ERROR-PROPAGATION-AWARE ROUTING WITH ADAPTIVE HORIZONS FOR MULTI-MODEL LLM AGENT DAG EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents increasingly execute multi-step workflows whose subtasks vary in difficulty and are connected by complex dependencies. Recent LLM-routing approaches reduce cloud cost by selecting models at the subtask level. However, none of these approaches jointly considers error propagation across dependent subtasks, dynamically controls how long routing decisions remain in effect, and concurrently executes runnable subtasks, which form a *frontier*. We present PARA, an Error-Propagation-Aware Routing framework with Adaptive horizons for multi-model LLM agents. PARA represents each request as a directed acyclic graph (DAG) of subtasks and concurrently executes each frontier. At each routing point, PARA jointly determines a model allocation scheme and an adaptive commitment horizon—the number of consecutive frontier waves for which the routing decision remains in effect before being reconsidered. This adaptive horizon balances routing overhead against the need to respond to evolving execution states and error propagation. Across seven benchmarks, PARA reduces cloud cost by 68.50% and end-to-end latency by 43.15% relative to all-cloud execution while remaining within 1.71–6.75 points of its task quality. Compared with the state-of-the-art methods, PARA improves average score by up to 15.02 points while reducing average latency and cloud cost by up to 46.06% and 41.51%, respectively.

1 INTRODUCTION

Recent advances in large language models (LLMs) have enabled autonomous AI agents, such as Auto-GPT (Significant-Gravitas, 2023), AutoGen (Wu et al., 2024), LangChain (LangChain, 2023), and BabyAGI (BabyAGI, 2023). These agents decompose high-level goals into workflows that combine reasoning, tool use, and final response generation (Yao et al., 2023; Wu et al., 2024). LLM applications span scientific analysis (Xie et al., 2024), medical question answering (Qian et al., 2024), and products such as BioNeMo (NVIDIA, 2024), IBM watsonx Assistant (IBM, 2024), and Grammarly (Grammarly, 2024). Cloud cost accumulates across the model calls required to complete each request, with token charges determined by the models used (OpenAI, 2023). Prior work reduces this cost by routing or cascading requests across models using predicted difficulty or confidence (Chen et al., 2023; Ding et al., 2023; Šakota et al., 2024; Aggarwal et al., 2024; Yue et al., 2024; Ramírez et al., 2024; Narasimhan et al., 2025). These approaches primarily address individual requests, whereas agent workflows contain dependent subtasks. We represent such a workflow as a DAG, where nodes are subtasks and edges specify dependencies. For example, comparing two medical claims may require independent evidence-gathering subtasks followed by a synthesis step that depends on both branches. Executing independent branches in parallel can reduce latency while preserving their dependencies. The executor pool may contain models of different sizes.

Extending hybrid routing from single-shot prompts to agent workflows creates two coupled challenges. First, subtask difficulty is non-uniform: a local model may be sufficient for some subtasks but fail on reasoning or synthesis subtasks. Second, an error at one subtask changes the inputs observed by its descendants; downstream errors may be recoverable by a stronger model or may become irrecoverable after several dependent steps. Thus, routing decisions should account for both the current subtask and the risk of downstream error propagation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Prior work (Shao et al., 2025; Narayan et al., 2025; Zhang et al., 2025; Ding et al., 2023; Su et al., 2025) only partially addresses this setting. Division of Thoughts (DoT) (Shao et al., 2025) decomposes requests into dependency graphs and routes individual subtasks between local and cloud models based on estimated subtask difficulty, while Minions (Narayan et al., 2025) uses a remote model to decompose a task into simpler subtasks over shorter document chunks, which are then executed in parallel by a local model. Router-R1 (Zhang et al., 2025) learns multi-round model routing and aggregation through sequential interactions. HybridLLM (Ding et al., 2023) performs request-level routing between local and cloud models, and ToolOrchestra (Su et al., 2025) uses reinforcement learning (RL) to dynamically orchestrate heterogeneous models. However, none of these approaches jointly considers error propagation across dependent subtasks, dynamically controls how long routing decisions remain in effect, and concurrently executes runnable subtasks. Our measurement study yields two insights: **Insight 1:** Mismatch rates differ across benchmarks but remain relatively stable across the reported frontier-wave groups; and **Insight 2:** Node executions exhibit both intrinsic model error and sensitivity to noisy parent inputs, with higher rates of irrecoverable errors among nodes at dependency depth at least three.

These insights motivate the design of PARA, an Error-Propagation-Aware Routing framework with Adaptive Horizons for multi-model LLM agent DAG execution. It jointly routes ready-to-execute subtasks to models and dynamically determines when to re-route, enabling cost-effective agent execution. As shown in Figure 1, an LLM-based planner produces a task DAG, the scheduler identifies the current frontier, and an LLM-based router selects both (i) a *model allocation scheme* and (ii) a *commitment horizon*, specifying how many frontier waves reuse that allocation before the next routing decision. The horizon controls both routing and replanning frequency. We train the router using RL to improve task score while reducing cloud cost. The selected candidate models execute subtasks within each frontier wave in parallel, up to the selected commitment horizon. At each horizon boundary, the planner regenerates the unexecuted DAG from cached outputs, and the router selects the next horizon and allocation.

