CONGRATTS: CONVERGENCE-GUIDED RESOURCE-AWARE TEST-TIME SCALING FOR LOCAL LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling (TTS) can improve language-model outputs by repeated output sampling or refinement, but the number of scaling rounds required varies across inputs: some outputs stabilize after only a few rounds, while others continue to change, and the cost of another round depends on system resources currently available. We introduce CONGRATTS (**CON**vergence-**G**uided **R**esource-**A**ware **TTS**), a framework for adaptive TTS of locally-hosted models. CONGRATTS decides after each scaling round whether to continue by jointly considering output convergence and live resource feasibility. The same framework supports both self-consistency and self-refinement scaling, without retraining, model modification, hidden-state access, or an external verifier model, and is evaluated across multiple small language models, related datasets, and distinct inference backends. In our evaluation, CONGRATTS reduces self-consistency scaling rounds by 48.2% with only a 0.39 percentage-point accuracy decrease relative to fixed-budget scaling, and self-refinement rounds by 88.3% while improving accuracy by 1.96 percentage points. Under controlled resource contention on both a MacBook Pro (M1 Max) and Jetson AGX Orin, it reduces latency by 34.4–87.6% and energy use by 31.6–87.9% relative to fixed-budget scaling, with only a 0.38 percentage-point average accuracy decrease across both platforms. Our code is available at: `https://anonymous.4open.science/r/congratts-E021`.

1 INTRODUCTION

Large language models (LLMs) now support a broad range of applications, including code generation, general reasoning, document question-answering, and retrieval-augmented generation (RAG) (Li et al., 2024a; Zhu et al., 2025). These applications are often served from the cloud, introducing potential issues around data privacy, connectivity, and infrastructure costs (Xue et al., 2024). Hosting LLMs locally provides stronger privacy and security guarantees (Qu et al., 2025), reduces dependence on network connectivity and vendor-side serving infrastructure, and lowers serving cost. However, memory, compute, and energy constraints generally limit local inference[1] to relatively small (*e.g.*, 1–2B-parameter) models (Liu et al., 2024; Alizadeh et al., 2024), whose capabilities lag behind those of frontier models.

Test-time scaling (TTS) can recover some of this capability by allocating additional computation during inference (Snell et al., 2024). Parallel **self-consistency (SC)** scaling uses repeated output sampling, generating independent candidate responses and aggregating their answers (Wang et al., 2023), whereas sequential **self-refinement (SR)** scales through iterative revision of a single response (Madaan et al., 2023). Their relative benefits are application-dependent; as Figure 1 illustrates, SC performs best in settings with well-defined outputs, such as on mathematical reasoning, while SR performs best on retrieval-based question-answering (QA), where an output can be progressively improved through feedback or additional evidence. Moreover, applying an unsuitable scaling method can be worse than performing no scaling at all; on the factual-retrieval QA benchmark, SC underperforms the unscaled base response. Consequently, supporting both parallel and sequential scaling can be beneficial across applications (Ren et al., 2025; Chang et al., 2025).

[1]We use *local inference* to mean executing an LLM directly on user-owned, commodity hardware, such as a Mac M-series system, a deployment pattern increasingly supported by dedicated LLM inference runtimes.

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.

One central challenge with TTS is evaluating, after each scaling round, whether further scaling is justified; for example, an SR response may improve after an early refinement but deteriorate under subsequent revisions. Across our evaluation, we find additional scaling is not uniformly beneficial (Appendix B.1), hence the appropriate stopping point depends on the output behavior observed as scaling proceeds (Wu et al., 2025; Zhou et al., 2026).

This challenge is especially important for resource-constrained inference, *i.e.*, with locally-hosted models, as scaling also entails variable system costs. Each sample or refinement round incurs overhead in latency, memory bandwidth, energy, and key–value (KV) cache capacity (Jin et al., 2025; Kumar et al., 2026), while available resources may vary during a request as foreground applications contend for CPU or accelerator time, memory pressure increases, or power and thermal limits are imposed (Laskaridis et al., 2024). Figure 2 illustrates how, for the same LLM, dataset, and scaling paradigm, both latency and energy consumption vary substantially across queries. Consequently, an adaptive stopping rule should continue only for as long as the current output has not yet converged and another scaling round remains feasible under current resource usage. Given that resource availability can change during query processing, an upfront-allocated scaling budget can become misaligned with the current system state; this motivates a causal, per-round decision driven by both the observed scaling history and the current resource state.

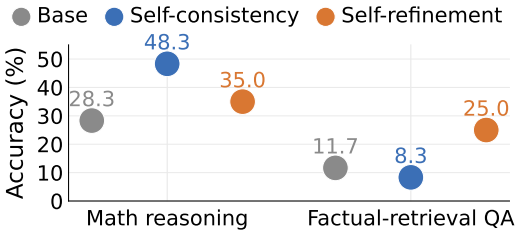

Figure 1: **Scaling paradigms are application-dependent.** Accuracy of the unscaled response (Base), 16-round self-consistency (SC), and 16-round self-refinement (SR) using `Qwen2.5-1.5B` on 60 MATH-500 and 60 CRAG queries. On CRAG, applying SC is worse than using the unscaled Base response.

We introduce *CONvergence-Guided Resource-Aware TTS*, or CONGRATTS, a lightweight framework for adaptive TTS with locally-hosted LLMs. At each scaling state, CONGRATTS determines whether to execute another round by jointly evaluating output convergence and live resource feasibility. We define convergence with respect to a fixed maximum scaling budget: the current answer is deemed converged if it agrees sufficiently with the answer selected at this budget. We use convergence rather than ground-truth correctness because convergence *(i)* depends only on how the model's output evolves under further scaling, whereas correctness is inherently tied to the task itself, *(ii)* requires no ground-truth labels or online evaluation and can be learned in a self-supervised manner, and *(iii)* is defined for every complete scaling run. CONGRATTS estimates convergence using only runtime-accessible features, and simultaneously evaluates whether another round is feasible under active resource constraints. The controller operates independently of the underlying LLM and requires no model modifications, access to hidden states, an external verifier, or additional LLM calls. This design also permits *on-device personalization*: the convergence model can be trained or calibrated from fixed-budget runs on a user's own queries, while resource costs are profiled on that device.

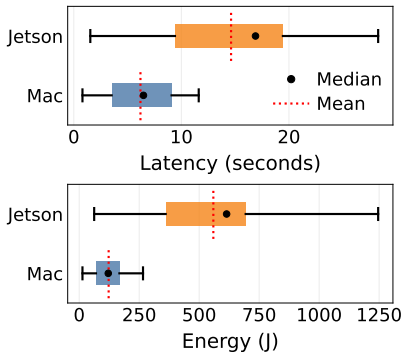

Figure 2: **Inference cost varies substantially across queries.** Inference latency and energy consumption for 60 MATH queries, using 16-step self-consistency with `Qwen2.5-1.5B`, on a MacBook Pro M1 Max and NVIDIA Jetson AGX Orin.

CONGRATTS handles both SC and SR scaling using the same underlying controller, without requiring retraining. It generalizes across four open-weight 1–2B-parameter LLMs, multiple reasoning and retrieval-augmented QA benchmarks, and distinct inference backends. Under SC, CONGRATTS reduces the mean number of scaling rounds by 48.2% while incurring a 0.39 percentage-point decrease in accuracy. Under SR, it reduces refinement rounds by 88.3% while improving accuracy by 1.96 percentage points, avoiding later refinements that degrade strong intermediate answers. Under controlled CPU and memory contention on both a MacBook Pro M1 Max and

NVIDIA Jetson AGX Orin, it reduces cumulative latency by 34.4–87.6% and energy consumption by 31.6–87.9% relative to fixed-budget scaling. Our contributions are:

- **Resource-aware scaling for constrained deployment.** We incorporate live resource feasibility directly into the adaptive scaling decision, and evaluate this under controlled resource contention on commodity hardware.
- **One controller for both dominant scaling paradigms.** Our framework supports both parallel SC and sequential SR scaling using the same underlying controller, without requiring retraining, reducing scaling rounds by 48.2% and 88.3%, respectively.
- **Generalization across deployment settings.** We show that our framework's convergence model, trained solely on `Qwen2.5-1.5B` (Qwen et al., 2025), generalizes without retraining to three additional LLMs and multiple datasets, while preserving a similar accuracy–compute profile under quantization. Our resource-feasibility formulation is also shared across deployments, while the exact resource bounds used, and any profiled per-round system costs, are configured separately for each model–platform pair.

2 Background and Related Work

Table 1: Comparison of related adaptive TTS methods. △ denotes partial or limited evidence.

Method	Scaling type	Decision point	External features	No changes to model	Self-labelled training	Portable (across models/tasks)	Includes resource usage
(Liu & Wang, 2025)	Sequential	Within-state	×	✓	✓	×	×
(Dai et al., 2026)	Parallel	Across samples	✓	✓	✓	✓	×
ASC (Aggarwal et al., 2023)	Parallel	Across samples	✓	✓	—	✓	×
ESC (Li et al., 2024b)	Parallel	Across samples	✓	✓	—	✓	×
RASC (Wan et al., 2025a)	Parallel	Across samples	△	✓	×	△	×
CGES (Aghazadeh et al., 2026)	Parallel	Across samples	△	✓	—	△	×
ReASC (Kim et al., 2026)	Parallel	Across samples	△	✓	—	△	×
CoRefine (Jin et al., 2026)	Sequential	Across refinements	△	✓	×	△	×
(Hammar et al., 2026)	Sequential	Across refinements	△	✓	×	△	△
(Chang et al., 2025)	Both	Within/Across rounds	△	✓	—	△	×
CONGRATTS	Both	Across rounds	✓	✓	✓	✓	✓

Table 1 compares representative adaptive TTS methods by scaling paradigm, decision point, model requirements, portability, and resource awareness. We summarize the key methods below and provide a more detailed discussion of related work in Appendix A.

Adaptive parallel & sequential scaling. Adaptive parallel scaling largely builds on SC, using answer agreement, posterior concentration, or token-level confidence to decide when to stop (Aggarwal et al., 2023; Li et al., 2024b; Aghazadeh et al., 2026). Other methods include rationale-quality scoring, response-level confidence calibration, process-reward models, or reinforcement-learned controllers (Wan et al., 2025a; Kim et al., 2026; Wan et al., 2025b; Dai et al., 2026). Sequential scaling has received less attention. CoRefine (Jin et al., 2026) trains a controller to halt, rethink, or explore alternatives using token-confidence statistics. Hammar et al. (Hammar et al., 2026) formulate refinement as an optimal-stopping problem that balances expected improvement against cost, but they rely on external feedback. Hybrid TTS (Chang et al., 2025) combines sequential revision with parallel exploration, using process-level verification to identify where further computation is needed. Existing methods typically specialize in a single scaling paradigm or rely on model-specific signals, external feedback, or separately trained control components. In contrast, CONGRATTS uses a single runtime-accessible convergence representation across SC and SR, transferring without retraining.

Within-state scaling & verification. Another related but distinct line of work reduces computation *within* a single generation by stopping reasoning early, detecting redundancy in reasoning, or verifying a completed response (Liu & Wang, 2025; Hosseini et al., 2026; Min et al., 2026; Wang et al., 2026; Liu et al., 2026; Sun et al., 2026; Tikhonov et al., 2026; Xie et al., 2026; Zhuang et al., 2026; Dip et al., 2026). These methods decide how much computation to spend *within* one generation, whereas our setting decides whether to execute another complete sample generation or refinement round. Both approaches are therefore complementary.

Systems-level optimizations for TTS. Recent systems work targets the execution cost of TTS on constrained hardware by optimizing memory management, scheduling, quantization, concurrency, and other hardware-specific optimizations (Chen et al., 2026; Hao et al., 2026; Wang et al., 2025b; Xiong et al., 2026a). These methods primarily improve *how* a chosen scaling workload is executed, not *if* another round should be executed.

3 DESIGNING CONGRATTS

We detail CONGRATTS, a resource-feasible adaptive TTS method for locally-hosted LLMs. Given a query, CONGRATTS decides after each scaling round whether to stop or continue, continuing only when the current answer is not yet predicted to have converged to a fixed-budget answer and another round can be executed under defined bounds on resource-usage.

3.1 CONVERGENCE TARGETS FROM FIXED-BUDGET RUNS

A natural way to formulate adaptive TTS is to ask whether the answer produced at an intermediate scaling round is correct. However, ground-truth correctness cannot be observed during deployment without external verification. Moreover, correctness is inherently task-specific, and a valid target is not always present for a given query. We therefore do not train CONGRATTS to predict correctness directly. Instead, we predict whether the current scaling state has converged to the output produced under a fixed maximum scaling budget. We further discuss this design choice and evaluate it in Appendices B.4 and C.4, respectively.

For each query q_i, we run the scaling procedure to a maximum budget T. For SC, $t \in \{1, \ldots, T\}$ denotes the number of sampled responses, and $a_{i,t}$ is the aggregated answer after t samples. For SR, $t \in \{0, \ldots, T\}$ denotes the number of refinement rounds, where $a_{i,0}$ is the initial model response and $a_{i,t}$ for $t \geq 1$ is the output after t refinements. In both regimes, $a_{i,T}$ is the fixed-budget reference output. We then define a binary convergence target at each valid scaling state t, $c_{i,t} \in \{0, 1\}$, where $c_{i,t} = 1$ indicates that the current output has sufficiently converged to the fixed-budget output. This formulation covers both scaling regimes considered in this work.

For SC-style scaling, the intermediate output is the current aggregated answer after t samples. When answers are discrete or can be normalized, such as numeric answers, equations, or multiple-choice options, we define convergence using normalized answer agreement:

$$c_{i,t} = \mathbf{1}\left[\mathrm{norm}(a_{i,t}) = \mathrm{norm}(a_{i,T})\right]$$

For SR-style scaling, the intermediate output is the answer obtained after t refinement rounds. Since these outputs may be free-form texts, exact lexical agreement with $a_{i,T}$ is overly strict. We instead compute a semantic similarity score $s^{\mathrm{sem}}_{i,t} = \cos\left(\phi(a_{i,t}), \phi(a_{i,T})\right)$, where $\phi(\cdot)$ is a text encoder and $\cos(\cdot, \cdot)$ denotes cosine similarity. We then convert this score into a binary convergence target:

$$c_{i,t} = \mathbf{1}\left[s^{\mathrm{sem}}_{i,t} \geq \alpha\right],$$

where α is a semantic-convergence threshold. This is a pointwise notion of convergence: a state may match the fixed-budget output even if a later, unexecuted state would diverge.

Thus, both SC and SR yield the same kind of supervision: a binary label indicating whether the current output has converged sufficiently to the fixed-budget output.

Since our target is constructed from completed fixed-budget runs, no ground-truth labels are required; a new dataset can be calibrated by running the scaling procedure to budget T on held-out queries, yielding supervision on whether further scaling is likely to change the final output.

3.2 CONVERGENCE ESTIMATION

Fixed-budget runs provide supervised examples for training a convergence estimator. For each query q_i and scaling state $t < T$, the input is a causal summary of the scaling history observed so far, and the target is the binary convergence label $c_{i,t}$ defined in Section 3.1. The estimator is trained offline and then used online within the runtime controller.

At each scaling state t, we compute a runtime feature vector $\mathbf{x}_{i,t}$ using only signals available from the scaling process. These features are shared across scaling regimes, and include budget-progress signals, answer-distribution statistics, token-level likelihood statistics, posterior confidence, running statistics, and consensus features. For SC, these statistics are computed over samples observed up to round t; for SR, they are computed over the sequence of refined answers up to round t. Thus, $\mathbf{x}_{i,t}$ never depends on future scaling outputs. Details about the features and how they are computed are provided in Appendix B.3.

We denote the convergence estimator by f_θ. Given $\mathbf{x}_{i,t}$, it outputs $\hat{c}_{i,t} = f_\theta(\mathbf{x}_{i,t})$, where $\hat{c}_{i,t}$ is the predicted probability that the current output has converged sufficiently to the fixed-budget output. Both scaling regimes share this interpretation: in SC, convergence means normalized agreement of answers with $a_{i,T}$; in SR, it means semantic similarity to $a_{i,T}$ of at least the fixed threshold α.

The final scaling state is excluded from training because its target is degenerate: $a_{i,T}$ is compared with itself, so $c_{i,T} = 1$ by construction. Here and below, t ranges over the valid scaling states defined for the corresponding regime. We define the training mask

$$m_{i,t} = \begin{cases} 1, & t < T, \\ 0, & t = T. \end{cases}$$

The estimator is trained offline using completed fixed-budget runs:

$$\mathcal{L}_{\text{conv}} = -\frac{1}{\sum_i \sum_t m_{i,t}} \sum_i \sum_t m_{i,t} \left[c_{i,t} \log f_\theta(\mathbf{x}_{i,t}) + (1 - c_{i,t}) \log \left(1 - f_\theta(\mathbf{x}_{i,t})\right) \right].$$

The learned estimator provides a causal convergence probability at each scaling round. For each scaling round, the controller combines this probability with the current resource state to decide whether or not to continue.

