THINK BEFORE SPENDING: VALUE-AWARE COMPUTATION CONTROL FOR LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time computation improves large language model (LLM) reasoning by generating additional evidence and exploring multiple reasoning paths, but incurs substantial generation cost. Existing adaptive computation methods mainly optimize resource allocation or termination after a reasoning strategy has been fixed, leaving the initial strategy choice unresolved. Reasoning strategies such as direct submission, evidence gathering, and consistency sampling offer different quality–cost trade-offs across requests, making the choice of strategy central to reducing unnecessary computation. To address this challenge, we present Compass, a value-aware computation controller for LLM reasoning. Compass is built on two key observations: (1) observable inference states contain signals about the expected quality and cost of candidate reasoning strategies, and (2) additional computation is valuable only when it improves upon the currently available answer. Based on these observations, Compass introduces two coupled decisions: quality-aware computation strategy selection and marginal-value continuation control. Specifically, Compass estimates the final quality and remaining token cost of candidate computation strategies before committing to expensive execution, selecting a low-cost strategy with near-optimal predicted quality. During execution, Compass dynamically reassesses continuation by estimating the probability of correcting the current answer against the risk of introducing an error and the additional computation cost. Across four datasets, three starting models, and two generation seeds, Compass reduces pooled output tokens per correct answer (TPC) by 20.1–88.1% relative to the evaluated sampling baselines, with macro-accuracy gains of 0.9–2.3 percentage points.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong reasoning capabilities through test-time computation beyond single-pass generation. Chain-of-thought prompting improves reasoning by eliciting intermediate steps (Wei et al., 2022), while search- and sampling-based methods explore multiple reasoning paths to enhance solution reliability, such as tree-based reasoning and Self-Consistency (SC) (Yao et al., 2023; Wang et al., 2023). Recent approaches further incorporate verification and refinement, where learned verifiers evaluate candidate solutions and refinement methods improve responses through feedback (Cobbe et al., 2021; Lightman et al., 2024; Madaan et al., 2023). These studies establish test-time computation as a promising paradigm for improving LLM reasoning without modifying model parameters.

Generating additional reasoning paths increases token consumption and can prolong inference, making it essential to reduce generation cost while maintaining reasoning accuracy. Recent studies have explored adaptive inference control from two perspectives. (1) *Computation allocation* methods focus on determining how much computation should be spent for each request. Adaptive-Consistency (AC) adjusts sampling budgets based on inference-time evidence (Aggarwal et al., 2023), while Strategic Scaling allocates computation across queries according to their difficulty and expected improvement (Zuo & Zhu, 2026). (2) *Computation termination* methods determine when an ongoing inference procedure should stop based on intermediate evidence, such as Early-Stopping Self-Consistency (ESC) and Certaindex (Li et al., 2024; Fu et al., 2025). Together, these efforts improve test-time efficiency by adapting computation allocation and termination decisions under different reasoning tasks.

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

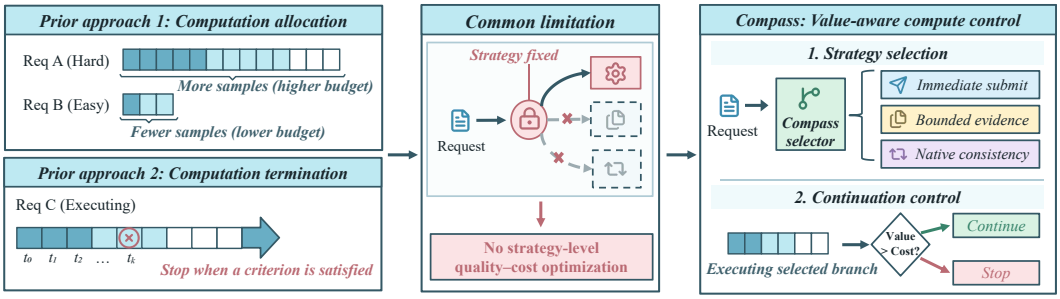


Figure 1: Overview of the computation-control problem addressed by Compass. The compared sampling methods allocate or terminate computation within a fixed reasoning strategy, whereas Compass first selects a value-aware strategy and then controls continuation by comparing expected value with cost.

These adaptive sampling methods mainly optimize computation after a reasoning strategy has been fixed, leaving the initial strategy choice unresolved. Different requests may favor different strategies, such as direct submission, evidence gathering, or consistency sampling. Because these strategies offer different quality–cost trade-offs across requests, deciding which strategy to execute is central to avoiding unnecessary computation. Once a strategy has been selected, the controller must also reassess whether additional computation can improve the current answer. This calls for two coupled decisions: choosing a computation strategy and controlling its continuation. Figure 1 summarizes this gap and the corresponding design direction: existing approaches adapt computation after fixing a strategy, whereas Compass first selects a value-aware strategy and then controls continuation according to marginal value.

Two observations guide our design. First, *observable inference states contain signals about the expected quality and cost of candidate computation strategies*. Query characteristics and the first generated answer provide evidence for comparing strategies before committing to further execution. Second, *additional computation is valuable only when it improves upon the currently available answer*. Further sampling may correct an incorrect answer, leave it unchanged, or introduce an error. Continuation therefore depends on the expected net correctness gain relative to the current answer and the additional token cost.

Building on these observations, we introduce **Compass**, a value-aware computation controller that couples *quality-aware computation strategy selection* with *marginal-value continuation control*. From the first-answer state, Compass compares immediate submission, bounded evidence gathering, and native Adaptive-Consistency using estimates of final-answer correctness and remaining output-token cost. It selects a low-cost strategy with near-best predicted quality. During evidence-branch execution, Compass reassesses continuation at supported checkpoints by weighing the probability of correcting the current answer against the risk of introducing an error and the additional token cost. The decisions are coupled because the initial strategy estimates account for downstream continuation control. Both decisions use observable inference signals and estimates learned from separate training tasks.