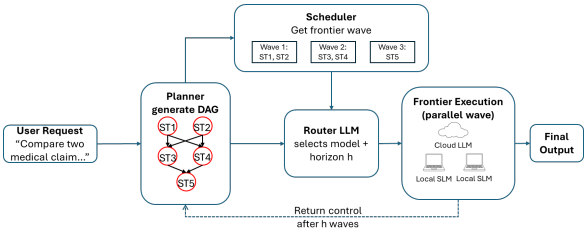

Figure 1: Dataflow of PARA.

Across seven benchmarks, PARA reduces cloud cost by 68.50% and end-to-end latency by 43.15% relative to all-cloud execution while remaining within 1.71–6.75 points of its task quality. Compared with the state-of-the-art methods (HybridLLM, Router-R1, and Minions), PARA improves average score by up to 15.02 points while reducing average latency and cloud cost by up to 46.06% and 41.51%, respectively. Our contributions are:

Error-propagation characterization. We characterize intrinsic node hardness and recoverable and irrecoverable propagation errors, showing how dependency context and commitment length shape effective model allocation in multi-step agent execution.

PARA framework. We introduce PARA, which jointly learns propagation-aware model allocation and adaptive commitment horizons, combining selective cloud execution with concurrent frontier processing to improve task score, latency, and cloud cost.

End-to-end evaluation. We evaluate PARA against state-of-the-art methods across seven benchmarks and heterogeneous model pools, complemented by ablation studies.

2 Measurement and Design Foundations

2.1 Measurement Setup

We implement profiling with `AutoGen` (Wu et al., 2024) and run all profiling experiments on a local device with an RTX 5090 GPU. We study two local–cloud model pairs: Qwen3-4B (local)/GPT-5 (cloud) and Gemma 3-4B (local)/Claude Sonnet 4.5 (cloud). Local small language models (SLMs) run via `llama.cpp`, while cloud LLMs are accessed through public APIs. Unless noted otherwise, we average results across both model pairs to avoid conclusions tied to a single model. We evaluate four benchmarks spanning math reasoning (GSM8K (Cobbe et al., 2021) (exact-match accuracy)),

multi-hop QA (HotpotQA (Yang et al., 2018) (F1)), discrete numerical reasoning (DROP (Dua et al., 2019) (F1)), and code generation (HumanEval (Chen et al., 2021) (pass@1)).

2.2 Mismatch Rates Across Benchmarks and Frontier Waves

To profile routing behavior in AI agents, we construct a subtask DAG for each query by prompting GPT-5 to (i) decompose the query into atomic subtasks plus a final synthesis node, and (ii) output directed precedence edges. We characterize subtask difficulty through local–cloud model disagreement by running every DAG node once with the local SLM and once with the cloud LLM. A node is considered a *mismatch* if its SLM output is not equivalent to its LLM output. Equivalence is determined by deterministic checks when available (normalized numeric match in math problems; unit tests for HumanEval) and otherwise by a GPT-5 equivalence judge (*same*/*different*). The *mismatch rate* is the fraction of nodes judged different.

Table 1: Mismatch rate by frontier-wave index.

Benchmark	**Wave 1**	**Wave 2–3**	**Wave 4+**
GSM8K	8.91%	10.12%	9.47%
HotpotQA	25.84%	28.73%	27.19%
DROP	29.36%	32.54%	31.11%
HumanEval	34.27%	37.02%	36.48%

Table 1 reports mismatch rates by frontier-wave group. Rates vary more across benchmarks than across the reported groups within a benchmark: GSM8K ranges from 8.91% to 10.12%, HotpotQA from 25.84% to 28.73%, DROP from 29.36% to 32.54%, and HumanEval from 34.27% to 37.02%. The persistent mismatch across frontier waves makes model allocation relevant throughout execution. A request-level decision cannot adapt to the changing mix of runnable subtasks and their intermediate inputs. PARA makes this adaptation efficient by sharing model preferences across nodes with the same task type and revisiting them at learned commitment boundaries. Section 4.4 shows that this design retains the score of per-node routing within 0.63 points while reducing latency and cloud cost.

Insight 1

• *Mismatch rates differ across benchmarks but remain relatively stable across the reported frontier-wave groups within each benchmark.*

2.3 Causes of Routing Failure

We study two sources of routing failure: intrinsic model limitations and sensitivity to parent outputs. For each query, we construct two traces over the same DAG: (i) a *reference trace*, in which the cloud LLM executes all nodes and we record their outputs, and (ii) a *noisy-parent trace*, in which the local SLM executes the DAG end to end. Both traces use the same node instructions and prompt template; the parent outputs inserted into the prompts differ. We compare node executions under these two input conditions without altering the DAG.

Table 2: Intrinsic model error and sensitivity to noisy parent inputs. All entries are percentages of nodes.

Dataset	**Clean inputs**	**Noisy inputs**			
	ME ↓	RB ↑	RC	IRC	IRC@Depth≥3
GSM8K	9.2%	86.3%	9.8%	3.9%	6.2%
HotpotQA	18.4%	56.4%	24.1%	19.5%	31.8%
DROP	20.1%	52.1%	25.6%	22.3%	35.9%
HumanEval	28.5%	58.3%	18.4%	23.3%	33.1%

We measure *intrinsic node hardness* by rerunning a node with the SLM and reference parent outputs. Disagreement with the node's reference output, evaluated as in Section 2.2, defines *intrinsic model error* (ME). To assess sensitivity to noisy parent inputs, we execute each node with both models using parent outputs from the local trace. This yields three outcomes: *Robust (RB)*, where the SLM matches the reference; *Recoverable (RC)*, where the SLM differs but the LLM matches; and *Irrecoverable (IRC)*, where both differ. These outcomes distinguish nodes that benefit from model escalation from nodes that require correction of upstream context, directly motivating propagation-aware routing and replanning.

