Adaptive Decentralized Multi-LLM Orchestration with Accelerated Consensus-Based SPSA

Kirill Chernikov
chernikov.kv@gmail.com
ITMO University
Saint Petersburg, Russia

Elizaveta Tarasova✉
elizaveta.tarasova@spbu.ru,
el.u.tarasova@gmail.com
Saint Petersburg State University
Saint Petersburg, Russia

Oleg Granichin
o.granichin@spbu.ru
Saint Petersburg State University
Saint Petersburg, Russia

Abstract

Adaptive routing in multi-LLM systems is difficult to evaluate reliably because different policies may observe different model outputs, latencies, costs, and judge scores. We introduce a replay-based evaluation protocol that fixes both the task stream and the potential outcome of each task–agent pair, allowing routing policies to be compared under identical conditions. Within this framework, we adapt an existing distributed accelerated consensus-based simultaneous perturbation stochastic approximation method (A-SPSA), rather than proposing a new optimization algorithm, to the online optimization of controller parameters used for processing-time and wait-time prediction. The method combines two-point zeroth-order feedback, accelerated surrogate updates, and consensus among distributed controllers. We evaluate A-SPSA against several distributed zeroth-order baselines in a discrete-event simulation with heterogeneous agents, and further validate the policies using cached outputs from real LLM endpoints. The experiments include aggregate comparison, component-level ablation, and hyperparameter sensitivity analysis. A-SPSA achieves the strongest overall balance between routing objective, deadline hit rate, and wait-time prediction error, while the ablation results show that lower intermediate prediction loss does not necessarily lead to better downstream routing decisions. The proposed protocol provides a reproducible basis for evaluating adaptive agent-routing policies under stochastic and partially observable conditions.

1 Introduction

LLM-based agents are increasingly deployed not as isolated models, but as components of larger agentic AI systems. Such systems may include multiple specialized LLM agents, routing or orchestration layers, external tools, quality evaluators, and online feedback loops. Their evaluation is therefore substantially more difficult than the evaluation of a single model on a static benchmark. The observed behavior depends not only on the quality of individual model outputs, but also on which agent is selected, how latency and cost are handled, how feedback is delayed, and how the system adapts over time. Recent work on LLM-based agent evaluation and agent benchmarks highlights the need to evaluate agents in interactive and dynamic settings rather than only through single-turn output quality [15, 26, 28].

A central difficulty is reproducibility. In adaptive multi-LLM orchestration, different routing policies may send the same task to different models and therefore observe different stochastic outputs, latencies, and quality signals. As a result, a naive comparison of policies can conflate the quality of the routing decision with random variation in model generation or API behavior. LLM-as-judge methods provide a scalable way to approximate quality signals, but they also introduce their own sources of bias and uncertainty, including judge-model preferences and sensitivity to prompting protocols [16, 27]. For this reason, evaluating an adaptive routing policy requires not only task-level performance scores, but also a protocol that controls which potential outcomes are available to each policy. Existing LLM routing methods address important cost–quality trade-offs in model selection. For example, cascaded and routing-based approaches can reduce inference cost while maintaining or improving response quality, and recent work studies routing from preference data and resource allocation in multi-agent LLM systems [2, 6, 20]. However, these works primarily focus on learning or designing routing mechanisms. The complementary question of how to reproducibly evaluate adaptive online routing policies under the same stochastic task–agent outcomes, delayed feedback, and judge-based quality signals remains less developed.

In this paper, we address this question using the decentralized adaptive multi-agent routing architecture introduced in [25] as a controlled system substrate. Unlike that study, the present work focuses on the reproducible evaluation of adaptive multi-LLM routing policies under stochastic outputs, partial observability, delayed feedback, and heterogeneous latency–quality trade-offs.

The adaptation mechanisms considered in this study come from distributed zeroth-order optimization. Such methods are natural candidates for online LLM orchestration because routing objectives are black-box, noisy, and observed only through feedback. We include simultaneous perturbation stochastic approximation and its consensus-based and accelerated variants [8, 9, 11, 12, 23, 24], as well as representative distributed zeroth-order and primal–dual baselines [1, 7, 14, 17, 22]. These methods are not introduced as new optimization algorithms in this paper; rather, they serve as adaptive routing policies whose behavior can be compared within a common evaluation protocol.

We propose a replay-based evaluation protocol for adaptive multi-LLM orchestration. The protocol fixes the task stream, precollects or simulates potential task–agent executions, and then replays the same potential outcomes for all routing policies. This design separates policy behavior from stochastic generation noise and makes it possible to compare policies under identical task streams, agent outputs, latency observations, and judge signals. The evaluation reports both operational routing metrics, such as routing utility and deadline hit rate, and intermediate adaptation metrics, such as prediction losses and stability across seeds.

Our contributions are as follows:

Eval. & Trustworthiness of Agentic AI @ KDD'26, Jeju, Republic of Korea 2026.

- We formulate adaptive multi-LLM routing as an evaluation problem for agentic AI systems with stochastic outputs, delayed feedback, and heterogeneous latency–quality tradeoffs.
- We introduce a replay-based evaluation protocol that fixes the task stream and caches task–agent executions, enabling adaptive routing policies to be compared under identical potential outcomes.
- We instantiate the protocol in a decentralized multi-controller routing system and compare several adaptive zeroth-order policies, including baseline distributed SPSA and distributed accelerated consensus-based A-SPSA.
- We analyze component-level behavior, step-size sensitivity, and a small real-endpoint feasibility setting, and discuss evaluation limitations such as judge bias, induced heterogeneity, and the limited scale of real-endpoint validation.

2 Evaluation Problem and System Setting

2.1 Evaluation Problem

We consider adaptive task routing in a multi-LLM orchestration system. Tasks arrive online and must be assigned to LLM agents using only the available history and the current system state. Let $T = T_1, T_2, \dots$ be the task stream, where task T_i arrives at time t_i^{arr}. When processing T_i, the system observes only $\mathcal{H}_i = T_1, \dots, T_i$ and feedback from previously completed tasks; future arrivals, API delays, LLM outputs, and quality values are unknown.

The main evaluation challenge is that different routing policies may assign the same task to different agents and therefore observe different responses, latencies, costs, and quality signals. A direct online comparison can thus mix policy quality with stochastic generation, API variability, and judge noise. We therefore treat a routing policy as $\pi : \mathcal{H}_i \times \mathcal{S}_i \to \mathcal{E}$, where $\mathcal{S}_i$ is the current system state and $\mathcal{E}$ is the set of LLM agents, and compare policies under a controlled task stream and controlled potential task–agent outcomes.

2.2 Adaptive Multi-LLM Routing Setting

The execution layer consists of LLM agents $\mathcal{E} = E_1, \dots, E_n$. Each agent represents an LLM model or service and may differ in response quality, latency, cost, specialization across task types, and availability. Following the distributed adaptive task-allocation setting of [25], we introduce controllers $\mathcal{C} = C_1, \dots, C_m$, which receive tasks, estimate their parameters, select LLM agents, and update local prediction models from feedback. Controllers communicate over $\mathcal{G}(\tau) = (\mathcal{C}, \mathcal{A}(\tau))$, where $\mathcal{A}(\tau)$ is the set of controller links at adaptation iteration τ.

