SAGE: LOOKAHEAD ACTION GATING FOR SAFE, BUDGET-AWARE EDGE–CLOUD AI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous AI agents execute requests through multi-step trajectories in which each action can be generated by a fast local edge-based small language model (SLM) or a stronger cloud large language model (LLM). Existing hybrid approaches primarily select models to ensure the same output quality to stronger LLM in each step or preserve trajectories from LLMs, without jointly considering the recoverability of an SLM's action in the future trajectory and its long-term impact on end-to-end latency and cloud cost. An imperfect SLM action may remain recoverable by future LLM actions, while a seemingly faster SLM step may induce additional steps and ultimately incur higher end-to-end latency than escalating to the LLM immediately. Moreover, using LLM now may leave insufficient budget for more important future escalations. To address the problems, we present a lookahead Safe and budget-aware Action Gating system for Edge-cloud agents (SAGE), that accounts for these long-term effects on the future trajectory under end-to-end latency and cloud-cost budgets. First, it estimates whether accepting SLM preserves the agent's ability to successfully complete the request. Second, it jointly considers this safety estimate, the remaining budgets, and anticipated future resource needs to decide whether to execute the local action or escalate to the cloud LLM. Across two SLM/LLM pairs and six benchmarks, SAGE preserves 96.87–98.06% of *LLM-Only* accuracy while reducing normalized cloud cost to 0.41–0.45, and reducing mean latency by 25.9–28.6%. Compared with a state-of-the-art method, SAGE improves accuracy by 3.21–3.26 percentage points, reduces mean latency by 35.6%, and reduces normalized cloud cost by 37.5–40.6%.

1 INTRODUCTION

LLMs enable autonomous AI agents that solve requests through iterative reasoning, generation, tool use, and interaction (Ouyang et al., 2022; Wei et al., 2022a; Significant-Gravitas, 2023; Wu et al., 2024; LangChain, 2023; AgentGPT, 2023; BabyAGI, 2023). Executing every step with a powerful cloud LLM can provide high quality but incurs API cost and network latency. Local edge-based SLMs, in contrast, can execute many steps with lower per-call latency and no cloud API charge (Chen et al., 2023; Ding et al., 2023; Yin et al., 2026; Šakota et al., 2024; Aggarwal et al., 2024; Yue et al., 2024; Ramírez et al., 2024; Narasimhan et al., 2025). This creates a natural hybrid design: use the SLM when possible and escalate to the LLM when necessary.

The challenge is that the effect of a routing decision is not confined to the current step. First, current-step quality does not directly determine final success. An SLM action that appears less accurate may still keep the trajectory recoverable, allowing later LLM actions to correct or complete the task. Conversely, an apparently plausible local action may move the agent into a state from which subsequent recovery is difficult. Second, current-step latency does not determine end-to-end latency. Although one SLM call is often faster than one LLM call, a weaker local action may require additional reasoning, retries, or tool interactions; several such steps can take longer than escalating to a stronger LLM immediately. Finally, cloud calls consume a finite budget: escalating now may improve the current step but leave insufficient latency or cloud-spend budget for more important future steps. Thus, a router should consider how its current decision affects future success and resource availability, rather than optimize only immediate quality, latency, or cost.

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.

Our pilot study reveals three key observations:
(1) independently plausible SLM outputs are not necessarily safe for the overall trajectory; we define an SLM action as *safe* if, after executing it, the agent can still successfully complete the request;
(2) local execution does not always reduce end-to-end latency, while LLM execution can incur substantial latency for modest accuracy gains; and
(3) later or recovery-limited steps are more likely to require LLM escalation, motivating end-to-end latency and cloud-cost budget reservation for future steps.
These observations motivate joint consideration of action recoverability and trajectory-level resource constraints. Accordingly, we present a lookahead Safe and budget-aware Action Gating system for Edge-cloud agents (SAGE), that accounts for these long-term effects on the future trajectory under end-to-end latency and cloud-cost budgets. First, it estimates whether accepting SLM preserves the agent's ability to successfully complete the request. Second, it jointly considers this safety estimate, the remaining budgets, and anticipated future resource needs to decide whether to execute the local action or escalate to the cloud LLM. In this way, SAGE makes each routing decision based not only on its immediate consequence but also on its expected impact on the remaining execution trajectory.

Most existing methods do not jointly consider the recoverability of a realized SLM action and its long-term impact on latency and cloud cost. Traditional hybrid LLM systems route queries based on predicted quality or confidence (Ong et al., 2025; Ding et al., 2023; Ramírez et al., 2024; Chen et al., 2023; Yue et al., 2024). Recent methods consider multi-step execution: Router-R1 learns to iteratively reason, invoke models, and aggregate their responses to optimize task performance and inference cost (Zhang et al., 2025), while SWE-Router uses a partial trajectory from several weak-model turns to decide whether to continue or switch to a stronger model (Son et al., 2026). MTRouter uses the execution history and candidate models to predict final task performance and selects a model under a total cost budget (Zhang et al., 2026). However, it evaluates candidate models based on the execution history without explicitly considering the current action or end-to-end latency. In contrast, SAGE considers both the current action and execution state to estimate the trajectory-level consequences, including future recoverability, latency, and cloud cost, of executing that *specific action* on SLM.

We evaluate SAGE on six benchmarks using two edge/cloud model pairs. It preserves 96.87–98.06% of *LLM-Only* accuracy while reducing normalized cloud cost to 0.41–0.45, and reducing mean latency by 25.9–28.6%. Compared with a state-of-the-art method, SAGE improves accuracy by 3.21–3.26 percentage points, reduces mean latency by 35.6%, and reduces normalized cloud cost by 37.5–40.6%.

Our contributions are three-fold:

- We formulate hybrid-agent routing as *action-level safe acceptance under end-to-end latency and cloud-cost budgets*, highlighting that a current routing decision affects future recoverability, execution latency, and available cloud budget.
- We design SAGE, which estimates the recoverability of an SLM action and combines it with runtime validity, execution state, and latency/cloud-spend constraints to make budget-aware accept-or-escalate decisions.
- We evaluate SAGE across six agent benchmarks and two SLM/LLM pairs, showing that it retains most of the cloud-only accuracy while moving a majority of generated tokens local and substantially reducing latency and cloud cost.