Table 2 reports ME under reference inputs and node outcomes under noisy inputs. *IRC@Depth≥3* denotes the IRC rate among nodes at dependency depth ≥ 3, measured in frontier waves from the

source nodes. Intrinsic ME is non-trivial across all datasets, ranging from 9.2% on GSM8K to 28.5% on HumanEval, identifying difficult nodes even when their parent outputs come from the cloud reference. Under noisy parents, propagation sensitivity varies substantially across workloads. GSM8K is relatively robust, with 86.3% RB and only 3.9% IRC, whereas HotpotQA, DROP, and HumanEval have much lower RB rates (52.1–58.3%) and substantially higher IRC rates (19.5–23.3%). Across all four benchmarks, nodes at dependency depth at least three have higher IRC rates than the overall node population: IRC@Depth$\geq$3 reaches 6.2% on GSM8K and 31.8–35.9% on the other three datasets. Thus, long fixed horizons can allow errors to propagate into irrecoverable downstream regions, whereas a fixed short horizon of h=1 repeatedly stalls execution after every wave for a new routing decision, incurring additional routing and synchronization overhead. This motivates adapting the horizon to propagation risk: use SLMs in robust regions, escalate recoverable nodes, and shorten commitments before irrecoverable errors propagate.

Insight 2

• *Node executions exhibit intrinsic model error and sensitivity to noisy parent inputs. Under noisy inputs, executions can be robust, recoverable, or irrecoverable; nodes at dependency depth at least three have higher IRC rates than the overall node population.*

2.4 From Insights to Design

Insight 2 motivates a router that accounts for both the current node and upstream execution state. The RL policy learns from end-to-end outcomes using node characteristics, dependency context, and completed outputs. This allows routing decisions to reflect the effect of earlier execution on later subtasks while balancing task score and cloud cost.

Selecting a separate model for every node creates an action space that changes with DAG size. Motivated by the variation observed in Insight 1, PARA instead shares model preferences among nodes with the same task type. This keeps the action dimension independent of the number of nodes. Section 4.4 compares task-type and per-node routing in score, latency, and cloud cost.

The measurements also motivate an adaptive commitment horizon. Table 2 shows higher IRC rates among nodes at dependency depth at least three, motivating reconsideration of routing as execution progresses. Task-type preferences can be reused across frontier waves because the task-tag vocabulary remains fixed even as runnable nodes change. The router therefore jointly selects these preferences and a commitment horizon h, specifying how many consecutive frontier waves reuse them before routing and the remaining DAG are reconsidered.

3 PARA Framework

PARA orchestrates a heterogeneous model pool for DAG-structured agent execution. As shown in Figure 2, PARA executes the following steps:

- *Task decomposition.* The planner represents each request as a subtask dependency DAG with task tags and explicit parent lists.
- *Orchestration core.* At each commitment boundary, an RL-based router outputs (i) a commitment horizon measured in waves and (ii) a task-conditioned routing preference that is held fixed throughout the commitment horizon. We use six task types, $\mathcal{T} = \{\textit{Retrieve}, \textit{Transform}, \textit{Math}, \textit{Code}, \textit{Reason}, \textit{Synthesize}\}$.
- *Multi-wave execution commitment loop.* Within the commitment horizon, the scheduler routes each runnable node to its preferred model from the *executor pool* and executes each frontier wave in parallel.

PARA then updates the remaining DAG at the horizon boundary and repeats the above process.

3.1 Task Decomposition

Given a user request q, the planner generates a directed acyclic graph $G = (V, E)$. Each node $v \in V$ contains a unique ID, a subtask instruction (the natural-language specification of the work performed by that node), a task type tag $\tau(v) \in \mathcal{T}$, and an ordered parent list $\textit{Parents}(v)$. Each directed edge $(u \to v) \in E$ indicates that the output of u becomes the input of v; the planner

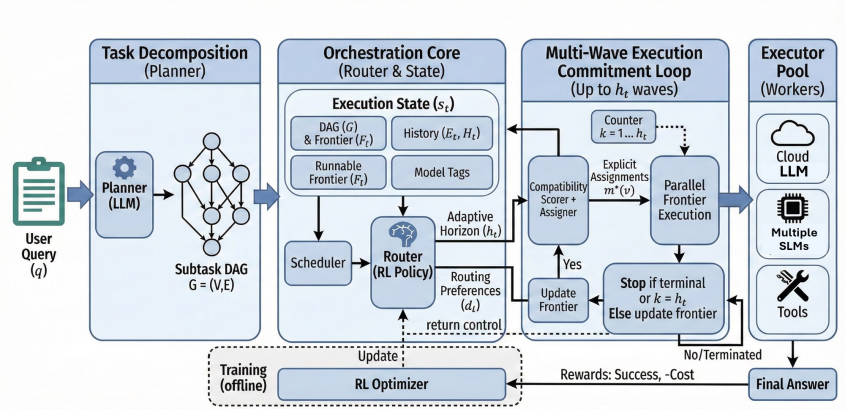

Figure 2: Overview of PARA. At each commitment horizon boundary, the planner updates the remaining DAG using cached outputs, and the router selects the next horizon and model preferences.

emits edges only through explicit parent lists. We use a small model as the planner, specifically Qwen3-4B, and train it via supervised fine-tuning (SFT) to decompose each request q into a subtask DAG. In addition to the DAG structure, the planner assigns each node a task tag $\tau(v) \in \mathcal{T}$. To construct the task-tag training data, we prompt GPT-5 to label every node in planner-produced DAGs based on its instruction and output. For example, an output containing retrieved passages is labeled RETRIEVE, whereas "compare the two evidence sets" is labeled REASON. These labels supervise the planner, which predicts one task type per node when generating a new DAG. Explicit parent lists give the scheduler a concrete representation of when each subtask becomes executable. A missing dependency can expose a node before its required context is available, making graph structure an important control point alongside model selection. Our edge-repair evaluation isolates this effect by adding the missing dependency at a matched execution snapshot and resuming from cached outputs. Appendix A reports the per-benchmark missing-dependency and edge-repair comparison.