3.3 Resource-Feasible Scaling Controller

At deployment, CONGRATTS uses the trained convergence estimator inside the scaling loop. The controller continues only when two conditions hold: the current output is not yet predicted to have converged, and a scaling round is feasible under the current resource state.

Given the convergence estimate $\hat{c}_{i,t} = f_\theta(\mathbf{x}_{i,t})$, the controller compares it against a state-dependent threshold $\psi_t = \psi_0 - \lambda \frac{t}{T}$, where ψ_0 is the base threshold and λ controls its decay rate. This schedule makes the stopping rule conservative early in the scaling rounds and gradually makes it more willing to stop as the maximum budget is approached.

Algorithm 1 CONGRATTS

Require: Query q_i, model M, max budget T, estimator f_θ, threshold ψ_t

for each valid scaling state $t < T$ **do**
  Obtain the current answer $a_{i,t}$
  Compute causal features $\mathbf{x}_{i,t}$
  $\hat{c}_{i,t} \leftarrow f_\theta(\mathbf{x}_{i,t})$
  Evaluate next-round feasibility $g_{i,t}$
  if $g_{i,t} = 0$ **or** $\hat{c}_{i,t} \geq \psi_t$ **then**
    return $a_{i,t}$
  end if
  Run next scaling round with M on q_i
  and obtain $a_{i,t+1}$
end for
return $a_{i,T}$

Let $g_{i,t}^{(r)} \in \{0, 1\}$, for $t < T$, denote whether another scaling round is permitted under resource constraint r, given the resource state observed at scaling state t, the configured resource bound, and, where applicable, the profiled per-round resource cost. Then, we define $g_{i,t} = \min_{r \in \mathcal{R}} g_{i,t}^{(r)}$, where $\mathcal{R}$ denotes the set of active resource constraints, such as CPU utilization, memory pressure, or explicit query-level energy and latency budgets. Consequently, $g_{i,t} = 1$ only when another round is permitted under all active resource constraints.

For a valid scaling state $t < T$, the runtime action is then

$$\text{action}_{i,t} = \begin{cases} \texttt{continue}, & \text{if } g_{i,t} = 1 \text{ and } \hat{c}_{i,t} < \psi_t, \\ \texttt{stop}, & \text{otherwise.} \end{cases}$$

Algorithm 1 summarizes our online controller. This same controller supports both SC and SR scaling. Both components address distinct failure modes: a convergence-only stopping rule is independent of live resource availability and cannot adapt to changing resource bounds during inference. The resource-feasibility formulation is shared across deployments, but its resource bounds and any profiled per-round costs are configured separately for each model–platform pair.

4 Implementation

Models, scaling, and benchmarks. We evaluate four instruction-tuned 1–2B-parameter LMs: `Qwen2.5-1.5B` (Qwen et al., 2025), `Llama-3.2-1B-Instruct` (Llama Team, 2024),

SmolLM2-1.7B-Instruct (Allal et al., 2025), and LFM2.5-1.2B-Instruct (Amini et al., 2025), including 8- and 4-bit bitsandbytes variants where supported. Unless otherwise stated, the maximum scaling budget is $T = 16$: 16 samples for SC and 16 refinements after the initial response for SR. This horizon captures most observed scaling gains (Appendix B.1). We evaluate SC on MATH-500 (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and SVAMP (Patel et al., 2021), and SR on the RAG datasets CRAG (Yang et al., 2024) and POPQA (Mallen et al., 2023). We use deterministic held-out subsets of up to 500 examples per benchmark (Appendix B.1).

Our SR setting is retrieval-augmented. The initial state $t = 0$ is produced by one answer-generation call; each subsequent refinement round consists of an action-selection call followed by a second call that generates a complete revised answer from the query and retrieved evidence, so t refinements require $1 + 2t$ model calls; full retrieval details are provided in Appendix B.1.

Convergence model and training. For each seed, we sample 300 queries from MATH and each SR dataset, then split them 80/20 for training and validation. We train a histogram-based gradient-boosting (HGB) convergence model jointly on the three 240-query training splits using Qwen2.5-1.5B outputs, then reuse it unchanged across downstream models, quantization levels, and both scaling paradigms. Its causal features capture scaling progress, answer-distribution and consensus statistics, and token-level log-probability statistics; the full feature set and feature-group ablations are given in Appendices B.3 and C.1. HGB inference adds negligible overhead relative to an LLM generation round[2]; alternative lightweight models are evaluated in Appendix C.8. We ablate the amount and source of training data in Appendices C.2 and C.3.

SC convergence uses normalized agreement with the T-round consensus. We ablate the SC agreement criterion in Appendix C.5. SR convergence uses cosine similarity of at least $\alpha = 0.90$ with the T-round refinement, computed using EmbeddingGemma-300M (Vera et al., 2025). This is consistent with semantic-equivalence methods used in prior refinement-stopping work (Yan et al., 2025). Threshold and embedding-model ablations appear in Appendices C.6 and C.7.

Baselines and calibration. For SC, we compare against ASC (Aggarwal et al., 2023), ESC (Li et al., 2024b), CGES (Aghazadeh et al., 2026), the unscaled response (Base), and 16-round scaling. For SR, we compare against Base, 4-, and 16-round scaling, semantic-answer-streak stopping (Yan et al., 2025), and edit-saturation stopping (Wu et al., 2026). Details are in Appendix B.5.

Thresholds are calibrated separately for SC and SR using held-out validation data: 60 MATH queries for SC and 60 queries for each of the SR datasets. SC calibration selects the lowest-round rule allowing at most one disagreement with the 16-round consensus; SR minimizes error to the earliest semantic-convergence state. Non-learned baselines are calibrated on the corresponding 300-query development sets. Further details and ablations are provided in Appendices B.6 and C.9.

Hardware evaluation. We evaluate resource-aware scaling on a MacBook Pro M1 Max and NVIDIA Jetson AGX Orin using 60 matched queries under idle, CPU, memory, and combined CPU–memory contention. We focus on CPU and memory contention because it reflects local inference alongside user-facing applications, whereas accelerator contention is often infeasible on the hardware considered. LLM generation on constrained hardware is also largely memory-bound (Laskaridis et al., 2024), while CPU contention can delay host-side sampling and scheduling. CPU contention uses five workload settings with workers active for 50–90% of each 100ms interval; memory pressure uses five settings spanning 45–95% of physical memory with stress-ng (King, 2026). Complete workload and hardware details are given in Appendix D.

We report accuracy, scaling rounds, end-to-end latency, energy, and resource-limit violations. CPU utilization above 90% is a violation on both platforms; on Jetson, less than 4GB of MemAvailable is a memory violation, while on macOS, a VM-pressure warning prevents further scaling and critical is treated as a violation. Where required, next-round feasibility uses the 95th-percentile incremental cost from disjoint profiling runs for each model–platform pair, with additional reserve for recent resource fluctuations. Full details are in Appendix D. All hardware inference uses llama.cpp with GGUF-format models (Gerganov & contributors, 2026).

[2]The stopping decision adds 0.56% and 2.52% of mean per-round inference latency on the MacBook Pro and Jetson, respectively (Appendix C.8).

Table 2: **Performance on MacBook Pro (M1 Max) & Jetson AGX Orin.** *Fixed* attempts all 16 rounds; + *Resource* adds proactive resource-aware stopping; and + *Conv.* additionally applies convergence-guided stopping. *Controller Miss*, *Base Invalid*, and *Post-Base Invalid* denote violations after adaptive continuation, during the initial inference, and during later retained states, respectively. Values are mean ± standard deviation across three runs.

Device	Scaling	Method	Acc. (%)	Rounds	Lat. (s)	Energy (J)	Controller Miss (%)	Base Invalid (%)	Post-Base Invalid (%)
MacBook Pro (M1 Max)	SC	Fixed	46.25 ± 1.25	11.73 ± 0.99	70.90 ± 4.69	1406.6 ± 40.2	–	3.89 ± 3.24	8.61 ± 3.39
		+ Resource	46.11 ± 0.87	11.56 ± 1.22	71.11 ± 8.55	1424.6 ± 105.7	3.19 ± 2.51	4.58 ± 5.91	3.19 ± 2.51
		+ Conv.	44.86 ± 0.48	6.89 ± 0.85	46.54 ± 5.76	962.4 ± 111.8	0.56 ± 0.24	0.14 ± 0.24	0.56 ± 0.24
	SR	Fixed	23.47 ± 0.24	13.30 ± 0.89	76.77 ± 1.16	2797.1 ± 321.9	–	0.42 ± 0.42	7.36 ± 8.06
		+ Resource	24.17 ± 0.00	12.87 ± 0.19	71.36 ± 1.22	2800.6 ± 24.0	1.25 ± 0.42	0.28 ± 0.24	1.25 ± 0.42
		+ Conv.	24.03 ± 1.05	3.16 ± 0.57	9.50 ± 3.29	342.0 ± 119.9	0.97 ± 0.48	0.69 ± 1.20	0.97 ± 0.48
Jetson AGX Orin	SC	Fixed	48.19 ± 0.24	12.29 ± 0.00	192.61 ± 1.01	7189.2 ± 55.6	–	21.11 ± 0.48	3.89 ± 0.48
		+ Resource	47.78 ± 0.64	10.28 ± 0.09	160.44 ± 2.22	5780.6 ± 99.1	2.64 ± 0.24	14.58 ± 0.42	2.64 ± 0.24
		+ Conv.	47.08 ± 0.42	6.13 ± 0.58	106.16 ± 8.80	3913.5 ± 319.8	2.50 ± 0.83	21.81 ± 0.64	2.50 ± 0.83
	SR	Fixed	22.64 ± 0.24	13.35 ± 0.02	90.80 ± 0.17	4180.6 ± 20.3	–	10.97 ± 0.64	14.03 ± 0.64
		+ Resource	21.11 ± 0.24	11.77 ± 0.03	79.54 ± 0.18	3562.4 ± 9.7	9.17 ± 0.42	0.69 ± 0.48	9.17 ± 0.42
		+ Conv.	23.06 ± 1.34	3.03 ± 0.52	11.55 ± 3.54	507.4 ± 161.9	1.94 ± 1.05	10.28 ± 0.24	1.94 ± 1.05

5 RESULTS

Main Evaluation. Table 2 evaluates scaling under live CPU and memory contention. *Fixed* attempts all 16 rounds, stopping only when a resource limit is reached; + *Resource* adds pro-active resource-aware stopping, and + *Conv.* adds convergence-guided stopping alongside this. *Base Invalid* records resource violations during the mandatory initial inference for each query, when the controller has not yet made a stopping decision. We report these cases separately and do not interpret differences in this metric as effects of the controller. Under sustained contention, the resource state at the start of a query may also reflect residual load from the preceding query.

The adaptive methods substantially reduce resource violations after the mandatory base inference. Compared to fixed scaling, *Post-Base Invalid* drops from 8.61% to 0.56% for Mac SC, and from 7.36% to 0.97% for SR, and 3.89% to 2.50% for Jetson SC, and 14.03% to 1.94% for Jetson SR, corresponding to relative reductions of 35.7–93.5%. Convergence guidance further drops *Controller Miss* across all four settings by avoiding later rounds that would exceed a resource limit.

These reductions coincide with substantial inference savings. Compared with fixed scaling, + *Conv.* reduces SC latency by 34.4% on the Mac and 44.9% on the Jetson, and energy by 31.6% and 45.6%, respectively, with accuracy decreases of 1.39 and 1.11 percentage points. For SR, latency drops by 87.6% and 87.3%, and energy by 87.8% and 87.9%, while accuracy increases by 0.56 and 0.42 percentage points, respectively. Over all four settings, end-to-end latency decreases by 34.4–87.6% and energy by 31.6–87.9%, with a mean decrease in accuracy of 0.38 percentage points.

Compared to resource-aware stopping alone, + *Conv.* reduces mean scaling depth by 40.4% for SC and 74.2–75.4% for SR, yielding a further 33.8–34.6% and 85.5–86.7% reduction in latency, respectively. Notably, for SR, latency and energy drop more sharply than refinement depth (85.5–87.8% versus 74.2–75.4%), showing that the cost of refinement varies substantially across rounds, increasing with later refinements as context accumulates.

Convergence Model Evaluation. Tables 3 and 4 evaluate our learned convergence model independently of live resource constraints. On SC, CONGRATTS is competitive with specialized adaptive-sampling baselines; however, its key distinction is that the same learned convergence model also supports sequential refinement without retraining. CONGRATTS preserves SC accuracy to within 0.39 percentage points of our fixed 16-round budget runs while reducing mean scaling rounds by 48.2%. Compared with our adaptive baselines, it uses fewer samples than ASC at slightly lower accuracy, outperforms ESC on both accuracy and sample count, and achieves slightly higher accuracy than CGES while using more samples. For SR, the initial retrieval-augmented refinement produces a large early accuracy gain, after which additional refinement produces smaller, query-dependent, and non-monotonic effects. CONGRATTS reaches 23.31% average accuracy using 1.88 refinement rounds, exceeding all fixed-round baselines while using substantially fewer rounds than the adaptive edit-saturation and semantic-streak baselines. These latter heuristics also benefit from avoiding over-refinement, improving accuracy by 1.44 and 1.93 percentage points over fixed-budget scaling, respectively; CONGRATTS primarily improves the accuracy–compute profile. This behavior may

Table 3: **Self-consistency (SC).** Accuracy (%) / mean scaling rounds. Values are mean ± sample standard deviation across three training/calibration seeds. ASC and ESC yield identical calibrated hyperparameters across seeds (ASC $\beta = 0.90$; ESC streak length $= 8$); no standard deviation is shown for these, nor for the fixed-budget methods. Bold denotes the highest mean accuracy in each row; underlining denotes the lowest mean rounds among adaptive methods. The final row reports macro-averages across all model–dataset pairs.

Model	Dataset	Base	Fixed16	ASC	CGES	ESC	CONGRATTS
Qwen2.5-1.5B	MATH-500	37.64 / 1.00	**48.35** / 16.00	48.08 / 11.14	47.53±0.27 / 7.63±0.52	47.25 / 11.03	47.53±0.73 / 10.11±1.21
	GSM8K	59.27 / 1.00	**73.80** / 16.00	**73.80** / 7.68	73.07±0.07 / 5.32±0.37	73.20 / 9.62	73.64±0.32 / 7.39±1.16
	SVAMP	75.33 / 1.00	82.00 / 16.00	81.67 / 6.22	83.11±0.19 / 4.22±0.33	82.33 / 9.02	**83.22**±0.19 / 5.92±1.26
Llama-3.2-1B	MATH-500	24.45 / 1.00	35.71 / 16.00	**35.99** / 12.53	34.34±0.55 / 9.04±0.54	34.34 / 11.55	34.52±1.11 / 11.43±1.13
	GSM8K	35.13 / 1.00	**50.73** / 16.00	50.67 / 10.72	49.73±0.13 / 7.28±0.42	50.60 / 10.80	50.31±0.57 / 9.73±1.31
	SVAMP	58.00 / 1.00	67.67 / 16.00	67.33 / 7.32	66.44±0.51 / 4.30±0.34	**68.67** / 9.60	67.78±0.51 / 6.45±1.22
LFM2.5-1.2B	MATH-500	51.10 / 1.00	**61.81** / 16.00	61.54 / 8.63	60.07±0.16 / 4.96±0.39	60.99 / 10.06	61.08±0.79 / 7.39±1.12
	GSM8K	72.73 / 1.00	78.73 / 16.00	78.80 / 6.17	**79.07**±0.13 / 3.69±0.25	78.93 / 9.12	78.93±0.33 / 5.40±1.03
	SVAMP	80.00 / 1.00	**86.67** / 16.00	**86.67** / 5.08	86.00±0.33 / 3.32±0.23	86.33 / 8.72	86.44±0.19 / 4.81±1.00
SmolLM2-1.7B	MATH-500	23.35 / 1.00	32.69 / 16.00	32.69 / 13.24	32.51±0.32 / 9.77±0.45	**33.24** / 12.01	32.05±0.57 / 12.45±1.07
	GSM8K	41.47 / 1.00	**57.47** / 16.00	57.33 / 10.89	56.29±0.33 / 7.36±0.38	55.87 / 10.85	56.60±0.87 / 9.99±1.27
	SVAMP	55.67 / 1.00	**73.67** / 16.00	**73.67** / 8.84	71.89±0.38 / 6.09±0.40	72.00 / 10.08	72.56±1.07 / 8.39±1.32
Avg. ΔAcc. (pp) / ΔRounds (%)		51.18 / 1.00 -11.26 / -93.8%	**62.44** / 16.00 reference	62.35 / 9.04 -0.09 / -43.5%	61.67±0.17 / 6.08±0.38 -0.77 / -62.0%	61.98 / 10.20 -0.46 / -36.3%	62.06±0.46 / 8.29±1.17 -0.39 / -48.2%

arise because the convergence model learns statistical cues of stabilization, causing it to stop at correct intermediate states before later refinement degrades the answer.