2 PILOT STUDY

This section provides the empirical grounding for SAGE using diagnostic pilot traces from HotPotQA (Yang et al., 2018) and GSM8K (Cobbe et al., 2021a). These benchmarks capture multi-step reasoning in which each request unfolds through intermediate states, tool-like operations, and partial outputs. Qwen3-4B runs locally on a single NVIDIA RTX 5090 host, and GPT-5 is accessed remotely through API calls. We compare six simple routing strategies. *SLM-Only* executes every step using the local model. *LLM-Only* executes every step using the cloud model. *Random* chooses a model independently at random for each step. *Hybrid-S* applies HybridLLM (Ding et al., 2023) independently to each step. *Hybrid-R* applies HybridLLM once to a request and uses the resulting model choice for all execution steps. *Oracle* is an offline upper bound that replays each trace and selects the cheapest step allocation that preserves 95% of the *LLM-Only* accuracy on the

Table 1: Pilot results on HotPotQA.

Scenario	Acc. (%)	Lat. (s)	LSSR (%)	Local (%)	Steps
SLM-Only	29.3	40.04	78.5	100.00	6.07
LLM-Only	45.1	52.15	92.7	0.00	4.23
Random	34.2	42.59	75.2	50.18	5.85
Hybrid-S	36.1	53.14	85.9	47.84	6.12
Hybrid-R	35.4	44.19	86.3	18.32	4.88
Oracle	43.3	42.42	100.0	53.76	4.54

Table 2: Pilot results on GSM8K.

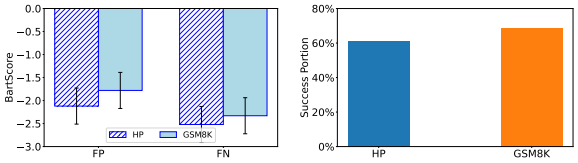
Scenario	Acc. (%)	Lat. (s)	LSSR (%)	Local (%)	Steps
SLM-Only	38.4	23.32	88.5	100.00	7.01
LLM-Only	85.3	36.25	95.9	0.00	5.31
Random	52.5	28.33	92.2	49.78	6.72
Hybrid-S	60.1	34.21	94.3	46.33	7.02
Hybrid-R	68.2	31.93	93.3	27.45	6.02
Oracle	83.4	26.73	100.0	62.76	5.82

corresponding dataset. For each request, we report end-to-end exact-match accuracy, wall-clock latency, the latency service-level-objective (SLO) satisfaction ratio (LSSR), the local-step ratio, and the average number of generated steps. We set the latency SLO of each request to twice of its average inference latency as in Li et al. (2023).

2.1 Safe Acceptance and Local Execution Opportunities

Tables 1 and 2 show a consistent gap between the routing methods and Oracle. *Random* routing provides only modest gains over *SLM-Only*, while *Hybrid-S* and *Hybrid-R* improve accuracy but remain substantially below Oracle. On HotPotQA, *Hybrid-S* reaches 36.1% accuracy versus 43.3% for the Oracle, despite Oracle using more local execution and achieving lower latency. On GSM8K, *Hybrid-S* and *Hybrid-R* reach 60.1% and 68.2%, respectively, compared with 83.4% for Oracle. The results show that an independently plausible action is not necessarily safe for the full execution trajectory.

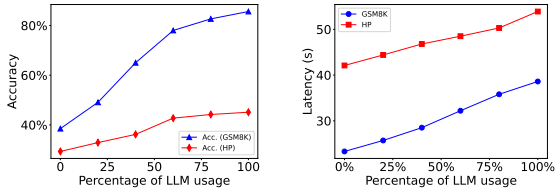
To make this concrete, we compare the decisions of *Hybrid-S* with *Oracle*. False positives (FP) occur when *Hybrid-S* keeps a step local but *Oracle* would escalate; false negatives (FN) do the reverse. We use BARTScore (Yuan et al., 2021) to measure local similarity between the SLM and LLM outputs at the step of FP and FN, respectively. Figure 1a shows strong overlap between the two distributions. High local similarity therefore does not reliably tell the controller if SLM is safe for the end-to-end outcome.



(a) Hybrid-S–Oracle FP/FN local output similarity. (b) Safe local assignment ratio: Oracle vs. LLM-Only.

Figure 1: FP/FN local output similarity and safe local execution opportunities.

Figure 1b shows the successful local assignment rate of the Oracle relative to *LLM-Only*. Oracle keeps 53.76% of HotPotQA steps and 62.76% of GSM8K steps local while remaining close to *LLM-Only* accuracy. In other words, there is clear opportunity to reduce cloud dependence and therefore reduce both cloud cost and remote communication load.



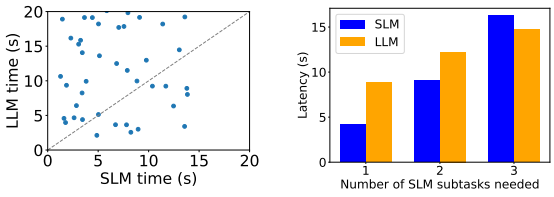
(a) Accuracy vs. LLM use. (b) Latency vs. LLM use.

Figure 2: Accuracy and latency versus LLM usage.

Observation 1. Local execution opportunities exist, but independent step scoring is insufficient because a plausible SLM output may not be safe for the overall trajectory.

2.2 Per-Step Latency Does Not Determine End-To-End Latency

Figures 2a and 2b show that accuracy improves non-linearly as LLM usage increases, while latency rises more steadily. After moderate LLM usage, additional cloud calls buy smaller accuracy gains while still paying the network and API overhead of remote inference. A useful controller must therefore decide *which* steps are safe to keep local.



(a) Single-subtask latency. (b) Multi-subtask latency.