Our contributions are threefold:

1. We formulate test-time computation control as a joint decision problem that extends beyond scaling a preselected reasoning strategy: the controller must choose which computation strategy to execute and continually assess whether further computation is worthwhile relative to the current answer.
2. We develop Compass, a value-aware controller that realizes this formulation through quality-aware computation strategy selection and marginal-value continuation control. Compass evaluates complete strategy outcomes, including downstream continuation, thereby coupling the two decisions.
3. Across four datasets, three starting models, and two generation seeds, Compass reduces pooled output tokens per correct answer (TPC) by 20.1–88.1% relative to the evaluated sampling baselines, with macro-accuracy gains of 0.9–2.3 percentage points.

2 Related Work

Test-Time Reasoning Strategies. Test-time computation improves the reasoning capability of fixed LLMs by introducing additional inference procedures, including intermediate reasoning, sampling, search, and verification. Chain-of-thought prompting elicits explicit reasoning steps, while Self-Consistency and Tree of Thoughts improve reasoning reliability by exploring multiple reasoning paths (Wei et al., 2022; Wang et al., 2023; Yao et al., 2023). Learned verifiers and feedback-based refinement further enhance reasoning by evaluating and revising candidate solutions (Cobbe et al., 2021; Lightman et al., 2024; Madaan et al., 2023). These strategies provide diverse ways of spending inference computation, with different quality and cost characteristics depending on tasks and available resources (Snell et al., 2025). Compass builds upon this space of reasoning strategies by considering computation strategy selection and continuation as part of test-time control.

Adaptive Computation Allocation. As test-time computation becomes more expensive, recent studies investigate how to allocate inference resources efficiently. Adaptive-Consistency adapts the number of samples according to accumulated answer evidence (Aggarwal et al., 2023). Strategic Scaling formulates computation allocation across requests as a sequential decision problem and allocates sampling opportunities through Elimination-SC (Zuo & Zhu, 2026). Model cascades further explore resource-aware inference by selecting models with different accuracy and cost profiles (Chen et al., 2023). These approaches demonstrate the effectiveness of adapting inference computation according to request difficulty, observed evidence, and resource constraints. Compass extends adaptive computation control to reasoning strategy decisions, where candidate computation strategies are compared based on their expected outcomes.

Adaptive Computation Termination. Another line of work focuses on reducing unnecessary inference by determining when additional computation should stop. ESC terminates Self-Consistency when sampled answers reach sufficient agreement, while Certaindex uses answer-distribution evidence to adapt reasoning computation (Li et al., 2024; Fu et al., 2025). These methods leverage intermediate signals to identify when further computation is unlikely to provide sufficient benefit. Studies of intrinsic self-correction show that revising answers without external feedback can introduce errors (Huang et al., 2024). These stopping rules assess continuation within a selected procedure; they do not compare the complete outcomes of alternative procedures from the same initial state. Compass incorporates this perspective by evaluating continuation according to the expected correctness improvement over the current answer.

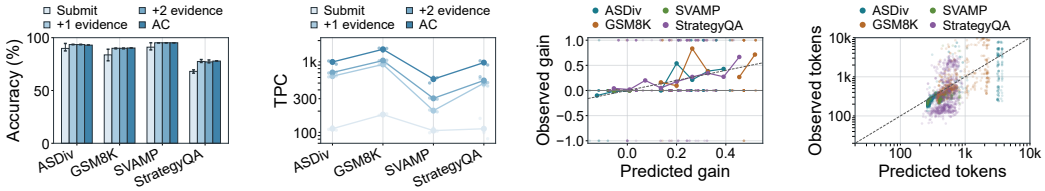
3 Motivation

3.1 Preliminaries

Test-Time Computation Setting. Test-time computation improves the reasoning ability of a fixed LLM by allocating additional inference resources after the initial generation. A reasoning strategy specifies how these resources are used, such as directly submitting the initial answer, acquiring additional evidence, or generating multiple reasoning paths for aggregation. For a request x, the inference state s records the generated answers and observable evidence collected during execution. A fixed aggregation rule maps s to the current answer $y(s)$ that would be submitted without further computation. Each executable computation branch a defines how additional evidence is acquired, intermediate results are aggregated, and execution is terminated, producing a final answer y_a with additional output-token cost $C(a, s)$. Different branches may incur different costs and achieve different answer quality, motivating adaptive computation control during inference. The reference answer Y is available only to the training scorer and independent evaluator, but is never exposed to the inference controller.

Computation Control Objective. The controller makes two sequential decisions during inference: selecting an initial computation branch after the first answer is generated and determining whether to continue the selected branch as new evidence becomes available. We evaluate the overall quality–cost trade-off using answer accuracy and output tokens per correct answer (TPC):

$$\mathrm{TPC} = \frac{\sum_i T_i}{\sum_i \mathbb{I}(\hat{y}_i = Y_i)}, \tag{1}$$



(a) Accuracy by branch **(b) TPC by branch** **(c) Gain calibration** **(d) Cost calibration**

Figure 2: Computation-control diagnostics on Seed-1. For fixed initial branches, downstream execution is retained: (a) mean accuracy with min–max ranges across starting models; (b) pooled TPC, with small dots indicating individual models. At each request's first supported checkpoint, (c) and (d) compare predicted and observed native-tail gain and cost; curves in (c) show binned means, and dashed lines indicate equality. Gains are expressed as probability differences. Colors identify branches in (a) and (b) and datasets in (c) and (d).

where $\mathbb{I}(\cdot)$ denotes the indicator function, T_i denotes the total output tokens consumed by all model calls for request i, including unsuccessful calls, and $\hat{y}_i$ is the final submitted answer. Accuracy measures final answer quality, while TPC measures the token cost required to obtain one correct answer over the evaluated request set.