Joint SC–SR training provides a more favorable shared operating point than either single-paradigm training source (Appendix C.3). Its gains hold across all four evaluated LLMs and held-out datasets, and are also largely preserved under quantization: without retraining or recalibration, CONGRATTS retains a similar accuracy–compute profile under both 8-bit and 4-bit inference (Table 5).

Full results on quantized LLMs are shown in Appendix E.1.

Table 4: **Self-refinement (SR).** Accuracy (%) / mean refinement rounds. Values are mean ± sample standard deviation across three training/calibration seeds. Bold denotes the highest mean accuracy in each row; underlining denotes the lowest mean round count among adaptive methods. The final row reports macro-averages across all model–dataset pairs.

Model	Dataset	Base	Fixed4	Fixed16	Edit Sat.	Semantic Streak	CONGRATTS
Qwen2.5-1.5B	CRAG	15.20 / 0.00	27.80 / 4.00	25.80 / 16.00	28.27±0.46 / 2.78±0.52	**29.13**±0.81 / 3.33±0.82	28.13±1.03 / 2.21±0.59
	POPQA	15.00 / 0.00	26.40 / 4.00	24.60 / 16.00	26.40±0.20 / 2.89±0.57	**26.80**±0.20 / 2.97±0.22	26.13±0.31 / 2.33±0.47
Llama-3.2-1B	CRAG	22.60 / 0.00	24.80 / 4.00	22.80 / 16.00	22.47±0.12 / 3.34±0.67	22.87±1.72 / 3.91±1.12	**25.87**±0.12 / 2.01±0.70
	POPQA	14.80 / 0.00	**23.40** / 4.00	21.60 / 16.00	21.73±2.10 / 2.99±0.58	20.87±2.57 / 2.91±0.26	21.80±0.60 / 1.69±0.45
LFM2.5-1.2B	CRAG	13.00 / 0.00	21.40 / 4.00	18.40 / 16.00	21.53±0.12 / 4.63±1.41	**24.20**±0.72 / 5.71±2.56	24.07±0.61 / 2.34±1.03
	POPQA	10.40 / 0.00	23.80 / 4.00	22.00 / 16.00	23.60±0.72 / 3.12±1.18	**24.07**±0.46 / 3.06±0.39	22.40±0.72 / 1.75±0.69
SmolLM2-1.7B	CRAG	17.00 / 0.00	14.80 / 4.00	14.40 / 16.00	17.80±2.09 / 3.88±0.23	18.47±1.81 / 5.07±2.07	**20.40**±0.53 / 1.52±0.58
	POPQA	14.80 / 0.00	20.20 / 4.00	**21.20** / 16.00	20.53±1.50 / 2.85±0.52	19.87±1.15 / 2.80±0.24	17.67±0.64 / 1.16±0.23
Avg. ΔAcc. (pp) / ΔRounds (%)		15.35 / 0.00 -6.00 / -100.0%	22.82 / 4.00 +1.47 / -75.0%	21.35 / 16.00 reference	22.79±0.72 / 3.31±0.36 +1.44 / -79.3%	23.28±0.81 / 3.72±0.76 +1.93 / -76.8%	**23.31**±0.16 / 1.88±0.55 +1.96 / -88.3%

6 Discussion

Joint stopping under live resource constraints. The central design principle in CONGRATTS is that every continuation is conditioned jointly on output convergence and the current resource state. This matters because a scaling round can remain feasible even when no longer necessary, or potentially beneficial even when no longer supportable under current contention. The hardware results make this distinction clear; relative to resource-aware stopping alone, adding convergence guidance reduces mean scaling depth by 40.4% for SC on both platforms and 74.2–75.4% for SR, with a further 33.8–34.6% and 85.5–86.7% reduction in latency, respectively (Table 2). Both signals are part of the same per-round continuation decision.

Scaling depth is not proportional to cost. Our results also show that reducing the number of rounds and reducing system cost are not equivalent. On SR, convergence guidance reduces refinement depth by 74.2–75.4% relative to resource-aware stopping alone, yet latency drops by 85.5–86.7% and energy by 85.8–87.8%. These differences are consistent with the increasing cost of later refinements as context accumulates, although other runtime effects may also contribute. On SC, reductions in

scaling depth and latency are much closer. Thus, scaling depth measures algorithmic computation, but does not fully capture its deployment cost; latency and energy must also be measured directly.

Portability across deployments. Our convergence model is trained using `Qwen2.5-1.5B` and reused unchanged across three additional LLMs and held-out benchmark queries; its behavior is also largely preserved under quantization (Table 5). The same frozen model is used with both Hugging Face Transformers (Wolf et al., 2020) and `llama.cpp`, demonstrating compatibility across the evaluated inference backends. Because our features rely only on generated outputs and token log-probabilities, they are compatible with standard LLM serving interfaces and do not require access to model-specific internal states. The joint continuation rule also remains unchanged across deployments, with resource bounds, measurements, and profiled per-round costs configured for each model–platform pair.

Table 5: **Quantization.** Accuracy (%) / mean scaling rounds, macro-averaged over `Qwen2.5-1.5B`, `Llama-3.2-1B`, and `LFM2.5-1.2B`, and all evaluated datasets. `SmolLM2-1.7B` is excluded below due to a failed 8-bit evaluation (Appendix E.1).

	SC		SR	
Precision	**Fixed16**	**CONGRATTS**	**Fixed16**	**CONGRATTS**
FP16	65.05 / 16.00	64.83 / 7.63	22.53 / 16.00	24.73 / 2.05
8-bit	64.24 / 16.00	64.23 / 7.74	21.70 / 16.00	24.31 / 3.33
4-bit	62.75 / 16.00	62.39 / 8.37	21.67 / 16.00	23.63 / 3.07

Serving-layer implications. Because CONGRATTS relies on observable scaling outputs and system measurements rather than model modification or hidden-state access, the continuation decision can remain outside the underlying LLM. The controller overhead is small relative to generation, adding only 0.56% and 2.52% of the average inference-round latency on the M1 Max and Jetson, respectively (Appendix C.8). This makes CONGRATTS complementary to methods that reduce computation within a generation: within-generation early stopping can reduce the cost of an individual round, while CONGRATTS determines whether the next complete round should execute. Such control could be integrated with serving schedulers that expose queueing, accelerator memory, or concurrency signals, although we do not evaluate these settings.

Limitations and future work. Our evaluation focuses on locally-deployable 1–2B-parameter LLMs and a maximum scaling budget of $T = 16$; larger models and longer scaling horizons may exhibit different convergence behavior. Given our convergence model consumes a fixed-dimensional representation at each scaling state, increasing T does not itself introduce additional long-sequence dynamics; substantially longer scaling is also impractical on the hardware under evaluation. We use CPU and memory as the only active feasibility constraints in our hardware experiments, while latency and energy are measured as outcomes rather than enforced as query-level budgets. We do not evaluate accelerator contention, which is particularly relevant to concurrent SC execution, where the continuation decision may need to account for a feasible batch of additional samples rather than a single next round. Although the joint continuation rule remains unchanged across deployments, resource bounds and profiled per-round costs are platform-specific and may require updating as runtime conditions evolve. Additionally, since feasibility is evaluated between rounds, an unexpected resource spike during an executing round can still cause a violation; conservative cost estimates and reserve margins reduce, but cannot eliminate, this risk. Online adaptation of these estimates could reduce the need for deployment-specific calibration. Evaluating CONGRATTS on additional hardware, including mobile devices, and under broader scaling configurations remains future work.

7 Conclusion

We presented CONGRATTS, a convergence-guided, resource-aware framework for adaptive test-time scaling on locally-hosted LLMs. CONGRATTS supports both parallel self-consistency (SC) and sequential self-refinement (SR) and generalizes across the evaluated language models and related benchmarks without retraining. It reduces SC scaling rounds by 48.2% with only a 0.39 percentage-point accuracy decrease, and reduces SR refinement rounds by 88.3% while improving accuracy by 1.96 percentage points. Under live resource contention on a MacBook Pro M1 Max and Jetson AGX Orin, CONGRATTS reduces latency by 34.4–87.6% and energy by 31.6–87.9% relative to fixed-budget scaling, with a 0.38 percentage-point average accuracy decrease. Our results show that convergence provides a general stopping signal for both parallel and sequential scaling, while live resource monitoring allows scaling decisions to respond to changing resource availability.

REPRODUCIBILITY STATEMENT

We provide the experimental details required to reproduce our results in the main paper and appendix. This includes model and dataset configurations, prompts and generation settings, classifier features and hyperparameters, calibration hyperparameters and process, baseline implementations, details on our evaluation process, quantization settings, and hardware and resource-contention configurations. We also document the resource-monitoring and profiling process, including the constraints and per-round cost estimates used by the resource-feasibility component. The evaluated models and benchmark datasets are publicly available. We also release the complete codebase used for training, evaluation, and our hardware experiments: `https://anonymous.4open.science/r/congratts-E021`.

AI USE STATEMENT

We used LLM-based tools to assist with writing and editing the manuscript, and with writing, debugging, and refining research code. All LLM-assisted text was reviewed and edited by the authors, as was all LLM-generated or modified code. LLMs were not relied upon to make any final scientific or experimental decisions. The authors take full responsibility for the final content of this work, including all text, claims, results, code, and other artifacts produced with the aid of LLMs.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's Sample Step by Step: Adaptive-Consistency for Efficient Reasoning and Coding with LLMs, 2023. URL https://arxiv.org/abs/2305.11860.

Ehsan Aghazadeh, Ahmad Ghasemi, Hedyeh Beyhaghi, and Hossein Pishro-Nik. CGES: Confidence-Guided Early Stopping for Efficient and Accurate Self-Consistency, 2026. URL https://arxiv.org/abs/2511.02603.

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S. Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.678. URL https://aclanthology.org/2024.acl-long.678/.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model, 2025. URL https://arxiv.org/abs/2502.02737.

Alexander Amini, Anna Banaszak, Harold Benoit, Arthur Böök, Tarek Dakhran, Song Duong, Alfred Eng, Fernando Fernandes, Marc Härkönen, Anne Harrington, Ramin Hasani, Saniya Karwa, Yuri Khrustalev, Maxime Labonne, Mathias Lechner, Valentine Lechner, Simon Lee, Zetian Li, Noel Loo, Jacob Marks, Edoardo Mosca, Samuel J. Paech, Paul Pak, Rom N. Parnichkun, Alex Quach, Ryan Rogers, Daniela Rus, Nayan Saxena, Bettina Schlager, Tim Seyde, Jimmy T. H. Smith, Aditya Tadimeti, and Neehal Tumma. LFM2 Technical Report, 2025. URL https://arxiv.org/abs/2511.23404.

Apple Inc. *powermetrics(1) - macOS Manual Page*. Apple Inc. macOS command-line utility for collecting CPU, GPU, SoC, and power-related statistics.

Kaiyan Chang, Yonghao Shi, Chenglong Wang, Hang Zhou, Chi Hu, Xiaoqian Liu, Yingfeng Luo, Yuan Ge, Tong Xiao, and JingBo Zhu. Step-level Verifier-guided Hybrid Test-Time Scaling for Large Language Models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18462–18477, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.931. URL https://aclanthology.org/2025.emnlp-main.931/.

Hao Mark Chen, Zhiwen Mo, Guanxi Lu, Shuang Liang, Lingxiao Ma, Wayne Luk, and Hongxiang Fan. FastTTS: Accelerating Test-Time Scaling for Edge LLM Reasoning, 2026. URL https://arxiv.org/abs/2509.00195.

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, 2021. URL https://arxiv.org/abs/2110.14168.

Runpeng Dai, Tong Zheng, Rui Liu, Chengsong Huang, and Hongtu Zhu. Small RL Controller, Large Language Model: RL-Guided Adaptive Sampling for Test-Time Scaling, 2026. URL https://arxiv.org/abs/2606.03102.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. BEST-Route: Adaptive LLM Routing with Test-Time Optimal Compute, 2025. URL https://arxiv.org/abs/2506.22716.

Sajib Acharjee Dip, Dawei Zhou, and Liqing Zhang. Think Again or Think Longer? Selective Verification for Budget-Aware Reasoning, 2026. URL https://arxiv.org/abs/2606.19808.

Georgi Gerganov and contributors. llama.cpp: LLM inference in C/C++, 2026. URL https://github.com/ggml-org/llama.cpp. GitHub repository.

Kim Hammar, Tansu Alpcan, and Emil C. Lupu. Optimal Stopping of Self-Refining Foundation Models, 2026. URL https://arxiv.org/abs/2608.10729.

Zixu Hao, Jianyu Wei, Tuowei Wang, Minxing Huang, Huiqiang Jiang, Shiqi Jiang, Ting Cao, and Ju Ren. Scaling LLM Test-Time Compute with Mobile NPU on Smartphones. In *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, pp. 2157–2172, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722127. doi: 10.1145/3767295.3769382. URL https://doi.org/10.1145/3767295.3769382.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Parsa Hosseini, Sumit Nawathe, Mahdi Salmani, Meisam Razaviyayn, and Soheil Feizi. Early Stopping for Large Reasoning Models via Confidence Dynamics, 2026. URL https://arxiv.org/abs/2604.04930.

Chengsong Huang, Langlin Huang, Jixuan Leng, Jiacheng Liu, and Jiaxin Huang. Efficient Test-Time Scaling via Self-Calibration, 2025a. URL https://arxiv.org/abs/2503.00031.

Jenny Y. Huang, Mehul Damani, Yousef El-Kurdi, Ramon Astudillo, and Wei Sun. Latency and Token-Aware Test-Time Compute, 2025b. URL https://arxiv.org/abs/2509.09864.

Shiyu Ji, Yixuan Wang, Yijun Liu, Qingfu Zhu, and Wanxiang Che. Seer Self-Consistency: Advance Budget Estimation for Adaptive Test-Time Scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42734–42747, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.2120. URL https://aclanthology.org/2026.findings-acl.2120/.

Chen Jin, Ryutaro Tanno, Tom Diethe, and Philip Teare. CoRefine: Confidence-Guided Self-Refinement for Adaptive Test-Time Compute, 2026. URL https://arxiv.org/abs/2602.08948.

Yunho Jin, Gu-Yeon Wei, and David Brooks. The Energy Cost of Reasoning: Analyzing Energy Usage in LLMs with Test-time Compute, 2025. URL https://arxiv.org/abs/2505.14733.

Mobina Kashaniyan and Ali Jannesari. Interpretable Adaptive Sampling for LLM Test-Time Scaling, 2026. URL https://arxiv.org/abs/2608.03961.

Junseok Kim, Nakyeong Yang, Kyungmin Min, and Kyomin Jung. Reliability-Aware Adaptive Self-Consistency for Efficient Sampling in LLM Reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21575–21590, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1085. URL https://aclanthology.org/2026.findings-acl.1085/.

Colin Ian King. stress-ng: A Tool to Stress Test a Computer System, 2026. URL https://github.com/ColinIanKing/stress-ng. Accessed: 2026-09-10.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. OverThink: Slowdown Attacks on Reasoning LLMs, 2026. URL https://arxiv.org/abs/2502.02542.

Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. MELTing Point: Mobile Evaluation of Language Transformers. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, pp. 890–907, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704895. doi: 10.1145/3636534.3690668. URL https://doi.org/10.1145/3636534.3690668.

Jiawei Li, Yizhe Yang, Yu Bai, Xiaofeng Zhou, Yinghao Li, Huashan Sun, Yuhang Liu, Xingpeng Si, Yuhao Ye, Yixiao Wu, Yiguan Lin, Bin Xu, Bowen Ren, Chong Feng, Yang Gao, and Heyan Huang. Fundamental Capabilities of Large Language Models and their Applications in Domain Scenarios: A Survey. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11116–11141, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.599. URL https://aclanthology.org/2024.acl-long.599/.

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, 2024b. URL https://arxiv.org/abs/2401.10480.

Kang Liu, YongKang Liu, Xiaocui Yang, Peidong Wang, Wen Zhang, Shi Feng, Yifei Zhang, and Daling Wang. NEAT: Neuron-based early exit for large reasoning models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24616–24627, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1231. URL https://aclanthology.org/2026.findings-acl.1231/.

Xin Liu and Lu Wang. Answer Convergence as a Signal for Early Stopping in Reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17896–17907, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.904. URL https://aclanthology.org/2025.emnlp-main.904/.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing Sub-Billion Parameter Language Models for On-Device Use Cases. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

AI @ Meta Llama Team. The Llama 3 Herd of Models, 2024. URL https://arxiv.org/abs/2407.21783.

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*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.546. URL https://aclanthology.org/2023.acl-long.546/.

Dehai Min, Giovanni Vaccarino, Huiyi Chen, Yongliang Wu, Gal Yona, and Lu Cheng. Stop When Reasoning Converges: Semantic-Preserving Early Exit for Reasoning Models, 2026. URL https://arxiv.org/abs/2605.17672.