Figure 3: Single- and multi-subtask latency.

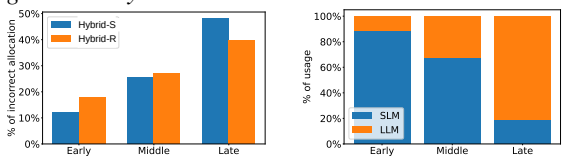
The one-step and multi-step latency plots in Figures 3a and 3b show that one local step is often faster than one remote step, but several local steps may be needed to do work that one remote step

can finish immediately. This means that local step does not automatically imply lower end-to-end latency once extra steps and extra tool iterations are counted.

Observation 2. *Local actions do not necessarily reduce end-to-end latency, as they may introduce additional steps that erase their per-step latency advantage. Conversely, LLM actions may provide only modest accuracy gains at substantially higher latency.*

2.3 Budget-Awareness

Whether to accept a local assignment depends on where the agent is in the trajectory and the remaining cloud budget available for recovery if needed. The stage-wise incorrect-assignment and Oracle model-usage results in Figures 4a and 4b show that incorrect decisions by *Hybrid-S* and *Hybrid-R* are more concentrated in later stages of execution, while Oracle increasingly favors the LLM as execution approaches completion. The results indicate that routing should account for how recoverable a local action is and whether sufficient cloud budget remains for future recovery.



(a) Incorrect decisions. (b) Oracle allocation.

Figure 4: Routing errors and Oracle model allocation.

Observation 3. *Accepting an SLM action depends on its position in the trajectory: later or recovery-limited states are more likely to require LLM escalation, motivating sufficient budget reservation for future steps.*

3 Methodology

The three observations suggest the design principle. SAGE should not allocate models only by local similarity or a fixed schedule. It should inspect the actual action that would be executed, estimate whether keeping that action local is safe for the overall task, and interpret that estimate through the remaining latency and cloud-cost budgets.

3.1 Problem Formulation

A request is executed as a sequence of structured actions. At step t, the controller observes $s_t = (x, \chi_t, \mathcal{H}_t, \mathcal{O}_t, b_t^{\text{lat}}, b_t^{\text{cost}}, m_t)$, where x is the user request, χ_t is the current compact state summary, $\mathcal{H}_t$ is the recent action history, $\mathcal{O}_t$ is the current tool state, b_t^{lat} and b_t^{cost} are the remaining latency and cloud-cost budgets, and m_t contains runtime metadata including the normalized step index, retry count, and action type. Each action follows a common schema $a_t = (\text{type}, \text{tool}, \text{args}, \text{text})$ so that the controller can evaluate model outputs before execution.

SAGE first obtains a local draft $a_t^S = M_S(s_t)$, and then decides whether to execute a_t^S or replace it with a cloud action $a_t^L = M_L(s_t)$. The local draft and verifier calls are part of the controller, so their latency (ℓ_t^{probe}) is charged to the remaining latency budget at the beginning of step t (b_t^{lat}) before the accept-or-escalate decision: $\bar{b}_t^{\text{lat}} = b_t^{\text{lat}} - \ell_t^{\text{probe}}$. However, the local SLM and verifier computation do not consume the cloud-cost budget: $\bar{b}_t^{\text{cost}} = b_t^{\text{cost}}$. The controller therefore compares accepting the local draft against the additional latency and remote spend incurred by escalating to the cloud.

SAGE is a draft–verify–route controller with two online steps: (1) estimate its safe-accept probability from model confidence and deterministic runtime signals (§3.2); and (2) compare that probability with a impact- and budget-aware threshold to accept the local action, escalates to the cloud, or falls back (§3.3). The safety/impact predictors are trained offline from replay traces (§3.4).

3.2 Safety Estimation

Verification metrics. SAGE generates one executable local draft a_t^S and K additional stochastic drafts $\tilde{a}_{t,1}, \ldots, \tilde{a}_{t,K}$ with the same prompt and different sampling seeds. Each action is normalized as $\sigma(a) = (\text{type}, \text{tool}, \text{canon}(\text{args}))$, where canon removes superficial argument-format differences. Two confidence signals are: $r_t^{\text{agree}} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{1}\big[\sigma(\tilde{a}_{t,k}) = \sigma(a_t^S)\big]$. Here, $\mathbf{1}[\cdot]$ is an indicator function that equals 1 when the two normalized actions match and 0 otherwise. Thus, $r_t^{\text{agree}} \in [0, 1]$ is the fraction of the K additional samples that agree with the executable local action. A larger value indicates that the local model produces a more consistent action under repeated sampling.

Second, we measure the model's *token-level confidence* in a_t^S. Let $y_1, \ldots, y_{n_t}$ denote the tokens of a_t^S. We compute $r_t^{\text{tok}} = \exp\Big(\frac{1}{n_t}\sum_{i=1}^{n_t} \log P(y_i \mid s_t, y_{<i})\Big)$, where $P(y_i \mid s_t, y_{<i})$ is the probability assigned by the local model to the generated token y_i, conditioned on the current state s_t and the preceding generated tokens $y_{<i}$. This quantity is the geometric mean of the token probabilities in a_t^S; a larger value indicates that the local model assigns higher probability to the action it generated. The two signals capture complementary information: r_t^{agree} measures the stability of the action across independent samples, whereas r_t^{tok} measures the model's confidence in the particular executable action a_t^S.

SAGE then asks the local model three binary questions. The first, *safe*, asks: "If a_t^S is executed now, will the request remain solvable within the remaining budgets?" The second, *progress*, asks: "Will a_t^S reduce the unresolved objective set?" The third, *remote-helpful*, asks: "Would replacing a_t^S with a remote-model step materially increase the likelihood of success?" For question q_j and sample k, it records $p_{t,j}^{(k)} = P(\texttt{Yes} \mid s_t, a_t^S, q_j)$. For each of the three questions, SAGE keeps the mean, minimum, and variance across the K samples: $\mu_{t,j} = \frac{1}{K}\sum_{k=1}^{K} p_{t,j}^{(k)}$, $m_{t,j} = \min_k p_{t,j}^{(k)}$, $\nu_{t,j} = \mathrm{Var}_k[p_{t,j}^{(k)}]$. The verifier block therefore contributes eleven model-based features: $v_t^{\text{model}} = [\, r_t^{\text{agree}}, r_t^{\text{tok}}, \mu_{t,1}, m_{t,1}, \nu_{t,1}, \ \mu_{t,2}, m_{t,2}, \nu_{t,2}, \mu_{t,3}, m_{t,3}, \nu_{t,3} \,]$.

