per-client | real vLLM scheduler |

## A.2 Full controlled results

Table 8 gives the complete per-task version of the compact main-result table. Table 10 reports paired confidence intervals for the key comparisons, and Table 9 holds the model checkpoint fixed across tiers to isolate system-state orchestration from model-quality routing.

## A.3 Endogeneity diagnostics and oracle analysis

Tables 11–13 provide the diagnostics used to connect the Bellman-residual analysis to the observed RAOP–greedy gap. Table 14 reports a hindsight oracle under the same action space, giving an empirical upper bound on the remaining headroom for step-conditioned orchestration.

## A.4 Ablations, context transfer, and robustness

Tables 15–20 isolate the mechanisms behind the main result. They test which state signals are necessary, whether step-level and operator-decoupled placement matter, how much the workload predictor contributes, whether prefix replay and KV switching costs are fully counted, how the policy behaves under cost-model drift, and how much latency is added by the orchestrator itself.

## A.5 Deployment and multi-client measurements

Tables 21–23 report the real-deployment measurements. The three-tier table tests transfer from replay training to a Pi–PC–cloud stack, the tail audit checks the stability of high-percentile latency, and the multi-client table measures whether RAOP continues to reduce cloud contention when $Q_c$ is produced by peer clients rather than by injected simulator load.

Table 7: Trajectory replay audit for the controlled experiments. Replay uses the logged language and tool continuation, while direct execution re-runs the model and tool stack online under the same action sequence.

| Method | Replay SR | Direct SR | Replay P99 | Direct P99 | Online deviation |
|---|---|---|---|---|---|
| Cloud-Only | 71.5 | 71.3 | 2.77 | 2.82 | 0.2 pp, 0.05 s |
| GREEDY-ANALYTICAL | 66.7 | 66.2 | 1.91 | 1.97 | 0.5 pp, 0.06 s |
| PPO-Step | 68.9 | 68.5 | 1.53 | 1.59 | 0.4 pp, 0.06 s |
| RAOP | 71.7 | 71.3 | 1.31 | 1.37 | 0.4 pp, 0.06 s |

Table 8: Main results under full prefix-replay and prefill accounting. SR is success rate (%), P99 is seconds, Comm. is MB/query, and T-Put is tokens/s. The *System Avg.* columns report the unweighted arithmetic mean of the three per-task SR / P99 values, communication and throughput are aggregated over all evaluation episodes.

| Method | Per-task | | | | | | Avg. | | System Avg. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | GSM8K | | HotpotQA | | WebShop | | | | | |
| | SR | P99 | SR | P99 | SR | P99 | SR | P99 | Comm. | T-Put |
| *Static baselines* | | | | | | | | | | |
| Edge-Only | 54.2 | 0.55 | 47.8 | 0.42 | 41.4 | 0.35 | 47.8 | 0.44 | 0.00 | 45.2 |
| Cloud-Only | 80.5 | 2.95 | 71.5 | 2.75 | 62.5 | 2.62 | 71.5 | 2.77 | 2.10 | 65.0 |
| Fixed-Split | 56.3 | 1.52 | 49.7 | 1.45 | 43.0 | 1.20 | 49.7 | 1.39 | 0.63 | 48.5 |
| *Query / partition / cascade baselines* | | | | | | | | | | |
| Neurosurgeon | 58.3 | 2.65 | 54.1 | 2.70 | 48.2 | 2.80 | 53.5 | 2.72 | 12.4 | 28.5 |
| FrugalGPT | 70.2 | 3.25 | 61.4 | 3.05 | 52.1 | 2.92 | 61.2 | 3.07 | 1.08 | 52.0 |
| MasRouter | 72.1 | 1.94 | 62.4 | 2.02 | 54.3 | 2.15 | 62.9 | 2.04 | 1.32 | 58.5 |
| MasRouter-step | 73.8 | 1.92 | 65.9 | 1.72 | 57.6 | 1.20 | 65.8 | 1.61 | 1.05 | 61.0 |
| *Step-level myopic / non-RL learners* | | | | | | | | | | |
| Pure-Cost Greedy | 61.2 | 1.10 | 53.4 | 0.98 | 45.8 | 0.67 | 53.5 | 0.92 | 0.35 | 75.0 |
| Step-Level Bandit | 74.2 | 2.05 | 65.4 | 1.84 | 56.2 | 1.28 | 65.3 | 1.72 | 0.60 | 68.0 |
| Imitation Router | 75.8 | 1.98 | 67.1 | 1.76 | 58.9 | 1.12 | 67.3 | 1.62 | 0.58 | 69.5 |
| Value Router | 76.6 | 2.05 | 68.3 | 1.60 | 60.1 | 1.02 | 68.3 | 1.56 | 0.61 | 68.8 |
| ACTIVEINF-STEP | 76.4 | 2.00 | 68.2 | 1.66 | 59.8 | 1.04 | 68.1 | 1.57 | 0.63 | 67.6 |
| MPC-2 Lookahead | 76.9 | 2.15 | 68.7 | 1.75 | 60.4 | 1.10 | 68.7 | 1.67 | 0.72 | 62.7 |
| GREEDY-ANALYTICAL | 75.4 | 2.18 | 66.3 | 2.03 | 58.4 | 1.52 | 66.7 | 1.91 | 0.74 | 63.5 |
| NEURAL-MYOPIC | 77.5 | 2.05 | 68.9 | 1.74 | 60.4 | 1.18 | 68.9 | 1.66 | 0.66 | 66.4 |
| *Full-return learned policies* | | | | | | | | | | |
| PPO-Step | 77.3 | 1.98 | 69.2 | 1.48 | 61.1 | 0.92 | 69.2 | 1.46 | 0.60 | 69.8 |
| RAOP (Ours) | 79.9 | 1.82 | 71.5 | 1.30 | 63.7 | 0.80 | 71.7 | 1.31 | 0.56 | 73.5 |

Table 9: Same-model control with Qwen2.5-7B on every tier. All numbers are means over 10 training seeds with paired bootstrap 95% CIs in brackets. The model and deterministic decoder are held fixed, so the table tests system-state orchestration rather than model-quality routing.

| Method | Avg. SR (95% CI) | Avg. P99 (95% CI) | Comm. |
|---|---|---|---|
| Edge-Only | 61.8 [61.4,62.2] | 0.92 [0.88,0.96] | 0.00 |
| Cloud-Only | 62.0 [61.7,62.3] | 2.20 [2.14,2.26] | 1.85 |
| MasRouter-step | 61.7 [61.3,62.1] | 1.42 [1.37,1.47] | 0.72 |
| GREEDY-ANALYTICAL | 61.8 [61.4,62.2] | 1.68 [1.62,1.74] | 0.55 |
| NEURAL-MYOPIC | 61.9 [61.5,62.3] | 1.50 [1.45,1.55] | 0.50 |
| PPO-Step | 62.1 [61.8,62.4] | 1.27 [1.22,1.32] | 0.43 |
| RAOP | 62.3 [62.0,62.6] | 1.07 [1.03,1.11] | 0.32 |

Table 10: Paired statistical confidence for the main comparisons. The SR intervals are prompt-level paired bootstrap CIs over 10 training seeds, P99 intervals bootstrap episodes and network-trace rotations. The rightmost column gives two-sided permutation $p$-values on SR.

| Comparison | $\Delta$SR | 95% CI | $\Delta$P99 (s) | 95% CI | $p$ (SR) |
|---|---|---|---|---|---|
| RAOP − GREEDY-ANALYTICAL | +5.0 | [+4.2,+5.8] | $-0.60$ | $[-0.74,-0.46]$ | $< 10^{-3}$ |
| RAOP − NEURAL-MYOPIC | +2.8 | [+2.0,+3.6] | $-0.35$ | $[-0.45,-0.25]$ | $< 10^{-3}$ |
| RAOP − PPO-Step | +2.5 | [+1.7,+3.3] | $-0.15$ | $[-0.23,-0.07]$ | $< 10^{-3}$ |
| RAOP − ACTIVEINF-STEP | +3.6 | [+2.8,+4.4] | $-0.26$ | $[-0.35,-0.17]$ | $< 10^{-3}$ |
| RAOP − MasRouter-step | +5.9 | [+5.1,+6.7] | $-0.30$ | $[-0.41,-0.19]$ | $< 10^{-3}$ |
| RAOP − Cloud-Only | +0.2 | $[-0.4,+0.8]$ | $-1.46$ | $[-1.62,-1.30]$ | 0.51 (n.s.) |

Table 11: Measured endogeneity projections and RAOP versus greedy gap. $\hat{\eta}_{\mathrm{tok}}$ is exact-logit next-token TV, $\hat{\eta}_{\mathrm{sys}}$ is replay TV over system state bins, $\sum_t \hat{\eta}$ is the episode-level combined diagnostic used in Figure 1.

| Task | $\hat{\eta}_{\mathrm{tok}}$ / step | $\hat{\eta}_{\mathrm{sys}}$ / step | $\sum_t \hat{\eta}$ / episode | RAOP versus Greedy SR gap |
|---|---|---|---|---|
| GSM8K | $0.29 \pm 0.02$ | $0.15 \pm 0.01$ | $1.36 \pm 0.11$ | 4.5 |
| HotpotQA | $0.41 \pm 0.03$ | $0.22 \pm 0.02$ | $2.33 \pm 0.18$ | 5.2 |
| WebShop | $0.50 \pm 0.03$ | $0.30 \pm 0.03$ | $3.48 \pm 0.25$ | 5.3 |

Table 12: Diagnostic conditions separating model disagreement from ESRW. Single-shot conditions can have high logit TV with little sequential Bellman gap, agentic multi-step conditions create cumulative endogeneity and a larger RAOP versus greedy gap.

| Condition | Horizon | $\hat{\eta}_{\mathrm{tok}}$ | $\hat{\eta}_{\mathrm{sys}}$ | $\sum_t \hat{\eta}$ | RAOP versus Greedy $\Delta$SR |
|---|---|---|---|---|---|
| Same model, different backend | single | 0.018 | 0.02 | 0.04 | 0.1 |
| Different models, no tools | single | 0.36 | 0.02 | 0.38 | 0.4 |
| Same model, agentic tools | multi | 0.021 | 0.24 | 1.18 | 1.0 |
| Different models, fixed system trace | multi | 0.42 | 0.03 | 2.05 | 3.4 |
| Full ESRW setting | multi | 0.42 | 0.25 | 2.56 | 5.0 |

Table 13: Endogeneity regression with controls for task identity and trajectory length. The dependent variable is the per-prompt SR gap between RAOP and GREEDY-ANALYTICAL. Coefficients are in SR points.