NVIDIA Corporation. tegrastats, 2026. URL https://docs.nvidia.com/jetson/archives/r38.4/DeveloperGuide/AT/JetsonLinuxDevelopmentTools/TegrastatsUtility.html. NVIDIA Jetson Linux system monitoring utility, accessed 2026-09-10.

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, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168/.

Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. Mobile Edge Intelligence for Large Language Models: A Contemporary Survey. *IEEE Communications Surveys & Tutorials*, 27(6):3820–3860, 2025. doi: 10.1109/COMST.2025.3527641.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report, 2025. URL https://arxiv.org/abs/2412.15115.

Jie Ren, Sercan Arik, Xinyun Chen, Jiefeng Chen, Chengrun Yang, Ruoxi Sun, and Jinsung Yoon. SETS: Leveraging Self-Verification and Self-Correction for Improved Test-Time Scaling. *Transactions on Machine Learning Research (TMLR)*, 2025. URL https://arxiv.org/pdf/2501.19306?

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MPNet: masked and permuted pre-training for language understanding. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Renliang Sun, Wei Cheng, Dawei Li, Haifeng Chen, and Wei Wang. Stop When Enough: Adaptive Early-Stopping for Chain-of-Thought Reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27250–27268, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1256. URL https://aclanthology.org/2026.acl-long.1256/.

Pavel Tikhonov, Ivan Oseledets, and Elena Tutubalina. Confidence Leaps in LLM Reasoning: Early Stopping and Cross-Model Transfer. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 602–616, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-381-4. doi: 10.18653/v1/2026.eacl-short.43. URL https://aclanthology.org/2026.eacl-short.43/.

Henrique Schechter Vera, Sahil Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, Daniel Cer, Alice Lisak, Min Choi, Lucas Gonzalez, Omar Sanseviero, Glenn Cameron, Ian Ballantyne, Kat Black, Kaifeng Chen, Weiyi Wang, Zhe Li, Gus Martins, Jinhyuk Lee, Mark Sherwood, Juyeong Ji, Renjie Wu, Jingxiao Zheng, Jyotinder Singh, Abheesht Sharma, Divyashree Sreepathihalli, Aashi Jain, Adham Elarabawy, AJ Co, Andreas Doumanoglou, Babak Samari, Ben Hora, Brian Potetz, Dahun Kim, Enrique Alfonseca, Fedor Moiseev, Feng Han, Frank Palma Gomez, Gustavo Hernández Ábrego, Hesen Zhang, Hui Hui, Jay Han, Karan Gill, Ke Chen, Koert Chen, Madhuri Shanbhogue, Michael Boratko, Paul Suganthan, Sai Meher Karthik Duddu, Sandeep Mariserla, Setareh Ariafar, Shanfeng Zhang, Shijie Zhang, Simon Baumgartner, Sonam Goenka, Steve Qiu, Tanmaya Dabral, Trevor Walker, Vikram Rao, Waleed Khawaja, Wenlei Zhou, Xiaoqi Ren, Ye Xia, Yichang Chen, Yi-Ting Chen, Zhe Dong, Zhongli Ding, Francesco Visin, Gaël Liu, Jiageng Zhang, Kathleen Kenealy, Michelle Casbon, Ravin Kumar, Thomas Mesnard, Zach Gleicher, Cormac Brick, Olivier Lacombe, Adam Roberts, Qin Yin, Yunhsuan Sung, Raphael Hoffmann, Tris Warkentin, Armand Joulin, Tom Duerig, and Mojtaba Seyedhosseini. EmbeddingGemma: Powerful and Lightweight Text Representations, 2025. URL https://arxiv.org/abs/2509.20354.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. Reasoning Aware Self-Consistency: Leveraging Reasoning Paths for Efficient LLM Sampling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3613–3635, Albuquerque, New Mexico, April

2025a. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.184. URL https://aclanthology.org/2025.naacl-long.184/.

Guangya Wan, Zixin Stephen Xu, Sasa Zorc, Manel Baucells, Mengxuan Hu, Hao Wang, and Sheng Li. BEACON: Bayesian Optimal Stopping for Efficient LLM Sampling, 2025b. URL https://arxiv.org/abs/2510.15945.

Junda Wang, Zhichao Yang, Dongxu Zhang, Sanjit Singh Batra, and Robert E. Tillman. ESTAR: Early-Stopping Token-Aware Reasoning For Efficient Inference, 2026. URL https://arxiv.org/abs/2602.10004.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers, 2020. URL https://arxiv.org/abs/2002.10957.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Make Every Penny Count: Difficulty-Adaptive Self-Consistency for Cost-Efficient Reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 6919–6932, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.383. URL https://aclanthology.org/2025.findings-naacl.383/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models, 2023. URL https://arxiv.org/abs/2203.11171.

Zili Wang, Tianyu Zhang, Haoli Bai, Lu Hou, Xianzhi Yu, Wulong Liu, Shiming Xiang, and Lei Zhu. Faster and Better LLMs via Latency-Aware Test-Time Scaling. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 17124–17137, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.928. URL https://aclanthology.org/2025.findings-emnlp.928/.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

Menghua Wu, Cai Zhou, Stephen Bates, and Tommi Jaakkola. Thought calibration: Efficient and confident test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14291–14305, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.722. URL https://aclanthology.org/2025.emnlp-main.722/.

Xuening Wu, Qianya Xu, Yanlan Kang, Zeping Chen, Yubin Liu, and Shenqin Yin. Do Language Models Converge to Themselves? Recursive Self-Refinement as Textual Relaxation, 2026. URL https://arxiv.org/abs/2607.22653.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed Resources For General Chinese Embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 641–649, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657878. URL https://doi.org/10.1145/3626772.3657878.

Yangxinyu Xie, Tao Wang, Soham Mallick, Yan Sun, Georgy Noarov, Mengxin Yu, Tanwi Mallick, and Edgar Dobriban. Statistical Early Stopping for Reasoning Models, 2026. URL https://arxiv.org/abs/2602.13935.

Jing Xiong, Qiujiang Chen, Fanghua Ye, Zhongwei Wan, Chuanyang Zheng, Chenyang Zhao, Hui Shen, Hanbo Li, Chaofan Tao, Haochen Tan, Haoli Bai, Lifeng Shang, Lingpeng Kong, and Ngai Wong. ATTS: Asynchronous Test-Time Scaling via Conformal Prediction, 2026a. URL https://arxiv.org/abs/2509.15148.

Juming Xiong, Kevin Guo, Congning Ni, Weixin Liu, Chao Yan, Katherine Brown, Avinash Baidya, Xiang Gao, Bradley Malin, and Zhijun Yin. Learning When to Sample: Confidence-Aware Selective Sampling for Efficient Chain-of-Thought Reasoning, 2026b. URL https://arxiv.org/abs/2603.08999.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. PowerInfer-2: Fast Large Language Model Inference on a Smartphone, 2024. URL https://arxiv.org/abs/2406.06282.

Tianqiang Yan, Ziqiao Lin, Lin Zhang, Zhenglong Sun, and Yuan Gao. Entrospect: Information-Theoretic Self-Reflection Elicits Better Response Refinement of Small Language Models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24563–24577, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1261. URL https://aclanthology.org/2025.findings-acl.1261/.

Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, Lingkun Kong, Brian Moran, Jiaqi Wang, Yifan Ethan Xu, An Yan, Chenyu Yang, Eting Yuan, Hanwen Zha, Nan Tang, Lei Chen, Nicolas Scheffer, Yue Liu, Nirav Shah, Rakesh Wanga, Anuj Kumar, Wen-tau Yih, and Xin Luna Dong. CRAG - comprehensive RAG benchmark. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Shu Zhou, Rui Ling, Junan Chen, Xin Wang, Tao Fan, and Hao Wang. When More Thinking Hurts: Overthinking in LLM Test-Time Compute Scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23967–23977, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1199. URL https://aclanthology.org/2026.findings-acl.1199/.

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. Large Language Models for Information Retrieval: A Survey. *ACM Trans. Inf. Syst.*, 44(1), November 2025. ISSN 1046-8188. doi: 10.1145/3748304. URL https://doi.org/10.1145/3748304.

Zhan Zhuang, Xiequn Wang, Zebin Chen, Feiyang Ye, Ying Wei, Kede Ma, and Yu Zhang. One-Token Verification for Reasoning Correctness Estimation, 2026. URL https://arxiv.org/abs/2603.01025.

APPENDIX

A ADDITIONAL BACKGROUND AND EXTENDED RELATED WORK

Allocating compute upfront A broad range of work on compute allocation for TTS predicts inference-compute budgets *upfront* ($i.e.$, from prefill and query features), often to preserve the latency benefits of parallel scaling. Compute-optimal scaling (Snell et al., 2024) assigns scaling strategies based on estimated query difficulty, although its strongest allocations rely on oracle or Monte Carlo estimates. DSC (Wang et al., 2025a) prompts an LLM to rank a batch of queries, partitions them by ranked difficulty, and initializes each *hard* query's sample count from previously 'nearby' samples; its allocation therefore relies explicitly on batch-level context. SeerSC (Ji et al., 2026) draws 64 short direct answers and maps their confidence-weighted entropy to a fixed subsequent budget, allowing the full reasoning stage to be generated in parallel but requiring substantial pilot inference and concurrent generation. BEST-Route (Ding et al., 2025) predicts query difficulty to jointly select a model and sample count, assuming a pool of candidate models and an upfront routing decision. Difficulty- and confidence-based allocation has also been explored through calibrated or learned confidence estimates; Self-Calibration (Huang et al., 2025a), for example, distills SC-derived confidence into the underlying model and requires additional model training. Huang et al. (Huang et al., 2025b) score candidate strategies using a learned predictor over query embeddings and strategy parameters together with offline latency estimates and external reward signals. Xiong et al. (Xiong et al., 2026b) instead select between single- and multi-path sampling from a completed reasoning state using model-specific probability information.

The above methods can improve coarse allocation, but either commit to a budget before observing how the output evolves, rely on batched queries or parallel generation, require model-specific signals, or introduce additional learned models. *CONGRATTS instead makes its decision after each scaling round using only runtime-accessible features revealed by the observed scaling history.*

Adaptive parallel TTS. Most work on adaptive TTS has focused on parallel sampling. ASC (Aggarwal et al., 2023), ESC (Li et al., 2024b), and CGES (Aghazadeh et al., 2026) dynamically terminate SC using posterior concentration, fixed agreement windows, and confidence-guided posterior estimates, respectively. RASC (Wan et al., 2025a) additionally scores the quality of generated rationales and trains a separate logistic-regression sufficiency model, introducing a learned rationale-quality component beyond answer-level statistics. ReASC (Kim et al., 2026) weights accumulated evidence using response-level confidence, relying on model confidence signals and confidence-distribution calibration. Difficulty-Adaptive SC (Wang et al., 2025a) combines adaptive sampling with batch-level query-difficulty estimates, while SeerSC (Ji et al., 2026) moves the adaptive decision before full generation to recover parallel latency. BEACON (Wan et al., 2025b) formulates sampling as Bayesian optimal stopping over candidate rewards, but requires candidates to be scored by a reward model and starts from an initial parallel sample set. Self-Calibration (Huang et al., 2025a) uses distilled model confidence and requires modification of the underlying model. Dai et al. (Dai et al., 2026) instead train a PPO controller over answer-count statistics; its continuation actions request multiple additional samples in parallel, and its policy depends on reward-weighted latency and computation penalties rather than measured runtime availability. Recent interpretable allocation methods similarly map query difficulty and confidence to sample budgets (Kashaniyan & Jannesari, 2026).

Adaptive sequential TTS Sequential TTS has received considerably less attention. CoRefine (Jin et al., 2026) trains a lightweight controller to halt, rethink, or explore an alternative based on token-confidence dynamics over the reasoning state. Although the backbone remains frozen, the controller relies on model probability information and on separately trained control parameters. Hammar et al. (Hammar et al., 2026) formulate iterative SR as an optimal stopping problem balancing expected improvement against cost, but assume external feedback from verifiers to drive each refinement. Hybrid TTS (Chang et al., 2025) adaptively revises individual reasoning rounds and combines sequential and parallel scaling, but uses an explicit process verifier to determine where refinement is required. In contrast, CONGRATTS uses the same runtime-accessible representation for both SC and SR, without retraining, model modification, internal-state access, or an external verifier.

Within-state scaling & verification. Another related line of work reduces computation *within* a single reasoning state rather than adapting the number of complete scaling rounds. Answer Convergence (Liu & Wang, 2025), CoDE-Stop (Hosseini et al., 2026), PUMA (Min et al., 2026), ES-

TAR (Wang et al., 2026), and NEAT (Liu et al., 2026) shorten reasoning through answer stability, semantic redundancy, learned stopping signals, or model activations. ESTAR, for example, explicitly trains the underlying reasoning model to emit stop tokens (Wang et al., 2026). In contrast, activation- or probability-based methods require access to internal model statistics that are unavailable through many inference backends. REFRAIN (Sun et al., 2026) uses a two-stage redundancy discriminator and adaptive threshold controller to stop a single reasoning state, while confidence-leap stopping (Tikhonov et al., 2026) terminates generation based on changes in answer probability. Statistical early-stopping methods monitor *uncertainty* during generation (Xie et al., 2026). OTV and SEVRA verify or route completed scaling runs (Zhuang et al., 2026; Dip et al., 2026). These methods reduce computation *within* a single generation, rather than deciding whether to generate another independent sample or perform another refinement round. Our method is complementary, controlling scaling across rounds without requiring accessing internal model state.

Systems-level optimizations for TTS. FastTTS (Chen et al., 2026) targets memory-constrained TTS on consumer GPUs through generator–verifier memory allocation, prefix reuse, and scheduling, but requires a separate verifier and optimizes execution of preselected search configurations rather than adapting to live resource availability. Hao et al. (Hao et al., 2026) accelerates parallel TTS on Qualcomm NPUs by exploiting underutilized matrix hardware and hardware-aware quantization, but primarily targets concurrent execution of fixed-scaling workloads. Wang et al. (Wang et al., 2025b) demonstrates that compute-optimal scaling need not be latency-optimal and optimizes branch- and sequence-level concurrency, but assumes available parallel execution resources and optimizes latency for a selected configuration. ATTS (Xiong et al., 2026a) addresses latency, synchronization, and memory overhead through asynchronous parallel and sequential scaling with speculative draft–target model pairs, requiring multiple models and substantial concurrent execution. Generally, work on efficient TTS serving demonstrates the importance of KV-cache management, verifier scheduling, parallelism, and hardware-specific execution (Chen et al., 2026; Hao et al., 2026; Wang et al., 2025b; Xiong et al., 2026a). These systems optimize *how a scaling workload executes* on the available hardware; CONGRATTS instead incorporates the current resource state into the decision of *whether that workload should continue*, jointly gating each additional round based on convergence and explicit feasibility constraints on latency, memory, CPU, and energy.

B Experimental and Reproducibility Details

B.1 Models, Generation, Training & Evaluation Data

We evaluate `Qwen2.5-1.5B` (Qwen et al., 2025), `Llama-3.2-1B-Instruct` (Llama Team, 2024), `SmolLM2-1.7B-Instruct` (Allal et al., 2025), and `LFM2.5-1.2B-Instruct` (Amini et al., 2025). Unless otherwise stated, we use a maximum scaling budget $T = 16$, corresponding to 16 sampled responses for SC and 16 refinement rounds following the initial response for SR.

Self-Consistency (SC) The SC evaluation uses MATH-500 (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and SVAMP (Patel et al., 2021). Mathematical reasoning is well-suited to SC because independently sampled solutions often yield a single normalizable final answer, making agreement among solutions an effective aggregation signal (Wang et al., 2023). MATH-500 contains competition-level mathematical problems, GSM8K focuses on multi-round arithmetic reasoning, and SVAMP evaluates compositional arithmetic reasoning. We restrict evaluation to examples with numeric answers; the resulting held-out sets contain 364 MATH-500, 500 GSM8K, and 300 SVAMP queries. For each seed, we additionally sample 300 queries from the MATH training split, balanced across difficulty levels where possible, and partition them into 240 training and 60 validation queries. The same set of evaluation queries is reused across all models and stopping methods within each seed and is disjoint from the examples used for training and calibration.

Each SC round independently samples one complete response to the same question and prompt. The responses generated after t rounds are grouped by their normalized final answer. The selected answer is the group with the largest number of samples; ties between equally sized groups are broken using the highest average token log-probability among their members. The representative response within the selected group is likewise the response with the highest average token log-probability.

Self-Refinement (SR). The SR evaluation uses CRAG (Yang et al., 2024) and POPQA (Mallen et al., 2023) in a retrieval-augmented question-answering setting. CRAG contains factual questions spanning multiple domains and varying retrieval requirements, while POPQA focuses on open-domain questions about long-tail entities. We include both because they exercise distinct forms of retrieval-augmented refinement. We construct held-out evaluation sets of 500 queries from each dataset. These evaluation queries are excluded from all seeded training and validation pools and reused across all models and stopping methods.