Each task is represented as $T_i = (t_i^{\text{arr}}, \text{type}_i, \phi_i, u_i, n_i)$, where t_i^{arr} is the arrival time, type_i is the task type, $\phi_i \in \mathbb{R}^d$ is a feature vector, $u_i \in [0, 1]$ is an urgency score, and n_i is the request size. Controller C_j estimates the expected processing time and acceptable waiting time and defines an internal deadline:

$$\hat{p}_i = \hat{p}(T_i; \theta_j^{(p)}), \quad \hat{w}_i = \hat{w}(T_i; \theta_j^{(w)}), \quad D_i = t_i^{\text{arr}} + \hat{p}_i + \hat{w}_i.$$

Here, $\theta_j^{(p)}$ and $\theta_j^{(w)}$ are local prediction parameters. The deadline D_i is an internal operational target rather than an externally imposed hard constraint. If T_i is assigned to E_k, the observed outcome is $Y_{ik} = (p_{ik}^{\text{real}}, q_{ik}^{\text{real}}, c_{ik}^{\text{real}}, y_{ik}^{\text{text}})$, including processing time, quality, cost, and the generated response.

2.3 Routing Decision and Feedback

For each task–agent pair (T_i, E_k), the controller computes

$$\text{score}(E_k, T_i) = \mathbf{w}^\top \mathbf{f}(E_k, T_i) - \kappa_1 d_{ik} - \kappa_2 \max\{0, r_{ik} - 1\}.$$

Here, $\mathbf{f}(E_k, T_i)$ may include relevance, availability, predicted quality, speed, queue state, cost, reputation, and urgency-aware terms, while $\kappa_1, \kappa_2 \geq 0$ weight deadline-related penalties. The predicted finish time $\hat{t}_{ik}^{\text{finish}}$, normalized deadline gap d_{ik}, time budget b_i, and time-budget ratio r_{ik} are

$$\hat{t}_{ik}^{\text{finish}} = \hat{t}_k^{\text{avail}} + \hat{p}_{ik}, \quad d_{ik} = \frac{\max\{0, \hat{t}_{ik}^{\text{finish}} - D_i\}}{b_i},$$

$$b_i = \max\{\hat{p}_i + \hat{w}_i, \varepsilon_b\}, \quad r_{ik} = \frac{\hat{t}_k^{\text{avail}} - t_i^{\text{arr}} + \hat{p}_{ik}}{b_i},$$

where $\varepsilon_b > 0$ ensures numerical stability. The selected agent is $k_i = \arg\max_{k \in 1,\dots,n} \text{score}(E_k, T_i)$.

After completion, the controller receives

$$O_i = \left(p_{ik_i}^{\text{real}}, w_i^{\text{real}}, q_{ik_i}^{\text{real}}, c_{ik_i}^{\text{real}}, \mathbb{I}\{C_i \leq D_i\}\right),$$

where C_i is the completion time and the last term indicates deadline satisfaction. This feedback updates local prediction models for processing time, acceptable waiting time, and quality. Since each controller observes only the tasks routed through it, local histories may be biased toward different task types and agent outcomes. To reduce this effect without centralizing data, controllers may periodically align prediction parameters through the communication graph. Later sections compare distributed zeroth-order mechanisms for this adaptation, including SPSA-based and accelerated SPSA-based variants.

2.4 Controller-Level Load Balancing

Incoming tasks are also distributed among controllers through decentralized load balancing. Controllers exchange information only with neighbors in $\mathcal{G}(\tau)$; the load of controller C_j can be represented as $x_j(t) = q_j(t)/\bar{p}_j$, where $q_j(t)$ is its weighted queue load and $\bar{p}_j$ is its average productivity. This balancing can be performed using the Local Voting Protocol or A-LVP [3, 4, 10]. Thus, controller-level balancing determines which local history is used for prediction, while agent-level routing determines the realized task outcome. Both affect the observed metrics and motivate the replay-based protocol introduced next.

3 Replay-Based Evaluation Protocol

The goal of the protocol is to compare adaptive routing policies under controlled stochastic conditions. If each policy is evaluated by an independent online run, differences in performance may be caused not only by routing decisions, but also by random LLM generation, API variability, or judge noise. We therefore fix both the task stream and the potential task–agent outcomes before comparing policies. All policies are evaluated on the same ordered task stream $\mathcal{T}_N = \{T_1, \dots, T_N\}$, where each task is represented as $T_i = (t_i^{\text{arr}}, \text{type}_i, \phi_i, u_i, n_i)$. For this stream and the set of LLM agents

$\mathcal{E} = \{E_1, \ldots, E_n\}$, we construct an outcome table

$$\mathcal{Y} = \Big\{ Y_{ik} = \big(p_{ik}^{\text{real}}, q_{ik}^{\text{real}}, c_{ik}^{\text{real}}, y_{ik}^{\text{text}}\big) : \\ i = 1, \ldots, N,\ k = 1, \ldots, n \Big\},$$

where p_{ik}^{real} is the observed processing time, $q_{ik}^{\text{real}} \in [0,1]$ is the quality signal, c_{ik}^{real} is the execution cost, and y_{ik}^{text} is the response returned by agent E_k for task T_i. In simulation mode, these outcomes are generated from fixed stochastic agent profiles under fixed random seeds. In real-endpoint mode, they are obtained in a pre-collection phase by sending each task to each agent and caching the resulting responses, latencies, token usage, costs, and judge scores.

During replay, policy π processes the fixed task stream sequentially. At step i, it selects an agent based on its own history and state,

$$E_{k_i}^{\pi} = \pi(\mathcal{H}_i^{\pi}, \mathcal{S}_i^{\pi}), \qquad Y_i^{\pi} = Y_{ik_i^{\pi}} \in \mathcal{Y}.$$

The selected cached outcome Y_i^{π} is then returned to the policy as feedback. Different policies may therefore produce different trajectories because they choose different agents and update their parameters differently, but they are evaluated against the same task stream and the same potential outcomes. This separates routing and adaptation behavior from repeated stochastic LLM calls.

The quality signal is also fixed at the task–agent level. In simulation mode, quality combines a binary success signal and a timeliness score, while in real-endpoint mode the response is additionally evaluated by an LLM judge:

$$\mathcal{T}_i = \exp\left(-\frac{\max\{0, p_{ik_i}^{\text{real}} - t_i^{\text{ref}}\}}{t_i^{\text{ref}}}\right),$$
$$Q_i = \alpha_Q S_i + (1-\alpha_Q)\mathcal{T}_i, \quad Q_i^{\text{real}} = \lambda_Q q_i^{\text{judge}} + (1-\lambda_Q)\mathcal{T}_i,$$

where $S_i \in \{0,1\}$ is the success indicator, t_i^{ref} is the reference processing time, $\alpha_Q \in [0,1]$ controls the contribution of the success component, $q_i^{\text{judge}} \in [0,1]$ is the judge score, and $\lambda_Q \in [0,1]$ is the judge-quality weight. Judge outputs are cached together with the corresponding task–agent response, so replay does not introduce additional judge calls.

We report both operational routing metrics and prediction losses. For a policy π, the main operational metrics are the mean routing objective and deadline hit rate:

$$\overline{R}^{\pi} = \frac{1}{N}\sum_{i=1}^{N} \text{score}(E_{k_i^{\pi}}, T_i), \quad \text{DHR}^{\pi} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\{C_i^{\pi} \le D_i\}. \tag{1}$$

Here, C_i^{π} is the completion time of task T_i under policy π, and D_i is the internal deadline. Higher values of both metrics are better.