3.2 Observations

The two decisions rely on different forms of inference evidence: branch selection requires comparing candidate strategies, while continuation control requires estimating the value of further computation relative to the current answer. The following observations motivate these two components.

Observation 1: Different reasoning strategies occupy distinct quality–cost operating points. A computation strategy determines both the evidence available to the final answer and the tokens spent obtaining it. Figure 2(a)–(b) compares strategies across four datasets, retaining each branch's downstream continuation rule. Direct submission is inexpensive; evidence gathering improves average accuracy at additional cost, with gains varying by dataset and starting model. Native AC consumes more tokens than either evidence branch without consistently improving accuracy. These distinct operating points motivate comparing complete strategies before selecting one for a request.

Observation 2: Continuation value varies with the current answer state. The value of further computation depends on whether continuation is likely to correct the current answer or introduce an error, and on the cost of the available continuation. Figure 2(c)–(d) compares predicted and observed correctness gains and tail-token costs across datasets, with cost shown on a log scale. These comparisons motivate continuation decisions based on marginal value relative to the current answer rather than on a fixed number of additional samples.

4 Compass Framework

Figure 3 presents the Compass framework, which integrates *quality-aware computation strategy selection* and *marginal-value continuation control*. After the first answer is generated, Compass first determines which computation strategy provides the most favorable predicted quality–cost trade-off, and then decides whether additional computation remains valuable during execution. The selector compares candidate strategies by their predicted final-answer correctness and remaining output-token cost, choosing a low-cost strategy with near-best predicted quality. Evidence-gathering strategies subsequently update the current answer and reassess whether further sampling is worthwhile. The two components are coupled through complete-strategy prediction: initial estimates include the execution outcomes produced by downstream continuation control.

4.1 Quality-Aware Computation Strategy Selection

Challenge and Approach. A fixed computation strategy can be inefficient because its preferred operating point depends on request difficulty and the reliability of the initial answer. The initial decision must compare strategies from the first-answer state s_0, while their final answers and remaining

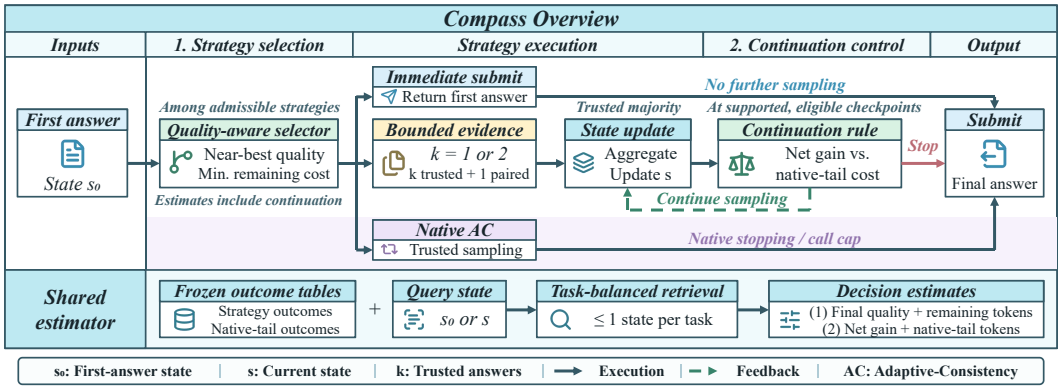


Figure 3: Overview of Compass. Quality-aware selection chooses among immediate submission, bounded evidence gathering, and native Adaptive-Consistency. Evidence branches update the answer and apply marginal-value continuation control. Both decisions use a shared task-balanced estimator built from frozen training outcomes.

costs depend on evidence yet to be acquired and subsequent aggregation and stopping decisions. Compass addresses this challenge by learning from complete-strategy executions on separate training tasks. It retrieves similar first-answer states to estimate each candidate's final-answer correctness and total remaining output-token cost, including downstream continuation.

Candidate Computation Strategies. The candidate set is $\mathcal{A} = \{a_{\text{sub}}, a_1, a_2, a_{\text{AC}}\}$. For each dataset, a trusted model is selected using training tasks, and its pairing with each starting model is fixed before evaluation. Branch a_{sub} submits $y(s_0)$ immediately. Evidence branch a_k, for $k \in \{1, 2\}$, collects k trusted-model answers and one answer from the other model in the pair, counting the first generated answer toward these totals when applicable. The fixed aggregation rule selects the trusted-model majority, while cross-model agreement informs continuation control. After this bounded acquisition stage, a_k enters the controller in Section 4.2. Branch a_{AC} directly samples the trusted model under Adaptive-Consistency (AC) (Aggarwal et al., 2023), following its native stopping rule until AC or the shared call cap terminates execution.

Complete-Strategy Outcome Estimation. For a branch a returning answer y_a at additional output-token cost $C(a, s_0)$, we define its true conditional outcome as

$$\begin{aligned} q_a(s_0) &= \Pr[y_a = Y \mid s_0], \\ c_a(s_0) &= \mathbb{E}[C(a, s_0) \mid s_0]. \end{aligned} \tag{2}$$

Since these quantities are unknown at inference time, Compass estimates them from separate training tasks as $\hat{q}_a(s_0)$ and $\hat{c}_a(s_0)$. For evidence branches, both quantities include subsequent sampling and stopping under Compass. The already-consumed first answer is excluded from the remaining-cost comparison.