Runtime verification. A thin adapter maps each benchmark to $a = (\text{type}, \text{tool}, \text{args}, \text{text})$. SAGE then adds deterministic signals for validity, executability, consistency, progress, and loops.

Runtime signals. SAGE augments model confidence with eight deterministic runtime signals that check whether the local draft a_t^S is valid, executable, consistent with the current state, and making progress. Let $\mathcal{R}(a)$ be a's required arguments, and $\mathcal{E}(a)$ be the state references it uses. The first four signals check whether the draft can be parsed, contains its required arguments, invokes an available tool, and uses valid state references, respectively:

$$r_t^{\text{schema}} = \mathbf{1}[\text{Parse}(a_t^S) \text{ succeeds}], \qquad r_t^{\text{tool}} = \mathbf{1}[\text{tool}(a_t^S) \in \mathcal{O}_t \text{ and enabled}],$$
$$r_t^{\text{args}} = \begin{cases} 1, & |\mathcal{R}(a_t^S)| = 0, \\ \frac{|\{u \in \mathcal{R}(a_t^S) : u \text{ present}\}|}{|\mathcal{R}(a_t^S)|}, & \text{otherwise}, \end{cases} \qquad r_t^{\text{ref}} = \begin{cases} 1, & |\mathcal{E}(a_t^S)| = 0, \\ \frac{|\{e \in \mathcal{E}(a_t^S) : e \text{ resolves}\}|}{|\mathcal{E}(a_t^S)|}, & \text{otherwise}. \end{cases} \tag{1}$$

If parsing fails, the schema-dependent signals are set to zero and the draft is rejected immediately. The next two signals measure state consistency and expected progress: $r_t^{\text{cons}} = 1 - \frac{N_t^{\text{viol}}}{N_t^{\text{chk}} + \epsilon}$, $r_t^{\text{prog}} = \frac{|U_t| - |U_t^{(a)}|}{|U_t| + \epsilon}$. N_t^{chk} and N_t^{viol} are the numbers of applicable and violated runtime constraints, respectively. U_t is the set of unresolved objectives (e.g., unanswered subquestions, unresolved variables, pending tool results, or finalization steps), and $U_t^{(a)}$ is the predicted set after executing a_t^S. A lightweight symbolic transition computes $U_t^{(a)}$ by checking action preconditions and updating the corresponding state; no additional model call is required. Finally, SAGE detects repeated actions and two-action cycles:

$$r_t^{\text{loop1}} = \max_{1 \le i \le W} \mathbf{1}[\sigma(a_t^S) = \sigma(a_{t-i})], \quad r_t^{\text{loop2}} = \max_{1 \le i \le W-1} \mathbf{1}\big[(\sigma(a_{t-1}), \sigma(a_t^S)) = (\sigma(a_{t-i-1}), \sigma(a_{t-i}))\big]. \tag{2}$$

Here, r_t^{loop1} identifies repetition of a recent action, while r_t^{loop2} identifies repetition of a recent two-action pattern.

SAGE also records a compact state descriptor capturing execution progress, remaining budgets, network conditions, and recent failures:

$$u_t = [\, \text{step_ratio}, \text{retry_count}, \text{side_effect_flag}, \text{final_flag}, b_t^{\text{lat}}/B_0^{\text{lat}}, b_t^{\text{cost}}/B_0^{\text{cost}}, \widehat{\text{RTT}}_t^{50}, \widehat{\text{RTT}}_t^{95}, \text{recent_tool_fail}, \text{recent_null_progress} \,]. \tag{3}$$

Here, step_ratio measures progress through the episode, and retry_count records the number of retries. The binary features side_effect_flag and final_flag indicate whether the proposed action can change external state (e.g., writing a file) and whether it is a final response, respectively. An incorrect action with a side effect may be harder to recover from. B_0^{lat} and B_0^{cost} are the end-to-end latency and cloud-cost budgets. The ratios $b_t^{\text{lat}}/B_0^{\text{lat}}$ and $b_t^{\text{cost}}/B_0^{\text{cost}}$ give the fractions of the initial latency and cloud-cost budgets remaining at step t. $\widehat{\text{RTT}}_t^{50}$ and $\widehat{\text{RTT}}_t^{95}$ are the estimated median and 95th-percentile cloud round-trip times. Finally, recent_tool_fail and $\text{recent_null_progress}$ indicate recent tool failure and lack of progress, respectively.

The full feature vector combines model confidence, runtime signals, and state information: $f_t = [\, v_t^{\text{model}}, r_t^{\text{schema}}, r_t^{\text{args}}, r_t^{\text{tool}}, r_t^{\text{ref}}, r_t^{\text{cons}}, r_t^{\text{prog}}, r_t^{\text{loop1}}, r_t^{\text{loop2}}, u_t \,]$. SAGE uses an XGBoost classifier to fuse these features, $\hat{p}_t = g_{\text{xgb}}(f_t)$, followed by isotonic calibration, $\tilde{p}_t = h_{\text{iso}}(\hat{p}_t)$. The resulting $\tilde{p}_t$ estimates the probability that accepting the local draft still allows the task to succeed within budget: $p_t^{\text{ref}} = P\big(\text{Replay}(s_t, a_t^S, \pi_{\text{ref}}) \text{ succeeds within budget} \mid f_t\big)$. Here, π_{ref} is the *LLM-Only* continuation policy: replay first forces a_t^S at step t and then uses the remote model M_L for all remaining steps under the same runtime and budgets. Thus, $\tilde{p}_t$ measures whether the local draft is safe to accept under subsequent cloud assistance, rather than its model-independent correctness. XGBoost is used because f_t is a compact tabular representation with short-history information already captured by u_t and the loop signals. Parsing failure causes immediate rejection.