For prediction quality, we use the processing-time loss and the one-sided wait-prediction loss:

$$F_1(\theta^{(p)}) = \mathbb{E}\left[\left(x_i^{\top}\theta^{(p)} - p_{ik_i}^{\text{real}}\right)^2\right],$$
$$F_2(\theta^{(w)}) = \mathbb{E}\left[\max\left\{w_i^{\text{real}} + \delta - x_i^{\top}\theta^{(w)}, 0\right\}^2\right]. \tag{2}$$

The loss F_1 measures processing-time prediction error. The loss F_2 penalizes underestimation of acceptable waiting time, which would otherwise produce overly tight internal deadlines and increase deadline misses. When a quality predictor is used, we additionally report

$$\text{QMSE}^{\pi} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{q}_{ik_i^{\pi}} - q_{ik_i^{\pi}}^{\text{real}}\right)^2. \tag{3}$$

All metrics are computed on the same task stream and outcome table for every policy, and results are aggregated across random seeds as mean ± standard deviation. Additional model and protocol details, including task-pool construction, judge prompting, fallback rules, routing sub-scores, calibration constants, and hyperparameter grids, are provided in Appendix A.

4 A-SPSA-Based Parameter Adaptation

We introduce a parameter-optimization mechanism for the routing controllers responsible for predicting task processing time and admissible waiting time. These parameters are used to compute the predicted finish time, the internal deadline, and the routing score; therefore, their adaptation directly affects the quality of task assignment to LLM agents. The mechanism is based on distributed accelerated consensus-based SPSA [8], which combines two-point zeroth-order evaluations, an accelerated surrogate update, and consensus-based parameter alignment among neighboring controllers.

Each controller C_j maintains two parameter vectors $\theta_j^{(p)} \in \mathbb{R}^d, \theta_j^{(w)} \in \mathbb{R}^d$. The vector $\theta_j^{(p)}$ is used for processing-time prediction, whereas $\theta_j^{(w)}$ is used for wait-time budget prediction. These parameters are optimized with respect to the objective functions defined in (2). In what follows, θ_j denotes either of these two parameter vectors, and $\widehat{F}_{j,\tau}(\theta_j)$ denotes the corresponding local empirical version of F_1 or F_2, computed over the rolling window of recent observations available to controller C_j. At each adaptation iteration τ, the controllers communicate over the graph $\mathcal{G}(\tau) = \big(C, \mathcal{A}(\tau)\big)$, where $\mathcal{A}(\tau)$ is the set of active inter-controller links and $b_{j\ell}(\tau)$ is the communication weight between controllers C_j and C_ℓ. For controller C_j, the A-SPSA update is written as

$$\begin{aligned}
&\widetilde{\theta}_{j,\tau} = \frac{\alpha_\tau\gamma_{\tau-1}z_{j,\tau-1} + \gamma_\tau\theta_{j,2\tau-1}}{\gamma_{\tau-1} + \alpha_\tau(\mu-\eta)},\\
&x_{j,2\tau} = \widetilde{\theta}_{j,\tau} + \beta_\tau^{+}\Delta_{j,\tau}, \qquad x_{j,2\tau-1} = \widetilde{\theta}_{j,\tau} - \beta_\tau^{-}\Delta_{j,\tau},\\
&\Delta_{j,\tau} \in \left\{-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right\}^d,\\
&\widehat{g}_{j,\tau} = \frac{\widehat{F}_{j,\tau}(x_{j,2\tau}) - \widehat{F}_{j,\tau}(x_{j,2\tau-1})}{\beta_\tau^{+} + \beta_\tau^{-}}\Delta_{j,\tau}\\
&\quad + \omega\sum_{\ell\in\mathcal{N}_j(\tau)} b_{j\ell}(\tau)\left(\widetilde{\theta}_{j,\tau} - \widetilde{\theta}_{\ell,\tau}\right),\\
&\theta_{j,2\tau} = \widetilde{\theta}_{j,\tau} - h\widehat{g}_{j,\tau}, \qquad \theta_{j,2\tau+1} = \theta_{j,2\tau},\\
&z_{j,\tau} = \frac{(1-\alpha_\tau)\gamma_{\tau-1}z_{j,\tau-1} + \alpha_\tau(\mu-\eta)\widetilde{\theta}_{j,\tau} - \alpha_\tau\widehat{g}_{j,\tau}}{\gamma_\tau}.
\end{aligned} \tag{4}$$

Here, $\widetilde{\theta}_{j,\tau}$ is the accelerated probing point at which controller C_j collects zeroth-order information. The points $x_{j,2\tau-1}$ and $x_{j,2\tau}$ are the two measurement points used within one adaptation iteration. They are constructed using the simultaneous perturbation vector $\Delta_{j,\tau}$, whose components are independent and take the two values

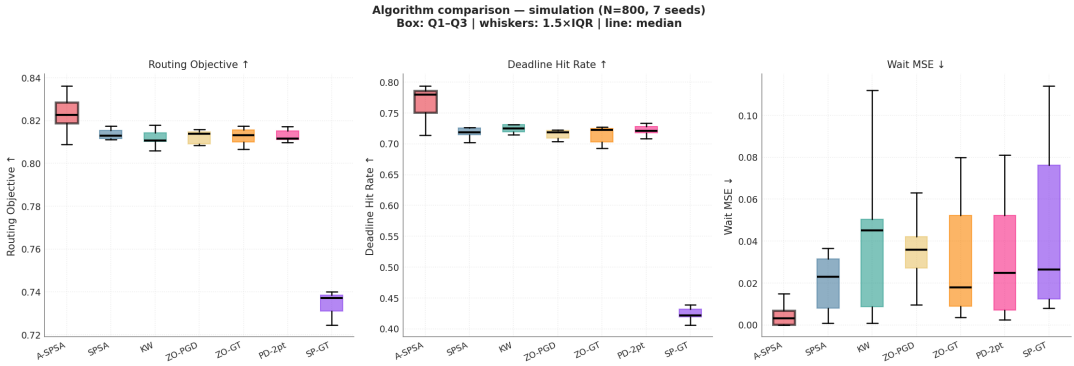

Figure 1: Numerical comparison over $N = 800$ tasks and seven independent seeds. Boxes show the interquartile range, center lines show medians, and whiskers extend to 1.5 times the interquartile range.

with equal probability. Thus, the update direction requires only two evaluations of the local empirical objective $\widehat{F}_{j,\tau}$, independently of the dimension d.

The estimate $\widehat{g}_{j,\tau}$ consists of two terms. The first term is the SPSA estimate of the local descent direction. The second term is the consensus correction based on the discrepancy between controller C_j and its neighbors $\mathcal{N}_j(\tau)$. The parameter $\omega > 0$ controls the strength of inter-controller alignment. Therefore, the update accounts for both local prediction error and consistency of controller parameters across the network.