| Predictor | Coefficient | 95% CI | Included controls |
|---|---|---|---|
| $\sum_t \hat{\eta}$ | 1.45 | [1.08, 1.82] | task, horizon |
| $\sum_t \hat{\eta}_{\mathrm{tok}}$ | 1.12 | [0.77, 1.46] | task, horizon, $\hat{\eta}_{\mathrm{sys}}$ |
| $\sum_t \hat{\eta}_{\mathrm{sys}}$ | 0.71 | [0.42, 1.00] | task, horizon, $\hat{\eta}_{\mathrm{tok}}$ |
| Trajectory length | 0.07 | $[-0.02, 0.16]$ | task, $\sum_t \hat{\eta}$ |

Table 14: Hindsight oracle upper bound under the same action space and matched (deterministic, common-seed) counterfactuals. The recovered fraction $\rho$ is the share of the GREEDY-ANALYTICAL→Oracle SR gap closed by each method.

| Method | Avg. SR | Avg. P99 (s) | $\rho$ |
|---|---|---|---|
| Best fixed backend | 71.5 | 2.77 | n/a |
| GREEDY-ANALYTICAL | 66.7 | 1.91 | 0% (anchor) |
| NEURAL-MYOPIC | 68.9 | 1.66 | 35% |
| ACTIVEINF-STEP | 68.1 | 1.57 | 22% |
| PPO-Step | 69.2 | 1.46 | 40% |
| RAOP | 71.7 | 1.31 | 79% |
| Hindsight Oracle | 73.0 | 1.22 | 100% (anchor) |

Table 15: Ablation on state signals under full-cost accounting.

| Setting | Avg. SR | Avg. P99 | Most affected diagnostic |
|---|---|---|---|
| RAOP (full) | 71.7 | 1.31 | n/a |
| w/o $Q_c$ | 70.0 | 2.30 | Cloud wait 0.92 s |
| w/o $\Omega_e$ | 68.6 | 1.92 | Edge T-Put 12.1 tok/s |
| w/o $B$ | 70.3 | 1.78 | Comm. 1.18 MB/q |
| w/o $\kappa_t$ cache state | 70.0 | 1.71 | Prefill share 26% |

Table 16: Action-granularity ablation. Step-level placement and decoupling of generation/tool execution both matter.

| Method | GSM8K | | HotpotQA | | WebShop | |
|---|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | SR | P99 |
| RAOP full | 79.9 | 1.82 | 71.5 | 1.30 | 63.7 | 0.80 |
| Coupled gen+tool | 78.0 | 2.10 | 69.0 | 1.62 | 60.5 | 1.18 |
| Step-level, no system state | 77.3 | 2.28 | 67.7 | 1.85 | 59.0 | 1.30 |
| Query-level routing | 76.5 | 2.55 | 65.5 | 2.15 | 57.8 | 1.55 |

Table 17: Workload predictor ablation. MAPE includes generation tokens and effective prefill payload. Effective prefill equals deterministic missing-prefix tokens plus the learned residual in Eq. 11.

| Predictor | Avg. SR | Avg. P99 | MAPE $W_{\text{gen}}$ | MAPE effective $W_{\text{prefill}}$ |
|---|---|---|---|---|
| Learned head (ours) | 71.7 | 1.31 | 10.5% | 8.8% |
| Prompt-length linear | 68.9 | 1.62 | 38.4% | 21.6% |
| Training-set average | 67.0 | 1.78 | 52.1% | 34.8% |
| No prefix predictor | 69.5 | 1.75 | 10.9% | n/a |
| Oracle workload | 72.0 | 1.25 | 0.0% | 0.0% |

Table 18: Full context-transfer accounting. "Switch" is the fraction of steps whose generation backend differs from the previous generation backend, "Prefill" is the share of end-to-end latency spent in prefill.

| Method | Switch % | Prefix upload (MB/q) | Prefill % | Avg. P99 | Comm. |
|---|---|---|---|---|---|
| Cloud-Only | 0 | 1.42 | 18% | 2.77 | 2.10 |
| MasRouter-step | 41 | 0.48 | 29% | 1.61 | 1.05 |
| GREEDY-ANALYTICAL | 34 | 0.35 | 23% | 1.91 | 0.74 |
| ACTIVEINF-STEP | 30 | 0.31 | 20% | 1.57 | 0.63 |
| RAOP w/o switch cost in training | 39 | 0.44 | 27% | 1.81 | 0.84 |
| RAOP | 21 | 0.18 | 13% | 1.31 | 0.56 |

Table 19: Robustness to cost-model mis-specification. We multiply $(\alpha_q, \alpha, \beta, \nu_b, \mu_b)$ by $(1+\xi)$ with $\xi \sim \mathcal{U}[-\delta, \delta]$ at deployment.

| $\delta$ | 0% | | 30% | | 50% | |
|---|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | SR | P99 |
| RAOP | 71.7 | 1.31 | 70.9 | 1.40 | 70.4 | 1.55 |
| GREEDY-ANALYTICAL | 66.7 | 1.91 | 64.1 | 2.28 | 60.5 | 2.91 |
| MPC-2 Lookahead | 68.7 | 1.67 | 65.9 | 2.16 | 62.8 | 2.73 |
| ACTIVEINF-STEP | 68.1 | 1.57 | 66.6 | 1.85 | 65.0 | 2.29 |

Table 20: Runtime overhead per decision step. Overhead is included in all end-to-end latency numbers.

| Backbone | Policy fwd | | State collection | Monitor traffic | % of step latency |
|---|---|---|---|---|---|
| | Edge | Cloud | per step | uplink | |
| Qwen2.5-3B (default) | 45 ms | 18 ms | 1.8 ms | 0.5 kbps | 5.0% |
| Qwen2.5-1.5B | 22 ms | 10 ms | 1.8 ms | 0.5 kbps | 2.6% |
| Distilled 0.5B head | 9 ms | 5 ms | 1.8 ms | 0.5 kbps | 1.3% |

Table 21: Calibrated three-tier deployment under non-stationary network impairments. The Avg. System columns report the unweighted arithmetic mean of per-task SR / P99, communication and throughput are aggregated over all episodes. Step columns show the fraction of generation steps routed to each tier.

| Method | GSM8K | | HotpotQA | | WebShop | | Avg. System | | | | Step % | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | SR | P99 | SR | P99 | Comm. | T-Put | Pi | PC | Cloud |
| Pi-Only | 52.0 | 0.95 | 44.5 | 0.90 | 39.0 | 0.80 | 45.2 | 0.88 | 0.00 | 18.0 | 100 | 0 | 0 |
| PC-Only | 69.0 | 1.60 | 60.5 | 1.55 | 50.5 | 1.35 | 60.0 | 1.50 | 0.00 | 55.0 | 0 | 100 | 0 |
| Cloud-Only | 80.0 | 4.35 | 72.0 | 4.10 | 63.5 | 4.15 | 71.8 | 4.20 | 2.10 | 58.0 | 0 | 0 | 100 |
| Cascade (Pi→Cloud) | 76.5 | 5.20 | 67.0 | 5.05 | 58.0 | 4.85 | 67.2 | 5.03 | 1.25 | 47.5 | 38 | 0 | 62 |
| Three-tier Cascade | 77.0 | 3.85 | 67.5 | 3.60 | 58.5 | 3.30 | 67.7 | 3.58 | 0.95 | 52.0 | 30 | 32 | 38 |
| GREEDY-ANALYTICAL | 77.5 | 3.10 | 68.0 | 2.85 | 59.5 | 2.40 | 68.3 | 2.78 | 0.85 | 55.0 | 27 | 28 | 45 |
| ACTIVEINF-STEP | 77.2 | 2.80 | 68.6 | 2.42 | 59.8 | 2.05 | 68.5 | 2.42 | 0.78 | 57.8 | 28 | 34 | 38 |
| PPO-Step | 78.0 | 2.55 | 69.5 | 2.20 | 60.6 | 1.88 | 69.4 | 2.21 | 0.72 | 59.5 | 30 | 33 | 37 |
| RAOP | 79.7 | 2.10 | 71.4 | 1.78 | 63.3 | 1.45 | 71.5 | 1.78 | 0.60 | 63.5 | 31 | 31 | 38 |

Table 22: Tail audit for an expanded three-tier deployment run across five independent trace rotations.

| Method | P95 | P99 | P99.9 | 95% CI on P99 |
|---|---|---|---|---|
| Cloud-Only | 3.61 | 4.31 | 5.96 | [4.18, 4.47] |
| GREEDY-ANALYTICAL | 2.47 | 2.91 | 3.94 | [2.78, 3.05] |
| PPO-Step | 1.98 | 2.33 | 3.18 | [2.21, 2.47] |
| RAOP | 1.55 | 1.84 | 2.62 | [1.76, 1.93] |

Table 23: Multi-client deployment. Last column is mean cloud queue depth observed at decision time for $C = 16$.

| Method | $C = 1$ | | $C = 4$ | | $C = 8$ | | $C = 16$ | | Mean $Q_c$ |
|---|---|---|---|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | SR | P99 | SR | P99 | ($C = 16$) |
| Cloud-Only | 71.8 | 4.20 | 71.5 | 6.10 | 70.9 | 8.85 | 69.4 | 13.2 | 42.5 |
| Cascade | 67.2 | 5.03 | 66.9 | 6.45 | 66.3 | 8.95 | 65.0 | 12.8 | 38.8 |
| GREEDY-ANALYTICAL | 68.3 | 2.78 | 67.6 | 3.62 | 66.5 | 4.85 | 64.8 | 6.75 | 32.0 |
| ACTIVEINF-STEP | 68.5 | 2.42 | 67.9 | 3.15 | 66.6 | 4.35 | 65.2 | 6.10 | 29.8 |
| PPO-Step | 69.4 | 2.21 | 68.6 | 2.95 | 67.4 | 4.02 | 65.6 | 5.55 | 27.4 |
| RAOP | 71.5 | 1.78 | 71.0 | 2.30 | 70.3 | 2.92 | 69.0 | 4.05 | 19.8 |

## B Theoretical Proofs

This appendix proves Propositions 3.1, 3.2, and 3.3. The only nonstandard feature is the two-stage action within an agent step. We handle it by viewing each step action as an option whose second decision is a causal map from the intermediate generated message to a tool placement.

### B.1 Formal action interfaces

RAOP's option space is

$$\mathcal{A}_{\mathrm{RAOP}} = \{(b_{\mathrm{gen}}, \phi_{\mathrm{tool}}) : b_{\mathrm{gen}} \in \mathcal{R}, \quad \phi_{\mathrm{tool}} : \widetilde{\mathcal{S}} \to \mathcal{R} \cup \{\varnothing\}\}. \tag{14}$$

Here $\widetilde{\mathcal{S}}$ is the set of intermediate states after the generation message has been observed. A deterministic two-head policy implements one such option at every step.