For each seed, we independently sample 300 additional CRAG and POPQA queries, then split each query-set into 240 training and 60 validation queries. For CRAG, we restrict the training and validation set to questions labeled *static* and exclude *false-premise* questions. This reduces ambiguity from time-varying answers and invalid question premises, giving more stable fixed-budget outputs from which to construct convergence targets.

The initial SR state $t = 0$ is generated by a single model call without retrieved evidence. Each subsequent refinement round $t \geq 1$ contains two model calls. First, the model receives the question, previous answer, accumulated evidence, available search results, and a compact KNOWN/MISSING state, and selects one of three actions: SEARCH, OPEN, or STOP. SEARCH operates over a frozen query-specific document collection, while OPEN adds the selected stored page to the accumulated evidence. Following a successful SEARCH, an unopened result is opened before another search is permitted, whenever one is available. Then, a second model call generates a complete revised answer from the original question and all evidence accumulated so far. Thus, executing t refinement rounds requires $1 + 2t$ model calls, and a complete 16-round run contains 17 answer states. The initial state $t = 0$ is generated without retrieval because subsequent retrieval actions are conditioned on the current answer and its KNOWN/MISSING state; $t = 0$ provides the common starting state from which retrieval-guided refinement begins.

The model-proposed STOP action does not itself terminate scaling. If unopened results remain, it is replaced by OPEN. Otherwise, it is replaced by a fallback SEARCH. Initial and revised answers are limited to 192 output tokens, while action-selection calls are limited to 96 tokens.

Retrieval. Both SR datasets use frozen retrieval results so that each method operates over the same available information. For CRAG, we load the benchmark-provided search results and retain each result's title, URL, snippet, and retrieved HTML. For POPQA, we use a frozen per-query collection containing result titles, URLs, snippets, and page text. This avoids the variation introduced by issuing live web requests during evaluation.

For each search action, candidate pages are ranked against the model-generated search query using a lightweight lexical relevance score over title and snippet text. The score combines query-term frequency, inverse document frequency, and length normalization, and we return the top 5 results. We retain at most 6,000 characters from an opened result and at most 18,000 characters of accumulated evidence in the answer context.

Training. For each training seed, we independently sample 300 queries from each of MATH, CRAG, and POPQA. Each dataset is split 80/20 into training and validation splits. The convergence classifier is fit on 240 MATH, 240 CRAG, and 240 POPQA queries, for a total of 720 training queries per seed. MATH provides the SC training runs, while CRAG and POPQA provide the SR training runs. All runs used to train the classifier are generated with `Qwen2.5-1.5B`, with a maximum scaling budget of 16 rounds.

The remaining 60 queries from each dataset are held out from classifier fitting and used to calibrate the stopping rule. SC calibration uses the 60-query MATH validation split. SR calibration combines the 60-query CRAG and 60-query POPQA validation splits, yielding 120 calibration queries. We calibrate SC and SR separately, producing distinct (ψ_0, λ) configurations for the two scaling regimes while keeping the jointly trained convergence classifier fixed.

The non-learned baselines require no classifier training, and are calibrated using the full 300-query development set available for the corresponding task. SC baselines are calibrated on the 300 MATH queries. SR baselines are calibrated separately for CRAG and POPQA using the 300-query set from each dataset. These development sets remain disjoint from all held-out evaluation queries.

Generation. Offline SC generation uses temperature 0.7, top-$p = 0.9$, and top-$k = 20$, with at most 512 new tokens per round. Offline SR generation uses temperature 0.3 and top-$p = 1.0$; the initial

answer and each revised answer are limited to 192 new tokens, while each action-selection call is limited to 96 tokens. Full prompt templates are reported in Appendix B.2 below.

Accuracy Across Scaling Rounds

Figure 3 shows how task accuracy changes with the number of SC rounds across our evaluated models and datasets, while Figure 4 shows the corresponding behavior for SR. These curves are computed from the completed fixed-16 runs and illustrate that the benefit of additional scaling varies substantially across models, datasets, and scaling depth. In particular, accuracy need not improve monotonically with additional rounds.

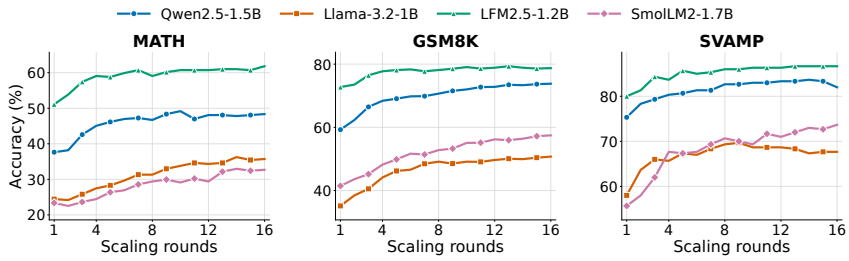

Figure 3: **SC accuracy across scaling rounds.** Accuracy as a function of the number of sampled responses for each model and SC benchmark, computed from the completed fixed-16 scaling runs. Additional samples provide diminishing and model–dataset-dependent gains. Accuracy is not uniformly monotonic with scaling depth.

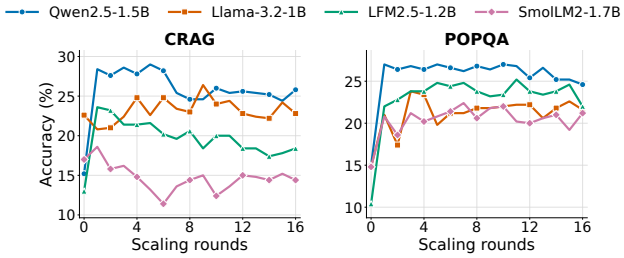

Figure 4: **SR accuracy across scaling rounds.** Accuracy as a function of refinement depth for each model and SR benchmark, computed from the completed fixed-16 scaling runs. Additional refinement does not consistently improve accuracy.

B.2 Prompt Templates

We report the prompt templates used for SC and SR below. Text enclosed in angle brackets denotes content inserted dynamically at inference time.

Self-consistency. SC uses the following system prompt for all reported generations.

SC system prompt

```
You are a careful reasoning assistant.
Follow the user's requested output format exactly.
Ensure the final answer is explicit.
```

Each sampled solution then uses the following reasoning prompt.

SC reasoning prompt

```
<QUESTION>

Solve the problem carefully. Be concise.
End with Final answer: <number>.
```

Self-refinement. SR alternates between an action-selection stage and an answer-generation stage. The action-selection stage uses the following system instruction.

SR action-stage system prompt

```
You control a research assistant with web tools.
Choose exactly one next action.
Follow the requested format exactly.
```

SR action-stage prompt

```
Question:

<QUESTION>

Current answer:

<CURRENT ANSWER>

Previous working state:

<WORKING STATE>

Evidence so far:

<EVIDENCE>

Search results available to open:

<SEARCH RESULTS>

Current action constraint:

<CONSTRAINT>

First, update a compact working state, then choose ONE action.

KNOWN: <1-3 short facts currently supported by evidence>

MISSING: <single most important unresolved fact or reasoning round>

ACTION: SEARCH

QUERY: <targeted query aimed at MISSING>

or

KNOWN: ...

MISSING: ...

ACTION: OPEN

RESULT_ID: <integer>

or

KNOWN: ...

MISSING: NONE
```

```
ACTION: STOP

Rules:

- KNOWN must contain only facts supported by the question/evidence,
  not guesses from the current answer.
- MISSING should identify one concrete fact/reasoning gap.
- SEARCH must directly target MISSING and should not repeat an
  earlier query.
- OPEN a promising unopened result when it may resolve MISSING.
- If the constraint says MUST OPEN, choose an unopened RESULT_ID.
- Keep KNOWN and MISSING concise; no additional prose or markdown.
```

The answer-generation stage uses a separate system instruction and prompt:

SR answer-stage system prompt

```
You answer factual questions using available evidence.
Give the shortest unambiguous answer possible.
End with exactly: FINAL ANSWER: <answer>
```

SR answer-stage prompt

```
Question:

<QUESTION>

Web evidence gathered so far:

<EVIDENCE>

Answer the question using the evidence when relevant.
If evidence is incomplete, make the best answer you can.

Finish with:

FINAL ANSWER: <short answer>
```

B.3 Convergence Features

Our HGB model uses a shared 20-dimensional feature schema for both SC and SR. Features are computed online at each valid scaling state using only information produced by the preceding scaling rounds; they require no hidden states, external verifier, or additional language model calls. The representation captures five broad aspects of convergence: position within the available scaling budget, concentration of the observed answer evidence, temporal stability of the selected answer, likelihood-based uncertainty of the currently selected response, and stability of answer-frequency margins.

The feature schema is shared semantically across SC and SR, although a number of features are computed differently where required by the different structures of each scaling paradigm. In SC, the current scaling state t provides a group of t sampled candidate responses, so the `answer_distribution_*` features summarize the empirical distribution over candidate-answer keys at the current prefix. In SR, only one new response is produced per refinement round, so the corresponding distribution is formed from the normalized answers observed from the initial response through state t. Thus, while SC aggregates over normalized parsed answers and SR iterates longer free-form responses, the model receives a comparable representation of whether its output answer has stabilized. We apply no explicit feature normalization or standardization: our classifier operates on features at their native scales, with missing values median-imputed from the training data.

We distinguish our answer-distribution features from the `answer_count_*` features. In SC, the latter summarizes the sequence of answers selected by the SC aggregator as the sample prefix grows. For example, if the selected SC answers after prefixes $1, \ldots, t$ are $a^1, \ldots, a^t$, the counts are computed from this sequence. In SR, the current answer at each refinement round is itself the selected answer, so the distribution and count families are based on the same refinement history, and some quantities therefore coincide algebraically.

The features are entirely scalar quantities available from the generated answers and their log probabilities. In our `llama.cpp` implementation, feature extraction adds on average 0.99ms per scaling round on the MacBook Pro and 2.17ms on the Jetson (Table 15).

Let T denote the maximum scaling budget and t the current round. For an empirical answer distribution containing K distinct answer states, n_k denotes the count of state k, and

$$N = \sum_{k=1}^{K} n_k, \qquad p_k = \frac{n_k}{N}, \qquad H(p) = -\sum_{k=1}^{K} p_k \log p_k.$$

Let $n_{(1)} \geq n_{(2)}$ denote the two largest answer counts, taking $n_{(2)} = 0$ when $K = 1$; their margin is $n_{(1)} - n_{(2)}$. Table 6 gives the resulting features.

The paradigm-specific definitions of answer-distribution confidence and uncertainty reflect the different evidence exposed by the two scaling paradigms. SC produces a growing set of candidate samples at each prefix, making normalized entropy a natural measure of how fragmented the current sample pool is. In contrast, SR produces a single evolving answer per round; its modal frequency $p_{\max}$ directly measures how often the sequence of refinement outputs has returned to its most common answer state. These quantities are therefore not assumed to be numerically interchangeable. Rather, they occupy the same semantic feature slots as measures of answer concentration. The learned convergence model can combine these signals with answer-change, streak, likelihood, and budget information, while the final stopping rule is calibrated separately for SC and SR. We further evaluate the importance of these features (or rather feature groups) in Appendix C.1.

B.4 Convergence-Target Construction

We define our stopping target based on convergence to the output produced by a given language model under a fixed maximum scaling budget, rather than on ground-truth correctness. We make this choice for two key reasons.

First, correctness is inherently task-specific; constructing a correctness target requires access to ground-truth answers and an evaluation rule that can determine whether a model's output is correct. While this is straightforward for tasks with exact or programmatically checkable answers, it is considerably less general for open-ended generation, subjective tasks, tasks with multiple valid outputs, partially correct responses, or other realistic deployment settings in which reference answers are unavailable. By contrast, convergence depends only on the model's own scaling history and can therefore be defined consistently across tasks and domains.

Second, every complete scaling run supplies a convergence target, whereas not every run supplies a correctness target. Given a fixed compute horizon, every query has an observable terminal model state, making it possible to define when scaling has converged toward that state regardless of whether the terminal answer is correct. For correctness, a scaling run may never contain a correct answer at any round. We further evaluate the impact of this design choice and provide the details of our findings in Appendix C.4.

For SC, let $a_{i,t}$ denote the aggregated consensus answer for query i after round t, and let $a_{i,T}$ denote its final consensus after T rounds. We define the pointwise convergence target as

$$c_{i,t} = \mathbf{1}\left[\mathrm{norm}(a_{i,t}) = \mathrm{norm}(a_{i,T})\right].$$

Thus, an SC round is labeled converged when the current consensus matches the answer selected after the full scaling budget.

However, exact equality is inappropriate for our SR benchmarks, which use free-form outputs. We therefore encode the intermediate response $a_{i,t}$ and final T-round response $a_{i,T}$ using a sentence-level embedding model $\phi(\cdot)$, and compute

$$s_{i,t}^{\mathrm{sem}} = \cos\left(\phi(a_{i,t}), \phi(a_{i,T})\right).$$

Table 6: **Features used by our convergence model.** The 20-dimensional schema is shared across SC and SR. Distributional features use the answer population naturally exposed by each paradigm, while retaining the common semantic interpretation of answer concentration and convergence.

Feature	Definition / interpretation
`scaling_round`	Current scaling state t
`current_round_fraction`	Fraction of the maximum scaling budget consumed, t/T.
`remaining_round_fraction`	Fraction of the maximum scaling budget remaining, $(T-t)/T$.
`answer_distribution_entropy`	Entropy $H(p) = -\sum_k p_k \log p_k$ of the current answer distribution. In SC, p is formed from candidate answers in the current sample prefix. In SR, p is formed from normalized refinement answers observed.
`answer_distribution_uncertainty`	Paradigm-specific concentration measure. For SC, this is $\frac{H(p)}{\log t}$ with $t \geq 3$. For SR, it is $1 - p_{\max}$, where $p_{\max} = \max_{1 \leq k \leq K} p_k$.
`answer_distribution_confidence`	Paradigm-specific concentration measure. For SC, this is $1 - \frac{H(p)}{\log t}$ with $t \geq 3$. For SR, it is $p_{\max}$.
`distinct_answer_count`	Number of distinct answer states: distinct candidate-answer keys in SC, or distinct refinement answers observed in SR.
`same_answer_streak`	Length of the current consecutive run for which the selected answer has remained the same. In SC this is the sequence of aggregate answers selected. In SR this is the sequence of normalized refinement answers.
`longest_same_answer_streak`	Maximum value of the same-answer streak attained up to state t.
`answer_changed`	Indicates that the currently selected answer differs from the answer at the preceding state. In SC, this compares consecutive selected SC answers. In SR, it compares consecutive parsed refinement-answer strings.
`chosen_avg_logprob`	Mean token log-probability of the current answer.
`chosen_min_token_logprob`	Minimum generated-token log-probability in the current response.
`running_mean_chosen_avg_logprob`	Running mean of `chosen_avg_logprob` up to state t.
`running_std_chosen_avg_logprob`	Running standard deviation of `chosen_avg_logprob` up to state t.
`answer_count_total`	Number of responses included in the answer count up to state t.
`answer_count_distinct`	Number of distinct selected-answer states represented in the counts.
`answer_count_entropy`	Entropy of the selected-answer frequency distribution.
`answer_count_margin`	Difference between the largest and second-largest answer counts.
`answer_count_margin_fraction`	Count margin normalized by the number of observations.
`answer_count_lock_gap`	Current margin minus the number of rounds remaining.

The corresponding pointwise convergence target is

$$c_{i,t} = \mathbf{1}\left[s_{i,t}^{\text{sem}} \geq \alpha\right], \qquad \alpha = 0.90.$$

For calibration, we define the earliest converged scaling state t_i^{conv} equivalently in SC and SR as

$$t_i^{\text{conv}} = \min\left\{t : c_{i,t} = 1\right\},$$

where t ranges over the valid scaling states for the corresponding regime. Since $c_{i,T} = 1$ by construction, t_i^{conv} is defined for every query. Notably, $c_{i,t}$ is defined pointwise: reaching convergence at one round does not mean all subsequent rounds are labeled converged.

The threshold $\alpha = 0.90$ requires strong semantic agreement while allowing lexical and formatting differences between refinements, and is consistent with semantic-equivalence thresholds used in prior refinement-stopping work (Yan et al., 2025).

The main results use `EmbeddingGemma-300M` (Vera et al., 2025) to construct the SR targets. The embedding model is used *only offline*: it is applied to completed fixed-budget scaling runs during label construction and is never invoked by the online stopping rule. We repeat target construction using alternative embedding models, and report the resulting behavior in Appendix C.7.

Using paradigm-specific definitions of the same target (*i.e.*, whether further scaling will materially change the fixed-budget output) allows SC and SR examples to be represented as binary convergence targets and learned jointly by the same classifier.