3.2 ORCHESTRATION CORE: ROUTER & STATE

Let $\mathcal{E}$ be the set of completed node IDs with cached outputs. For the current graph G_t, the runnable frontier is $F = \{v \in V(G_t) \setminus \mathcal{E} : \text{Parents}_{G_t}(v) \subseteq \mathcal{E}\}$. The scheduler launches the nodes in F in parallel, caches their outputs, and updates $\mathcal{E} \leftarrow \mathcal{E} \cup F$ before computing the next frontier. The index t denotes a routing decision, which can commit to multiple frontier waves.

At each routing decision time t, the router observes state s_t: the request, DAG topology (IDs, tags, and parent lists encoding the edges), current frontier, summaries of completed outputs, model executor profiles $\phi(m)$ and token prices $(c_{\text{in}}(m), c_{\text{out}}(m))$. For example, node `v3` may be a REASON node whose parents are `v1` and `v2`. The router produces an internal reasoning trace and a machine-readable action $a_t = (h_t, P_t)$. Only the structured action is parsed by the execution system; the reasoning text is used by the policy to form the decision but does not directly control execution.

Here $h_t \in \{1, \dots, H_{\max}\}$ is the number of frontier waves to execute before the next routing decision, and $P_t \in \mathbb{R}^{|\mathcal{T}| \times |\mathcal{M}|}$ scores each model in the executor pool $\mathcal{M}$ for each task type. We use task-conditioned model preferences rather than per-node model assignments to keep the action dimension independent of DAG size and to share routing behavior across semantically similar subtasks. For each runnable node v, let $\tau(v)$ and $\ell(v)$ denote its task type and prompt length. Among models that can accommodate $\ell(v)$, PARA selects the highest-scoring model in P_t:

$$m^*(v) = \arg\max_{m \in \mathcal{M}_v} P_t[\tau(v), m], \qquad \mathcal{M}_v = \{m : \ell(v) \le L_{\max}(m)\}. \tag{1}$$

PARA executes up to h_t frontier waves without invoking the router again. At the next commitment boundary, the planner regenerates the remaining DAG using cached outputs; completed node IDs and outputs are preserved. Planner calls are included in end-to-end latency.

For example, two independent RETRIEVE nodes can feed a REASON node. Retrieval runs concurrently; once both outputs are cached, the reasoning node becomes runnable. A commitment spanning these waves reuses one routing decision while applying different task-type preferences. At its boundary, cached outputs inform the next DAG and model allocation.

3.3 ROUTER TRAINING

We initialize the Qwen3-4B router by supervised fine-tuning on GPT-5-generated state–action pairs. We then apply Group Relative Policy Optimization (GRPO) (Shao et al., 2024). For each training request, GRPO samples a group of complete execution trajectories. Each trajectory includes horizon choices, model allocations, and boundary replanning until termination. Group-normalized returns provide the advantage signal for the policy update; a Kullback-Leibler divergence penalty regularizes the policy relative to a frozen reference.

The reward combines task score and measured cloud cost:

$$R = \begin{cases} r_{\text{succ}} - \alpha \sum_t \Delta\text{Cost}_t, & \text{valid trajectory,} \\ -1, & \text{invalid router action.} \end{cases} \tag{2}$$

Here r_{succ} is the terminal task-quality reward and ΔCost_t is the cloud cost accrued during commitment t. An invalid router action terminates the episode. We select α on held-out validation requests and use $\alpha = 0.7$ in the main experiments. The terminal reward credits complete trajectories, teaching the router to balance final task score against cumulative cloud cost. We evaluate the latency benefits of adaptive commitments and frontier parallelism end to end.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP AND BASELINES

Inference experiments run in an AutoGen agent on a single RTX 5090: local models execute on-device and the cloud model is accessed via its public API. We evaluate PARA in two executor settings: **Pool** setting and **Single** setting. In the Pool setting, the router selects from four local SLMs (Qwen3-0.6B, Gemma 3-4B, Qwen3-8B, and gpt-oss-20B) as well as one cloud-based LLM (GPT-5). In the Single setting, routing is restricted to Qwen3-8B and GPT-5. We evaluate seven benchmarks spanning multi-hop QA, math and discrete numerical reasoning, code generation, and tool use: HotpotQA (Yang et al., 2018) (F1), GSM8K (Cobbe et al., 2021) (exact match), DROP (Dua et al., 2019) (F1), HumanEval (Chen et al., 2021) (pass@1), MATH (Hendrycks et al., 2021b) (exact match), ToolBench (Qin et al., 2023) (task success rate under its evaluation protocol), and FRAMES (Krishna et al., 2025) (answer accuracy). We use official evaluation scripts/verifiers from each benchmark when available.