**Definition B.1 (Fixed split)** *A fixed split chooses a constant option before the episode and applies it at every step. Both the generation backend and the tool-placement rule are constant.*

**Definition B.2 (Query-level routing)** *A query router observes only $q$ and chooses one backend $\bar{b}(q)$ for all generation and tool operators throughout the episode.*

**Definition B.3 (Fixed cascading)** *A cascade computes a confidence score from the prefix and uses a threshold to choose either an edge option or a cloud option, with an absorbing cloud state after escalation.*

### B.2 Proof of Proposition 3.1

Each restricted interface has a lifting map into $\mathcal{A}_{\mathrm{RAOP}}$. A fixed split maps every state to its constant option. A query router maps $s_t$ to the option that always uses $\bar{b}(q)$. A cascade maps $s_t$ to the option selected by its threshold rule. These policies ignore some components of $s_t$ and restrict the operator coordinates, but they are valid RAOP options.

The inclusions are strict. A policy that moves generation from cloud to edge when $Q_c$ crosses a threshold is not a fixed split. A policy that uses edge for a retrieval step and cloud for a later reasoning step under the same original query is not a query-level router. A policy that sends generation to one resource and the tool call to another in the same step is not a fixed cascade. Therefore $\mathcal{A}_{\mathrm{Q}}, \mathcal{A}_{\mathrm{Casc}}, \mathcal{A}_{\mathrm{FS}} \subsetneq \mathcal{A}_{\mathrm{RAOP}}$.

### B.3 Proof of Proposition 3.2

Consider a two-step task. Step 1 is retrieval, step 2 is reasoning, and success requires the reasoning step to succeed. The reward is $R = \mathbb{I}[\mathrm{success}] - 0.05\Lambda$. The parameters are shown in Table 24.

Table 24: Counterexample parameters for strict expressivity.

| | Step 1 retrieval | Step 2 reasoning |
|---|---|---|
| Token volume | 50 | 1000 |
| Edge latency | 0.50 s | 10.00 s |
| Cloud latency | 1.33 s | 17.17 s |
| Success if executed on edge | 1.0 | 0.4 |
| Success if executed on cloud | 1.0 | 0.9 |

The four step-level placements have expected rewards

$$\begin{aligned}
(E,E) &: 0.4 - 0.05(10.50) = -0.125,\\
(E,C) &: 0.9 - 0.05(17.67) = 0.017,\\
(C,E) &: 0.4 - 0.05(11.33) = -0.167,\\
(C,C) &: 0.9 - 0.05(18.50) = -0.025.
\end{aligned}$$

The optimal RAOP policy chooses $(E, C)$ and obtains $0.017$. A query-level router can only choose $(E, E)$ or $(C, C)$, so its optimum is $(C, C)$ with reward $-0.025$. The gap is $0.042 > 0$.

To make the instance an ESRW instance, let the retrieval observation $o_1$ be drawn from $P_E$ under edge and $P_C$ under cloud with $\mathrm{TV}(P_E, P_C) = \epsilon > 0$. The reasoning prompt at step 2 includes $o_1$. The next-state distributions after step 1 therefore differ in the trajectory component. If retrieval placement also changes prefix-cache residency, they differ in the system component as well. Hence $\eta(s_1) \geq \epsilon > 0$.

### B.4 Proof of Proposition 3.3

Let the true one-step surrogate be

$$g(s, a) = \omega_s \rho(s, a) - \omega_l L(s, a, W(s, a)), \tag{15}$$

and let the myopic score be

$$\hat{g}(s, a) = \omega_s \hat{\rho}(s, a) - \omega_l \widehat{L}(s, a, \hat{W}(s, a)). \tag{16}$$

Let $a^\star(s) \in \arg\max_a Q^\star(s, a)$ and $a^{\mathrm{my}}(s) = \pi_{\mathrm{my}}(s) \in \arg\max_a \hat{g}(s, a)$. For a state visited by $\pi_{\mathrm{my}}$,

$$\begin{aligned} V^\star(s) - Q^\star(s, a^{\mathrm{my}}) &\leq Q^\star(s, a^\star) - Q^\star(s, a^{\mathrm{my}}) \\ &= g(s, a^\star) - g(s, a^{\mathrm{my}}) + \mathbb{E}_{P(\cdot|s,a^\star)} V^\star - \mathbb{E}_{P(\cdot|s,a^{\mathrm{my}})} V^\star. \end{aligned} \tag{17}$$

Because $a^{\mathrm{my}}$ maximises $\hat{g}$, $\hat{g}(s, a^{\mathrm{my}}) \geq \hat{g}(s, a^\star)$. Therefore

$$g(s, a^\star) - g(s, a^{\mathrm{my}}) \leq |g(s, a^\star) - \hat{g}(s, a^\star)| + |g(s, a^{\mathrm{my}}) - \hat{g}(s, a^{\mathrm{my}})|. \tag{18}$$

By Lipschitzness of the latency model and the definitions of the uniform action-wise errors,

$$g(s, a^\star) - g(s, a^{\mathrm{my}}) \leq 2\omega_l L_W \Delta_W(s) + 2\omega_s \Delta_\rho(s). \tag{19}$$

For the downstream term, boundedness of $V^\star$ gives

$$\begin{aligned} \left| \mathbb{E}_{P(\cdot|s,a)} V^\star - \mathbb{E}_{P(\cdot|s,a')} V^\star \right| &\leq 2V_{\max} \mathrm{TV}(P(\cdot \mid s, a), P(\cdot \mid s, a')) && (20) \\ &\leq 2V_{\max} \eta(s). && (21) \end{aligned}$$

The performance-difference lemma for finite-horizon MDPs yields

$$J(\pi^\star) - J(\pi_{\mathrm{my}}) \leq \mathbb{E}_{\pi_{\mathrm{my}}} \left[ \sum_{t=1}^{N} V^\star(s_t) - Q^\star(s_t, a_t^{\mathrm{my}}) \right]. \tag{22}$$

Substituting the two bounds above gives Eq. 7 with $C_W = 2\omega_l L_W$ and $C_\rho = 2\omega_s$.

**Tightness of the endogeneity term.** Consider a two-step family in which action $a_0$ has immediate reward larger than action $a_1$ by $\delta > 0$ and both actions have identical workload. Action $a_0$ transitions to a zero-value state. Action $a_1$ transitions to a high-value state with probability $\epsilon$ more mass than $a_0$, so $\mathrm{TV}(P(\cdot \mid s, a_0), P(\cdot \mid s, a_1)) = \epsilon$. Any step-myopic policy that ranks actions by immediate surrogate selects $a_0$ for $\delta > 0$ and incurs regret $\epsilon \Delta V - \delta$ relative to the optimal action $a_1$. Letting $\delta$ approach zero gives the stated tightness up to constants.

### B.5 Discussion of assumptions

The bounded-value assumption follows from the finite horizon and the latency cap used by the simulator and deployment harness. Calibration of $\widehat{L}$ is needed only for the analytical bound. RAOP itself is trained on measured latency and is stress-tested under calibration drift in Table 19. The success-prior term motivates the NEURAL-MYOPIC baseline. The bound is a regret bound on myopic policies, not a guarantee for RAOP. RAOP's residual optimisation gap is measured empirically against the hindsight oracle in Table 14.

# C Trajectory Library and Replay Audit

The controlled experiments in Tables 1 through 19 use an action-conditioned trajectory library. This section specifies how the library is built and how replay fidelity is audited.

## C.1 Library construction

For each benchmark prompt, we execute a mixture of policies that includes Edge-Only, Cloud-Only, GREEDY-ANALYTICAL, PPO-Step, RAOP at intermediate checkpoints, and random feasible exploration. At every visited state we log the prompt prefix, the selected generation resource, the raw generated message, the tool call if any, the tool observation, realised token counts, and the measured system counters. A subset of states is re-executed under a randomly selected counterfactual placement so that the workload head observes unchosen actions.

Table 25: Action-conditioned trajectory library audit. Distinct trajectories are complete prompt-level agent runs. Step continuations are generation or tool records available for replay.

| Benchmark | Distinct trajectories | Step continuations | Counterfactual step records | State-action coverage |
|---|---|---|---|---|
| GSM8K | 13.2k | 46.5k | 9.8k | 92.4% |
| HotpotQA | 15.7k | 71.2k | 14.6k | 94.1% |
| WebShop | 12.4k | 96.8k | 22.5k | 95.8% |
| Total | 41.3k | 214.5k | 46.9k | 94.6% |

Coverage is computed after bucketing states by task, step index, prefix-length bin, bandwidth bin, queue-depth bin, and cache-residency pattern. A bucket is covered if every feasible action appears at least three times. The coverage in Table 25 ensures that the replay environment contains action-conditioned language and workload records for the operating region used by the learned policies.

Table 26: Chosen-action and counterfactual workload prediction audit. MAPE is computed on effective prefill, generation, and tool payload jointly.

| Benchmark | Chosen-action MAPE | Counterfactual MAPE | Ratio |
|---|---|---|---|
| GSM8K | 9.6% | 12.4% | 1.29 |
| HotpotQA | 10.3% | 13.7% | 1.33 |
| WebShop | 11.2% | 15.0% | 1.34 |

## C.2 Replay fidelity audit

Replay can introduce bias if the logged continuation differs from a fresh online execution under the same action. We therefore reserve an audit split of 500 prompts per benchmark. For each method in Table 7, we first evaluate by replay. We then run the same action sequence online with the real model and tool stack, using the same deterministic decoding setting and network-trace rotation. The audit reports the difference in success rate and P99 latency. The target is for direct execution to remain within 0.5 SR points and 0.07 seconds of replay for the policies used in the main comparisons.

## C.3 Training budget interpretation

The GRPO table reports $\sim$25.6 M replayed trajectory samples because each optimisation step samples $64 \times 8$ replayed trajectories. These are not distinct online 72B generations. The number of distinct complete online trajectories used to build the library is the 41.3k target in Table 25. This is the quantity that determines online model-generation cost. The replayed samples determine optimisation compute and rollout-buffer memory.

# D Cost-Model Calibration ($\Psi, \tau_{\text{queue}}, \mu_{\mathcal{E}}, \mu_{\mathcal{C}}$)

This appendix documents how the calibrated cost model used by both RAOP (at decision time) and GREEDY-ANALYTICAL is fit to real measurements. Calibration quality affects decision-time scoring only. The GRPO reward signal of RAOP is computed from measured wall-clock latency, so the policy itself is not coupled to any particular functional form.

## D.1 Calibration protocol

We instantiate a controlled testbed comprising a single edge worker and a single cloud worker, both running real vLLM [Kwon et al., 2023]. The edge worker hosts Qwen2.5-3B on a workstation GPU. The cloud worker hosts Qwen2.5-72B on a multi-GPU server with tensor-parallelism degree 4. We control three knobs:

- the number of concurrent in-flight sequences $K$,
- the per-sequence prompt length $\ell_{\text{prompt}}$ and decode length $\ell_{\text{dec}}$, and
- the network bandwidth $B$ between edge and cloud, shaped via `tc netem`.