B.5 BASELINES

For SC, we compare our convergence model against adaptive SC (ASC) (Aggarwal et al., 2023), early-stopping SC (ESC) (Li et al., 2024b), and confidence-guided (Aghazadeh et al., 2026) early stopping (CGES), as well as fixed 1-, 4-, 8-, and 16-round configurations. ASC uses the evolving sampled-answer distribution to estimate whether the current majority is sufficiently reliable and stops when its confidence criterion is satisfied. ESC stops once the selected answer remains unchanged for a calibrated streak length k. CGES constructs a confidence-weighted distribution over candidate answers from token-level generation confidence and stops once the leading candidate exceeds a calibrated threshold. We use its log-normalized geometric-confidence variant, which provides a length-normalized confidence score from token probabilities without requiring the use of an external verifier or reward model.

For each seed, the SC baselines are calibrated on the full 300-query MATH development set. We select the configuration that produces the lowest mean number of rounds, subject to at most 5 disagreements with the corresponding fixed-16 answer, matching the disagreement rate used to calibrate our stopping rule. We break ties by higher agreement. If no configuration satisfies our agreement constraint, we select the configuration with the highest agreement and then the lowest mean rounds. We calibrate our ASC and CGES stopping thresholds over the shared SC threshold grid, 0.900–0.999, and ESC patience over $k \in \{1, \ldots, 8\}$.

For SR, baseline calibration is performed separately for each dataset: CRAG baselines use the 300-query CRAG development set, and POPQA baselines use the 300-query POPQA development set. These baselines require no learned model fitting, so the full development sets are used only to select their stopping hyperparameters. Our calibration target is t_i^{conv}, the earliest SR scaling state for which the cosine similarity to the fixed-budget response reaches $\alpha = 0.90$. We select the baseline parameters that minimize the mean absolute difference between the resulting stopping round and this target, weighting early and late stopping equally; ties are broken by lower mean stopping state.

We evaluate our convergence model on SR against the corresponding fixed-round configurations and two adaptive stopping rules that require no additional LM calls: *(i)* semantic answer streak computes the cosine similarity between consecutive refinements and stops after that similarity exceeds a calibrated threshold for a calibrated number of consecutive rounds (Yan et al., 2025), and *(ii)* edit saturation computes the normalized character-level Levenshtein distance between consecutive responses and stops once that magnitude remains below a calibrated threshold for k consecutive refinements (Wu et al., 2026). We calibrate the semantic-streak cosine-similarity threshold over the shared SR grid spanning 0.150–0.900 and patience values $k \in \{1, \ldots, 5\}$. For edit saturation, we search the stopping threshold over the unique normalized edit-distance magnitudes observed in the calibration data and use the same patience grid $k \in \{1, \ldots, 5\}$.

B.6 CALIBRATION PROCEDURE

For each scaling state, the convergence model produces $\hat{c}_{i,t} = f_\theta(\mathbf{x}_{i,t})$. Given a base threshold ψ_0 and decay λ, we then define

$$\psi_t = \psi_0 - \lambda \frac{t}{T}$$

The resulting convergence-based stopping round is

$$\hat{t}_i = \min\left(\{t < T : \hat{c}_{i,t} \geq \psi_t\} \cup \{T\}\right),$$

where t ranges over the valid scaling states for the corresponding regime.

SC and SR use separate calibration objectives because their convergence targets encode different output structures. Let $\mathcal{V}$ denote the corresponding validation set. For SC, validation agreement is

$$\frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \mathbf{1}\left[a_{i,\hat{t}_i} = a_{i,T}\right].$$

For SC, we select the stopping rule with the lowest mean number of rounds while allowing at most one validation query to deviate from its fixed-budget consensus. Requiring perfect agreement would allow a single atypical query to force a substantially more conservative stopping rule. If no candidate satisfies this constraint, we select the stopping rule with the highest agreement and break ties by lower mean rounds.

For SR, calibration directly targets the earliest semantic-convergence round by minimizing:

$$\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}}\left|\hat{t}_i - t_i^{\mathrm{conv}}\right|,$$

breaking ties by lower mean rounds. SC calibration, therefore, prioritizes preservation of the final discrete answer, whereas SR calibration prioritizes stopping near the first refinement that is semantically equivalent to the fixed-budget response.

For SC, we search the base stopping threshold ψ_0 over a grid spanning 0.900–0.999. For SR, we search ψ_0 over a grid spanning 0.150–0.900. In both scaling regimes, we jointly search λ over the predefined decay grid spanning 0–0.10.

We repeat training and calibration using three seeds. Downstream metrics are first computed for each seed–model–dataset combination and then macro-averaged across model–dataset combinations.

C Validating the Design of CONGRATTS

C.1 Contribution of Runtime Feature Groups

We evaluate the contribution of the runtime features through a leave-one-group-out ablation. The 20 features are partitioned into five exhaustive, non-overlapping groups:

- **Answer uncertainty & diversity (6 features):** answer-distribution entropy, uncertainty, and confidence; the number of distinct answers in the current distribution; and the distinct-answer count and entropy over the sequence of selected answers.
- **Consensus strength (4 features):** the total answer count, the margin between the two most frequent answers, the normalized count margin, and the remaining-budget lock gap.
- **Answer stability (3 features):** both the current and longest same-answer streak, and the answer-change indicator.
- **Token likelihood (4 features):** the average and minimum token log-probabilities of the selected response and the running mean and deviation of its average token log-probability.
- **Scaling position (3 features):** the current scaling round and the fractions of the maximum scaling budget consumed and remaining.

For each ablation, we remove one group, retrain the convergence model, and independently recalibrate its SC and SR stopping rules, while all other experimental settings remain unchanged.

Table 7: **Leave-one-group-out feature ablation.** Each variant removes one complete feature group and is independently retrained and recalibrated. Accuracy (%) and mean scaling rounds are reported.

	SC		SR	
Variant	**Acc.**	**Rounds**	**Acc.**	**Rounds**
Full	62.06	8.29	23.31	1.88
No uncertainty/diversity	61.99	9.90	23.37	1.83
No consensus strength	62.01	8.18	23.22	1.76
No answer stability	62.15	8.71	23.38	2.67
No token likelihood	62.21	8.43	22.79	1.92
No scaling position	62.18	8.35	23.44	1.95

Table 7 shows different patterns across SC and SR. For SC, removing answer uncertainty and diversity has the largest effect on compute, increasing the mean from 8.29 to 9.90 rounds while decreasing accuracy by only 0.07 percentage points. Removing answer stability also increases computation, to 8.71 rounds, whereas consensus strength, token likelihood, and scaling position have comparatively small effects. This suggests that SC stopping intuitively depends most on whether the sampled answer distribution has concentrated and stabilized.

For SR, removing answer stability increases the mean refinement depth from 1.88 to 2.67 rounds with little change in accuracy, while removing token-likelihood features produces the largest accuracy decrease, from 23.31% to 22.79%. The remaining feature groups have smaller marginal effects.

Overall, the ablation shows a clear difference across scaling paradigms: distributional uncertainty contributes most strongly to SC stopping, whereas temporal stability and token likelihood contribute more strongly to SR. This supports a shared representation that accommodates the distinct convergence characteristics of SC and SR.

C.2 Sensitivity to Training-Set Size

We evaluate the sensitivity of the convergence model to training-set size using subsets of 30, 60, 120, and 240 training queries from each of our training datasets, corresponding to 90, 180, 360, and 720 total training queries. Calibration, test queries, model hyperparameters, and other experimental settings are unchanged. Table 8 reports macro-averaged results across model–dataset combinations, plus sample standard deviations across the three training seeds.

Table 8: **Sensitivity to convergence-model training-set size.** Accuracy (%) and mean scaling rounds are reported as mean ± sample standard deviation across three training seeds.

Queries per dataset	Total Queries	SC		SR	
		Acc.	Rounds	Acc.	Rounds
30	90	62.04 ± 0.04	8.84 ± 1.01	22.78 ± 0.63	1.67 ± 0.29
60	180	62.19 ± 0.26	8.85 ± 1.25	23.09 ± 0.17	2.03 ± 0.83
120	360	62.07 ± 0.19	8.19 ± 1.11	23.38 ± 0.40	2.32 ± 0.92
240	720	62.06 ± 0.46	8.29 ± 1.17	23.31 ± 0.16	1.88 ± 0.55

Performance remains stable across the evaluated training-set sizes. For SC, accuracy varies only from 62.04–62.19%, with mean computation between 8.19 and 8.85 rounds. For SR, accuracy increases from 22.78% with 90 total training queries to 23.38% with 360, while the full 720-query condition achieves a similar 23.31%. Mean refinement depth ranges from 1.67 to 2.32 rounds. Overall, the full training set is not required to recover the main accuracy–compute profile: 90 total training queries yields comparable downstream behavior.

C.3 Contribution of SC and SR Training Data

We vary the scaling paradigms represented in the convergence model's training data to assess how training-source composition affects performance and cross-paradigm transfer. We train on either SC data only, SR data only (from both SR datasets), or the joint SC+SR training set used in our main experiments. For each resulting model, we evaluate it separately on the held-out SC and SR evaluations, directly measuring transfer across scaling paradigms. SC stopping is calibrated on the held-out SC validation split, while SR stopping is calibrated on the combined held-out validation splits from both SR datasets. All other calibration and evaluation settings are unchanged. Table 9 reports the resulting accuracy and scaling depth for each training-source configuration.

Table 9: **Sensitivity to convergence-model training source.** Accuracy (%) and mean scaling rounds are reported as mean ± sample standard deviation across three training seeds.

Training data	Evaluation	Accuracy (%)	Rounds
SC only	SC	61.59 ± 0.83	7.72 ± 1.33
SC only	SR	20.27 ± 2.73	3.84 ± 2.94
SR only	SC	62.44 ± 0.06	12.09 ± 1.02
SR only	SR	23.52 ± 0.17	2.45 ± 0.83
SC+SR	SC	62.06 ± 0.46	8.29 ± 1.17
SC+SR	SR	23.31 ± 0.16	1.88 ± 0.55

Training on a single paradigm changes the resulting accuracy–compute profile. On SC, training only on SC data uses fewer rounds than joint training (7.72 versus 8.29) but reduces accuracy by 0.47 percentage points. Training only on SR data transfers to SC with similar accuracy, but is substantially more conservative at 12.09 rounds. On SR, training only on SR data achieves 23.52% accuracy at 2.45 rounds, while joint training achieves a similar 23.31% at 1.88 rounds. In contrast, training only on SC data and evaluating on SR reduces accuracy to 20.27% and increases refinement depth to 3.84 rounds. Cross-paradigm transfer is asymmetric: SR-only training preserves SC accuracy at substantially higher compute cost, whereas SC-only training degrades both accuracy and compute for SR. Joint training maintains a stronger shared accuracy–compute profile across both paradigms.

C.4 Convergence versus Correctness as the Stopping Target

We evaluate our design choice by replacing the convergence target with one derived from ground-truth correctness. Let $z_{i,t} \in \{0, 1\}$ indicate whether the output at scaling state t is correct. We define the correctness-derived stopping target as

$$t_i^{\text{corr}} = \begin{cases} \min\{t : z_{i,t} = 1\}, & \text{if a correct output occurs,} \\ 0, & \text{otherwise.} \end{cases}$$

For SC, $t = 0$ lies outside the valid scaling-state range, so the earliest valid state is $t = 1$. For SR, $t = 0$ corresponds to the initial model response, so an initially correct response naturally has $t_i^{\text{corr}} = 0$. For queries that never produce a correct output, assigning $t_i^{\text{corr}} = 0$ instead encourages the earliest possible stopping point during calibration.

For the learned controller, we replace the convergence label with per-state correctness while retaining the same 20 causal features, model architecture, seeds, and threshold grids. For adaptive baselines that depend on calibration, we recalibrate their stopping hyperparameters against the same correctness-derived target. Held-out runs and evaluation procedures are otherwise unchanged.

Table 10: **Convergence versus correctness-based stopping targets for SC.** Entries report downstream accuracy (%) / mean samples ± sample standard deviation over three training/calibration seeds, if relevant.

Method	Convergence	Correctness
ASC	62.35 / 9.04	62.35 / 9.04
CGES	61.67 ± 0.17 / 6.08 ± 0.38	59.10 / 3.69
ESC	61.98 / 10.20	51.18 / 1.00
CONGRATTS	62.06 ± 0.46 / 8.29 ± 1.17	62.47 ± 0.05 / 9.91 ± 0.58

For SC, the learned controller is relatively insensitive to the target definition. Our model's result changes from 62.06% accuracy at 8.29 samples under agreement supervision to 62.47% at 9.91 samples under correctness supervision. ASC is unchanged, while CGES and ESC become substantially more aggressive under the correctness target, reducing compute at the cost of lower downstream accuracy. Overall, the learned model achieves a similar level of accuracy without requiring access to task-specific correctness labels.

Table 11: **Semantic-convergence versus correctness-based stopping targets for SR**. Entries report macro-averaged accuracy (%) / refinement rounds as mean ± sample standard deviation across three training/calibration seeds.

Method	Convergence	Correctness
Edit Sat.	22.79 ± 0.72 / 3.31 ± 0.36	22.75 ± 0.04 / 1.01 ± 0.00
Semantic Streak	23.28 ± 0.81 / 3.72 ± 0.76	22.75 ± 0.00 / 1.02 ± 0.00
CONGRATTS	23.31 ± 0.16 / 1.88 ± 0.55	19.28 ± 0.74 / 0.46 ± 0.06

The distinction is substantially more important for SR. Under semantic-convergence supervision, ours obtains 23.31% accuracy using 1.88 refinement rounds. With correctness supervision, the controller reduces computation to only 0.46 refinement rounds on average, but accuracy falls to 19.28%. The calibrated heuristic baselines show the same: edit saturation falls from 3.31 to 1.01 refinements, while semantic same-answer streak falls from 3.72 to 1.02, with no corresponding improvement in accuracy.

This behavior follows from the sparsity of the correctness target in SR. Only 28.1% of the analyzed SR runs contain a correct intermediate answer, compared with 65.7% for SC. For most SR runs, refinement does not change an initially incorrect answer to a correct one; consequently, a correctness-derived stopping rule is biased toward stopping at the earliest possible point.

These results motivate using convergence as our target on both practical and statistical grounds. Practically, agreement is more general: it requires neither task-specific ground-truth answers nor a task-specific correctness evaluator. Statistically, it provides a target for every completed scaling run, including runs that remain incorrect throughout. For SC, this label-free target yields a downstream operating point similar to that of direct correctness supervision. For SR, it is substantially better behaved because it remains informative even when binary correctness is sparse or undefined.

C.5 SC Agreement Threshold: Ablations

We ablate the calibration disagreement rate used for adaptive SC by varying the number of validation disagreements permitted with the fixed-budget SC answer. Our learned model is calibrated on a 60-query validation set, so allowing d disagreements corresponds to a minimum agreement of (60-d)/60.

We evaluate $d \in \{0, 1, 2, 3\}$. The SC baselines are recalibrated using the same agreement rates on their 300-query calibration set, corresponding to 0, 5, 10, and 15 allowed disagreements. Table 12 reports the resulting accuracy–compute profiles.

Table 12: **SC calibration-agreement ablation.** Entries report accuracy (%) / mean samples, with mean $\pm$ sample standard deviation across three training/calibration seeds where applicable. We calibrate our learned model on 60 validation queries and the baselines on all 300 train+validation queries. We permit d disagreements on our 60-query validation set and the equivalent $5d$ disagreements on the 300-query baseline calibration set.

d	ASC	CGES	ESC	CONGRATTS
0	62.45 / 10.36	62.07 ± 0.18 / 7.37 ± 0.64	61.98 / 10.20	62.42 ± 0.16 / 9.84 ± 1.16
1	62.35 / 9.04	61.67 ± 0.17 / 6.08 ± 0.38	61.98 / 10.20	62.06 ± 0.46 / 8.29 ± 1.17
2	62.35 / 9.04	61.48 ± 0.20 / 5.45 ± 0.42	61.98 / 10.20	62.06 ± 0.46 / 8.29 ± 1.17
3	62.35 / 9.04	60.53 ± 1.24 / 4.54 ± 0.74	61.98 / 10.20	61.66 ± 0.35 / 7.11 ± 0.50

The calibration exposes a clear accuracy–compute trade-off. Requiring exact agreement (d=0) makes the adaptive stopping substantially more conservative: our method uses an average of 9.84 samples and achieves 62.42% accuracy. Allowing one disagreement reduces the mean sample count to 8.29, while retaining 62.06% accuracy. Increasing this to two disagreements leaves our accuracy unchanged at 62.06% and selects the same mean stopping budget, whereas allowing three disagreements reduces the mean budget further to 7.11 samples at the cost of a drop to 61.66% accuracy.

CGES is particularly sensitive, decreasing from 7.37 samples under exact agreement to 4.54 samples under three allowed disagreements, and reducing accuracy from 62.07% to 60.53%. ASC changes only under the strictest setting, while ESC selects the same setting throughout.