3.3 Impact- and Budget-Aware Routing

The goal of SAGE's budget-aware routing policy is to decide whether to accept the local draft or escalate to the cloud while preserving the remaining latency and cloud-cost budgets. It combines the safe-accept probability $\tilde{p}_t$ with state impact and the resource burden of cloud escalation to derive a dynamic acceptance threshold.

Impact and budget reservation. SAGE uses a separate impact model to estimate the damage caused by incorrectly accepting a local action. Its feature vector contains state and action-type features:

$$u_t^{\text{crit}} = [\,\text{step_ratio}, \text{retry_count}, \text{side_effect_flag}, \text{final_flag}, |U_t|, \\ b_t^{\text{lat}}/B_0^{\text{lat}}, b_t^{\text{cost}}/B_0^{\text{cost}}, \widehat{\text{RTT}}_t^{95}, \text{recent_tool_fail}, \text{recent_null_progress}\,]. \tag{4}$$

The impact score is produced by an ℓ_2-regularized logistic regressor $c_t = \sigma(w_c^\top u_t^{\text{crit}})$. We use a linear impact model so its coefficients reveal how factors such as task progress, retries, side effects, and remaining recovery opportunities affect the controller's decisions. A larger c_t raises the safety threshold for accepting a local draft.

To avoid exhausting the cloud budget before later critical steps, SAGE also reserves resources for likely future escalations. We index the current state by a stage bucket $b_t = (\text{progress decile}, \text{action type})$ and store the 0.9-quantile of the number of future critical escalations in the training traces: $N_{\text{crit,rem}}(b_t) = Q_{0.9}[\#\text{ future critical remote steps} \mid b_t]$. Thus, $N_{\text{crit,rem}}(b_t)$ estimates a conservative number of critical steps that are likely to require cloud escalation after the current stage b_t. The corresponding reserves are $R_t^{\text{cost}} = N_{\text{crit,rem}}(b_t)\,\bar{C}_L$, $R_t^{\text{lat}} = N_{\text{crit,rem}}(b_t)\,\bar{L}_L$, where $\bar{C}_L$ and $\bar{L}_L$ are the average cost and latency of one remote step. The budgets available for the current escalation after preserving the reserve are $\tilde{b}_t^{\text{cost}} = \max(\bar{b}_t^{\text{cost}} - R_t^{\text{cost}}, 0)$, $\tilde{b}_t^{\text{lat}} = \max(\bar{b}_t^{\text{lat}} - R_t^{\text{lat}}, 0)$.

Escalation burden. To estimate the incremental burden of escalating this step, SAGE predicts the remote output length ($\hat{n}_t^{L,\text{out}}$) with an exponential-moving-average table keyed by $(\text{model pair}, \text{action type}, \text{prompt length bucket})$. It then computes $\hat{C}_t^{\uparrow} = \pi_L^{\text{in}} n_t^{\text{in}} + \pi_L^{\text{out}} \hat{n}_t^{L,\text{out}}$, $\hat{L}_t^{\uparrow} = \widehat{\text{RTT}}_t^{95} + \widehat{\text{Queue}}_t + \frac{\hat{n}_t^{L,\text{out}}}{\hat{\rho}_t^L}$, where n_t^{in} is the number of input tokens, π_L^{in} and π_L^{out} are the cloud prices per input and output token, respectively, and $\hat{\rho}_t^L$ is the estimated cloud generation rate in tokens per second. SAGE summarizes the relative burden of escalation as $\phi_t = \lambda_c \frac{\hat{C}_t^{\uparrow}}{\bar{b}_t^{\text{cost}} + \epsilon} + \lambda_l \frac{\hat{L}_t^{\uparrow}}{\bar{b}_t^{\text{lat}} + \epsilon}$.

Dynamic acceptance threshold. Critical states receive a larger risk weight, $w_t = 1 + \alpha c_t + \beta\,\mathbf{1}[\text{side_effect_flag}] + \gamma\,\mathbf{1}[\text{final_flag}]$, and SAGE sets $\tau_t^\star = 1 - \frac{\phi_t}{w_t}$, $\tau_t = \text{clip}\,(\tau_t^\star, \tau_{\min}, \tau_{\max})$. Thus, when cloud escalation is affordable (ϕ_t is small), SAGE requires high local safety; when cloud resources are scarce (ϕ_t is large), it accepts more local drafts to conserve budget. Higher impact increases the threshold and makes local acceptance more conservative. Without clipping, the rule $\tilde{p}_t \geq \tau_t$ is equivalent to $w_t(1 - \tilde{p}_t) \leq \phi_t$, which compares the risk of local acceptance with the resource burden of cloud escalation.

Final routing decision. SAGE first applies deterministic checks. It marks an action as a hard failure if the action cannot be parsed, lacks a required argument, calls an unavailable tool, references an unresolved state object, or repeats a two-step loop:

$$\mathsf{HF}_t = \mathbf{1}\begin{bmatrix} r_t^{\text{schema}} = 0 \ \vee \ r_t^{\text{args}} < 1 \ \vee \ r_t^{\text{tool}} = 0 \\ \vee \ r_t^{\text{ref}} < 1 \ \vee \ r_t^{\text{loop2}} = 1 \end{bmatrix}. \tag{5}$$

Cloud escalation is feasible only if it fits the non-reserved budgets: $\mathsf{F}_t = \mathbf{1}\left[\hat{C}_t^{\uparrow} \le \tilde{b}_t^{\text{cost}} \ \wedge \ \hat{L}_t^{\uparrow} \le \tilde{b}_t^{\text{lat}}\right]$. The final decision is

$$a_t = \begin{cases} M_L(s_t), & \text{if } \mathsf{HF}_t \wedge \mathsf{F}_t, \\ \text{SafeFallback}(s_t), & \text{if } \mathsf{HF}_t \wedge \neg\mathsf{F}_t, \\ a_t^S, & \text{if } \neg\mathsf{HF}_t \wedge \tilde{p}_t \ge \tau_t, \\ M_L(s_t), & \text{if } \neg\mathsf{HF}_t \wedge \tilde{p}_t < \tau_t \wedge \mathsf{F}_t, \\ \text{SafeFallback}(s_t), & \text{otherwise.} \end{cases} \tag{6}$$

SAFEFALLBACK attempts deterministic parser repair when possible; otherwise it terminates without executing an unsafe action.