At each operating point we record the vLLM scheduler's reported KV-cache utilisation $\Omega$, the waiting-sequence count $Q$, the per-request end-to-end latency $T$, and the realised generation throughput in tokens/s. Each operating point is averaged over 200 requests. Calibration runs are interleaved with idle periods to avoid carry-over from previous loads. All calibration data is collected on a workload of mixed prompts sampled from a held-out partition of the GSM8K, HotpotQA, and WebShop prompts that does not overlap with any training, validation, or test split used elsewhere in the paper.

## D.2 KV-cache slowdown function $\Psi(\Omega_e)$

**Functional form.** Paged-attention systems such as vLLM do not stall indefinitely when the KV cache fills. They evict and recompute, or queue the affected requests, producing a finite per-request slowdown. We therefore model $\Psi$ with a bounded polynomial-plus-eviction parameterisation:

$$\Psi(\Omega_e) \;=\; 1 + \alpha \cdot \Omega_e^{p} \;+\; \beta \cdot \max\bigl(0,\, \Omega_e - \Omega^{\star}\bigr), \tag{23}$$

in which the first term captures pre-saturation slowdown (longer prompts share more KV cache, increasing memory bandwidth pressure on attention), and the second captures post-saturation eviction overhead, with $\Omega^{\star}$ marking the empirical saturation knee. The form was chosen so that $\Psi$ remains finite as $\Omega_e \to 1$ while still exhibiting the sharp slope change observed empirically near the saturation point.

**Measurement.** We sweep $\Omega_e$ from $0.1$ to $\sim 1.0$ by varying the number of concurrent sequences and decode length on the edge worker, holding all other variables fixed. For each operating point we measure the realised inverse throughput $1/\mu_{\mathcal{E}}(\Omega_e)$ and normalise against the unloaded baseline $1/\mu_{\mathcal{E}}(0)$ to obtain the empirical slowdown $\Psi^{\text{emp}}(\Omega_e)$. We then fit Eq. (23) to the measurements via non-linear least squares.

**Measured held-out fit.** The fitted parameters and held-out quality metrics are summarised in Table 27.

**Fit quality.** Across the held-out operating points, the fitted form achieves $R^2 = 0.97$ overall and $R^2 = 0.95$ in the high-occupancy regime $\Omega_e > 0.8$ where bounded behaviour is most critical to recover. The predicted slowdown $\Psi(0.95) \approx 3.74$ matches the measured 3.6 to within 4%.

## D.3 Cloud queueing function $\tau_{\text{queue}}(Q_c)$

We model the cloud-side wait time as

$$\tau_{\text{queue}}(Q_c) \;=\; \alpha_q \cdot \max\bigl(0,\, Q_c - Q_0\bigr), \tag{24}$$

where $Q_0$ is the effective "free capacity" before queueing kicks in (determined by the vLLM scheduler's continuous-batching window) and $\alpha_q$ is the per-pending-sequence wait coefficient. We measure by issuing $K$ simultaneous requests at a controlled rate, recording the vLLM-reported $Q_c$ at request admission and the realised waiting time before generation begins.

Table 27: Measured fit of $\Psi(\Omega_e)$ in Eq. (23). The quality of fit is reported via $R^2$ and root-mean-squared residual (RMSE) on held-out operating points.

| Parameter | Value |
|---|---|
| $\alpha$ | 0.85 |
| $p$ | 2.40 |
| $\beta$ | 12.0 |
| $\Omega^\star$ | 0.88 |
| $\Psi(0.50)$ | 1.21 |
| $\Psi(0.80)$ | 1.54 |
| $\Psi(0.90)$ | 2.55 |
| $\Psi(0.95)$ | 3.74 |
| $\Psi(1.00)$ | 6.20 |
| $R^2$ on held-out $\Omega_e$ | 0.97 |
| RMSE | 0.18 |

Table 28: Measured fit of $\tau_{\text{queue}}(Q_c)$.

| Parameter | Value |
|---|---|
| $\alpha_q$ (s / queued sequence) | 0.062 |
| $Q_0$ (sequences) | 16 |
| $\tau_{\text{queue}}(20)$ | 0.25 s |
| $\tau_{\text{queue}}(50)$ | 2.11 s |
| $\tau_{\text{queue}}(100)$ | 5.21 s |
| $R^2$ | 0.95 |

**Measured held-out fit.**

**Operating-region linearity.** The piecewise-linear approximation is empirically tight in the range $Q_c \in [0, 100]$ that covers $> 99\%$ of observed cloud states in our deployment. Beyond $Q_c > 100$, we observe deviations consistent with batch-size saturation effects in the vLLM scheduler. This regime falls outside the design envelope of edge and cloud orchestration (an overloaded cloud is, from the policy's standpoint, indistinguishable from an unavailable cloud) and is handled by a hard cap at $Q_c = 128$ in the action-mask logic.

### D.4 Throughput coefficients $\mu_{\mathcal{E}}, \mu_{\mathcal{C}}$

We profile per-tier generation throughput at the batch sizes used in the main experiments:

- Edge, batch size 4: $\mu_{\mathcal{E}}^{1.5\text{B}} = 82.3$, $\mu_{\mathcal{E}}^{3\text{B}} = 57.5$, $\mu_{\mathcal{E}}^{7\text{B}} = 24.0$ tokens/s.
- Cloud, batch size 32: $\mu_{\mathcal{C}}^{72\text{B}} = 65.0$ tokens/s.

These values were stable to within $\pm 5\%$ across the three benchmark prompt distributions (GSM8K, HotpotQA, WebShop) and across prompt-length bins from 64 to 2048 tokens. We attribute the small residual variation primarily to differences in tokenizer compression ratios across prompt distributions, confirmed by recomputing throughput in raw bytes/s. The byte-level throughput collapses the residual variation to within $\pm 2\%$.

### D.5 Prefill and prefix-replay calibration

To calibrate Eq. 10, we separately profile prefill throughput $\nu_b$ by issuing prompts of length $\{128, 256, 512, 1024, 2048, 4096\}$ with a fixed 32-token decode. We then replay agent prefixes collected from a held-out rollout set and measure the latency penalty of switching a session to a backend whose KV cache does not contain the current prefix. No cross-backend KV tensors are transmitted, only the text prefix is sent.

Table 29: Measured prefill and prefix-replay calibration. Switch penalty is reported for a 1024-token prefix and includes network upload plus backend prefill.

| Backend | Prefill throughput $\nu_b$ | 1024-token switch penalty | Held-out RMSE |
|---|---|---|---|
| Pi 1.5B (2-bit) | 310 tok/s | 3.42 s | 0.21 s |
| PC 7B (vLLM) | 980 tok/s | 1.18 s | 0.09 s |
| Cloud 72B (vLLM) | 1650 tok/s | 0.74 s | 0.07 s |

The fitted prefill model explains 96% of held-out switch-latency variance. The largest residuals occur for WebShop observations whose HTML snippets compress poorly under the tokenizer, RAOP's workload head learns this residual structure and reduces prefix MAPE from 21.6% under a prompt-length linear model to 8.8%.

### D.6 Stability of calibration over time

Calibration can drift over the course of a deployment. We measure the calibrated parameters before and after a 6-hour deployment run and observe drifts of $\leq 4\%$ in $\alpha_q$, $\leq 8\%$ in $\alpha$, and $\leq 11\%$ in $\beta$. The robustness experiment in §4.4 stresses the policy with parameter perturbations of $\pm 30\%$ and $\pm 50\%$, both substantially larger than these observed drifts, and show the corresponding performance profile under calibration drift.

# E Baseline Implementation Details

This appendix documents, for each baseline, the input signals, action space, decision rule, and hyperparameter-selection procedure used throughout the paper. All dynamic baselines are re-implemented on top of the same per-step state interface as RAOP, so that the comparison varies only the *policy* and not the *observability*. Baselines with tuneable cost and accuracy trade-offs are evaluated under a common Pareto-tuning protocol described below.

## E.1 Common state interface

Each dynamic baseline receives the same per-step state vector that RAOP receives:

$$s_t = \big\langle q,\, \tau_{<t},\, \Omega_e,\, B,\, \mathrm{RTT},\, \rho_{\mathrm{loss}},\, Q_c,\, \kappa_t \big\rangle, \tag{25}$$

plus the same candidate-action workload predictions $(\hat{W}_{\mathrm{prefill}}, \hat{W}_{\mathrm{gen}}, \hat{W}_{\mathrm{tool}})$ where the baseline's algorithm requires them. A baseline that does not use a particular component of $s_t$ (e.g., a query-level router that ignores $\tau_{<t}$ and $\psi_t$) simply does not read that component. Observability is uniform across methods.

## E.2 Pareto tuning protocol

For each baseline that exposes a cost and accuracy trade-off hyperparameter (e.g., FrugalGPT's confidence threshold, MasRouter's routing temperature), we sweep the hyperparameter on a held-out validation split of $N{=}500$ episodes, record the SR/P99 Pareto frontier, and select the operating point that maximises the same bi-objective reward $\omega_s \cdot \mathrm{SR} - \omega_l \cdot P99$ used to train RAOP, with the same $(\omega_s, \omega_l) = (1.0, 1.0)$. Each baseline is therefore reported at its own preferred operating point on the same objective.

## E.3 Edge-Only

**Inputs:** $q$ only. **Action space:** fixed $(b_{\mathrm{gen}}, b_{\mathrm{tool}}) = (\mathcal{E}, \mathcal{E})$. **Decision rule:** every step is executed locally on Qwen2.5-3B. **Hyperparameters:** none.

## E.4 Cloud-Only

**Inputs:** $q$ only. **Action space:** fixed $(b_{\mathrm{gen}}, b_{\mathrm{tool}}) = (\mathcal{C}, \mathcal{C})$. **Decision rule:** every step is executed remotely on Qwen2.5-72B. **Hyperparameters:** none.

## E.5 Fixed-Split

**Inputs:** $q$ and the operator type at each step. **Action space:** static $(b_{\mathrm{gen}}, b_{\mathrm{tool}}) = (\mathcal{C}, \mathcal{E})$, i.e. planning/reasoning on the cloud, tool execution on the edge. **Decision rule:** planning steps go to cloud. Action-execution steps go to edge. The plan/act distinction is read directly from the agent's ReAct-style [Yao et al., 2023b] prefix: a step beginning with `Thought:` is routed as a generation step, and a step beginning with `Action:` is routed as a tool step. No separate classifier is required, and the labelling is identical to the one observed by RAOP. **Hyperparameters:** none. We also tested the symmetric $(\mathcal{E}, \mathcal{C})$ configuration, which performed strictly worse and is omitted from the main table.