We use one allowed disagreement in the main experiments. This provides a middle operating point between exact agreement, which incurs substantially higher inference cost, and lower calibration thresholds, which begin to sacrifice downstream accuracy.

C.6 SR Semantic Threshold: Ablations

We ablate the semantic convergence threshold α used by our SR implementation; an intermediate answer is considered semantically converged when its embedding cosine similarity to the final fixed-budget answer is at least α. We vary α from 0.80 to 0.95 and recalibrate edit saturation, semantic same-answer streak, and our convergence model for each setting. All other components of the calibration and evaluation procedure are held fixed. Table 13 reports the resulting stopping behavior.

Table 13: **SR semantic-convergence threshold (α).** Entries report macro-averaged accuracy (%) / refinement rounds as mean $\pm$ sample standard deviation across three training/calibration seeds.

α	Edit Sat.	Semantic Streak	CONGRATTS
0.80	22.92 ± 0.51 / 2.00 ± 0.28	22.79 ± 0.49 / 2.12 ± 0.33	22.89 ± 0.50 / 1.06 ± 0.23
0.85	22.54 ± 0.18 / 2.35 ± 0.53	22.91 ± 0.32 / 2.51 ± 0.53	22.63 ± 0.48 / 1.50 ± 0.65
0.90	22.79 ± 0.72 / 3.31 ± 0.36	23.28 ± 0.81 / 3.72 ± 0.76	23.31 ± 0.16 / 1.88 ± 0.55
0.95	23.02 ± 0.34 / 5.57 ± 1.38	23.57 ± 0.34 / 5.02 ± 0.66	23.35 ± 0.14 / 4.39 ± 1.81

Increasing α makes semantic convergence more demanding and consequently shifts all three adaptive methods toward later stopping. For our method, the mean number of refinement rounds increases from 1.06 at α = 0.80 to 4.39 at α = 0.95, while accuracy remains within a comparatively narrow range. Semantic same-answer streak increases from 2.12 to 5.02 refinement rounds, and edit saturation increases from 2.00 to 5.57.

We use α = 0.90 in the main experiments as a fixed definition of semantic convergence, consistent with semantic-equivalence thresholds used in prior refinement-stopping work (Yan et al., 2025). The ablation shows that the principal effect of α is to control the aggressiveness of refinement: lower thresholds admit substantially earlier stopping, whereas stricter thresholds spend progressively more computation for relatively modest changes in task accuracy. With the main setting, our method achieves 23.31% accuracy using only 1.88 refinement rounds on average, compared with 23.28% at 3.72 rounds for semantic same-answer streak and 22.79% at 3.31 rounds for edit saturation.

Table 14: **Sensitivity to the embedding model used in SR.** Entries report macro-averaged accuracy (%) / refinement rounds as mean ± sample standard deviation across three training/calibration seeds.

Embedding model	Edit Sat.	Semantic Streak	CONGRATTS
`EmbeddingGemma-300M`	22.79 ± 0.72 / 3.31 ± 0.36	23.28 ± 0.81 / 3.72 ± 0.76	23.31 ± 0.16 / 1.88 ± 0.55
`MiniLM-L6-v2`	22.48 ± 0.73 / 2.58 ± 0.31	23.49 ± 0.94 / 3.77 ± 0.38	23.32 ± 0.21 / 1.52 ± 0.49
`BGE-small-v1.5`	22.70 ± 0.45 / 2.41 ± 0.58	22.65 ± 0.54 / 2.47 ± 0.65	23.24 ± 0.03 / 1.23 ± 0.26
`MPNet-Base-v2`	22.64 ± 0.77 / 2.74 ± 0.50	23.68 ± 0.60 / 3.71 ± 0.29	23.08 ± 0.07 / 1.93 ± 0.08

C.7 SR Embedding Model: Ablations

Our SR stopping target relies on embedding similarity to determine when an intermediate answer has become semantically equivalent to the fixed-budget answer. We test whether the resulting stopping behavior depends strongly on the particular embedding encoder used to construct this target.

We compare four modern, lightweight embedding models: `EmbeddingGemma-300M` (Vera et al., 2025), `all-MiniLM-L6-v2` (Wang et al., 2020), `BGE-small-en-v1.5` (Xiao et al., 2024), and `MPNet-Base-v2` (Song et al., 2020). For each encoder, we independently regenerate the semantic-convergence targets and recalibrate the SR stopping rules. For CONGRATTS, we additionally retrain the joint controller using the corresponding encoder-specific convergence labels. All other components are held fixed, including the evaluated data, features, classifier architecture, seeds, threshold grids, and semantic threshold $\alpha = 0.90$.

Table 14 reports the resulting accuracy and refinement depth for each encoder.

The results show that CONGRATTS is largely robust to the choice of semantic encoder at our chosen operating point; accuracy varies only from 23.08% to 23.32% across the models, while mean refinement cost remains between 1.23 and 1.93 rounds.

The calibrated heuristic baselines exhibit greater variation in their compute operating points. For example, the semantic same-answer streak ranges from 2.47 to 3.77 mean refinement rounds across encoders, while accuracy remains between 22.65% and 23.68%. Edit saturation similarly varies from 2.41 to 3.31 refinements while remaining within a relatively narrow accuracy range.

Importantly, cosine similarities are not calibrated to a common scale across embedding models. This means that a fixed numerical threshold, such as $\alpha = 0.90$, does not represent exactly the same degree of semantic equivalence across encoders. Despite this, CONGRATTS remains stable across four substantially different embedding families.

C.8 Convergence-Model Architecture

We use a histogram-based gradient-boosting (HGB) classifier as the convergence model. HGB requires no feature normalization and has negligible overhead relative to LM generation. The classifier uses `max_iter=80`, `learning_rate=0.05`, and `max_leaf_nodes=15`. We impose no explicit maximum depth and use the default L2 regularization of 0. We show a breakdown of the latency overheads of our stopping model, compared to the latency of an average LLM inference round, on our evaluated hardware in Table 15. The stopping decision adds only 0.56% and 2.52% of the latency of an average LLM inference round on the MacBook Pro and Jetson, respectively.

Table 15: **Runtime overhead of the stopping model.** Latency is measured during SR execution on the MacBook Pro M1 Max and Jetson AGX Orin. Values are the mean ± standard deviation across all recorded rounds from three runs. The stopping decision adds only 0.56% and 2.52% of the latency of an average LLM inference round on the Mac and Jetson, respectively.

Component	**M1 Max**	**Jetson AGX Orin**
LLM inference round (ms)	2974 ± 2847	3656 ± 3012
Feature extraction (ms)	0.99 ± 2.66	2.17 ± 3.69
Model inference (ms)	15.63 ± 12.32	89.98 ± 83.03
Cumulative (ms)	16.63 ± 12.58	92.15 ± 82.92
Decision / LLM round (%)	0.56	2.52

We now evaluate our chosen model architecture against three alternatives: XGBoost, a feed-forward multilayer network (MLP), and a gated recurrent unit (GRU). The four models all receive the same

20-dimensional feature representation and are trained using the same targets, training splits, and random seeds. HGB, XGBoost, and the MLP independently classify the feature vector corresponding to each scaling round. The GRU instead receives the same feature vectors as a causal sequence. Each architecture is trained independently for each training seed, and its SC and SR stopping rules are separately recalibrated using the same calibration as in the main experiment.

As shown in Table 16, downstream performance is highly stable across model families. On SC, accuracy ranges only from 61.44% to 62.18%, while mean compute remains between 7.44 and 8.64 rounds. XGBoost closely matches the HGB model, achieving 62.14% accuracy at 8.64 rounds compared with 62.06% at 8.29 rounds. The MLP and GRU similarly produce only small changes in the resulting operating point.

Table 16: **Sensitivity to the convergence-model architecture.** All models use the same features and are independently recalibrated. Values are mean ± sample standard deviation across three training/calibration seeds.

Scaling	Model	Accuracy (%)	Rounds
SC	HGB	62.06 ± 0.46	8.29 ± 1.17
	XGBoost	62.14 ± 0.50	8.64 ± 1.32
	MLP	61.44 ± 1.00	7.44 ± 1.67
	GRU	62.18 ± 0.23	8.55 ± 0.83
SR	HGB	23.31 ± 0.16	1.88 ± 0.55
	XGBoost	23.29 ± 0.13	1.54 ± 0.35
	MLP	22.92 ± 0.92	1.94 ± 1.11
	GRU	23.33 ± 0.09	2.33 ± 0.65

The same pattern holds for SR. Accuracy varies only from 22.92% to 23.33% across all four architectures. XGBoost closely matches HGB while stopping earlier, achieving 23.29% accuracy at 1.54 refinements compared with 23.31% at 1.88 refinements. The MLP achieves 22.92% at 1.94 refinements, while the causal GRU achieves 23.33% at 2.33 refinements.

Overall, the ablations show that CONGRATTS is not strongly dependent on a particular convergence-model architecture.

C.9 Calibration Set Sensitivity Analysis

SC stopping-rule calibration uses the 60-query MATH validation split, while SR calibration uses the 60-query validation splits from both RAG evaluation datasets. The non-learned baselines are instead calibrated on the full 300-query development set for each corresponding dataset.

To evaluate whether baseline performance depends materially on the calibration set, we recalibrate each baseline using only the corresponding 60-query validation split. For SR, this comparison is performed separately for each dataset, consistent with the dataset-specific baseline calibration used in the main experiments. The validation subset, calibration objectives, and method-specific hyperparameter grids remain fixed.

Table 17: **Effect of calibration-set size for SC baselines.** Entries report accuracy (%) / samples, with mean ± sample standard deviation across three calibration seeds where configurations vary. The 60-query condition uses only the held-out validation split, while the 300-query condition uses the full train/validation calibration set.

Method	60	300
ASC	62.35 / 9.04	62.35 / 9.04
CGES	62.07 ± 0.18 / 7.37 ± 0.64	61.67 ± 0.17 / 6.08 ± 0.38
ESC	61.89 ± 0.16 / 9.84 ± 0.62	61.98 / 10.20

SC performance is broadly stable across calibration-set sizes. ASC selects the same operating point in both conditions. CGES calibrated on all 300 examples uses 1.29 fewer samples on average, with a 0.40 percentage-point reduction in accuracy, while ESC changes only modestly, gaining 0.09 percentage points at the cost of 0.36 additional samples. Thus, increasing the calibration pool primarily shifts the selected accuracy–compute operating point rather than producing a systematic degradation in baseline quality.

Table 18: **Effect of calibration-set size for SR baselines.** Entries report macro-averaged accuracy (%) / refinement rounds as mean ± sample standard deviation across three calibration seeds. The 60-query condition uses only the held-out validation split, while the 300-query condition uses the full train/validation calibration set.

Method	60	300
Edit Sat.	22.89 ± 0.60 / 3.78 ± 0.50	22.79 ± 0.72 / 3.31 ± 0.36
Semantic Streak	23.09 ± 0.83 / 4.15 ± 0.23	23.28 ± 0.81 / 3.72 ± 0.76

The same pattern holds for SR. With 300-query calibration, edit saturation reduces its mean refinement budget from 3.78 to 3.31 rounds with only a 0.10 percentage-point change in accuracy, while

Table 19: **Hardware and resource-contention configuration.** CPU and memory contention vary across five workload settings. Resource feasibility is evaluated from live system measurements before each additional scaling round.

Platform	Resource	Available	Applied workload	Feasibility rule
MacBook Pro M1 Max	CPU	10 CPU cores	5 workers, 50–90% compute duty cycle	90% utilization limit + profiled cost/reserve
	Memory	32 GB unified memory	45–95% of physical memory	VM-pressure `warning` threshold
Jetson AGX Orin	CPU	12 CPU cores	12 workers, 50–90% compute duty cycle	90% utilization limit + profiled cost/reserve
	Memory	64 GB unified memory	45–95% of physical memory	4 GB available-memory threshold + profiled cost/reserve

semantic streak improves on both axes, increasing from 23.09% to 23.28% accuracy while reducing mean refinement rounds from 4.15 to 3.72.

Overall, these results indicate that the main conclusions are not driven by the larger calibration set provided to the baselines. Our baseline performance remains similar when restricted to the same 60-query validation subset used to calibrate the learned model, while access to the full 300-query pool generally yields comparable / more compute-efficient operating points.

D Hardware and Resource-Aware Evaluation Details

Main hardware experiments. The hardware experiments are run on both a MacBook Pro (M1 Max) and an NVIDIA Jetson AGX Orin Developer Kit.

The M1 Max has 10 CPU cores, a 32-core integrated GPU, and 32GB of unified memory, and runs macOS 26.6.2. The Jetson AGX Orin has 12 ARM Cortex-A78AE CPU cores and 64GB of memory, and runs JetPack 6.2.2 with Linux for Tegra (L4T) 36.5.0 and CUDA 12.6.Table 19 summarizes both hardware platforms, the contention workloads used, and our configured resource bounds.

We evaluate four resource conditions: idle, CPU contention, memory pressure, and combined CPU–memory contention. Each benchmark contains 60 matched queries. For CPU contention, we use five increasing workload settings ranging from 50% to 90%. Background workers repeatedly execute arithmetic expressions for the corresponding fraction of each 100ms interval and are idle for the remainder; we use 5 workers on the M1 Max and 12 on the Jetson. This produces controlled competition for CPU execution time without fixing the resulting whole-system utilization, which is measured independently during the experiment.

For memory pressure, we use five increasing settings spanning 45–95% of total physical memory. We use `stress-ng` (King, 2026) with a single virtual-memory worker to allocate the requested amount and repeatedly access the allocated pages for the complete query. This maintains an active physical-memory workload. This workload stresses each platform's main memory: unified memory on the M1 Max and system RAM on the Jetson.

The combined condition applies the corresponding CPU and memory workloads simultaneously.

Within each profile, each of the five settings is assigned to 12 of the 60 queries and remains fixed throughout each query. We leave 10 minutes between benchmark profiles to allow the platform to return to a comparable idle and thermal state.

We sample CPU utilization for the system and inference process, available and resident memory, swap usage, and platform-supported energy and thermal measurements throughout inference. The system state is sampled every 50ms; CPU feasibility uses a 1s rolling mean to avoid reacting to short utilization spikes. Before each scaling round, the most recent resource measurements are supplied to the feasibility controller. On the Mac M1 Max, CPU, GPU, ANE, and thermal measurements are collected using `powermetrics` (Apple Inc.) at 100ms intervals. On the Jetson AGX Orin, utilization, memory, power, and thermal measurements are collected using `tegrastats` (NVIDIA Corporation, 2026) at 100ms intervals. We run the Jetson AGX Orin in `MAXN` power mode to avoid a fixed power cap during contention experiments. Both `powermetrics` and `tegrastats` report instantaneous power. We integrate these measurements over the inference time to obtain per-query energy consumption, which is reported in joules (J) throughout.

We measure end-to-end system energy over each query interval without subtracting an idle-power baseline. Thus, reported energy includes normal background system consumption and reflects the energy consumed by the platform while serving the query, with *Fixed*, + *Resource*, and + *Conv.* evaluated under the same definition. We reduce short-term noise by using rolling or recent CPU and

memory observations alongside a quantile-based reserve for recent fluctuations, platform-specific profiling of marginal round costs, and the above-mentioned 10min recovery period between benchmark profiles. We further aggregate our results across profiles and controller-seed variants. Uncontrolled background activity may nevertheless introduce run-to-run variance.

For any active resource constraint that requires it, the controller incorporates a profiled estimate of the incremental cost of one additional scaling round. Before the resource-pressure evaluation, we run separate profiling and record round-level latency, CPU utilization, memory, and energy. These profiling queries are disjoint from the queries used in the hardware evaluation. From these runs, we estimate the 95th-percentile incremental round latency $\Delta^{95}_{\mathrm{lat}}$, CPU requirement $\Delta^{95}_{\mathrm{cpu}}$, and energy usage $\Delta^{95}_{\mathrm{energy}}$. On the Jetson, we also estimate the 95th-percentile incremental memory requirement $\Delta^{95}_{\mathrm{mem}}$. The resulting estimates are specific to the model and hardware platform. In our main hardware evaluation, we use CPU and memory as our active constraints, while latency and energy are measured as outcomes. The same profiled latency and energy costs can optionally be used to enforce query-level, user- or application-defined limits.

For readability, we omit the query index i in this section. In our main hardware evaluation, CPU and memory are the active constraints, so next-round feasibility is defined as

$$g_t = \min\left(g_t^{\mathrm{cpu}}, g_t^{\mathrm{mem}}\right).$$

Considering CPU feasibility, let u_t denote the 1s rolling whole-system CPU utilization after scaling state t, $\Delta^{95}_{\mathrm{cpu}}$ the profiled incremental CPU requirement of another round, and r_t^{cpu} a reserve for recent upward fluctuations in utilization. CPU feasibility is then defined as

$$g_t^{\mathrm{cpu}} = \mathbf{1}\left[u_t + \Delta^{95}_{\mathrm{cpu}} + r_t^{\mathrm{cpu}} \le u_{\max}\right],$$

where $u_{\max}$ is the configured CPU-utilization limit. The reserve r_t^{cpu} is obtained from the 95th percentile of recent positive changes in the rolling CPU signal over the preceding eight samples.