3.4 TRAINING AND OVERHEAD

SAGE trains only an XGBoost safety predictor, a logistic-regression impact predictor, and an isotonic safety calibrator; all other equations are deterministic feature or routing computations. From logged state s_t, replay under the fixed LLM continuation policy gives

$$y_t = \mathbf{1}[\text{Replay}(s_t, a_t^S, \pi_{\text{ref}}) \text{ succeeds within budget}], \tag{7}$$

and impact uses

$$z_t = \mathbf{1}[\text{Replay}(s_t, a_t^S, \pi_{\text{ref}}^{(1)}) \text{ fails within horizon } h = 2], \tag{8}$$

where $\pi_{\text{ref}}^{(1)}$ takes one corrective remote step before resuming π_{ref}. Requests are split 70/15/15% for train/validation/test. XGBoost uses 200 depth-4 trees and learning rate 0.05; the impact model uses ℓ_2 regularization, and routing hyperparameters are selected on validation data.

The controller is < 1 MB and executes on CPU in sub-millisecond time; offline overhead is dominated by replay label generation. Online overhead comes from the executable local draft and K verification samples. Escalated steps therefore pay both local probing and remote inference; ℓ_t^{probe} is deducted before routing, and all local/verifier time is included in reported latency.

4 EVALUATION

SAGE is implemented around AutoGen (Wu et al., 2024). We use Qwen3-4B (Qwen Team, 2025) or Gemma 3 4B (Gemma Team, 2025) locally on one RTX 5090 and GPT-5 (OpenAI, 2025) or Claude Sonnet 4.5 (Anthropic, 2025) remotely through provider APIs. All local drafting/verifier time is included in latency, while cloud cost counts remote API calls only.

We evaluate SAGE on benchmarks spanning math reasoning (GSM8K (Cobbe et al., 2021b)), multi-hop QA (HotPotQA (Yang et al., 2018)), discrete numerical reasoning (DROP (Dua et al., 2019)), code generation (HumanEval (Chen et al., 2021)), interactive web navigation (WebShop (Yao et al., preprint)), and competition-level mathematical reasoning (MATH (Hendrycks et al., 2021)). This setting reflects an edge/cloud deployment: cloud escalation incurs token charges, WAN round-trip time, provider queueing, and remote decoding, whereas local acceptance uses only the local host. HotPotQA, GSM8K, DROP, HumanEval, and MATH use SAGE's common structured action interface, while WebShop retains its native web-interaction actions and observations. Prompts, tool interfaces, step caps, and stopping criteria are fixed across methods.

We compare SAGE against *SLM-Only*, *LLM-Only*, *Random*, a HybridLLM-style independent router denoted *Hybrid*, and an adapted *Router-R1*. Router-R1 uses the same two-model pool, runtime interface, and budget information as SAGE. We include all routing and aggregation overhead for Router-R1 and all local-probe latency for SAGE. The primary metrics are end-to-end task accuracy, wall-clock latency, local-token ratio, and normalized cloud spend. Accuracy follows each benchmark's standard metric: EM/F1 for HotPotQA and DROP, exact answer accuracy for GSM8K and MATH, pass@1 for HumanEval, and success rate for WebShop. Latency includes local generation, verifier computation, tool execution, network time, provider queueing, and optional escalation. *Local-token ratio* is measured as the percentage of generated tokens produced locally, while *cloud spend* is normalized to the cloud spend of *LLM-Only*. Unless otherwise noted, we set the latency budget to the median *LLM-Only* latency, the normalized cloud-spend budget to 0.50, and the number of stochastic verifier samples to $K = 3$.

Table 3: Accuracy (%) for Qwen3-4B + GPT-5.

Benchmark	SLM	Rand.	Hybrid	R1	SAGE	LLM
HotPotQA	48.21	60.18	64.73	71.95	**75.04**	78.62
GSM8K	88.74	90.26	91.48	94.31	**95.17**	96.84
DROP	63.52	72.84	77.09	82.43	**84.95**	86.71
HumanEval	66.07	73.81	78.57	84.52	**88.10**	92.26
WebShop	36.91	50.44	58.63	66.57	**71.28**	74.38
MATH	62.48	74.36	79.92	87.24	**92.06**	94.13
Average	60.99	70.32	75.07	81.17	**84.43**	87.16

Table 4: for Gemma 3 4B + Claude Sonnet 4.5.

Benchmark	SLM	Rand.	Hybrid	R1	SAGE	LLM
HotPotQA	51.04	63.29	67.42	73.86	**76.83**	79.46
GSM8K	89.20	90.88	92.74	94.65	**95.22**	95.92
DROP	66.38	75.66	79.41	83.94	**85.86**	87.34
HumanEval	71.30	77.42	81.58	85.73	**88.46**	89.78
WebShop	39.27	53.18	60.84	69.21	**74.59**	78.21
MATH	75.60	79.43	82.91	87.04	**92.72**	93.06
Average	65.46	73.31	77.48	82.40	**85.61**	87.30

Table 5: System metrics for Qwen3-4B + GPT-5.

Method	Acc.	Lat. (s)	Local (%)	Spend
SLM-Only	60.99	6.42	100.00	0.00
Random	70.32	8.24	50.06	0.54
Hybrid	75.07	8.61	44.18	0.47
Router-R1	81.17	11.93	36.42	0.69
SAGE	**84.43**	**7.68**	**56.38**	**0.41**
LLM-Only	87.16	10.37	0.00	1.00

Table 6: for Gemma 3 4B + Claude Sonnet 4.5.