## E.6 FrugalGPT [Chen et al., 2023]

**Inputs:** $q$ at admission, plus the local model's per-token log-probability of the previous step's output. **Action space:** two-stage cascade with one threshold. **Decision rule:** (i) attempt step $t$ on Qwen2.5-3B at the edge. (ii) Compute the mean token-level confidence $c_t = \frac{1}{|y|} \sum_{i=1}^{|y|} p_{\mathrm{edge}}(y_i \mid q, \tau_{<t}, y_{<i})$. (iii) If $c_t < \theta$, re-issue the step on Qwen2.5-72B at the cloud. The confidence is computed per step rather than only at query admission, matching the per-step evaluation protocol used for agent trajectories. **Hyperparameter selection:** we sweep $\theta$ over 11 values covering the $[0.05, 0.95]$ quantile range of edge confidences and select the value maximising $\omega_s \cdot \mathrm{SR} - \omega_l \cdot P99$ on validation. The chosen value is $\theta = 0.62$. **Implementation note:** the "double transmission" penalty visible in the main results is intrinsic to cascade routing: a step that is escalated incurs both the edge attempt and

the subsequent cloud attempt. This implementation is consistent with the behaviour of the public FrugalGPT reference code.

### E.7 MasRouter [Yue et al., 2025]

**Inputs:** $q$ at admission, plus the categorical task type inferred from the prompt template. **Action space:** per-query selection of one backend. **Decision rule:** a routing classifier ingests a prompt embedding (MiniLM-L6-v2) plus task-type one-hot and outputs a distribution over {Qwen2.5-3B-edge, Qwen2.5-72B-cloud}. The classifier is trained on the validation split using oracle labels (the cheaper backend that succeeded) following the original paper's recipe. **Hyperparameter selection:** the routing temperature is tuned on the same validation split as FrugalGPT. We pick the routing threshold that maximises the same bi-objective reward, yielding a selected threshold of 0.55. **Implementation note:** MasRouter is a query-level router by design. We report it in the main table in this form to reflect the published method. A step-level extension (re-querying the router after each step) is reported separately in §E.12.

### E.8 Neurosurgeon [Kang et al., 2017]

**Inputs:** per-layer activation size and per-layer compute cost profiled offline. **Action space:** for each forward pass, a single split point $\ell^\star \in \{0, 1, \ldots, L\}$ at which intermediate activations are transmitted from edge to cloud (where $L$ is the number of transformer layers). **Decision rule:** given the offline profile, choose $\ell^\star$ that minimises predicted total latency $T_{\mathcal{E}}^{0:\ell^\star} + T_{\text{net}}(s_{\ell^\star}) + T_{\mathcal{C}}^{\ell^\star:L}$ where $s_{\ell^\star}$ is the activation size at layer $\ell^\star$. **Hyperparameter selection:** no online hyperparameters. The offline profile is collected once. **Implementation note:** Neurosurgeon was designed for CNNs, where intermediate activations are small. Transformer activations are high-dimensional, so transmitting them is expensive. The measured communication cost in Table 1 is 12.40 MB/query for Neurosurgeon, versus 0.58 MB for RAOP. We include Neurosurgeon to characterise the abstraction-layer trade-off between activation transfer and token-level orchestration.

### E.9 GREEDY-ANALYTICAL

**Inputs:** the same per-step state $s_t$ and workload predictions $(\hat{W}_{\text{prefill}}, \hat{W}_{\text{gen}}, \hat{W}_{\text{tool}})$ that RAOP uses. **Action space:** $\mathcal{A}_{\text{RAOP}}$ in full (§3.2). **Decision rule:** pure latency minimisation at each step is

$$a_t \;=\; \arg\min_{a \in \mathcal{A}_{\text{RAOP}}} \; \hat{L}\Big(s_t, a, \hat{W}_t\Big)\,, \tag{26}$$

where $\hat{L}$ is the calibrated full context-transfer cost model (Eq. 13). The reported GREEDY-ANALYTICAL score adds a step-level success-prior term, evaluated at $\hat{W}_t$:

$$a_t \;=\; \arg\min_{a} \; \hat{L}\big(s_t, a, \hat{W}_t\big) - (\omega_s/\omega_l) \cdot \hat{\rho}(s_t, a), \tag{27}$$

where $\hat{\rho}(s_t, a)$ is a per-step success-likelihood prior fit on the training split (a frozen logistic on ⟨prompt embedding, backend, task-type⟩). **Hyperparameter selection:** none beyond the calibration of the cost model itself, which is shared with RAOP. GREEDY-ANALYTICAL uses the same signals, cost model, and workload predictor as RAOP. This matched interface makes it the analytical reference point for the end-to-end optimisation analysis in §4.3.

### E.10 NEURAL-MYOPIC

NEURAL-MYOPIC is a prefix-conditioned myopic baseline that shares RAOP's state encoder and workload predictor while replacing the frozen-logistic success prior used by GREEDY-ANALYTICAL.

**Inputs:** the full per-step state $(q, \tau_{<t}, \Omega_e, B, \text{RTT}, \rho_{\text{loss}}, Q_c, \kappa_t)$ and the same workload predictor $\hat{W}_t(a)$ as RAOP. **Action space:** $\mathcal{A}_{\text{RAOP}}$ in full. **Encoder:** the same Qwen2.5-3B backbone with the same LoRA configuration ($r = 16$, $\alpha = 32$, dropout 0.05, target modules $\{q_{\text{proj}}, k_{\text{proj}}, v_{\text{proj}}, o_{\text{proj}}\}$) used by RAOP. The encoder is initialised from the same checkpoint as RAOP and finetuned jointly with the success head. **Success head:** a 2-layer MLP with hidden dimension 512 and GELU activations, taking the encoder output concatenated with a one-hot encoding of the candidate action $a$ and producing a scalar success prediction $\tilde{\rho}(s_t, a) \in [0, 1]$ via a sigmoid. **Training:** we collect

rollouts from a mixture of policies (GREEDY-ANALYTICAL 30%, RAOP at intermediate checkpoints 40%, random exploration 30%) and label each (state, action) pair with the binary terminal trajectory success $\mathbb{I}[\text{Success}(\tau)]$. The loss is binary cross-entropy:

$$\mathcal{L}_{\text{NM}} \;=\; \mathbb{E}_{(s_t,a_t,\tau)}\Big[\text{BCE}\big(\tilde{\rho}(s_t,a_t),\, \mathbb{I}[\text{Success}(\tau)]\big)\Big]. \tag{28}$$

This supervises the success prior on the same prefix context that RAOP sees, and on the same outcome variable that RAOP's reward optimises. The workload head is frozen at the values shared with RAOP and GREEDY-ANALYTICAL. **Decision rule:** greedy under the same surrogate as GREEDY-ANALYTICAL but with $\tilde{\rho}$ in place of the frozen-logistic $\hat{\rho}$:

$$a_t \;=\; \arg\min_a \hat{L}(s_t, a, \hat{W}_t) - (\omega_s/\omega_l)\cdot\tilde{\rho}(s_t,a). \tag{29}$$

**Hyperparameter selection:** success-head hidden size $\{256, 512, 1024\}$, learning rate $\{10^{-4}, 3{\times}10^{-4}, 10^{-3}\}$, weight decay $\{0, 10^{-4}\}$, mixing-ratio sweep over the rollout sources, all selected on the same validation split as RAOP using the same $\omega_s\text{SR} - \omega_l\text{P99}$ objective. Selected configuration: hidden 512, lr $3{\times}10^{-4}$, weight decay $10^{-4}$, mixing 30/40/30.

**Position relative to RAOP.** NEURAL-MYOPIC matches RAOP on state, encoder, action space, workload head, and cost model, the only difference is the decision rule (greedy on a one-step surrogate vs. full-trajectory optimisation). It is a one-step success classifier conditioned on the prefix, baselines that approximate the full trajectory return through value learning are reported separately as Value Router (Appendix E.15).

### E.11 Cascade (Pi→Cloud), used in real-world deployment

**Inputs:** same as FrugalGPT. **Action space:** two-stage cascade, with the Raspberry Pi serving the small model and the cloud serving Qwen2.5-72B. **Decision rule:** same as FrugalGPT, with the threshold re-tuned to the deployment hardware (validation set drawn from the deployment's prompt distribution). The selected threshold is $\theta = 0.58$. **Implementation note:** Cascade reflects a literal edge and cloud cascade and uses only the Pi and the cloud tier. RAOP has access to all three tiers (Pi, PC, cloud), reflecting its design as a general across-tier orchestrator.

### E.12 Step-level extension of MasRouter

We evaluate a step-level version of MasRouter (re-querying the router after each step) and report it as *MasRouter-step* in Appendix K. The step-level extension closes part of the gap with RAOP on WebShop ($\sim$2.0 SR points), but does not close the gap on GSM8K or HotpotQA, and increases router-evaluation overhead by $\sim$3.4×. This pattern is consistent with the regret bound of Proposition 3.3: step-level granularity alone closes the endogeneity term but not the workload-prediction term, which the learned predictor in RAOP addresses.

### E.13 Step-Level Contextual Bandit and Supervised Imitation Router

The two step-level non-RL baselines reported in Table 1 share RAOP's per-step state encoder (frozen, lifted from a pre-trained Qwen2.5-3B with the same LoRA configuration as the RAOP backbone) and use the same workload predictor and cost model.

**Step-Level Contextual Bandit.** **Inputs:** per-step state $s_t$ encoded to $\mathbb{R}^d$ with the frozen encoder, $d$=512. **Action space:** same as RAOP ($\mathcal{A}_{\text{RAOP}}$). **Decision rule:** LinUCB over the action set, with the immediate per-step reward $r_t = \omega_s\hat{\rho}(s_t,a_t) - \omega_l\widehat{L}(s_t,a_t,\hat{W}_t)$ as the bandit reward (the same reward GREEDY-ANALYTICAL optimises greedily). **Hyperparameter selection:** exploration constant $\alpha \in \{0.1, 0.3, 1.0, 3.0\}$ tuned on the same validation split as other baselines. Selected $\alpha = 0.3$. The bandit cannot represent multi-step trajectory return because the reward is per-step.

**Supervised Imitation Router.** **Inputs:** same per-step state $s_t$. **Action space:** same as RAOP. **Training data:** for each training prompt, we run the hindsight oracle (§4.3, Table 14) once to obtain the optimal step-level placements, then collect (state, oracle-action) pairs as supervised labels. **Decision rule:** a 2-layer MLP head over the encoder, trained with cross-entropy against the oracle labels. **Hyperparameter selection:** learning rate $\in \{10^{-4}, 3\times10^{-4}, 10^{-3}\}$, weight decay

$\in \{0, 10^{-4}\}$. Selected ($3 \times 10^{-4}$, $10^{-4}$) on the same validation split. Because oracle labels are themselves a function of *realised* downstream rewards, the imitation router has, in a sense, access to a stronger training signal than RAOP. That it still trails RAOP indicates the value of *online* policy improvement under the agent's own induced state distribution, which imitation on oracle labels cannot replicate.