The variable $z_{j,\tau}$ is the surrogate center. It accumulates information from previous iterations and is used to construct the next probing point. The accelerated dynamics are governed by $z_{j,\tau}$ together with the sequences α_τ and γ_τ, defined by

$$\frac{\alpha_\tau^2}{1+\varepsilon} h = (1-\alpha_\tau)\gamma_{\tau-1} + \alpha_\tau(\mu - \eta),$$
$$\gamma_\tau = (1-\alpha_{\tau-1})\gamma_{\tau-1} + \alpha_{\tau-1}(\mu - \eta).$$

Here, h is the descent step size, μ corresponds to the strong convexity parameter of the regularized objective, $\eta \in (0, \mu)$ is a stabilization parameter, and $\varepsilon > 0$ accounts for noise and nonstationary effects. In the implementation, α_τ may additionally be capped from above to avoid overly aggressive acceleration at the beginning of the online trajectory, when the rolling window contains only a limited number of observations.

The same update is applied separately to $\theta_j^{(p)}$ and $\theta_j^{(w)}$. For $\theta_j^{(p)}$, the local oracle $\widehat{F}_{j,\tau}$ corresponds to the processing-time prediction error. For $\theta_j^{(w)}$, it corresponds to the one-sided wait-prediction loss. After the update, the new parameters are used in subsequent routing decisions: they affect the predicted finish time, the internal deadline, and the final routing score.

Adaptation is performed periodically after new observations have been collected. Between adaptation iterations, controllers use the current values of $\theta_j^{(p)}$ and $\theta_j^{(w)}$ to route incoming tasks. Once tasks are completed, feedback is added to the rolling window, and the next A-SPSA update is computed from the updated local history:

$$\text{routing} \longrightarrow \text{feedback} \longrightarrow \text{rolling window update} \longrightarrow$$
$$\longrightarrow \text{A-SPSA adaptation} \longrightarrow \text{next routing decisions}.$$

Under the replay-based evaluation protocol, all policies use the same task stream and the same task–agent outcome table. Hence, the A-SPSA-based policy does not receive additional access to the environment; it differs only in the parameter-update mechanism. Other distributed zeroth-order methods are treated as experimental baselines and introduced in the experimental section.

5 Experiments

We evaluate A-SPSA in three stages. First, we compare it with distributed zeroth-order baselines in a reproducible numerical simulation. We then examine the contribution of its main components and its sensitivity to the step-size scale. Finally, we evaluate the routing policies using responses collected from real LLM endpoints. Detailed task-generation rules, agent profiles, hyperparameter configurations, and supplementary trajectories are provided in Appendix B.

5.1 Numerical Evaluation

Setup. The simulation includes three controllers and five heterogeneous agents. Tasks arrive according to a Poisson process with rate $\lambda = 1.25$ tasks per second and are sampled from five categories: PROGRAMMING, QA, SUMMARIZATION, TRANSLATION, and TOOL_USE, with prompts drawn from five established benchmarks (Appendix B.1, Table 3). Agent outcomes are generated from task-dependent latency, capability, and semantic profiles that induce a speed–quality trade-off.

The main comparison uses $N = 800$ tasks and seven independent seeds. Parameter adaptation is performed every five tasks using a rolling window of 25 completed tasks. A-SPSA is compared with consensus-based SPSA, Kiefer–Wolfowitz (KW), zeroth-order projected gradient descent (ZO-PGD), zeroth-order gradient tracking (ZO-GT), a two-point primal–dual method (PD-2pt), and single-point gradient tracking (SP-GT). All methods use identical task streams and task–agent outcomes and differ only in the controller-parameter update.

Figure 1 reports Routing Objective, Deadline Hit Rate, and Wait MSE over the seven runs. A-SPSA achieves the highest median Routing Objective and Deadline Hit Rate and the lowest Wait MSE. The largest separation is observed for Deadline Hit Rate, suggesting

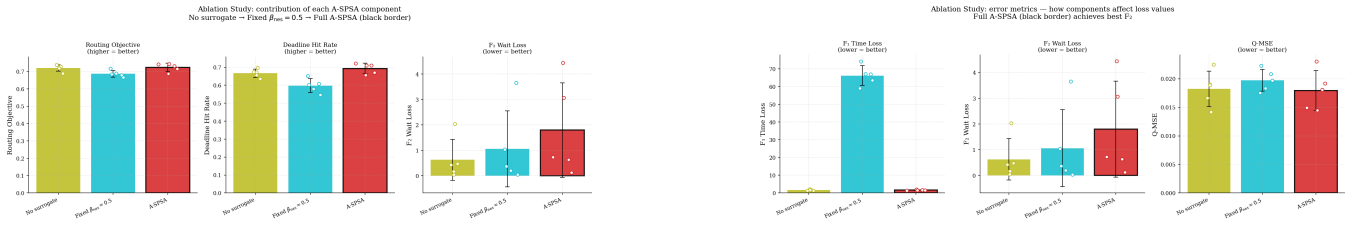

Figure 2: Component-level evaluation of A-SPSA. The left panel reports Routing Objective, Deadline Hit Rate, and Wait MSE; the right panel reports additional prediction-error metrics.

that the accelerated update leads to more effective completion-time and deadline estimates in the considered setting. SP-GT exhibits the weakest operational performance and the largest degradation relative to the remaining methods.

5.2 Component-Level Evaluation of A-SPSA

We compare the full A-SPSA update with two variants: a no-surrogate variant without accelerated momentum and a variant with a fixed acceleration coefficient. Each configuration is tuned independently on the same validation stream and evaluated under the same task and system configuration.

Figure 2 shows that the full A-SPSA variant achieves the highest Routing Objective and Deadline Hit Rate. The no-surrogate variant attains a lower intermediate Wait MSE, but this reduction does not translate into better downstream routing performance. Fixing the acceleration coefficient degrades both operational metrics and increases prediction error. Table 1 reports the corresponding numerical values.

Table 1: Ablation results reported as mean ± standard deviation.

Variant	Routing obj. ↑	DHR ↑	Wait MSE ↓
No surrogate	0.720 ± 0.018	0.667 ± 0.023	**0.627 ± 0.805**
Fixed acceleration	0.687 ± 0.019	0.598 ± 0.039	1.060 ± 1.496
A-SPSA	**0.724 ± 0.024**	**0.694 ± 0.029**	1.796 ± 1.864

5.3 Hyperparameter Sensitivity of A-SPSA

Figure 3 examines the sensitivity of A-SPSA to the step-size scale a in $\alpha_\tau = a/(\tau + 11)^{0.602}$.

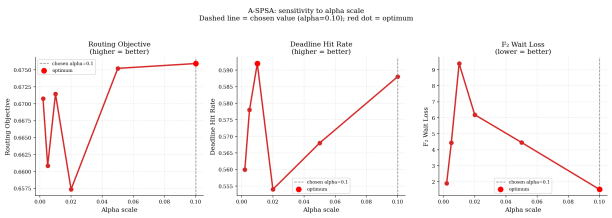

Figure 3: Sensitivity of A-SPSA to the step-size scale a. The vertical dashed line marks the selected value $a = 0.1$.

Small values of a lead to under-adaptation, whereas Routing Objective and Wait MSE improve as the step-size scale increases. Deadline Hit Rate remains close to its maximum over the upper part of the tested range. The results indicate that the selected value $a = 0.1$ provides strong performance without requiring fine-grained tuning.