We report (i) *task score* via the benchmark metric/verifier on the final response, (ii) *end-to-end latency* as wall-clock time from the start of request execution to terminal completion, including planner/router execution, scheduling and synchronization, and cloud API request/response time, and (iii) *cloud cost* as total token charges for all cloud API calls. We report unweighted means across benchmarks and compute percentage reductions from these means. Cloud cost is normalized separately for each benchmark to All-LLM $= 100$; calculations use unrounded values before display.

To train the Qwen3-4B planner, we collect training pairs $(q, G^\star)$, where $G^\star$ is generated by GPT-5 using a constrained decomposition prompt. The planner is then fine-tuned to imitate $G^\star$ using token-level cross-entropy loss. To construct the task-tag training data, we prompt GPT-5 to label every node in planner-produced DAGs for a 1,000-request trace sample drawn from HotpotQA, GSM8K, DROP, and HumanEval. The router is initialized from Qwen3-4B and fine-tuned on GPT-5-generated routing traces before GRPO training. We collect supervised (s_t, a_t) pairs from 16K requests: 8K HotpotQA (50%), 4K GSM8K (25%), and 4K APPS (25%) (Hendrycks et al., 2021a). Starting from this checkpoint, the router is optimized with GRPO over complete request trajectories using veRL (Sheng et al., 2025). Training uses one NVIDIA A100 80GB GPU; the RTX 5090 is used only for inference.

We compare PARA against All-SLM, All-LLM, Random, HybridLLM, Router-R1, Minions, GPT-5 ReAct, and GPT-5 Direct. All-SLM and All-LLM execute every subtask with the local SLM and cloud LLM, respectively. Random assigns each node to the cloud LLM with probability 0.5. HybridLLM (Ding et al., 2023) routes each request to a model; Router-R1 (Zhang et al., 2025) interleaves reasoning and sequential model calls over the full model pool. Minions (Narayan et al., 2025) uses a cloud model to decompose work into subtasks that local models execute in parallel. GPT-5 ReAct (Yao et al., 2023) alternates reasoning, tool calls, and observations through the shared

Table 3: End-to-end evaluation and seven-benchmark means. C_t is latency in seconds; C_a is cloud cost normalized to All-LLM = 100.

Method	HotpotQA			GSM8K			DROP			HumanEval		
	Score	C_t	C_a	Score	C_t	C_a	Score	C_t	C_a	Score	C_t	C_a
All-SLM	58.27	15.38	0.00	83.61	4.46	0.00	72.48	7.25	0.00	54.88	9.41	0.00
Random	60.12	21.08	51.64	82.44	8.13	48.18	73.26	11.92	47.42	66.46	13.98	54.91
HybridLLM (original)	65.65	23.91	46.55	86.26	9.86	30.12	77.91	13.26	39.44	69.51	15.74	52.88
Router-R1 (original)	71.09	31.72	52.10	91.07	12.44	34.89	84.02	16.71	44.72	76.83	20.83	58.26
Minions	69.82	39.14	28.04	90.35	10.77	34.19	82.11	18.53	27.15	71.34	24.56	41.36
PARA (single)	75.84	17.44	34.40	95.62	7.89	18.24	87.98	9.26	28.64	82.32	11.35	37.21
PARA (pool)	76.92	18.67	31.27	96.13	6.51	16.58	89.04	9.84	26.04	83.54	11.79	33.83
All-LLM	78.63	32.46	100.00	97.84	18.92	100.00	91.72	19.37	100.00	87.20	19.62	100.00
HybridLLM (matched)	71.85	20.32	41.90	91.69	8.38	27.11	84.03	11.27	35.50	76.83	13.38	47.59
Router-R1 (matched)	74.09	30.13	49.50	92.07	11.82	33.15	86.02	15.87	42.48	80.49	19.79	55.35
GPT-5 Direct	83.97	14.61	37.52	98.38	9.46	19.90	93.79	9.69	31.25	90.85	10.79	40.60
GPT-5 ReAct	80.77	41.80	104.69	98.06	22.18	102.49	92.55	24.29	103.91	88.41	25.52	105.07

Method	MATH			ToolBench			FRAMES			**Average**		
	Score	C_t	C_a	Score	C_t	C_a	Score	C_t	C_a	Score	C_t	C_a
All-SLM	47.32	10.11	0.00	36.89	24.61	0.00	25.00	48.85	0.00	**54.06**	**17.15**	**0.00**
Random	62.17	15.78	46.26	48.34	30.37	52.77	37.38	69.46	65.26	**61.45**	**24.39**	**52.35**
HybridLLM (original)	65.48	18.52	47.15	52.09	33.48	61.40	44.90	78.73	60.13	**65.97**	**27.64**	**48.24**
Router-R1 (original)	73.26	24.20	52.73	60.22	41.62	67.18	57.40	101.19	67.14	**73.41**	**35.53**	**53.86**
Minions	66.93	19.33	39.83	53.74	46.18	53.02	50.36	108.73	48.44	**69.24**	**38.18**	**38.86**
PARA (single)	80.96	13.88	35.09	67.83	26.21	45.78	68.08	59.01	43.19	**79.80**	**20.72**	**34.65**
PARA (pool)	82.18	13.27	31.90	69.11	25.42	41.62	70.02	58.65	39.27	**80.99**	**20.59**	**31.50**
All-LLM	88.93	26.84	100.00	73.58	38.95	100.00	75.97	97.38	100.00	**84.84**	**36.22**	**100.00**
HybridLLM (matched)	74.67	15.74	42.44	61.45	28.46	55.26	54.00	66.92	54.12	**73.50**	**23.50**	**43.42**
Router-R1 (matched)	77.26	22.99	50.09	64.22	39.54	63.82	71.00	96.13	63.78	**77.88**	**33.75**	**51.17**
GPT-5 Direct	91.70	17.45	38.28	74.92	45.31	103.12	66.02	51.50	33.51	**85.66**	**22.69**	**43.45**
GPT-5 ReAct	90.04	33.48	104.79	76.22	51.66	106.24	74.03	124.04	104.53	**85.73**	**46.14**	**104.53**

tool interface. GPT-5 Direct makes one GPT-5 call without DAG planning or model routing, with tools available through the shared interface.