Training records contain each branch's final correctness and additional output tokens. When constructing an evidence branch's record, the downstream controller excludes the queried training task from its estimates. This links the selector's targets to the continuation behavior used during execution. We use 64 training tasks per dataset and four sample-sequence offsets per task; estimation tables and controller parameters remain fixed during evaluation.

Initial-state features describe the prompt and first answer: answer validity, length-normalized generation confidence, truncation, and prompt and output lengths. For Python solutions, syntax-tree features capture response structure while excluding identifier names and literal values. Estimation tables are conditioned on dataset, starting model, and candidate branch.

Compass applies a task-balanced empirical estimator within each table, implemented with nearest-neighbor retrieval. Features are scaled by their training-state standard deviations, and retrieval selects at most one nearest state from each of up to 24 distinct tasks. For an outcome or cost target z,

$$\hat{z}(s) = \frac{\sum_{j \in \mathcal{N}(s)} z_j + 4\bar{z}_{\text{task}}}{|\mathcal{N}(s)| + 4}, \tag{3}$$

where $\mathcal{N}(s)$ is the selected neighborhood, z_j is the recorded target, and $\bar{z}_{\text{task}}$ averages the per-task target means in the table. Task-balanced retrieval limits each task's contribution, while the four-observation prior stabilizes estimates with small neighborhoods. Tables with fewer than eight distinct supporting tasks are marked unsupported. Section 4.2 reuses this estimator for continuation outcomes.

Quality-Constrained Selection. Compass constructs an admissible set $\mathcal{A}_{\text{safe}}(s_0)$ from supported candidates using a_1 as the reference. Immediate submission requires the starting model to be the trusted model and $\hat{q}_{\text{sub}}(s_0) \geq \hat{q}_1(s_0)$. Any strategy with $\hat{c}_a(s_0) > \hat{c}_1(s_0)$ must additionally have positive task-paired correctness gain after subtracting a standard-error-based heuristic margin. The paired comparison uses outcomes from the same training tasks.

For a nonempty admissible set, Compass selects the minimum-cost strategy subject to a predicted-quality tolerance:

$$\begin{aligned} a^*(s_0) = \underset{a \in \mathcal{A}_{\text{safe}}(s_0)}{\arg\min}\ & \hat{c}_a(s_0), \\ \text{subject to}\quad & \hat{q}_a(s_0) \geq \max_{b \in \mathcal{A}_{\text{safe}}(s_0)} \hat{q}_b(s_0) - \epsilon. \end{aligned} \tag{4}$$

We set $\epsilon = 0.01$, retaining strategies within one percentage point of the best admissible correctness prediction. Quality determines eligibility, and remaining token cost determines selection. If the admissible set is empty, Compass enters a_1.

4.2 Marginal-Value Continuation Control

Challenge and Approach. The correctness of the current answer is unknown during inference, and additional samples can either correct it or introduce an error. Compass evaluates continuation by estimating both transitions relative to immediate submission and comparing their net gain with the remaining token cost. The reference execution is a *native consistency tail*: trusted-model sampling from the current state until AC or the shared call cap stops execution.

Net Correctness Gain. Let $y(s)$ be the current answer and y_{tail} the answer returned by native consistency completion. The rescue and regression probabilities are

$$\begin{aligned} r(s) &= \Pr[y(s) \neq Y,\ y_{\text{tail}} = Y \mid s], \\ g(s) &= \Pr[y(s) = Y,\ y_{\text{tail}} \neq Y \mid s]. \end{aligned} \tag{5}$$

Their difference gives the expected correctness improvement over immediate submission:

$$\Delta(s) = \mathbb{E}[\mathbb{I}(y_{\text{tail}} = Y) - \mathbb{I}(y(s) = Y) \mid s] = r(s) - g(s). \tag{6}$$

Here, $\mathbb{I}(\cdot)$ denotes the indicator function. The indicator difference captures the change in correctness induced by continuation: it is positive for error correction, negative for error introduction, and zero when correctness is unchanged.

Training records pair the current answer with its native-tail outcome and record the additional output tokens consumed. Compass estimates rescue, regression, and tail cost using Equation 3. Continuation features extend the first-answer features with sample counts, the fraction admitted to aggregation, vote concentration and margin, answer entropy, and cross-model agreement. The estimation tables additionally distinguish checkpoint and agreement state. Correctness labels provide training targets; inference features are computed from the answers already generated.

Cost-Aware Continuation. Let $\hat{\Delta}(s) = \hat{r}(s) - \hat{g}(s)$ be the estimated net correctness gain and $\hat{c}_{\text{tail}}(s)$ the estimated remaining native-tail output tokens. At a supported checkpoint where learned stopping is permitted, Compass stops when

$$\hat{\Delta}(s) \leq \tau + \lambda \frac{\hat{c}_{\text{tail}}(s)}{1000}, \tag{7}$$

where τ is a minimum-gain margin and λ specifies the required correctness gain per thousand additional output tokens. We use $\tau = 0$ and $\lambda = 0.02$: continuation predicted to consume 1,000 tokens requires a net correctness gain above two percentage points.

Each checkpoint forecasts native completion from its current state. When execution continues, Compass acquires further samples and updates its predictions at subsequent checkpoints. Initial strategy

estimates incorporate these repeated decisions, while each continuation estimate evaluates the native tail available from the current state.

Checkpointed Execution. For evidence branch a_k, learned decisions occur after the initial acquisition stage with k trusted-model answers and subsequently at trusted-sample counts $\{3, 5, 9, 17, 31\}$. At each checkpoint, Compass updates the aggregated answer and inference state before applying the continuation rule. Native AC stopping and the shared call cap take precedence; cross-model disagreement prevents learned early stopping at later checkpoints.