Method	Acc.	Lat. (s)	Local (%)	Spend
SLM-Only	65.46	6.88	100.00	0.00
Random	73.31	9.11	50.12	0.56
Hybrid	77.48	9.02	45.72	0.50
Router-R1	82.40	12.84	38.64	0.72
SAGE	**85.61**	**8.27**	**58.63**	**0.45**
LLM-Only	87.30	11.58	0.00	1.00

4.1 End-To-End Performance

Tables 3 and 4 report benchmark-wise accuracy at the default settings. Across both model pairs, *LLM-Only* provides the highest accuracy, while SAGE is the strongest hybrid policy. On Qwen3-4B/GPT-5, SAGE achieves 84.43% accuracy versus 87.16% for *LLM-Only*; on Gemma 3 4B/Claude Sonnet 4.5, it achieves 85.61% versus 87.30%. SAGE gains most over independent *Hybrid* routing on WebShop and HotPotQA, where incorrect actions can affect subsequent observations, while remaining close to *LLM-Only* on GSM8K, HumanEval, and MATH.

Tables 5 and 6 show that SAGE recovers most of the cloud-only accuracy while substantially reducing latency and cloud usage. For Qwen3-4B/GPT-5, SAGE reduces mean latency from 10.37 s to 7.68 s (25.9%), keeps 56.38% of generated tokens local, and reduces normalized spend to 0.41. For Gemma 3 4B/Claude Sonnet 4.5, SAGE reduces latency from 11.58 s to 8.27 s (28.6%), with 58.63% local usage and 0.45 normalized spend.

Compared with *Router-R1*, SAGE simultaneously improves accuracy, latency, and cloud spend. For Qwen3-4B/GPT-5, SAGE improves accuracy by 3.26 percentage points (81.17% to 84.43%), reduces latency by 35.6% (11.93 s to 7.68 s), and reduces normalized spend by 40.6% (0.69 to 0.41). For Gemma 3 4B/Claude Sonnet 4.5, it improves accuracy by 3.21 percentage points (82.40% to 85.61%), reduces latency by 35.6% (12.84 s to 8.27 s), and reduces normalized spend by 37.5% (0.72 to 0.45). Router-R1 repeatedly reasons about routing, invokes selected models, and aggregates their responses, introducing additional model calls and hence latency and inference cost. SAGE instead evaluates the realized local action for trajectory-level safety, enabling more accurate action-level routing while invoking the cloud model only when escalation is needed. Random improves over SLM-Only but trails Hybrid by 4.75 and 4.17 accuracy points across the two model pairs. Hybrid achieves similar latency to Random with lower cloud spend. Compared with Hybrid, SAGE improves accuracy by 9.36 and 8.13 percentage points, reduces latency by 10.8% and 8.3%, increases local token usage by 12.20 and 12.91 percentage points, and reduces normalized cloud spend by 12.8% and 10.0% for the two model pairs, respectively.

4.2 Sensitivity Analysis

Figure 5 shows the macro-average accuracy versus the normalized cost target and Figure 6 shows the Expected Calibration Error (ECE) and latency versus the number of verifier samples K of the two local/remote model pairs of SAGE.

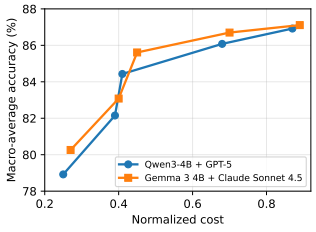


Figure 5: Accuracy versus normalized cost budget.

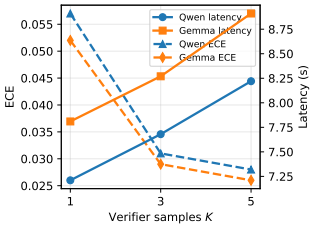


Figure 6: ECE and latency versus verifier samples K.

They illustrate how SAGE responds to the budget target and verifier sampling budget. As the normalized cost target increases, SAGE uses the cloud model more frequently and approaches the *LLM-Only*'s accuracy ceiling. For Qwen3-4B/GPT-5, macro-average accuracy increases from 78.92% at a cost target of 0.25 to 86.92% at 1.00. For Gemma 3 4B/Claude Sonnet 4.5, accuracy increases from 80.26% to 87.11%. The verifier sweep shows diminishing returns from additional samples. Increasing K from 1 to 3 improves accuracy and ECE for both model pairs, whereas increasing it to 5 provides only marginal additional gains. For Qwen3-4B/GPT-5, ECE decreases from 0.057 at $K{=}1$ to 0.031 at $K{=}3$ and 0.028 at $K{=}5$, while latency increases from 7.21 s to 7.68 s and 8.22 s, respectively. For Gemma 3 4B/Claude Sonnet 4.5, ECE similarly decreases from 0.052 to 0.029 and 0.026, while latency increases from approximately 7.78 s to 8.22 s and 8.78 s, respectively.

4.3 Ablation Study

Table 7: Ablation results at the default operating point.

Ablation	Acc. (%)	Lat. (s)	Local (%)	Spend	ECE
Full SAGE	**85.02**	7.98	57.51	0.43	**0.030**
Linear fusion	84.28	7.95	57.86	0.43	0.041
No calibration	83.77	7.92	59.20	0.41	0.080
No reserve	84.11	7.86	58.40	0.43	0.041
Fixed threshold	83.54	7.89	60.44	0.40	0.074
No impact	84.02	7.83	58.96	0.42	0.039
No model verifier	83.69	7.71	58.77	0.42	0.049
No runtime checks	83.88	7.60	58.32	0.42	0.046
No hard checks	83.61	7.59	58.08	0.43	0.044

Table 7 isolates the contribution of each major SAGE component. The largest regressions arise from removing calibration and replacing the budget-aware threshold with a fixed threshold. Without calibration, macro-average accuracy drops to 83.77% and ECE worsens from 0.030 to 0.080. A fixed threshold similarly reduces accuracy, showing that SAGE benefits from both a calibrated safety estimate and a state-dependent decision threshold. Removing budget reservation or the impact model causes smaller but consistent degradations, while removing model-based verifier features, runtime checks, or hard checks also reduces accuracy. Full SAGE is the only configuration that simultaneously exceeds 85% macro-average accuracy, keeps a majority of generated tokens local, and maintains strong calibration.