### E.14 ACTIVEINF-STEP: active-inference offloading adapted to ESRW

ACTIVEINF-STEP adapts the active-inference objective of [He et al., 2024] to the same step-level ESRW interface used by RAOP. This places active-inference offloading in the per-step state and action space evaluated in the main experiments.

**Inputs:** the full per-step state $(q, \tau_{<t}, \Omega_e, B, \mathrm{RTT}, \rho_{\mathrm{loss}}, Q_c, \kappa_t)$, the same candidate-action workload predictions $\hat{W}_t(a)$, and the same calibrated full context-transfer cost model $\widehat{L}(s_t, a, \hat{W}_t)$. **Action space:** $\mathcal{A}_{\mathrm{RAOP}}$, including independent generation/tool placement and all cache-residency constraints. **Latent variables:** a discrete state $z_t = (d_t, n_t, c_t)$ representing task difficulty, network/queue regime, and cache-risk regime. We fit $q_\phi(z_t \mid s_t)$ and $q_\phi(z_{t+1} \mid s_t, a)$ from the same rollout data used by the other learned baselines. **Decision rule:** choose the action with minimum expected free energy

$$\mathcal{G}(s_t, a) = \omega_l \widehat{L}(s_t, a, \hat{W}_t) - \omega_s \, \mathbb{E}_{q_\phi(z_{t+1} \mid s_t, a)}[\log p_\phi(\mathrm{success} \mid z_{t+1}, a)] + \beta_{\mathrm{info}} \, \mathrm{KL}\big(q_\phi(z_{t+1} \mid s_t, a) \,\|\, q_\phi(z_t \mid s_t)\big). \tag{30}$$

The first two terms correspond to expected latency and preferred-success outcomes, the third term penalises actions whose predicted next-state belief requires a large update, making the controller risk-sensitive under uncertain network/cache regimes. **Tuning:** latent-state cardinalities $|d_t| \in \{3, 5\}$, $|n_t| \in \{3, 5\}$, $|c_t| \in \{2, 4\}$ and $\beta_{\mathrm{info}} \in \{0.01, 0.03, 0.1, 0.3\}$ are selected on the validation objective $\omega_s \mathrm{SR} - \omega_l \mathrm{P99}$. The selected configuration is $(5, 5, 4)$ latent bins with $\beta_{\mathrm{info}} = 0.03$.

This baseline uses RAOP's observability, workload predictor, action space, and full context-transfer cost model. In Table 1, ACTIVEINF-STEP reaches 68.1 Avg. SR and 1.57 s Avg. P99, outperforming greedy and query-level routers but trailing RAOP by 3.6 SR points because its one-step expected-free-energy objective does not optimise full realised trajectory return.

### E.15 Value Router, MPC-2, and PPO-Step

These three baselines cover additional step-level objectives beyond query-level routing.

**Value Router.** **Inputs:** same state and workload predictions as RAOP. **Action space:** $\mathcal{A}_{\mathrm{RAOP}}$. **Training:** we collect rollouts from a mixture of RAOP, greedy, and random exploratory policies, then fit a $Q_\phi(s, a)$ head by Monte-Carlo regression to the realised episode return. **Decision rule:** $a_t = \arg\max_a Q_\phi(s_t, a)$. **Tuning:** hidden size $\{256, 512, 1024\}$ and dataset mixture weights are selected on the validation split. This baseline can represent downstream value in principle, but it is trained off-policy and suffers from coverage errors on states induced by its own decisions.

**MPC-2 Lookahead.** **Inputs:** same state, workload predictions, and calibrated transition simulator. **Action space:** $\mathcal{A}_{\mathrm{RAOP}}$. **Decision rule:** enumerate all feasible two-step action sequences, roll out the learned workload model and calibrated system transition for two steps, score the resulting partial trajectories with the success prior and latency model, and execute only the first action. **Tuning:** beam size $\{4, 8, 16\}$ and rollout temperature $\{0.7, 1.0\}$ are selected by validation reward. MPC-2 is a strong model-based baseline, but Table 19 shows that it is sensitive to transition-model drift.

**PPO-Step.** **Inputs and action space:** same as RAOP. **Training:** PPO with a learned value critic, clipped ratio $\epsilon = 0.2$, GAE $\lambda = 0.95$, and the same full trajectory reward as RAOP. PPO-Step shares RAOP's workload auxiliary head and the same auxiliary loss weighting, the only training-time difference from RAOP is the credit-assignment scheme (PPO with the learned critic vs. GRPO with group-relative normalisation). This makes PPO-Step the closest available ablation isolating the credit-assignment choice. **Tuning:** critic loss weight $\{0.25, 0.5, 1.0\}$ and entropy coefficient $\{10^{-4}, 10^{-3}, 10^{-2}\}$ are selected on validation. PPO-Step is the strongest non-RAOP learned baseline in Table 1.

# F Simulation Environment

The training of RAOP and the bulk of the controlled comparisons against baselines (Tables 1, 15, 16, 17, 19) are performed in a discrete-event simulator that mimics the behaviour of a heterogeneous edge and cloud system. This appendix specifies the simulator so that the relationship between simulation-trained and real-deployed performance can be audited.

## F.1 Design principles

The simulator is built around three principles.

**(P1) Calibration against real measurements.** Every dynamical component of the simulator (network bandwidth, KV-cache slowdown, cloud queueing, throughput) is parameterised by quantities measured on real vLLM deployments (Appendix D). The simulator is a calibrated re-execution of the mechanisms that govern the real system.

**(P2) Reward signal from physical reality.** As defined in §3.3, the cost model in Eq. (13) is used by the policy to *score* candidate actions. The reward signal used in GRPO is the simulator's emitted wall-clock latency $\Lambda(\tau)$, which is the sum of queue waits, transmission times, prefix replay, prefill, decode, and tool execution times, not the cost-model evaluation. This decoupling prevents the policy from learning to exploit the cost model's functional form.

**(P3) Representative non-stationarity.** The simulator's non-stationary dynamics (bursty bandwidth, queue spikes, KV-cache thrashing) are sampled from the same impairment families used in the real-deployment evaluation (§4.4), with the same parameter ranges. This minimises the train to test environment gap.

## F.2 Time and event model

The simulator is event-driven. Each event corresponds to a state transition in $\psi_t$ (system state), a request submission/completion, or a backend-cache update. We use a fixed time step of $\Delta t = 5\,\text{ms}$ for state updates of $\Omega_e$ and $Q_c$ (matching the polling frequency of the real client handler) and a token-level granularity for generation events. Concurrency at the cloud is modelled via a sequence of in-flight requests, each consuming one "slot" of cloud capacity and contributing to $Q_c$. For each episode we additionally maintain $\kappa_t(b)$, the longest prefix materialised on backend $b$, and update it after every successful prefill or decode event.

## F.3 Network impairment model

Network conditions enter the simulator through three observable signals $(B, \mathrm{RTT}, \rho_{\text{loss}})$.

**Bandwidth.** We discretise bandwidth into five ordinal levels $\{1, 3, 5, 7, 9\}$ corresponding nominally to $\{2, 6, 10, 30, 60\}$ Mbps. Per-step transmission cost is computed as $T_{\text{net}}(t) = \text{payload}(t)/B$. The payload includes prefix text uploaded on cache miss, tool-call arguments/results, and generated responses returned to the client, each converted with the measured average bytes/token (3.6 for Qwen2.5 BPE).

**Time-varying behaviour.** Within an episode, $B$ follows a Markov chain over the five levels with transition probabilities calibrated to a real WAN trace (Appendix F.6). The expected dwell time at each level is approximately $\{2.1, 1.8, 5.6, 4.2, 3.0\}$ seconds. The trace was recorded over 12 hours of mixed-day traffic on a residential broadband connection.

**Burst spikes.** With probability $0.018$ per second, a burst event is triggered. During a burst (duration sampled from $\text{Lognormal}(\mu{=}-1.0, \sigma{=}0.4)$), $B$ is divided by a factor sampled from $\text{Uniform}(2, 5)$. This emulates the intermittent congestion observed in the real-deployment trace (§4.4).

**Loss and retransmission.** We model packet loss at a base rate $\rho_{\text{loss}} = 0.5\%$ that doubles during bursts. Retransmissions inflate effective transmission time by $1/(1 - \rho_{\text{loss}})$ in expectation, which we apply multiplicatively to $T_{\text{net}}$.

### F.4 Cloud queue dynamics

The cloud server processes requests in a continuous-batching scheduler. We model this as a discrete queue with effective batch capacity $Q_0 = 16$ (the value calibrated in Appendix D.3). When $|Q_c| \leq Q_0$ a new request is admitted immediately, otherwise it waits until a slot frees. Per-request inference time is the sum of prefill $W_{\text{prefill}}/\nu_{\mathcal{C}}$ and decode $W_{\text{gen}}/\mu_{\mathcal{C}}$, multiplied by $(1 + 0.18 \cdot |Q_c|/Q_0)$ to model batch-throughput degradation as concurrency rises. Background load (other tenants) is injected as exogenous arrivals at rate $\lambda_{\text{bg}}$, which we vary across training episodes to expose the policy to the full range of $Q_c$ values it encounters in deployment. Default training distribution: $\lambda_{\text{bg}} \sim \text{Uniform}(0.5, 8.0)$ requests/s.

### F.5 Edge KV-cache dynamics

The simulator tracks the per-edge KV-cache occupancy $\Omega_e \in [0, 1]$ as a function of active sequences and their token lengths, following the paged-attention bookkeeping of vLLM. When a sequence completes, its blocks are released. When a new sequence is admitted, blocks are allocated. If the backend cache lacks the current prefix, the simulator first charges a text-prefix upload (if remote) and prefill, it then updates $\kappa_t$ for that backend. Per-sequence block counts are computed from realised token counts using the Qwen2.5 model's KV dimensions (36 layers $\times$ 2 KV heads $\times$ 128 dims, 16-bit). The slowdown factor $\Psi(\Omega_e)$ is applied to both prefill and decode time of every sequence currently in flight.

### F.6 Calibration trace

We record a 12-hour mixed-traffic trace on a residential broadband connection during typical working hours. Bandwidth samples are collected at 200 ms granularity using `iperf3`, and RTT samples at 50 ms granularity using `ping`. The trace exhibits characteristic non-stationarity (median RTT 55 ms, P95 210 ms, with intermittent burst excursions to 800 ms), which we replay with random rotation across training episodes. Trace statistics are summarised in Table 30.

Table 30: Calibration trace statistics.

| Statistic | Value |
|---|---|
| Duration | 12.0 h |
| Mean bandwidth | 18.5 Mbps |
| P5 / P95 bandwidth | 2.4 / 48.2 Mbps |
| Median RTT | 55 ms |
| P95 RTT | 210 ms |
| Burst event rate | 0.018/s |
| Mean burst duration | 0.45 s |

# G Hyperparameters

This appendix consolidates all hyperparameters of RAOP and its training procedure. Hyperparameter values were selected on the validation split prior to test-time evaluation and then shared across tasks.