If the estimator lacks sufficient support, the initial checkpoint uses agreement-based stopping, whereas later checkpoints continue native consistency. Whenever execution stops, Compass submits the current answer $y(s)$ returned by the fixed aggregation rule.

5 Evaluation

5.1 Experimental Setup

Tasks and Models. We evaluate Compass on four reasoning benchmarks covering mathematical and commonsense reasoning. Mathematical reasoning includes GSM8K (Cobbe et al., 2021), ASDiv (Miao et al., 2020), and SVAMP (Patel et al., 2021); commonsense reasoning uses StrategyQA (Geva et al., 2021). We use Qwen3-4B-Instruct-2507, Qwen3-8B, and Llama-3.1-8B-Instruct (Qwen Team, 2025b;a; Meta, 2024) as starting models. Compact labels Q4, Q8, and L8 refer to these models, respectively. The evaluation contains 872 problems, with the same task identities used for both model-generation seeds.

Baselines. We compare Compass with Self-Consistency (SC) (Wang et al., 2023), Adaptive-Consistency (AC) (Aggarwal et al., 2023), Early-Stopping Self-Consistency (ESC) (Li et al., 2024), training-profiled Certaindex, and profiled Strategic Scaling (Fu et al., 2025; Zuo & Zhu, 2026). SC uses 40 calls; the other baselines adapt termination or allocation. These baselines sample the starting model, while Compass can acquire auxiliary-model evidence. All methods share prompting, answer extraction, correctness evaluation, and the 40-call cap. Appendix A.4 gives the profiles and decision rules.

Evaluation Protocol. We evaluate all methods through offline replay over real model-generated answer banks. Seed-1 uses model-generation seed 2053 and Seed-2 uses seed 2063, with the same 872 task identities and separately generated answer banks. Within each seed, methods consume the same recorded answers and prefixes in the same order, without access to unconsumed samples. Training and evaluation tasks are disjoint, and the controller never receives evaluation labels; the shared interface and accounting rules are detailed in Appendices A.2 and A.3.

Metrics. We report answer accuracy and pooled output tokens per correct answer (TPC) to evaluate answer quality and output-token efficiency, respectively. Accuracy is the macro mean over the 12 dataset–model cells, while TPC pools total consumed output tokens and correct answers over all requests. Output tokens include auxiliary-model generations and are not weighted by model size. Detailed timeout, invalid-output, and baseline-profile settings appear in Appendix A. In a warmed local CPU traversal, controller execution takes 11.98 ms per request on average and 17.69 ms at the 95th percentile, including aggregation and instrumentation; model inference is outside the timing scope (Appendix C).

5.2 End-to-End Efficiency and Accuracy

Token efficiency. Across the two seeds, Compass attains pooled TPC of 601.6 and 595.3, a 20.1–88.1% reduction relative to the five starting-model sampling baselines. Table 1 resolves this aggregate result by dataset and model: Compass uses fewer tokens per correct answer than SC, AC, ESC, and Certaindex in all 12 cells. Its pooled TPC reduction is 45.3–46.3% relative to AC and 20.1–20.7% relative to training-profiled Strategic Scaling. In the three cells where the starting model is also an 8B trusted model, Compass reduces TPC relative to AC by 44.1% on GSM8K-Q8, 32.0% on ASDiv-L8, and 49.9% on StrategyQA-L8.

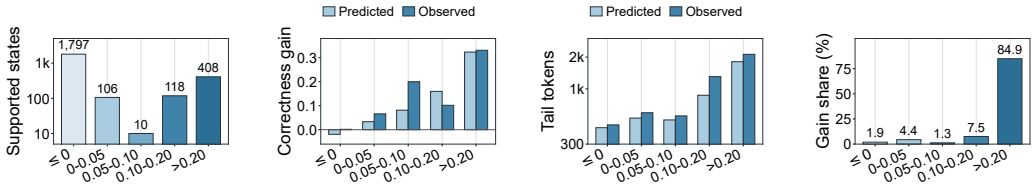
Reasoning accuracy. Compass's two-seed macro accuracies are 88.84% and 89.07%, 0.9–2.3 percentage points above the evaluated baselines. Table 2 shows where the gain arises: relative to AC,

Table 1: Output tokens per correct answer (TPC, lower is better), averaged over the two model-generation seeds. Q4, Q8, and L8 denote the three starting models. Certaindex and Strategic Scaling use their training-profiled configurations.

Method	GSM8K			ASDiv			SVAMP			StrategyQA		
	Q4	Q8	L8	Q4	Q8	L8	Q4	Q8	L8	Q4	Q8	L8
SC	6721.0	6679.1	6636.6	4309.1	4408.4	4078.5	4039.2	4534.5	4210.6	6021.6	4139.1	2833.2
AC	1119.2	1374.7	1859.8	659.1	780.8	910.3	519.9	616.8	861.2	1758.1	1353.5	895.4
ESC	1723.0	2065.9	2771.7	851.3	1185.3	1237.0	646.5	985.0	1193.8	2655.8	1920.2	1333.6
Certaindex	1294.4	1486.2	3042.1	943.8	919.1	959.8	518.5	663.1	1251.7	1831.6	1198.1	969.5
Strategic Scaling	685.8	682.5	1865.2	714.4	440.6	422.3	397.3	602.3	901.3	596.2	410.9	1129.3
Compass	**851.2**	**769.1**	**928.3**	**569.0**	**648.2**	**618.6**	**99.3**	**208.4**	**196.0**	**598.0**	**470.8**	**448.7**

Table 2: Accuracy (%, higher is better) for each dataset–model pair, averaged over two generation seeds. Q4, Q8, and L8 denote the starting models. All baseline configurations are fixed across seeds.