5.4 Real-LLM Evaluation

Setup. We evaluate the routing policies on 200 tasks from two independent streams with seeds {11, 42} using five Groq endpoints:

- `llama-3.1-8b-instant`
- `openai/gpt-oss-20b`
- `allam-2-7b`
- `llama-4-scout-17b-16e-instruct`
- `qwen/qwen3-32b`

The experiment follows the shared-execution-cache protocol from Section 3. Each task is first submitted to every endpoint, and the generated response and judge-based quality score are cached. The routing policies are then replayed against the same cache without additional LLM calls.

API latency is replaced by a sample from the corresponding controlled agent profile, while quality is computed from the actual generated response. Adaptation is performed after each task, yielding approximately 186 updates after the warm-up period.

Figure 4 shows the online evolution of Deadline Hit Rate. A-SPSA exhibits an initial adaptation transient, but after sufficient feedback has accumulated it recovers and maintains the highest sustained deadline performance over most of the remaining task stream.

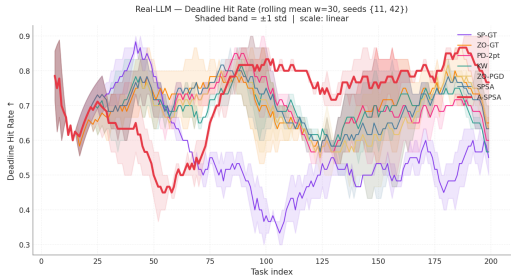

Figure 4: Deadline Hit Rate in the real-LLM evaluation. Curves show rolling means with a window of 30 tasks; shaded regions show one standard deviation over seeds {11, 42}.

Table 2: Real-LLM results over 200 tasks, reported as mean ± standard deviation over seeds $\{11, 42\}$.

Metric	A-SPSA	ZO-GT	KW	PD-2pt	ZO-PGD	SPSA	SP-GT
Routing Objective ↑	**0.839 ± 0.001**	0.835 ± 0.002	0.834 ± 0.004	0.834 ± 0.008	0.832 ± 0.009	0.831 ± 0.010	0.764 ± 0.007
DHR ↑	**0.722 ± 0.004**	0.695 ± 0.021	0.680 ± 0.042	0.688 ± 0.053	0.677 ± 0.032	0.690 ± 0.049	0.562 ± 0.011
Wait MSE ↓	**0.162 ± 0.093**	0.189 ± 0.141	0.239 ± 0.234	0.213 ± 0.154	0.241 ± 0.195	0.243 ± 0.176	0.223 ± 0.172

Table 2 shows that A-SPSA achieves the highest Routing Objective and Deadline Hit Rate and the lowest Wait MSE. ZO-GT provides the closest competing result, whereas SP-GT remains substantially below the other methods in both operational metrics.

Code is available at https://github.com/newpotatato/decentralized-llm-spsa.

6 Trustworthiness, Biases, and Limitations

The replay-based protocol improves comparability by exposing all routing policies to the same task streams and potential task–agent outcomes. However, it does not remove all sources of bias. The numerical evaluation relies on hand-designed agent profiles, while the real-LLM experiment uses a limited set of Groq endpoints and only two task streams. In addition, API latency is replaced by controlled samples, so the real-output evaluation does not capture provider-side congestion, rate limits, or failures. Quality estimates depend on the selected LLM judge and may inherit its preferences or model-specific biases. The evaluated task categories and datasets also cover only a subset of practical orchestration workloads. Finally, the experiments use three controllers and five agents; scalability to larger controller networks, denser communication graphs, and broader model pools remains to be studied. The consensus-correction term in the A-SPSA update (Eq. 4) assumes synchronous communication, without an explicit communication-delay term. This matches the current co-located setting, where communication cost is negligible relative to the adaptation interval. In a deployment with physically separate controllers, round-trip parameter-exchange time could become comparable to or larger than the adaptation interval, requiring an explicit delay-aware term (e.g., stale neighbor estimates $\theta_{\ell,\tau-d_{j\ell}}$) and a quantitative treatment of communication overhead, which we leave to future work.

7 Conclusion

We presented a replay-based framework for evaluating adaptive multi-LLM routing policies under controlled task streams and task–agent outcomes. Within this framework, we adapted accelerated consensus-based SPSA to the online optimization of controller prediction parameters. Across numerical simulations, component-level analysis, and real-LLM outputs, A-SPSA achieved the strongest overall balance between routing utility, deadline compliance, and wait-time prediction error. The results also show that lower intermediate prediction loss does not necessarily imply better downstream routing performance. Future work will extend the evaluation to larger controller networks, additional LLM providers, judge-robustness checks, naturally observed latency profiles, and more diverse agentic workloads.

Acknowledgments

This work was supported by the Ministry of Economic Development of the Russian Federation (Agreement No. 139-15-2025-007, dated April 16, 2025; ID: 000000C313925P3O0002).

References

[1] Arya Akhavan, Massimiliano Pontil, and Alexandre B. Tsybakov. 2021. Distributed Zero-Order Optimization under Adversarial Noise. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 10209–10220. https://proceedings.neurips.cc/paper_files/paper/2021/file/5487e79fa0ccd0b79e5d4a4c8ced005d-Paper.pdf

[2] Alfonso Amayuelas, Jingbo Yang, Saaket Agashe, Ashwin Nagarajan, Antonis Antoniades, Xin Eric Wang, and William Wang. 2025. Self-Resource Allocation in Multi-Agent LLM Systems. arXiv:2504.02051 [cs.MA] doi:10.48550/arXiv.2504.02051

[3] Natalia Amelina, Alexander Fradkov, Yuming Jiang, and Dimitrios J. Vergados. 2015. Approximate Consensus in Stochastic Networks With Application to Load Balancing. *IEEE Transactions on Information Theory* 61, 4 (2015), 1739–1752. doi:10.1109/TIT.2015.2406323

[4] Natalia Amelina, Oleg Granichin, and Aleksandra Kornivetc. 2013. Local Voting Protocol in Decentralized Load Balancing Problem with Switched Topology, Noise, and Delays. In *Proceedings of the 52nd IEEE Conference on Decision and Control*. 4613–4618. doi:10.1109/CDC.2013.6760604

[5] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732* (2021). doi:10.48550/arXiv.2108.07732

[6] Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Transactions on Machine Learning Research* (2024). Featured Certification. doi:10.48550/arXiv.2305.05176

[7] Songsong Cheng, Xin Yu, Yuan Fan, and Gaoxi Xiao. 2023. Zeroth-order Gradient Tracking for Distributed Constrained Optimization. *IFAC-PapersOnLine* 56, 2 (2023), 5197–5202. 22nd IFAC World Congress. doi:10.1016/j.ifacol.2023.10.115

[8] Victoria Erofeeva, Oleg Granichin, and Anna Sergeenko. 2026. Accelerated Consensus-Based SPSA Algorithm for Multisensor Multitarget Tracking Problem. *IEEE Trans. Automat. Control* (2026). doi:10.1109/TAC.2026.3678473

[9] Victoria Erofeeva, Oleg Granichin, Munira Tursunova, Anna Sergeenko, and Yuming Jiang. 2022. Accelerated Simultaneous Perturbation Stochastic Approximation for Tracking Under Unknown-but-Bounded Disturbances. In *Proceedings of the 2022 American Control Conference*. 1582–1587. doi:10.23919/ACC53348.2022.9867491