PARA (single) uses task-type routing and adaptive horizons with only Qwen3-8B and GPT-5. Per-node PARA instead selects a model separately for each node while retaining the planner, model pool, adaptive horizons, and scheduler. Results appear in Section 4.4.

The *original* baselines retain their earlier implementations; *matched* variants share the same 16K SFT requests, GRPO rollout budget, executor pool, and validation-based operating-point selection where applicable. DAG-based comparisons share planner-produced task instances.

4.2 Accuracy, Cost, and Latency

Table 3 compares all seven benchmarks, with PARA (pool) as the primary configuration. It averages 80.99 score, 20.59 s latency, and 31.50 normalized cloud cost. Relative to All-LLM, PARA reduces latency by 43.15% and cloud cost by 68.50% while remaining within 1.71–6.75 points of its per-benchmark task scores. PARA recovers 87.50% of the difference between the All-SLM and All-LLM macro-average scores. Mean score gains over original HybridLLM, original Router-R1, and Minions are 15.02, 7.58, and 11.76 points, respectively, with latency and cloud cost reductions of up to 46.06% and 41.51%. Against matched HybridLLM and Router-R1, score gains remain 7.49 and 3.11 points, with lower mean latency and cloud cost. Random scores 61.45 at 52.35 normalized cloud cost.

PARA (single) averages 79.80 score, 20.72 s latency, and 34.65 normalized cloud cost. PARA (pool) improves score by 1.19 points and reduces normalized cloud cost by 3.15 points, with similar mean latency. Score and cloud cost improve on every benchmark. Compared with sequential GPT-5 ReAct, PARA (pool) delivers its 80.99 mean score using only 44.63% of the latency and 30.14% of the cloud cost. GPT-5 Direct averages 85.66 score, 22.69 s latency, and 43.45 normalized cloud cost.

Relative to Direct, PARA reduces mean latency by approximately 9.23% and cloud cost by 27.51%, while remaining within 4.67 score points. These results demonstrate efficient use of the local model pool alongside selective cloud execution.

On FRAMES, which is excluded from training, PARA (pool) achieves 70.02%, exceeding original HybridLLM, original Router-R1, and Minions by 25.12, 12.62, and 19.66 points, respectively, with lower latency. PARA is within 0.98 points of matched Router-R1 (71.00%) while reducing latency by 38.99% and cloud cost by 38.43%. This combination of strong task score and lower execution overhead demonstrates effective transfer to an unseen benchmark.

4.3 Generalizability Evaluation

We test PARA's generalization with Pool-B (LFM2-0.7B/Phi4-4B/Ministral3-8B/Ministral3-14B + Claude Sonnet 4.5), and directly apply the policy trained with the original pool (Pool-A) to Pool-B without retraining. Table 4 reports the six-benchmark comparison on HotpotQA, GSM8K, DROP, HumanEval, MATH, and ToolBench. Under Pool-A, PARA improves mean benchmark score by +13.34, +10.44, and +6.74 points over original HybridLLM, Minions, and original Router-R1, respectively, while incurring only 0.30× the All-LLM cloud cost and 0.55× the All-LLM end-to-end latency. Without retraining, PARA continues to perform well under Pool-B, improving mean benchmark score by +10.78 points over HybridLLM, +6.34 points over Minions, and +3.17 points over Router-R1, while using only 0.35× the All-LLM cost and 0.59× the All-LLM latency. PARA retains its advantage over Router-R1, whose model descriptors and sequential routing support generalization across model pools (Zhang et al., 2025). Pool-B changes both local executors and the cloud model. The retained gains demonstrate that the learned allocation and commitment policy transfers across model combinations while maintaining efficient execution.

Table 4: Transfer across model pools on six benchmarks (excluding FRAMES). ΔScore is the mean gain over original baselines; cloud cost and latency are relative to All-LLM (= 1).

Pool	ΔScore Hybrid	ΔScore Minions	ΔScore Router-R1	Cost (×LLM)	Latency (×LLM)
Pool-A	+13.34	+10.44	+6.74	0.30	0.55
Pool-B	+10.78	+6.34	+3.17	0.35	0.59

4.4 Ablation Studies

SFT-only and SFT+GRPO share the supervised checkpoint, planner, executor pool, and requests. Per-node PARA changes allocation granularity while retaining adaptive horizons. Table 5 reports seven-benchmark means in (a), SFT-only results in (b), per-node results in (c), and GRPO gains over SFT-only in (d). Relative gains use the SFT-only score as the denominator; C_t is in seconds. Full PARA results are in Table 3. GRPO yields a seven-benchmark gain of 6.25 points (8.37% relative), with per-benchmark relative gains of 7.36–9.49%.