Method	GSM8K			ASDiv			SVAMP			StrategyQA		
	Q4	Q8	L8	Q4	Q8	L8	Q4	Q8	L8	Q4	Q8	L8
SC	92.0	90.6	86.9	95.9	94.1	92.8	94.7	91.3	88.5	70.7	75.2	78.3
AC	92.2	90.4	86.9	95.9	93.9	92.6	94.7	91.3	88.9	71.1	75.2	78.1
ESC	92.0	90.6	86.9	95.9	94.1	92.6	94.7	91.3	88.5	71.1	75.2	77.7
Certaindex	91.8	90.2	86.9	95.9	93.4	92.2	94.7	91.3	88.5	71.3	75.6	77.5
Strategic Scaling	90.4	88.5	86.3	95.9	93.8	89.5	95.7	91.3	90.4	71.3	76.4	77.9
Compass	**90.2**	**89.6**	**89.8**	**93.2**	**92.8**	**92.2**	**95.2**	**95.2**	**95.2**	**79.1**	**77.3**	**77.5**



(a) Supported states **(b) Gain calibration** **(c) Cost calibration** **(d) Gain contribution**

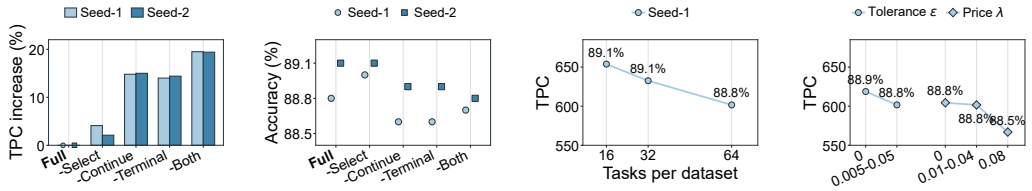
Figure 4: Continuation estimates at the first supported checkpoint of 2,439 Seed-1 requests, targeting native AC completion relative to immediate submission: (a) state counts; (b) correctness gain; (c) remaining tokens; (d) share of state-count-weighted observed gain. Bins use predicted gain; gains are probability differences.

Compass averages 3.5 additional points on SVAMP and 3.2 on StrategyQA across starting models and seeds. At SVAMP-L8 and StrategyQA-Q4, it reaches 95.2% and 79.1%, exceeding the strongest baseline in each cell by 4.8 and 7.8 points, respectively.

5.3 Continuation Value and Cost

Where continuation helps. Figure 4(a), (b), and (d) locate the value of native-tail continuation across 2,439 first supported checkpoints. The nonpositive predicted-gain bin contains 73.7% of states but has an average observed correctness gain of 0.0017. The > 0.20 bin contains 16.7% of states yet contributes 84.9% of the state-count-weighted observed gain. Continuation therefore has markedly different value depending on the current state.

Gain versus remaining cost. Figure 4(b) and (c) show why gain alone does not determine the continuation decision. In the highest-gain bin, predicted and observed correctness gains are 0.3235 and 0.3309, while observed native-tail cost is 2,125.1 tokens, compared with 456.2 in the nonpositive-gain bin. Predicted cost likewise rises from 429.1 to 1,809.3 tokens. The same states that offer the most potential gain also require substantially more computation, making gain and remaining cost joint inputs to the decision.



(a) Ablation TPC **(b) Ablation accuracy** **(c) Training support** **(d) Controller settings**

Figure 5: Component contributions and robustness: (a) and (b) TPC increase relative to Full Compass and macro accuracy across two seeds; (c) and (d) Seed-1 TPC under support and parameter sweeps, with accuracy labels. "-Continue" fixes initial actions; "-Terminal" removes native terminal value. Full results and paired intervals: Appendix B.

5.4 Component Analysis

Selection and continuation. Figure 5(a) and (b) compare the two decisions under matched tasks, sample order, and the same 40-call cap. Holding initial actions fixed, replacing value-aware continuation with native AC raises TPC by 14.8% and 15.0% on the two seeds. Within the same selected evidence branches, value-aware continuation therefore improves token efficiency over native AC completion. Removing strategy selection while retaining continuation raises TPC by 4.1% and 2.1%. Both variants remain near Full Compass in macro accuracy, locating complementary TPC gains in continuation and initial strategy choice.

Combined contribution and terminal value. Disabling both decisions in Figure 5(a) and (b) raises TPC by 19.4–19.5%, more than removing either one alone. Removing native terminal value and refitting the selector raises TPC by 14.0–14.4%. Full Compass has the lowest TPC on both seeds, with all five variants remaining within 88.6–89.1% macro accuracy. The increase for the refitted terminal-value variant further supports valuing continuation relative to native completion. Together, the ablations favor the complete controller, which evaluates both the initial strategy and its downstream execution.

5.5 Robustness Analysis

Training support. Figure 5(c) varies training support for both estimator stages. Raising support from 16 to 32 to 64 tasks per dataset lowers TPC from 653.8 to 632.5 to 601.6, an 8.0% reduction from the smallest setting, while macro accuracy remains between 88.8% and 89.1%. The monotonic TPC reduction, together with the narrow accuracy range, indicates that greater training support primarily improves efficiency in this sweep. The default 64-task configuration achieves the lowest TPC among the tested support settings.