5 Related Work

Hybrid LLM systems. Routing selects a model before generation (Ong et al., 2025; Ding et al., 2023), while cascading starts with a cheaper model and escalates unreliable outputs (Ramírez et al., 2024; Chen et al., 2023). FrugalGPT, HybridLLM, and RouteLLM optimize query-level cost/quality (Chen et al., 2023; Ding et al., 2023; Ong et al., 2025); other work aggregates models or uses iterative local–cloud collaboration (Jiang et al., 2023; Narayan et al., 2025). SAGE instead evaluates a realized local action by looking ahead to its effect on eventual task success and the remaining latency and cloud-cost budgets.

Agents and dynamic routing. CoT, ReAct, and Reflexion support iterative reasoning and interaction (Wei et al., 2022b; Yao et al., 2023; Shinn et al., 2023), while agent frameworks support multi-step tool use (Significant-Gravitas, 2023; Wu et al., 2024; LangChain, 2023). DoT decomposes requests into dependency DAGs and routes their nodes between local and cloud models (Shao et al., 2025). Router-R1 learns sequential model routing (Zhang et al., 2025), while STRMAC, MasRouter, and related methods learn state-aware agent/model selection policies (Wang et al., 2025; Yue et al., 2025; Panayotov & Emanuilov, 2025); ToolOrchestra dynamically selects heterogeneous models and tools (Su et al., 2025). SAGE instead looks ahead from an already-generated local action to assess whether accepting it leaves the remaining trajectory solvable within the budgets.

6 Conclusion

SAGE looks ahead to assess how accepting a realized local action would affect the remaining trajectory, then routes the action based on whether the request can still succeed within its budgets. By combining calibrated safety estimation, deterministic runtime checks, and budget-aware routing, SAGE decides whether to accept each realized local action or escalate to the cloud while accounting for its impact on future recoverability, latency, and cloud spend. Across six benchmarks and two SLM/LLM pairs, SAGE remains close to *LLM-Only* accuracy while keeping a majority of generated tokens local and substantially reducing cloud spend and end-to-end latency. These results demonstrate the potential of lookahead action-level routing for efficient edge/cloud agent execution.

AI Assistance Disclosure

Generative AI tools were used to improve the clarity, grammar, and presentation of portions of the manuscript. The authors reviewed and revised all AI-assisted text and take full responsibility for the final content.

References

AgentGPT. Agentgpt: Autonomous ai agent framework. https://www.agentgpt.com, 2023.

Aggarwal, P., Madaan, A., Anand, A., Potharaju, S. P., Mishra, S., Zhou, P., Gupta, A., Rajagopal, D., Kappaganthu, K., Yang, Y., Upadhyay, S., Faruqui, M., and Mausam. Automix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Anthropic. Introducing Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, 2025.

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

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

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021a.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021b.

Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Rühle, V., Lakshmanan, L. V., and Awadallah, A. H. Hybrid llm: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2023.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL-HLT*, pp. 2368–2378, 2019.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Jiang, D., Ren, X., and Lin, B. Y. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (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/.

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

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., and Stoica, I. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679. USENIX Association, 2023.

Narasimhan, H., Jitkrittum, W., Rawat, A. S., Kim, S., Gupta, N., Menon, A. K., and Kumar, S. Faster cascades via speculative decoding. In *International Conference on Learning Representations*, 2025.

Narayan, A., Biderman, D., Eyuboglu, S., May, A., Linderman, S., Zou, J., and Ré, C. 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.

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, 2025.

OpenAI. Introducing GPT-5. `https://openai.com/index/introducing-gpt-5/`, 2025.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

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

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Šakota, M., Peyrard, M., and West, R. 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.

Shao, C., Hu, X., Lin, Y., and Xu, F. 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.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. 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.

Son, S., Yoon, S., Tang, J., Wang, S., Wolf, L., and Bogunovic, I. SWE-Router: Routing in multi-turn agentic software engineering tasks. *arXiv preprint arXiv:2607.00053*, 2026.

Su, H., Diao, S., Lu, X., Liu, M., Xu, J., Dong, X., Fu, Y., Belcak, P., Ye, H., Yin, H., Dong, Y., Bakhturina, E., Yu, T., Choi, Y., Kautz, J., and Molchanov, P. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Wang, J., Zhao, S., Wang, H., Fan, Y., Zhang, L., Liu, Y., and Liu, T. Optimal-agent-selection: State-aware routing framework for efficient multi-agent collaboration. *arXiv preprint arXiv:2511.02200*, 2025.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022a.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022b.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *Proceedings of the Conference on Language Modeling (COLM)*, 2024.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yao, S., Chen, H., Yang, J., and Narasimhan, K. Webshop: Towards scalable real-world web interaction with grounded language agents. In *ArXiv*, preprint.

Yin, W., Xu, M., Li, Y., and Liu, X. Llm as a system service on mobile devices. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2026.

Yuan, W., Neubig, G., and Liu, P. Bartscore: Evaluating generated text as text generation. *Advances in Neural Information Processing Systems*, 34:27263–27277, 2021.

Yue, M., Zhao, J., Zhang, M., Du, L., and Yao, Z. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. In *International Conference on Learning Representations*, 2024.

Yue, Y., Zhang, G., Liu, B., Wan, G., Wang, K., Cheng, D., and Qi, Y. 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.

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

Zhang, Y., Li, H., Wang, Z., Feng, S., Yang, X., Wang, D., Zhang, B., Bai, L., and Hu, S. MTRouter: Cost-aware multi-turn LLM routing with history–model joint embeddings. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44206–44226. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.2045.