Task-type routing. PARA achieves 80.99 mean score while reducing latency from the per-node variant's 25.74 s to 20.59 s and normalized cloud cost from 34.02 to 31.50. It retains the per-node score of 81.62 within 0.63 points, saving 20.01% latency and 7.41% cloud cost. Sharing preferences among nodes of the same task type provides efficient model allocation as the frontier changes.

Adaptive commitment. Compared with fixed $h = 1$, the learned horizon reduces latency by 25.83% and cloud cost by 29.25%, with scores of 80.99 and 81.84, respectively. At the other extreme, fixed $h = \infty$ reaches 66.26 score at 18.43 s latency and 29.21 normalized cloud cost. Adaptive commitment gains 14.73 score points over this long commitment for 2.16 s additional latency. The learned horizon combines efficient reuse of routing decisions with timely reconsideration of model allocation and the remaining DAG.

Frontier execution. Parallel frontier execution reduces latency from the sequential scheduler's 33.74 s to 20.59 s, a 1.64× speedup, while maintaining similar task score and cloud cost. Together, the ablations show how task-type routing, adaptive commitments, and frontier parallelism contribute to efficient execution.

Table 5: Training, routing, and execution ablations.

(a) Seven-benchmark mean

Variant	Score	C_t (s)	C_a
Adaptive horizon	80.99	20.59	31.50
SFT-only	74.74	22.24	35.26
Per-node	81.62	25.74	34.02
Fixed $h = 1$	81.84	27.76	44.52
Fixed $h = \infty$	66.26	18.43	29.21
Sequential scheduler	80.61	33.74	31.71

(b) SFT-only

Benchmark	Score	C_t (s)	C_a
HotpotQA	70.96	20.16	34.40
GSM8K	89.54	7.03	17.41
DROP	82.37	10.63	28.64
HumanEval	77.44	12.73	37.89
MATH	75.68	14.33	36.69
ToolBench	63.23	27.45	47.86
FRAMES	63.96	63.34	43.96

(c) Per-node

Benchmark	Score	C_t (s)	C_a
HotpotQA	77.42	23.34	33.77
GSM8K	96.23	8.14	17.91
DROP	89.54	12.30	28.12
HumanEval	84.15	14.74	36.54
MATH	82.93	16.59	34.45
ToolBench	70.11	31.78	44.95
FRAMES	71.00	73.31	42.41

(d) GRPO–SFT

ΔScore	Rel. (%)
5.96	8.40
6.59	7.36
6.67	8.10
6.10	7.87
6.50	8.59
5.88	9.30
6.07	9.49

5 Related Work

Structured agent execution. Chain-of-thought (CoT), ReAct, and Reflexion organize reasoning and interaction, while multi-agent systems coordinate specialized agents (Wei et al., 2022; Yao et al., 2023; Shinn et al., 2023; Wu et al., 2024). DoT (Shao et al., 2025) is closest structurally: it decomposes requests into dependency DAGs, exposes parallel subtasks, and routes nodes between local and cloud models based on estimated difficulty. However, it does not explicitly account for error propagation along dependency edges. In contrast, PARA distinguishes intrinsic node difficulty from propagation sensitivity and adapts routing as DAG execution evolves.

Hybrid local–cloud inference. Request-level methods such as FrugalGPT and HybridLLM route or cascade models using difficulty, confidence, or learned scores (Chen et al., 2023; Ding et al., 2023; Ong et al., 2025), while multi-call methods aggregate outputs from multiple models (Jiang et al., 2023). Minions (Narayan et al., 2025) uses iterative local–remote collaboration. Router-R1 (Zhang et al., 2025) extends one-shot routing to multi-round RL-based routing by interleaving reasoning and model invocation, but does not consider dependency DAGs with concurrently runnable subtasks and error propagation along edges. PARA instead performs propagation-aware routing over a planner-produced DAG and adaptively determines when routing decisions should be reconsidered.

Multi-agent routing and orchestration. STRMAC (Wang et al., 2025) and MasRouter (Yue et al., 2025) learn agent/model selection policies. Panayotov & Emanuilov (2025) study adaptive routing between agents. ToolOrchestra (Su et al., 2025) trains a lightweight RL orchestrator to dynamically invoke heterogeneous models and tools, optimizing task success and execution efficiency, but follows a sequential reasoning–tool-calling paradigm rather than explicitly modeling dependency DAGs. In contrast, PARA executes concurrently runnable DAG nodes as frontier waves and jointly learns task-conditioned model assignment and an adaptive commitment horizon, adapting to execution state without rerouting after every wave.

6 Conclusion

Existing hybrid agent routers overlook error propagation across subtask dependencies and make routing decisions one subtask or frontier at a time. PARA combines error-propagation-aware task-type model allocation, adaptive commitment horizons, and concurrent frontier execution for DAG-structured agents. Across seven benchmarks, SAGE reduces cloud cost by 68.50% and latency by 43.15% relative to All-LLM while closing 87.50% of the mean accuracy gap between All-SLM and All-LLM. It also achieves higher mean accuracy, lower latency, and lower cloud cost than HybridLLM, Router-R1, and Minions. Transfer and ablation results show that these gains persist across model pools and execution settings.

AI Assistance Disclosure

Generative AI tools helped improve clarity, grammar, and presentation. The authors reviewed and revised all AI-assisted text and take full responsibility for the content.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. Automix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

BabyAGI. Babyagi: Simple autonomous ai agents. `https://www.babyagi.com`, 2023.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2023.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL-HLT*, pp. 2368–2378, 2019.

Grammarly. Grammarly: Writing enhancement tool. `https://www.grammarly.com/`, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021b.