Controller parameters. Figure 5(d) varies quality tolerance and continuation price one at a time, fixing initial actions for the price sweep. For tested tolerances in $\{0.005, 0.01, 0.02, 0.05\}$ and prices in $\{0.01, 0.02, 0.04\}$, Compass maintains 601.6 TPC and 88.8% accuracy. These tested tolerances span a tenfold range, and the prices span a fourfold range. The matching outcomes show that the reported aggregate quality–cost balance is reproduced across several discrete settings of each parameter.

6 Conclusion

We introduced Compass, a value-aware computation controller that couples quality-aware computation strategy selection with marginal-value continuation control. Using observable inference states, Compass selects a low-cost strategy with near-best predicted quality and reassesses further sampling by weighing the expected net correctness gain against the additional token cost. Initial strategy estimates include downstream continuation outcomes, linking the two decisions. Across four datasets, three starting models, and two generation seeds, Compass reduces pooled TPC by 20.1–88.1% relative to the evaluated sampling baselines, with macro-accuracy gains of 0.9–2.3 percentage points.

AI USE STATEMENT

During manuscript preparation, the authors used generative artificial intelligence (AI) tools for language editing, manuscript organization, figure and table refinement, interpretation of experimental results, reference checking, and LaTeX debugging. AI-assisted suggestions for presentation and analysis were checked against the source data and implementation by the authors. The authors take responsibility for the final text, figures, claims, and experimental results.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.761. URL https://aclanthology.org/2023.emnlp-main.761/.

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

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

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling LLM reasoning with Certaindex. In *Advances in Neural Information Processing Systems*, volume 38, pp. 141987–142025, 2025. doi: 10.52202/085713-4751. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/d037fd021c9aace128b8ce25001cdb6c-Paper-Conference.pdf.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did Aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021. doi: 10.1162/tacl_a_00370. URL https://aclanthology.org/2021.tacl-1.21/.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01798.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ndR8Ytrzhh.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.17651.

Meta. Llama-3.1-8B-Instruct model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, 2020. doi: 10.18653/v1/2020.acl-main.92. URL https://aclanthology.org/2020.acl-main.92/.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, 2021. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168/.

Qwen Team. Qwen3-8B model card, 2025a. URL https://huggingface.co/Qwen/Qwen3-8B.

Qwen Team. Qwen3-4B-Instruct-2507 model card, 2025b. URL https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2201.11903.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.10601.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.12721.

A Evaluation Protocol and Reproducibility

This appendix specifies the shared evaluation configuration, answer processing, metric aggregation, and baseline settings used in Section 5.1.

A.1 Evaluation Seeds

Both evaluation settings use the same 872 problems: 256 each from GSM8K, ASDiv, and StrategyQA, and 104 from SVAMP. Separately generated answer banks use seed 2053 for Seed-1 and 2063 for Seed-2. Table 3 summarizes the shared configuration.

The trusted models are Q8 for GSM8K, L8 for ASDiv and StrategyQA, and Q4 for SVAMP. Each starting model is paired with the trusted model. When the starting model is itself the trusted model, the auxiliary model is Q4 for GSM8K and Q8 for the other three datasets. These pairings are selected on training tasks and frozen before evaluation.

Table 3: Frozen configuration shared by the two evaluation seeds.

Configuration	Seed-1 (2053)	Seed-2 (2063)
Model-generation seed	2053	2063
Evaluation problems	Same 872 problems	
Prompts and evaluator	Shared	
Model pairings and Compass parameters	Frozen and shared	
Strategic Scaling allocation-order seed	2053	2053

Table 4: Training-profiled baseline parameters shared by both evaluation seeds. The average budget is the total Strategic Scaling pool cap divided by its request count.

Dataset	Model	Certaindex	Strategic Scaling		
		threshold	Minimum	Vote fraction	Average budget
GSM8K	Q4	0.5	4	0.5	4
	Q8	0.5	4	0.5	4
	L8	0.8	4	0.7	16
ASDiv	Q4	1.0	4	0.7	8
	Q8	0.5	4	0.5	4
	L8	0.5	4	0.5	4
SVAMP	Q4	0.5	4	0.5	4
	Q8	0.5	4	0.7	8
	L8	0.8	4	0.9	8
StrategyQA	Q4	0.5	4	0.5	4
	Q8	0.6	4	0.5	4
	L8	0.7	8	0.7	16

A.2 Answer Processing

Execution. Each model call uses its recorded completion prompt and task-specific stop sequences, without an additional policy-specific prompt. Arithmetic programs execute in subprocesses separate from model inference and private scoring, with a three-second timeout, a two-second central processing unit (CPU) time limit, and a 512 MiB address-space limit.

Answer aggregation. Numeric aggregation excludes invalid extractions. The shared text-majority interface retains all extracted text, including empty strings; ESC and Strategic Scaling additionally require valid, nonempty answers for their unanimity and elimination tests. Invalid samples remain in token accounting and may prevent agreement. Correctness is evaluated after policy submission.

A.3 Metrics and Paired Comparisons

Accuracy is the macro mean over the 12 dataset–model cells. **TPC** pools total consumed output tokens and correct answers over all requests. Output tokens include auxiliary-model generations without model-size weighting. All consumed model calls, including invalid outputs and failed executions, are counted; unused samples and data-collection costs are excluded. **Paired component analyses** resample tasks within each dataset, keeping all model records for a task together.

A.4 Baseline Rules and Parameters

Adaptive-Consistency. Let u and v count the most frequent and second-most frequent extracted answers, with $v = 0$ if only one answer has appeared. AC stops when $1 - I_{0.5}(u+1, v+1) \geq 0.95$ or at the common cap, where I is the regularized incomplete beta function.

ESC. ESC checks nonoverlapping windows of five samples, submitting the window's answer if all five valid answers agree; otherwise it reaches the cap and submits the overall majority.