[10] Victoria Erofeeva, Oleg Granichin, and Elena Volodina. 2024. Accelerated Decentralized Load Balancing in Multi-Agent Networks. *IEEE Access* 12 (2024), 161954–161967. doi:10.1109/ACCESS.2024.3488399

[11] Oleg Granichin and Natalia Amelina. 2015. Simultaneous Perturbation Stochastic Approximation for Tracking Under Unknown but Bounded Disturbances. *IEEE Trans. Automat. Control* 60, 6 (2015), 1653–1658. doi:10.1109/TAC.2014.2359711

[12] Oleg Granichin, Victoria Erofeeva, Yury Ivanskiy, and Yuming Jiang. 2021. Simultaneous Perturbation Stochastic Approximation-Based Consensus for Tracking Under Unknown-But-Bounded Disturbances. *IEEE Trans. Automat. Control* 66, 8 (2021), 3710–3717. doi:10.1109/TAC.2020.3024169

[13] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, Vancouver, Canada, 1601–1611. doi:10.18653/v1/P17-1147

[14] Jinlong Lei, Han-Fu Chen, and Hai-Tao Fang. 2016. Primal–Dual Algorithm for Distributed Constrained Optimization. *Systems & Control Letters* 96 (2016), 110–117. doi:10.1016/j.sysconle.2016.07.009

[15] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan

Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations*. https://openreview.net/forum?id=zAdUB0aCTQ

[16] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2511–2522. doi:10.18653/v1/2023.emnlp-main.153

[17] Elissa Mhanna and Mohamad Assaad. 2023. Single Point-Based Distributed Zeroth-Order Optimization with a Non-Convex Stochastic Objective Function. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 24701–24719. https://proceedings.mlr.press/v202/mhanna23a.html

[18] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 1797–1807. doi:10.18653/v1/D18-1206

[19] NousResearch. 2024. Hermes Function-Calling V1. https://huggingface.co/datasets/NousResearch/hermes-function-calling-v1. Hugging Face dataset repository, accessed: 2026-06-12.

[20] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs from Preference Data. In *International Conference on Learning Representations*. https://openreview.net/forum?id=8sSqNntaMr

[21] OPUS Project. 2020. OPUS-100 Corpus. https://opus.nlpl.eu/opus-100.php. Accessed: 2026-06-12.

[22] Anit Kumar Sahu, Dusan Jakovetic, Dragana Bajovic, and Soummya Kar. 2018. Distributed Zeroth Order Optimization Over Random Networks: A Kiefer-Wolfowitz Stochastic Approximation Approach. In *Proceedings of the 57th IEEE Conference on Decision and Control*. IEEE, 4951–4958. doi:10.1109/CDC.2018.8619044

[23] Anna Sergeenko, Victoria Erofeeva, Oleg Granichin, Olga Granichina, and Anton Proskurnikov. 2021. Convergence Analysis of Weighted SPSA-based Consensus Algorithm in Distributed Parameter Estimation Problem. *IFAC-PapersOnLine* 54, 7, 126–131. 19th IFAC Symposium on System Identification SYSID 2021. doi:10.1016/j.ifacol.2021.08.346

[24] James C. Spall. 1992. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *IEEE Trans. Automat. Control* 37, 3 (1992), 332–341. doi:10.1109/9.119632

[25] Elizaveta Tarasova, Victoria Erofeeva, Oleg Granichin, and Kirill Chernikov. 2025. Decentralized Adaptive Task Allocation for Dynamic Multi-Agent Systems. *Scientific Reports* 15 (2025), 39226. doi:10.1038/s41598-025-21709-9

[26] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on Evaluation of LLM-based Agents. arXiv:2503.16416 [cs.AI] doi:10.48550/arXiv.2503.16416

[27] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, Vol. 36. https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html

[28] Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2025. Agent-as-a-Judge: Evaluate Agents with Agents. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*, Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (Eds.). PMLR, 80569–80611. https://proceedings.mlr.press/v267/zhuge25a.html

A Additional Modeling and Calibration Details

This appendix summarizes the modeling choices and calibration details omitted from the main text for compactness.

A.1 Task Complexity and Semantic Features

Each task T_i is associated with a semantic feature vector $\phi_i \in \mathbb{R}^5$, where $\phi_{i,\ell}$ is the normalized keyword-frequency score along task axis ℓ and $\sum_\ell \phi_{i,\ell} = 1$. In the main text, the prediction feature vector is restricted to the one-hot task-type encoding $x_i = e_{\text{type}_i} \in \mathbb{R}^5$, while semantic variation within a task type is represented separately through ϕ_i and the scalar complexity score

$$h_i = \text{clip}\big(2 + 4\phi_{i,\text{type}_i} + 2\ell_i + b_i^{\text{kw}},\ 1,\ 10\big),$$
$$\ell_i = \min(|w_i|/800,\ 1.5),$$
$$b_i^{\text{kw}} = 0.4|\mathcal{K}_i|,$$

Here $|w_i|$ denotes the prompt length in characters, and $\mathcal{K}_i$ is the set of technical complexity indicators present in the prompt. In the experiments, the keyword set is

$$\mathcal{K}_i \subseteq \{\texttt{optimize}, \texttt{pipeline}, \texttt{debug}, \texttt{production}, \texttt{integration}, \texttt{benchmark}\}.$$

Thus, h_i serves as a compact proxy for task difficulty, combining declared task type, prompt length, and the presence of complexity-related keywords.

A.2 Choice of Forgetting Factor in RLS

All three online predictors use recursive least squares with exponential forgetting factor $\beta = 0.99$. This value was chosen as a compromise between stability and responsiveness. Its effective memory window is approximately

$$\frac{1}{1-\beta} = 100$$

recent observations. With five task types and roughly balanced arrivals, this corresponds to approximately 20 recent examples per type, which is sufficient for stable linear updates while remaining responsive to gradual drift in agent behavior.

Larger values, such as $\beta = 0.999$, were found to react too slowly to changes in latency or output quality, while smaller values, such as $\beta = 0.95$, led to high-variance updates in the early phase, when the number of available samples per type is still limited.

A.3 Quality Signal Calibration

After task completion, the judge module computes a composite quality score

$$Q_i = \alpha_Q S_i + (1 - \alpha_Q)\mathcal{T}_i,$$

where $S_i \in \{0, 1\}$ indicates task success and

$$\mathcal{T}_i = \exp\Big(-\frac{\max(0,\ p_{i,k_i}^{\text{real}} - t_i^{\text{ref}})}{t_i^{\text{ref}}}\Big),$$
$$t_i^{\text{ref}} = \tau_{k_i}^{\text{avg}}(\text{type}_i)\big(1 + h_i/10\big).$$

The coefficient $\alpha_Q = 0.6$ was selected on a held-out calibration set. This choice reflects the asymmetric value of the two components: an incorrect response is typically useless regardless of latency, whereas a correct but moderately delayed response may still be valuable. In preliminary tests, smaller values of α_Q overemphasized timeliness, whereas larger values made latency almost irrelevant.