IBM. Watson Assistant: AI for Customer Service. `https://www.ibm.com/cloud/watson-assistant/`, 2024.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL `https://aclanthology.org/2023.acl-long.792/`.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4745–4759. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.243. URL `https://aclanthology.org/2025.naacl-long.243/`.

LangChain. Langchain: A framework for building applications with large language models. https://www.langchain.com, 2023.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. In *International Conference on Learning Representations*, 2025.

Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Ré. Minions: Cost-efficient collaboration between on-device and cloud language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 45682–45719, 2025.

NVIDIA. Bionemo: Ai for drug discovery and molecular biology. https://www.nvidia.com/en-us/drug-discovery/bionemo/, 2024.

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

OpenAI. Openai api pricing, 2023. URL https://openai.com/pricing.

Theodor Panayotov and Ivo Emanuilov. Adaptive routing protocols for determining optimal paths in ai multi-agent systems: a priority-and learning-enhanced approach. *arXiv preprint arXiv:2503.07686*, 2025.

Jili Qian, Zhengyu Jin, Quan Zhang, Guoqing Cai, and Beichang Liu. A liver cancer question-answering system based on next-generation intelligence and the large model med-palm 2. *International Journal of Computer Science and Information Technology*, 2(1):28–35, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Guillem Ramírez, Alexandra Birch, and Ivan Titov. Optimising calls to large language models with uncertainty-based two-tier selection. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024.

Marija Šakota, Maxime Peyrard, and Robert West. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 606–615, 2024. doi: 10.1145/3616855.3635825.

Chenyang Shao, Xinyuan Hu, Yutang Lin, and Fengli Xu. Division-of-thoughts: Harnessing hybrid language model synergy for efficient on-device agents. In *Proceedings of the ACM on Web Conference 2025*, pp. 1822–1833, 2025.

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. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pp. 1279–1297, 2025. doi: 10.1145/3689031.3696075.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Significant-Gravitas. Autogpt. https://github.com/Significant-Gravitas/AutoGPT, 2023.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Jingbo Wang, Sendong Zhao, Haochun Wang, Yuzheng Fan, Lizhe Zhang, Yan Liu, and Ting Liu. Optimal-agent-selection: State-aware routing framework for efficient multi-agent collaboration. *arXiv preprint arXiv:2511.02200*, 2025.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *Proceedings of the Conference on Language Modeling (COLM)*, 2024.

Zikai Xie, Xenophon Evangelopoulos, Ömer H Omar, Alessandro Troisi, Andrew I Cooper, and Linjiang Chen. Fine-tuning gpt-3 for machine learning electronic and functional properties of organic molecules. *Chemical science*, 15(2):500–510, 2024.

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. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

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

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. In *International Conference on Learning Representations*, 2024.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025. doi: 10.18653/v1/2025.acl-long.757.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

A PLANNER RELIABILITY AND EDGE REPAIR

We evaluate missing-dependency repair. At a matched execution snapshot, we repair the missing edge, retain the cached prefix, and resume execution. Table 6 reports the percentage of requests with missing dependencies, whole-benchmark scores before and after repair, and score gains.

Table 6: Impact of repairing missing dependencies. Original and repaired scores are whole-benchmark scores; recovered points are their difference before rounding.

Benchmark	**Requests with missing dependencies (%)**	**Original score**	**Repaired score**	**Recovered points**
HotpotQA	2.79	76.92	77.20	0.28
GSM8K	2.48	96.13	96.25	0.12
DROP	4.03	89.04	89.44	0.40
HumanEval	6.10	83.54	84.15	0.61
MATH	5.58	82.18	82.88	0.70
ToolBench	5.58	69.11	69.81	0.70
FRAMES	7.28	70.02	71.00	0.97
Macro-average	**4.83**	**80.99**	**81.53**	**0.54**

B RUNTIME RELIABILITY: TIMING AND CONCURRENCY

B.1 REPEATED LATENCY

The reported timing setup uses 50 requests per benchmark and five repeats per request, giving 350 request pairs across seven benchmarks. Table 7 gives an approximation from the aggregate values: the displayed per-benchmark latencies in Table 3 imply a mean Single–Pool difference of $\bar{d} \approx 0.127\,\text{s}$. Using the reported paired SD $s_d = 1.20\,\text{s}$, the normal approximation $\bar{d} \pm 1.96 s_d/\sqrt{350}$ gives $[0.001, 0.253]\,\text{s}$. This calculation assumes that the aggregate difference and SD describe the same matched requests. We do not infer statistical significance from this approximation.

Table 7: Approximation of the Single–Pool latency difference from aggregate values, using 350 request pairs and the reported paired SD.

Comparison	Difference (s)	Repeats	SD (s)	95% interval (s)
Single minus pool	0.127	5	1.20	$[0.001, 0.253]$

B.2 CLOUD CONCURRENCY CAP

To evaluate the cloud concurrency cap, we run a subset of experiments with requests, router/planner, local concurrency, and retry policy fixed while limiting simultaneous cloud calls to 1, 2, or 4. Table 8 reports the results. Its latency values follow $T(c) = 11.85 + 9.60/c$ seconds, with cap 4 giving 14.25 s. The score remains 82 across the three caps; added delay is $T(c) - T(4)$.

Table 8: Effect of the cloud-concurrency cap on score, cost, and latency.

Cloud cap	Score	C_a	Latency (s)	Added delay (s)
1	82	30.21	21.45	7.20
2	82	29.38	16.65	2.40
4	82	31.15	14.25	0.00