## G.1 Policy backbone and architecture

Table 31: Policy backbone and architecture.

| | |
|---|---|
| Backbone | Qwen2.5-3B-Instruct |
| LoRA rank $r$ | 16 |
| LoRA $\alpha$ | 32 |
| LoRA dropout | 0.05 |
| LoRA target modules | $\{q_{\text{proj}}, k_{\text{proj}}, v_{\text{proj}}, o_{\text{proj}}\}$ |
| Trainable parameters | $\sim$94 M (LoRA + heads) |
| Frozen parameters | $\sim$3.0 B |
| Policy head | 2-layer MLP, hidden 512, GELU |
| Workload head | 2-layer MLP, hidden 256, GELU, output 3 |
| State encoding | concat of $(q, \tau_{<t})$ tokens<br>with $(\Omega_e, B, \text{RTT}, \rho_{\text{loss}}, Q_c, \kappa_t)$ features<br>projected to embedding dim |

## G.2 GRPO training

Table 32: GRPO training hyperparameters.

| | |
|---|---|
| Optimiser | AdamW |
| Learning rate | $3 \times 10^{-4}$ |
| Weight decay | $10^{-2}$ |
| $(\beta_1, \beta_2)$ | $(0.9, 0.999)$ |
| Gradient clipping | 1.0 |
| LR schedule | cosine, 3% warmup |
| Group size $G$ | 8 |
| Clipping parameter $\epsilon$ | 0.2 |
| Entropy coefficient $\lambda_{\text{ent}}$ | $10^{-3}$ |
| Sampling temperature $\tau$ | $1.0 \to 0.7$ (linear, first 20% of training) |
| Reward clipping | $[-10, 10]$ before group normalisation |
| Batch size (prompts per update) | 64 |
| Effective trajectories per update | 512 ($= 64 \times G$) |
| Total optimisation steps | 50,000 |
| Total replayed trajectory samples | $\sim$25.6 M |
| Distinct online trajectories | 41.3k |
| Online 72B generations during GRPO | 0 |
| Workload-loss weight $\lambda_W$ | 0.1 |
| Workload loss | Huber on $\log(1 + W)$, $\delta = 1.0$ |

**Compute budget.** Training runs on $8\times$ A100 80 GB GPUs. Total wall-clock time per run is $\sim$ 38 hours. The 38-hour figure covers policy optimisation on replayed trajectories and simulator timing. It does not include online 72B generation for every GRPO rollout. Online model execution is paid once when building the trajectory library described in Appendix C. Memory peaks at $\sim 56$ GB per GPU, dominated by the replay buffer for $G = 8$ trajectories per prompt. We use `bf16` mixed precision and gradient checkpointing. The LoRA adapter keeps optimiser state size modest.

### G.3 Reward weights

The default operating point reported in the main paper is $(\omega_s, \omega_l) = (1.0, 1.0)$ where the success indicator is binary $\{0, 1\}$ and $L_{\text{total}}$ is the wall-clock latency in seconds. A sensitivity analysis is in Appendix I.

### G.4 Inference-time configuration

Table 33: Inference-time policy configuration.

| | |
|---|---|
| Sampling | Greedy decoding (argmax over actions) |
| Action masking | Yes (enforces feasibility: exactly one backend per operator, no $\varnothing$ for required ops) |
| Workload predictor | Forward pass on each step's state |
| Step latency budget | 50 ms (planner overhead), enforced by early termination if exceeded (rare, $< 0.1\%$ of steps) |

# H Statistical Analysis and Paired Comparisons

All main experiments use 10 independent training seeds $\{0, \ldots, 9\}$ and 2,000 held-out episodes per benchmark per method. Evaluation prompts and network-trace rotations are paired across methods, so method differences are estimated as within-prompt contrasts.

## H.1 Per-seed standard deviations on the main results

Table 34 reports the per-seed standard deviation of the system-average SR and P99 in Table 1 across the ten training seeds. The seed-level standard deviations are substantially smaller than the paired prompt-level differences in Table 2, supporting the use of paired bootstrap CIs rather than seed-level $t$-tests.

Table 34: Per-seed standard deviations for system-average SR and P99 in Table 1 over 10 training seeds.

| Method | Avg. SR | SR std | Avg. P99 | P99 std |
|---|---|---|---|---|
| Cloud-Only | 71.5 | 0.18 | 2.77 | 0.030 |
| MasRouter-step | 65.8 | 0.27 | 1.61 | 0.038 |
| GREEDY-ANALYTICAL | 66.7 | 0.29 | 1.91 | 0.045 |
| Value Router | 68.3 | 0.34 | 1.56 | 0.050 |
| ACTIVEINF-STEP | 68.1 | 0.36 | 1.57 | 0.052 |
| PPO-Step | 69.2 | 0.40 | 1.46 | 0.055 |
| RAOP | 71.7 | 0.32 | 1.31 | 0.043 |

## H.2 Paired bootstrap and permutation tests

For each comparison, we compute the prompt-level difference in success and resample prompts with replacement within each benchmark and seed. P99 CIs are computed by resampling episodes and trace rotations. The paired permutation $p$-values for SR are $p < 10^{-3}$ for RAOP over GREEDY-ANALYTICAL, $p < 0.01$ for RAOP over PPO-Step, $p < 10^{-3}$ for RAOP over ACTIVEINF-STEP, and $p < 10^{-3}$ for RAOP over MasRouter-step. We report CIs in Table 2 rather than relying on a small-$n$ seed-level $t$-test.

## H.3 Variance in the endogeneity analysis

The confidence bands in Figure 1 are computed by bootstrapping prompts within endogeneity bins and then bootstrapping seeds. The monotone trend is preserved within every seed, with Spearman correlation between episode-level $\sum_t \hat{\eta}$ and RAOP versus greedy SR gap of 0.71, 0.74, and 0.79 on GSM8K, HotpotQA, and WebShop respectively.

## H.4 WebShop with graded native reward

We also train and evaluate with the native graded WebShop reward $r \in [0, 1]$ in addition to the binary threshold $r \geq 0.8$.

Table 35: WebShop with binary success and native graded reward.

| Method | Binary ($r \geq 0.8$) | | Graded ($r \in [0, 1]$) | |
|---|---|---|---|---|
| | SR | P99 | Mean reward | P99 |
| GREEDY-ANALYTICAL | 58.4 | 1.52 | 0.692 | 1.55 |
| ACTIVEINF-STEP | 59.8 | 1.04 | 0.708 | 1.07 |
| PPO-Step | 61.1 | 0.92 | 0.721 | 0.95 |
| RAOP | 63.7 | 0.80 | 0.748 | 0.83 |

### H.5 Tail-statistic stability in deployment

With 2,000 deployment episodes per method, P99 is estimated from approximately 20 tail samples per method per trace rotation. We report P95, P99, and bootstrap CIs in Table 36.

Table 36: Tail-statistic stability for the real three-tier deployment.

| Method | Avg. P95 | Avg. P99 | Bootstrap 95% CI on P99 |
|---|---|---|---|
| Cloud-Only | 3.58 | 4.20 | [4.05,4.37] |
| Cascade | 4.22 | 5.03 | [4.82,5.24] |
| GREEDY-ANALYTICAL | 2.36 | 2.78 | [2.63,2.94] |
| ACTIVEINF-STEP | 2.10 | 2.42 | [2.27,2.58] |
| PPO-Step | 1.92 | 2.21 | [2.08,2.36] |
| RAOP | 1.51 | 1.78 | [1.69,1.88] |

# I Reward Sensitivity

Section 4.4 reports a compact reward-weight sensitivity result. This appendix provides the full sweep, traces the Pareto curve, and discusses the choice of operating point for different deployment scenarios.

## I.1 Full sweep

We sweep $(\omega_s, \omega_l)$ over a $5 \times 5$ grid covering the ranges $\omega_s \in \{0.5, 0.75, 1.0, 1.25, 1.5\}$ and $\omega_l \in \{0.25, 0.5, 1.0, 1.5, 2.0\}$, retraining a fresh policy for each setting (15 effective points after collapsing redundant ratios). Average SR and Avg. P99 across the three benchmarks are reported in Table 37.

Table 37: Reward weight sweep. Each entry is (Avg. SR / Avg. P99). The balanced configuration $(1.0, 1.0)$ used in the main paper is highlighted.

| $\omega_s \backslash \omega_l$ | 0.25 | 0.5 | 1.0 | 1.5 | 2.0 |
|---|---|---|---|---|---|
| 0.5 | 70.50/2.10 | 70.10/1.72 | 69.20/1.34 | 68.80/1.10 | 67.10/0.92 |
| 0.75 | 71.40/1.96 | 71.00/1.62 | 70.40/1.30 | 69.50/1.02 | 68.20/0.90 |
| 1.0 | 72.15/1.78 | 71.95/1.46 | **71.70/1.31** | 70.50/1.03 | 69.95/0.88 |
| 1.25 | 72.40/1.72 | 72.20/1.39 | 71.85/1.20 | 70.80/1.01 | 70.10/0.88 |
| 1.5 | 72.65/1.67 | 72.30/1.34 | 72.00/1.18 | 71.10/0.99 | 70.30/0.86 |

## I.2 Operating-point selection for different deployments

The choice of $(\omega_s, \omega_l)$ should reflect deployment priorities. We provide three reference points:

- **Latency-critical (interactive UX).** Pick $(\omega_s, \omega_l) = (1.0, 2.0)$. Avg. P99 $\approx 0.88$ s, Avg. SR $\approx 69.95\%$. Recommended when the application's UX SLA dominates accuracy concerns.
- **Balanced (default, used in the paper).** $(1.0, 1.0)$. The knee of the Pareto curve.
- **Accuracy-critical (offline batch / research deployment).** $(1.0, 0.5)$. Avg. SR $\approx 71.95\%$ at the cost of Avg. P99 $\approx 1.46$ s. Recommended when latency is not user-facing.

## I.3 Reward shaping discussion

Two alternative reward shapes were evaluated: (i) replacing $L_{\text{total}}$ with $\log(1 + L_{\text{total}})$ to flatten the latency penalty in the tail, and (ii) thresholding the success indicator at a partial-credit level (e.g., 0.5 for partial HotpotQA matches). Neither produced a meaningful improvement: (i) reduced Avg. P99 by $\sim 0.1$ s but at $-1.2$ SR points, and (ii) increased SR by $\sim 0.5$ points but introduced training instability. We retain the simple linear shape in the main paper.

# J Training Hyperparameter Sensitivity

This appendix sweeps the three GRPO/architecture hyperparameters the policy is most plausibly sensitive to: LoRA rank $r$, GRPO group size $G$, and entropy coefficient $\lambda_{\text{ent}}$. All other hyperparameters are held at their defaults (Appendix G). Each cell aggregates 10 seeds.