A.4 Quality Predictor Features

The online quality predictor uses the feature vector

$$z_{ik} = \big[e_{\text{type}_i}^\top,\ \phi_{i,\text{type}_i},\ \bar{\psi}_k,\ \hat{p}_i,\ \hat{w}_i\big]^\top \in \mathbb{R}^9,$$

where $e_{\text{type}_i} \in \mathbb{R}^5$ is the one-hot task-type indicator, $\phi_{i,\text{type}_i} \in [0, 1]$ is the semantic alignment score along the declared task axis,

$$\bar{\psi}_k = \frac{1}{5}\sum_{\ell=1}^{5} \psi_{k,\ell}$$

is the mean capability score of agent E_k, and $\hat{p}_i, \hat{w}_i$ are the current processing-time and wait-tolerance predictions. This design preserves type information while keeping the feature dimension small.

A.5 Routing Sub-Scores

The routing feature vector is $\mathbf{f} = [R, A, \hat{Q}, s_t, s_q, s_c, s_r, s_u]^\top$. Its components are defined as follows.

Semantic relevance.

$$R_{ki} = \frac{\phi_i^\top \psi_k}{\|\phi_i\| \, \|\psi_k\| + 10^{-8}}.$$

This is the cosine similarity between task semantics and the capability profile of agent E_k.

Availability and queue features.

$$A_{ki} = \exp\Big(-\frac{t_{\text{free}}}{\max(\hat{w}_i + 0.5,\ 0.5)}\Big), \qquad t_{\text{free}} = \hat{t}_k^{\text{avail}} - t \geq 0,$$
$$s_q = \exp\Big(-\frac{t_{\text{free}}}{\max(\hat{w}_i + 0.5,\ 0.5)}\Big).$$

Although A_{ki} and s_q share the same closed form, they are retained as separate features so that their aggregate contribution can be tuned through distinct weights during routing.

Processing-speed feature.

$$s_t = \exp\Big(-\frac{\hat{p}_{i,k}}{\max(\hat{p}_i + 0.5,\ 0.5)}\Big).$$

Here $\hat{p}_{i,k}$ is an agent-specific processing-time estimate. In the implementation, it is constructed as a convex combination of a type-level RLS prediction rescaled by the agent's relative latency and a profile-based estimate adjusted for prompt length and semantic mismatch.

Cost feature.

$$s_c = \exp(-c_k/0.3).$$

This form imposes an exponential penalty on expensive agents while preserving some differentiation across the tested price range.

Reputation feature.

$$s_r = \frac{1}{2} r_{jk}^{\text{local}}(\text{type}_i) + \frac{1}{2} r_k^{\text{global}}(\text{type}_i).$$

The local and global terms are updated as

$$r_{jk}^{\text{local}}(\ell) \leftarrow (1 - \alpha_t)\, r_{jk}^{\text{local}}(\ell) + \alpha_t Q_t, \qquad \alpha_t = 0.1 + 0.4 c_t,$$
$$r_k^{\text{global}}(\ell) \leftarrow 0.90\, r_k^{\text{global}}(\ell) + 0.10\, \text{clip}\big(0.55 \hat{\rho}_{k,\ell} + 0.45 P_k^{\text{succ}}(\ell),\ 0.05,\ 0.98\big),$$

where Q_t is the observed quality score of the completed task, $c_t \in [0, 1]$ is the judge confidence, $\hat{\rho}_{k,\ell}$ is the empirical posterior success estimate, and $P_k^{\text{succ}}(\ell)$ is the benchmark-based prior.

Urgency-aware deadline feature.

$$s_u = \exp\big(-(1 + 3\varsigma_i) d_k\big)\ \exp\big(-0.45 \varsigma_i \max(0, r_k - 1)\big).$$

This term penalizes deadline overruns and budget violations more strongly for urgent tasks.

A.6 Penalty Terms and Service Budget

The deadline gap and time-budget ratio are defined by

$$d_k = \frac{\max(0, \hat{t}_k^{\text{finish}} - D_i)}{b_i}, \qquad r_k = \frac{\hat{t}_k^{\text{avail}} - t + \hat{p}_{i,k}}{b_i},$$
$$\hat{t}_k^{\text{finish}} = \hat{t}_k^{\text{avail}} + \hat{p}_{i,k}, \qquad b_i = \max(\hat{p}_i + \hat{w}_i,\ 0.75),$$

where the floor of 0.75 in b_i prevents numerical instability for very short tasks.

The routing penalties use

$$\kappa_1 = 0.30 + 0.55 \varsigma_i, \qquad \kappa_2 = 0.08.$$

Thus, deadline overruns become more strongly penalized for urgent tasks, while time-budget overruns receive a smaller secondary penalty.

A.7 Weight Vector and Adaptive Blending

The baseline weight vector is

$$\tilde{\mathbf{w}} = (0.17, 0.10, 0.13, 0.20, 0.15, \omega_{\text{cost}}, 0.06, 0.19), \qquad \omega_{\text{cost}} = 0.12,$$

and the operational weight vector is obtained by ℓ_1 normalization:

$$\mathbf{w} = \frac{\tilde{\mathbf{w}}}{\|\tilde{\mathbf{w}}\|_1}.$$

Under high load, the routing policy blends $\mathbf{w}$ toward a throughput-oriented profile. System pressure is measured by

$$\pi = \text{clip}\Big(0.45 f_{\text{busy}} + 0.35 \frac{\min(q_{\text{press}}, 1.5)}{1.5} + 0.20 \varsigma_i,\ 0, 1\Big),$$
$$f_{\text{busy}} = \frac{1}{n} \sum_k \mathbf{1}[a_k > t],$$
$$q_{\text{press}} = \frac{1}{n} \sum_{k=1}^{n} \frac{\max(0, \hat{t}_k^{\text{avail}} - t)}{b_i + 0.5},$$

where f_{busy} is the fraction of agents currently busy and q_{press} is the mean normalized queue pressure. If $\pi > \theta_\pi = 0.62$, the blend factor is

$$\lambda = \frac{\pi - \theta_\pi}{1 - \theta_\pi},$$

and the effective weight vector becomes

$$\mathbf{w}_{\text{eff}} = (1 - \lambda)\mathbf{w} + \lambda \mathbf{w}_{\text{em}},$$

where

$$\mathbf{w}_{\text{em}} = (0.05, 0.08, 0.07, 0.28, 0.20, 0.02, 0.08, 0.24).$$

This emergency profile shifts mass away from quality and cost features and toward processing-speed and urgency-aware routing.

A.8 Choice of the Hinge Margin in F_2

The wait-tolerance objective is

$$F_2(\theta^{(w)}) = \mathbb{E}\Big[\max\big\{w_i^{\text{real}} + \delta - x_i^\top \theta^{(w)},\ 0\big\}^2\Big],$$

with $\delta = 0.1$ s. This margin serves two purposes. First, it reduces sensitivity to small measurement fluctuations in wait-time observations. Second, it introduces a mild conservative bias, encouraging the system to avoid overly tight deadlines. Smaller values of δ led to unstable behavior under noise, whereas larger values produced systematically slack wait-time predictions.

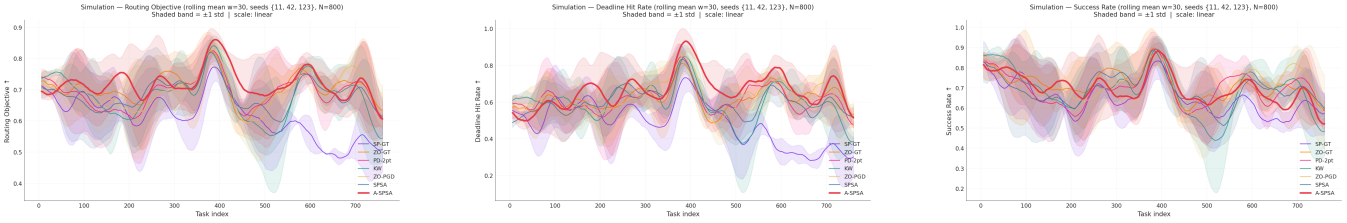

Figure 5: Evolution of numerical metrics over the task stream for seeds $\{11, 42, 123\}$.