Certaindex. Certaindex uses normalized certainty $1 - H/\log n$, where H is the empirical answer entropy after n calls and each invalid extraction is a separate category. The evaluated profiled configuration checks every five samples using Table 4.

Table 5: Component ablations. Acc is macro accuracy (%); TPC is pooled output tokens per correct answer, and ΔTPC is its relative change from Full Compass within each seed. "Both" disables strategy selection and value-aware continuation.

Variant	Seed-1 (2053)			Seed-2 (2063)		
	TPC↓	ΔTPC	Acc↑	TPC↓	ΔTPC	Acc↑
Full Compass	**601.6**	**0.0%**	**88.8**	**595.3**	**0.0%**	**89.1**
w/o Strategy Selection	626.1	+4.1%	89.0	608.0	+2.1%	89.1
w/o Value Continuation	690.8	+14.8%	88.6	684.9	+15.0%	88.9
w/o Terminal Value	685.6	+14.0%	88.6	681.0	+14.4%	88.9
w/o Both	719.2	+19.5%	88.7	710.6	+19.4%	88.8

Table 6: Robustness of Compass on Seed-1. Accuracy is macro-averaged (%); TPC is output tokens per correct answer; fallback is unsupported/eligible checkpoints excluding native stops. Grouped ranges combine the tested settings with identical outcomes.

Sweep	Setting	TPC↓	Acc↑	Unsupported / eligible
Training support	16	653.8	89.1	1,112/4,049 (27.5%)
	32	632.5	89.1	418/4,019 (10.4%)
	64	**601.6**	**88.8**	**229/3,841 (6.0%)**
Quality tolerance ϵ	0	618.7	88.9	n/a
	0.005–0.05	**601.6**	**88.8**	**n/a**
Continuation price λ	0	604.4	88.8	n/a
	0.01–0.04	**601.6**	**88.8**	**n/a**
	0.08	566.9	88.5	n/a

Strategic Scaling uses an Elimination-SC core that visits active requests in a fixed hash order and adds one sample per request per round. After the minimum sample count, a request leaves the pool when its leading valid answer reaches the profiled vote fraction, or at the 40-call cap. Sampling ends when the pool is empty or its budget is exhausted. Each dataset–model cell receives a pool budget equal to its request count times the profiled average budget in Table 4.

B Component and Robustness Results

B.1 Component Comparisons

Table 5 uses matched tasks, sample order, and the shared 40-call cap; ΔTPC is relative to same-seed Full Compass. The no-continuation variant fixes initial actions and completes evidence branches with native AC. The no-terminal-value variant uses agreement stopping and refits the selector, so its initial choice also changes.

Paired uncertainty. For the variants in Table 5, 20,000 Seed-2 task-clustered bootstrap resamples at the nominal 99.1667% level give TPC-reduction intervals of $[1.27\%, 3.37\%]$ versus w/o Strategy Selection and $[7.37\%, 18.26\%]$ versus w/o Terminal Value. Tasks are resampled within datasets, keeping all model records together. Both TPC intervals exclude zero; the accuracy-difference intervals include zero.

B.2 Starting-to-Trusted AC

Starting-to-Trusted AC consumes the starting model's first answer, then samples the same dataset-specific trusted model available to Compass until native AC stops or the shared 40-call cap is reached. The final answer is the trusted-model majority. It uses the same frozen model pairings and answer banks as the end-to-end evaluation.

Table 7: Starting-to-Trusted AC comparison. Acc is macro accuracy (%); TPC is pooled output tokens per correct answer.

	Seed-1 (2053)		Seed-2 (2063)	
Method	**Acc↑**	**TPC↓**	**Acc↑**	**TPC↓**
Compass	88.84	601.63	89.07	595.34
Starting-to-Trusted AC	89.07	1088.98	88.76	1062.77

Table 7 shows that Compass reduces TPC by 44.8% and 44.0% on the two seeds, respectively, with both methods attaining approximately 89% macro accuracy.

B.3 Training Support and Controller Settings

Table 6 varies Seed-1 training support, $\epsilon \in \{0, 0.005, 0.01, 0.02, 0.05\}$, and $\lambda \in \{0, 0.01, 0.02, 0.04, 0.08\}$ one factor at a time; the price sweep fixes initial actions. Unsupported fallback falls from 1,112/4,049 at 16 tasks per dataset to 229/3,841 at 64. The denominators differ because policy trajectories change. Grouped parameter rows denote tested discrete settings with identical outcomes.

C Controller Accounting and Overhead

Token accounting. On Seed-2, Compass consumes 1,370,469 output tokens, 10,444,450 input tokens, and 12,323 model calls over 2,616 model–problem instances. Its macro output cost per request is 469.94 tokens; this macro mean differs from the pooled numerator used for TPC.

Controller execution. A warmed local CPU traversal of the controller takes 11.98 ms per request on average and 17.69 ms at the 95th percentile, including aggregation and instrumentation. Individual predictor calls average 1.65 ms, and the serialized predictor occupies 9,429,945 bytes. These measurements profile controller execution, with model inference and grading outside the timing scope.

The traversal covered 2,616 requests and 14,305 predictor calls.

D Scope and Limitations

Evaluation scope. The evaluation covers arithmetic and commonsense reasoning with fixed-cap sample banks and two generation seeds on shared task identities. The training-support sweep characterizes the tested support range. Compass's quality and continuation estimates depend on the available training support and native-tail outcomes.

Deployment scope. TPC measures unweighted output-token consumption; Appendix C reports controller CPU timings. End-to-end serving also depends on batching, contention, model switching, and prefill/decode costs outside the replay evaluation. Online serving and training-cost amortization are directions for deployment studies.