Table 38: Sensitivity of the policy to training hyperparameters. "Default" denotes the value used in the main paper.

| Hyperparameter | Value | Avg. SR | Avg. P99 (s) | vs. default |
|---|---|---|---|---|
| LoRA rank $r$ | 8 | 70.85 | 1.30 | $-0.85$ SR |
| | 16 (default) | 71.7 | 1.31 | n/a |
| | 32 | 71.80 | 1.23 | $+0.10$ SR |
| Group size $G$ | 4 | 70.40 | 1.35 | $-1.30$ SR |
| | 8 (default) | 71.7 | 1.31 | n/a |
| | 16 | 71.85 | 1.22 | $+0.15$ SR |
| $\lambda_{\text{ent}}$ | $10^{-4}$ | 70.95 | 1.28 | $-0.75$ SR |
| | $10^{-3}$ (default) | 71.7 | 1.31 | n/a |
| | $10^{-2}$ | 71.10 | 1.26 | $-0.60$ SR |

The sensitivity is mild: doubling or halving each of the three hyperparameters perturbs Avg. SR by less than 1.3 points, and all variants outperform the strongest non-learned baseline (GREEDY-ANALYTICAL, 66.7 SR). The chosen defaults sit on a plateau rather than at a sharp optimum.

# K Generalization Experiments

This appendix expands on the brief generalisation results in §4.4. We evaluate three forms of generalisation: across tasks, across model families, and across hardware tiers.

## K.1 Cross-task generalisation

We split the three benchmarks into pairs and train a policy on each pair, then evaluate on the held-out task. Workload predictors are not re-trained on the held-out task. Results are in Table 39.

Table 39: Cross-task generalisation. "In-dist" is the average performance on the training task pair. "OOD" is performance on the held-out task. Recovery $= \mathrm{SR}_{\mathrm{OOD}}/\mathrm{SR}_{\mathrm{target,full}}$, where $\mathrm{SR}_{\mathrm{target,full}}$ is the SR of the fully trained policy on the target task, taken from Table 1, this measures how much of the in-distribution performance on the target task is recovered when the target task is held out of training.

| Train→Test | In-dist | | OOD | | Recovery |
|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | |
| {GSM8K, HotpotQA} → WebShop | 75.7 | 1.56 | 59.5 | 0.88 | 93% |
| {GSM8K, WebShop} → HotpotQA | 71.8 | 1.31 | 64.5 | 1.40 | 90% |
| {HotpotQA, WebShop} → GSM8K | 67.6 | 1.05 | 73.5 | 1.95 | 92% |

The OOD recovery is consistently $\geq 90\%$. Inspection of the failure cases reveals that most degradation comes from the workload predictor mis-estimating the held-out task's typical token distribution, which inflates the regret-bound's first term. P99 is largely preserved because the policy still respects the cost model's structure even when its workload estimates are off, indicating cost-aware decisions under prediction uncertainty.

## K.2 Cross-model generalisation

We freeze the trained orchestration policy and substitute the underlying LLMs at every tier from the Qwen2.5 family to the Llama-3.1 family (8B at edge, 70B at cloud). The throughput coefficients $\mu_{\mathcal{E}}, \mu_{\mathcal{C}}$ are re-profiled but the policy itself is not re-trained. Results are in Table 40.

Table 40: Cross-model generalisation. The policy is trained on Qwen2.5 and tested on Llama-3.1 with no re-training.

| Setting | Qwen2.5 (in-dist) | | Llama-3.1 (OOD) | | Recovery |
|---|---|---|---|---|---|
| | SR | P99 | SR | P99 | |
| GSM8K | 79.9 | 1.82 | 73.0 | 1.95 | 91% |
| HotpotQA | 71.5 | 1.30 | 63.5 | 1.46 | 89% |
| WebShop | 63.7 | 0.80 | 56.0 | 0.86 | 88% |
| Avg. | 71.7 | 1.31 | 64.2 | 1.42 | 90% |

The policy retains $\geq 88\%$ of in-distribution SR across all three tasks under model substitution. The P99 increases slightly on average (1.42 vs. 1.31 s), reflecting Llama-3.1's slightly different throughput profile at comparable size, but remains well below all per-task baselines in Table 1. This indicates that RAOP's decisions are predominantly driven by the structure of the agent trajectory and the system state, not by idiosyncratic features of the training-time LLM.

## K.3 Cross-hardware generalisation

The real-deployment results in §4.4 already exercise cross-hardware generalisation: training is in simulation but evaluation is on a Raspberry Pi 5, a desktop GPU, and a cloud server. The retained SR (71.5 vs. 71.7 in simulation) and the moderate P99 inflation (1.78 vs. 1.31 s) demonstrate end-to-end hardware-stack transfer without on-target retraining. We attribute the residual gap primarily to (i) bandwidth jitter that exceeds the training-trace tail, and (ii) Ollama's quantised inference stack on

the Pi having a slightly different throughput-vs-batch curve than the linear model assumed in the simulator.

### K.4 Generalisation boundaries

The following two failure modes delimit the observed generalisation range:

- **Drastic backbone capacity changes.** Replacing the cloud backend with a model two orders of magnitude smaller (e.g., Qwen2.5-7B at the cloud tier) breaks the policy's learned trade-off, because the training distribution of $(\mu_{\mathcal{E}}, \mu_{\mathcal{C}})$ no longer covers the deployment ratio. SR drops to $\sim 60\%$ in this regime. A short fine-tune of 2,000 steps with a small mix of target-hardware rollouts recovers $\geq 96\%$ of original performance.
- **Non-Markovian network dynamics.** The training simulator's first-order Markov bandwidth model does not capture diurnal or weekly patterns. On a deployment that exhibits strong daily patterns, the policy's decisions are still locally good but the long-term expected-cost calibration drifts. This is best addressed at the systems layer (e.g., per-time-of-day calibration of $\tau_{\text{queue}}$ and $B$) rather than at the policy layer.

These limitations are consistent with the assumption boundaries of our formulation in §3.1, in which $P_{\text{sys}}$ is modelled as Markov and the throughput parameters $(\mu_{\mathcal{E}}, \mu_{\mathcal{C}})$ are calibrated once. They delimit deployment regimes that require recalibration or adaptation at production scale.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main scope of the paper: formalizing agent serving as an endogenous, sequentially revealed workload, introducing RAOP as a step-level edge–cloud orchestration policy, and evaluating full-trajectory success, tail latency, and communication. The method is formalized in Section 3, the main controlled and deployment results are reported in Section 4, and the conclusion states the current scope and future extensions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion notes that the current evaluation focuses on text-based agents and a bounded set of deployment tiers, and identifies richer multimodal tools, privacy-aware rewards, and multi-tenant fairness as future directions. Appendix K further discusses generalization boundaries, including drastic backbone-capacity changes and non-Markovian network dynamics that may require recalibration or adaptation.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Section 3 states the formal ESRW controlled-trajectory MDP and the theoretical claims, including containment, strict expressivity, and the Bellman-regret bound for myopic ESRW policies. Appendix B provides the formal action-interface definitions, the proofs of Propositions 3.1, 3.2, and 3.3, and a discussion of the assumptions behind the bounded-value and calibrated-latency conditions.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4.1 specifies the benchmarks, model tiers, evaluation protocol, baselines, metrics, seed protocol, paired evaluation, and replay audit. Appendices C, D, E, F, G, and H provide the trajectory-library construction, replay audit, cost-model calibration, baseline implementation details, simulator design, training hyperparameters, and statistical procedures.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: At submission time, the paper does not provide an anonymized public code release or a downloadable replay library with exact reproduction scripts. The paper uses public benchmarks and publicly described model families, and it provides detailed methodological, simulation, calibration, and hyperparameter information to support reproduction.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setup specifies GSM8K, HotpotQA, and WebShop; the edge, cloud, and three-tier model configurations; the controlled replay protocol; the deployment setup; all major baselines; the metrics; the number of training seeds and held-out episodes; and the paired bootstrap and permutation testing procedures. Appendices E through H give implementation, simulator, hyperparameter, and variance details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports paired bootstrap confidence intervals for success rate and P99 latency, and permutation tests for key success-rate comparisons. Section 4.1 describes the paired prompt and network-trace seed protocol, Table 2 reports the main paired confidence intervals, and Appendix H provides additional per-seed standard deviations, bootstrap details, permutation tests, and tail-statistic stability.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the model sizes and serving stacks used in the two-tier simulation and the three-tier deployment. Appendix G reports the policy-training compute budget, including 8 A100 80 GB GPUs, approximately 38 hours per run, peak memory, precision, optimizer state considerations, and the distinction between online trajectory collection and replayed GRPO optimization.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The work evaluates resource orchestration for LLM agents using previously released benchmarks, publicly described model families, simulator calibration, and deployment measurements. It does not introduce human-subject experiments or unsafe data collection, and it discusses relevant deployment concerns, including privacy-aware rewards, multi-tenant fairness, and system recalibration boundaries.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The work can improve practical LLM-agent serving by reducing tail latency, communication, and cloud contention, which may make agent systems more usable and less resource-intensive. Potential negative impacts include privacy risks from moving prompts, tool outputs, and cache state across tiers; unequal service quality under multi-tenant contention; and over-optimization for latency at the expense of safety or fairness.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a new high-risk pretrained language model, image generator, scraped dataset, or dual-use model artifact. RAOP is an orchestration method evaluated with existing models, benchmarks, and deployment traces.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [No]

Justification: The paper cites the benchmarks, model families, serving systems, and related methods used in the experiments, including GSM8K, HotpotQA, WebShop, Qwen2.5, Llama, vLLM, and prior baselines. However, the current manuscript does not explicitly enumerate the licenses or terms of use for each existing dataset, model, and software asset. Adding an asset-and-license table would allow this answer to be changed to [Yes].

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces an action-conditioned trajectory library for controlled replay evaluation. Appendix C documents how the library is constructed, what is logged at each state, how coverage is measured, how counterfactual placements are included, and how replay fidelity is audited. The paper also reports library size, step-continuation counts, counterfactual records, and workload-prediction audits.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not report crowdsourcing, newly recruited participants, or human-subject experiments. Evaluation uses benchmark tasks, model and tool execution, replay audits, simulator calibration, and deployment measurements.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The reported work does not involve newly recruited human subjects, crowdsourcing workers, or participant studies. It uses existing benchmark prompts, real model and tool execution, replay traces, and deployment measurements, so IRB approval is not applicable to the reported experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

Justification: LLMs are central to the method and evaluation. The paper specifies the LLM-agent setting, the Qwen2.5-based edge and cloud models, the Qwen2.5-3B policy encoder, the three-tier deployment models, the real LLM continuations stored in the trajectory library, and the use of vLLM workers in simulation calibration and deployment.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.