A.9 Why These Details Are Omitted from the Main Text

The system model in the main paper is intentionally limited to the core definitions required to understand the routing mechanism and the optimization objectives. The detailed forms of the routing sub-scores, reputation updates, coefficient choices, and blending heuristic are included here to document the full implementation while keeping the main text concise.

B Additional Experimental Details

B.1 Task Generation and Agent Profiles

In the numerical experiments, task types are sampled with probabilities 0.25, 0.20, 0.20, 0.15, and 0.20 for PROGRAMMING, QA, SUMMARIZATION, TRANSLATION, and TOOL_USE, respectively. Prompts are sampled without replacement within each run from the datasets listed in Table 3. Each task is additionally assigned a complexity score sampled uniformly from $[1, 5]$, an urgency value in $[0, 1]$, a semantic feature vector, and an arrival time generated from exponentially distributed inter-arrival intervals.

Table 3: Task sources used in the numerical experiments.

Task type	Dataset	Pool size
PROGRAMMING	MBPP [5]	974
QA	TriviaQA [13] unfiltered dev	11 322
SUMMARIZATION	XSum [18] test	11 334
TRANSLATION	OPUS-100 [21] en–fr test	2 000
TOOL_USE	Hermes-FC [19] train	1 893

Each agent is represented by a task-dependent latency, capability, and semantic profile. The baseline latency and maximum output length are summarized in Table 4.

Table 4: Agent profiles used in the numerical experiments.

Agent	Baseline latency (s)	Maximum tokens
llama3-8b	1.5 ± 0.4	80
mistral-7b	2.0 ± 0.5	180
qwen2.5-72b	2.0 ± 0.5	300
qwen2.5-tooluse	2.5 ± 0.6	380
deepseek-coder	2.5 ± 0.6	512

Table 5: Hyperparameters selected on the independent validation stream.

Method	a	b	$\beta_{\max}$
A-SPSA	0.10	0.30	0.30
SPSA	0.70	0.05	—
KW	1.00	0.30	—
ZO-PGD	0.50	0.10	—
ZO-GT	0.10	0.30	—
PD-2pt	0.05	0.10	—
SP-GT	0.10	0.30	—

For task T_i assigned to agent E_k, the service time is generated as

$$p_{ik}^{\text{real}} = \tau_k(\text{type}_i)\big(1 + 0.07h_i\big)\big(1 + 0.10 \log n_i\big)\big(1 + 0.12 d_{ik}^{\text{sem}}\big)\varepsilon_{ik},$$

where h_i is task complexity, n_i is request size, d_{ik}^{sem} is semantic mismatch, and ε_{ik} follows a log-normal distribution with parameters $(0, 0.16)$. With probability 0.08, the result is additionally multiplied by a random factor sampled uniformly from $[1.25, 1.9]$.

The task-success probability depends on task-type capability, semantic match, task complexity, waiting time, and urgency, with an additional Gaussian perturbation. Agent profiles are calibrated so that faster agents have smaller output budgets and lower expected success probabilities.

B.2 Hyperparameter Selection

Hyperparameters are selected independently for each method on a validation stream that does not overlap with the test runs. The validation stream contains 100 tasks. The search considers $a \in \{0.05, 0.1, 0.2, 0.5, 0.7, 1.0\}$ and $b \in \{0.05, 0.1, 0.2, 0.3\}$. For A-SPSA, the search additionally includes $\beta_{\max} \in \{0.3, 0.5, 0.7\}$. The configuration maximizing validation Success Rate is fixed before the main evaluation. The resulting values are reported in Table 5.

B.3 Numerical Adaptation Trajectories

Figure 5 shows the evolution of Routing Objective, Deadline Hit Rate, and Success Rate over $N = 800$ tasks for seeds $\{11, 42, 123\}$. The curves report rolling means with a window of 30 tasks, while shaded regions indicate one standard deviation.

The methods exhibit similar short-term fluctuations because they process the same task streams. Over longer intervals, A-SPSA generally remains in the upper part of the comparison group and recovers after local performance drops, whereas SP-GT and, on some intervals, KW exhibit prolonged degradation. These per-seed

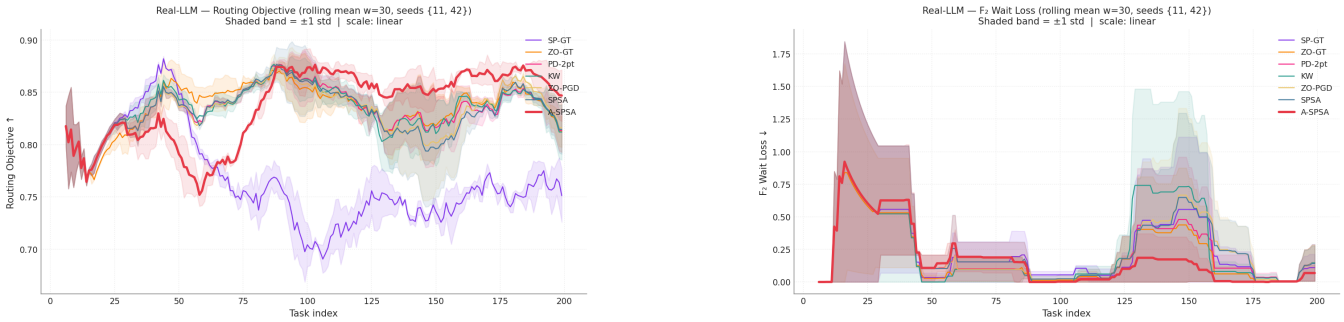

Figure 6: Supplementary real-LLM adaptation trajectories. Curves show rolling means with a window of 30 tasks; shaded regions show one standard deviation over seeds $\{11, 42\}$**.**

trajectories provide the temporal view underlying the aggregate summary in Figure 1: rather than reporting only a single terminal value, they show how each method's Routing Objective, Deadline Hit Rate, and Success Rate evolve as the rolling window accumulates observations. The relative ordering of methods visible here is consistent with the seed-aggregated ranking reported in Section 5.1.

B.4 Additional Real-LLM Results

During pre-collection, each of the 200 tasks is submitted to all five Groq endpoints, producing 200×5 cached task–agent outcomes per task stream. Each cached record includes the generated response and judge-based quality score. During replay, latency is sampled from the corresponding controlled agent profile.

Figure 6 reports the supplementary Routing Objective and Wait MSE trajectories. Together with Figure 4, they illustrate the initial A-SPSA adaptation transient and its subsequent recovery.

Taken together with Figure 4 in the main text, these supplementary trajectories indicate that A-SPSA's numerical-simulation advantage extends to real generated content: after the initial adaptation transient, Routing Objective and Wait MSE stabilize in the same relative ordering observed in Table 2, suggesting that the method's behavior in Section 5.4 is not an artifact of the synthetic agent profiles used in the numerical evaluation.