We consider a completed state invalid if observed whole-system CPU utilization exceeds 90% during our evaluation. Profiled next-round CPU cost and a reserve for recent fluctuations are used to decide whether another scaling round should be attempted.

On the M1 Max, memory feasibility is determined directly from the native macOS VM-pressure state[3] rather than from a profiled incremental memory cost. At scaling state t, we define

$$g_t^{\mathrm{mem}} = \begin{cases} 1, & \text{if VM pressure is } \texttt{normal}, \\ 0, & \text{if VM pressure is } \texttt{warning} \text{ or } \texttt{critical}. \end{cases}$$

Consequently, reaching `warning` prevents another scaling round, while reaching `critical` is recorded as a memory-resource violation. The same `warning` stopping bound applies to both fixed-budget and adaptive scaling.

On the Jetson, memory feasibility is evaluated based on the available memory and the profiled incremental memory requirement for the next scaling round. We reserve 4GB of memory for the system and permit further scaling when the remaining capacity above this reserve exceeds the profiled 95th-percentile memory cost of an additional round, plus a reserve for recent fluctuations in available memory. Let m_t^{avail} denote `MemAvailable` after scaling state t, $\Delta^{95}_{\mathrm{mem}}$ the profiled incremental memory requirement of another round, r_t^{mem} the reserve for recent decreases in available memory, and $m_{\min} = 4\mathrm{GB}$ the minimum available-memory threshold. Memory feasibility is then defined as

$$g_t^{\mathrm{mem}} = \mathbf{1}\left[m_t^{\mathrm{avail}} - \Delta^{95}_{\mathrm{mem}} - r_t^{\mathrm{mem}} \ge m_{\min}\right]$$

The reserve r_t^{mem} is obtained from the 95th percentile of recent decreases in available memory over the preceding eight samples.

We do not apply the profiled next-round CPU or (Jetson) memory feasibility predictor for the fixed-budget baselines. We retain and count as invalid any resource-bound violations, and we exclude subsequent scaling rounds. On the M1 Max, native VM-pressure `warning` prevents further scaling,

[3] We query the native macOS/XNU `sysctl` interface, `kern.memorystatus_vm_pressure_level`, at runtime to obtain the system-wide VM-pressure state (`normal`, `warning`, or `critical`).

but is not counted as invalid itself; `critical` is treated as a violation. Hence, all methods use the same definitions of an observed CPU or memory violation.

Hardware experiments use `llama-cpp-python` 0.3.35 (Gerganov & contributors, 2026). We use a 16-bit floating-point (FP16) `Qwen2.5-1.5B-Instruct` GGUF model without weight quantization. We use a context length of 4096 for SC and 16384 for SR, a prompt-evaluation token count of $n_{\text{batch}} = 512$, full layer offload to the available accelerator (`n_gpu_layers`=-1), and the default `llama.cpp` CPU-thread count (half of available). Hardware generation uses a separately fixed configuration of temperature 0.7, top-$p = 0.9$, and top-$k = 20$, shared by all compared methods. The benchmark explicitly applies temperature scaling, top-k filtering, and top-p filtering to the logits before sampling so that differences in `llama.cpp` sampler defaults do not alter the decoding procedure. The hardware experiments evaluate relative controller behavior under a common on-device workload, rather than a direct numerical correspondence to the offline evaluation.

E Extended Experimental Results

E.1 Complete Results with Quantized Models

We evaluate the performance of our convergence model on 8- and 4-bit quantized variants of each LLM from our main evaluation, using `bitsandbytes`[4] for quantization. By default, `bitsandbytes` 8-bit vector-wise quantization handles outlier activations separately using 16-bit matrix multiplication. We disable this outlier handling, as the associated 16-bit matrix multiplication adversely affects latency on our GPU architecture[5]. For `SmolLM2-1.7B-Instruct` (Allal et al., 2025), however, disabling outlier handling leads to a substantial drop in accuracy. We therefore omit 8-bit results for this model.

For this evaluation, the trained convergence model and calibrated rules are transferred unchanged from the full-precision configuration to 8-bit and 4-bit inference. No retraining or recalibration is performed. The generation parameters, queries, and other settings used in our main evaluation are also left entirely unchanged.

Quantization preserves a similar accuracy–compute trade-off without retraining or recalibration. Tables 20 and 21 report the complete SC results for all four models at 4-bit and 8-bit precision, respectively. For SC, CONGRATTS reduces the mean number of scaling rounds by 51.6% at 8-bit and 43.8% at 4-bit relative to 16-round scaling, with accuracy decreases of only 0.01 and 0.37 percentage points, respectively. Under 8-bit quantization, CONGRATTS uses fewer rounds than ASC (7.74 vs. 8.41) and ESC (9.96), while maintaining comparable accuracy. CGES is more aggressive at 5.63 rounds but incurs a larger drop in accuracy. Under 4-bit, CONGRATTS similarly uses slightly fewer rounds than ASC (8.99 vs. 9.64) while retaining higher average accuracy than CGES and ESC.

The pattern is stronger for SR. CONGRATTS reduces refinement rounds by 79.2% at 8-bit and 83.3% at 4-bit while improving accuracy by 2.61 and 2.22 percentage points, respectively, relative to 16-round scaling. It also achieves both higher average accuracy and fewer refinement rounds than edit saturation and semantic streak at both precisions. Tables 22 and 23 report the corresponding complete SR results for all four models. These results show that the learned stopping rule retains similar behavior under quantization without target-precision retraining or recalibration.

[4] `https://huggingface.co/docs/bitsandbytes/en/index`

[5] `https://github.com/bitsandbytes-foundation/bitsandbytes/issues/1867`

Table 20: **Self-consistency (SC), 4-bit quantization.** Accuracy (%) / mean scaling rounds. Adaptive configurations are trained or calibrated on full-precision data and frozen before quantized evaluation. CGES and CONGRATTS report mean ± sample standard deviation across three calibration/training seeds, each evaluated on all quantized runs. ASC ($\beta = 0.90$) and ESC (streak length $= 8$) produce identical results across seeds and are shown without ±. Bold denotes the highest mean accuracy in each row; underlining denotes the lowest mean round count among adaptive methods. The final row reports macro-averages across all model–dataset pairs.

Model	Dataset	Base	Fixed16	ASC	CGES	ESC	CONGRATTS
Qwen2.5-1.5B	MATH-500	32.69 / 1.00	44.78 / 16.00	45.33 / 11.88	44.96±0.16 / 8.71±0.37	45.33 / 11.40	**45.42**±0.57 / 11.00±1.14
	GSM8K	56.00 / 1.00	**73.67** / 16.00	73.47 / 8.70	73.64±0.14 / 6.31±0.41	73.53 / 10.00	73.53±0.52 / 8.45±1.32
	SVAMP	73.33 / 1.00	83.00 / 16.00	82.33 / 6.62	**84.22**±0.19 / 5.11±0.31	82.33 / 9.30	83.00 / 6.74±1.28
Llama-3.2-1B	MATH-500	23.08 / 1.00	34.34 / 16.00	34.34 / 12.96	34.25±0.42 / 9.89±0.46	32.69 / 11.70	**34.52**±0.42 / 12.19±1.18
	GSM8K	35.93 / 1.00	**49.00** / 16.00	48.73 / 10.91	48.62±0.10 / 7.92±0.41	47.67 / 10.77	48.47±0.24 / 10.07±1.33
	SVAMP	52.67 / 1.00	65.33 / 16.00	**65.67** / 8.34	62.33±0.33 / 5.22±0.40	64.33 / 9.77	63.67±1.20 / 7.22±1.23
LFM2.5-1.2B	MATH-500	45.60 / 1.00	55.22 / 16.00	54.67 / 9.11	**55.49**±0.48 / 5.84±0.37	54.67 / 10.44	55.31±0.32 / 8.24±0.99
	GSM8K	68.33 / 1.00	**76.07** / 16.00	75.80 / 6.69	**76.07**±0.18 / 4.18±0.24	75.93 / 9.29	75.71±0.04 / 6.05±1.04
	SVAMP	77.67 / 1.00	**83.33** / 16.00	83.00 / 5.83	81.44±0.19 / 3.83±0.27	82.00 / 8.98	81.89±0.19 / 5.41±1.06
SmolLM2-1.7B	MATH-500	12.91 / 1.00	**26.37** / 16.00	25.82 / 13.96	25.64±0.16 / 10.93±0.48	25.27 / 12.18	25.73±0.42 / 13.31±1.12
	GSM8K	32.13 / 1.00	47.00 / 16.00	46.87 / 11.60	46.73±0.31 / 8.49±0.43	46.60 / 11.12	**47.31**±0.37 / 10.65±1.33
	SVAMP	53.33 / 1.00	67.67 / 16.00	**68.00** / 9.07	67.00±0.33 / 6.29±0.34	67.33 / 9.99	66.78±0.51 / 8.52±1.28
Avg. ΔAcc. (pp) / ΔRounds (%)		46.97 / 1.00 -11.85 / -93.8%	**58.82** / 16.00 reference	58.67 / 9.64 -0.15 / -39.8%	58.37±0.09 / 6.89±0.37 -0.45 / -56.9%	58.14 / 10.41 -0.68 / -34.9%	58.45±0.33 / 8.99±1.19 -0.37 / -43.8%

Table 21: **Self-consistency (SC), 8-bit quantization.** Accuracy (%) / mean scaling rounds. Training, calibration, and aggregation follow the same procedure as in our 4-bit experiment. CGES and CONGRATTS report mean ± sample standard deviation across three calibration/training seeds; ASC and ESC yield identical results across seeds. SmolLM2-1.7B is excluded because no 8-bit evaluation data is available. Bold denotes the highest mean accuracy in each row; underlining denotes the lowest mean round count among adaptive methods. The final row reports macro-averages across all model–dataset pairs.

Model	Dataset	Base	Fixed16	ASC	CGES	ESC	CONGRATTS
Qwen2.5-1.5B	MATH-500	37.09 / 1.00	46.43 / 16.00	46.43 / 10.97	**47.53**±0.27 / 7.69±0.41	45.33 / 10.97	46.25±0.32 / 10.08±1.13
	GSM8K	60.80 / 1.00	72.93 / 16.00	**73.40** / 7.80	72.22±0.31 / 5.31±0.38	72.47 / 9.58	72.56±0.48 / 7.32±1.17
	SVAMP	73.33 / 1.00	83.67 / 16.00	83.33 / 5.96	82.67±0.33 / 4.16±0.33	83.67 / 8.96	**84.00**±0.33 / 5.90±1.13
Llama-3.2-1B	MATH-500	22.25 / 1.00	33.52 / 16.00	33.79 / 12.72	34.25±0.63 / 9.23±0.38	**35.71** / 11.71	34.71±0.16 / 11.99±1.20
	GSM8K	35.67 / 1.00	48.67 / 16.00	**48.93** / 10.80	48.40±0.29 / 7.79±0.43	48.27 / 10.79	48.53±0.13 / 10.00±1.29
	SVAMP	60.00 / 1.00	68.00 / 16.00	68.00 / 7.34	67.33±0.00 / 4.59±0.38	**68.33** / 9.56	67.67±0.33 / 6.52±1.15
LFM2.5-1.2B	MATH-500	50.27 / 1.00	**58.79** / 16.00	58.52 / 8.79	57.42±0.48 / 4.93±0.38	58.52 / 10.24	58.61±0.42 / 7.45±0.95
	GSM8K	71.13 / 1.00	79.80 / 16.00	**80.00** / 6.08	79.69±0.31 / 3.71±0.22	79.80 / 9.15	79.20±0.53 / 5.45±0.97
	SVAMP	82.00 / 1.00	86.33 / 16.00	86.33 / 5.24	85.89±0.19 / 3.28±0.23	**86.67** / 8.73	86.56±0.19 / 4.98±1.09
Avg. ΔAcc. (pp) / ΔRounds (%)		54.73 / 1.00 -9.51 / -93.8%	64.24 / 16.00 reference	64.30 / 8.41 +0.06 / -47.4%	63.93±0.19 / 5.63±0.35 -0.31 / -64.8%	**64.31** / 9.96 +0.07 / -37.7%	64.23±0.15 / 7.74±1.11 -0.01 / -51.6%

Table 22: **Self-refinement (SR), 4-bit quantization.** Accuracy (%) / mean refinement rounds. Adaptive configurations are calibrated on full-precision data and frozen before being evaluated in quantized mode. Edit saturation, semantic streak, and CONGRATTS report mean ± sample standard deviation across three calibration/training seeds. Bold denotes the highest mean accuracy in each row; underlining denotes the lowest mean round count among adaptive methods. The final row reports macro-averages across all model–dataset pairs.

Model	Dataset	Base	Fixed4	Fixed16	Edit sat.	Semantic streak	CONGRATTS
Qwen2.5-1.5B	CRAG	12.00 / 0.00	26.20 / 4.00	23.40 / 16.00	25.40±0.35 / 2.54±0.65	**26.67**±0.61 / 3.38±0.85	24.47±0.61 / 2.47±0.63
	POPQA	14.80 / 0.00	24.20 / 4.00	23.60 / 16.00	23.87±0.90 / 2.48±0.61	**24.33**±0.31 / 2.78±0.33	**24.33**±0.31 / 3.12±0.56
Llama-3.2-1B	CRAG	16.80 / 0.00	21.20 / 4.00	**23.40** / 16.00	20.80±0.20 / 3.23±0.41	21.87±1.10 / 4.25±1.46	22.67±0.58 / 2.17±0.80
	POPQA	11.80 / 0.00	20.80 / 4.00	21.00 / 16.00	21.20±1.31 / 2.91±0.62	21.80±0.69 / 3.00±0.20	**22.07**±0.83 / 5.83±1.25
LFM2.5-1.2B	CRAG	11.40 / 0.00	19.40 / 4.00	16.60 / 16.00	23.53±0.64 / 4.00±0.85	22.47±0.31 / 4.89±2.08	**25.47**±0.23 / 2.60±1.14
	POPQA	9.00 / 0.00	21.20 / 4.00	22.00 / 16.00	22.20±0.53 / 2.91±0.86	22.40±0.72 / 3.01±0.31	**22.80**±0.60 / 2.22±0.73
SmolLM2-1.7B	CRAG	14.40 / 0.00	13.60 / 4.00	16.20 / 16.00	16.60±2.46 / 4.94±1.05	18.27±0.23 / 5.88±1.94	**19.87**±0.83 / 1.61±0.49
	POPQA	13.80 / 0.00	17.00 / 4.00	14.20 / 16.00	**17.60**±0.20 / 2.69±0.55	16.73±0.61 / 2.78±0.28	16.47±0.31 / 1.39±0.29
Avg. ΔAcc. (pp) / ΔRounds (%)		13.00 / 0.00 -7.05 / -100.0%	20.45 / 4.00 +0.40 / -75.0%	20.05 / 16.00 reference	21.40±0.58 / 3.21±0.38 +1.35 / -79.9%	21.82±0.04 / 3.75±0.75 +1.77 / -76.6%	**22.27**±0.14 / 2.68±0.71 +2.22 / -83.3%

Table 23: **Self-refinement (SR), 8-bit quantization.** Accuracy (%) / mean refinement rounds. Training, calibration, and aggregation follow our 4-bit experiment. Edit saturation, semantic streak, and CONGRATTS report mean ± sample standard deviation across three calibration/training seeds. `SmolLM2-1.7B` is excluded because no 8-bit evaluation data is available. Bold denotes the highest mean accuracy in each row; underlining denotes the lowest mean round count among adaptive methods. The final row reports macro-averages across all model–dataset pairs.

Model	Dataset	Base	Fixed4	Fixed16	Edit sat.	Semantic streak	CONGRATTS
`Qwen2.5-1.5B`	CRAG	13.20 / 0.00	26.40 / 4.00	23.40 / 16.00	26.53±0.31 / 3.18±0.42	**27.40**±0.35 / 3.67±1.03	25.07±0.23 / 2.80±0.73
	POPQA	15.00 / 0.00	26.80 / 4.00	24.60 / 16.00	24.73±0.58 / 3.26±0.70	25.53±0.70 / 3.12±0.24	**27.40**±0.53 / 4.24±0.92
`Llama-3.2-1B`	CRAG	17.80 / 0.00	**23.00** / 4.00	22.00 / 16.00	**23.00**±0.69 / 3.41±0.73	21.93±0.99 / 3.79±1.13	21.73±0.70 / 2.39±0.82
	POPQA	13.00 / 0.00	21.00 / 4.00	19.40 / 16.00	20.73±0.64 / 3.26±0.70	20.60±1.64 / 3.03±0.32	**21.33**±0.76 / 4.59±1.17
`LFM2.5-1.2B`	CRAG	14.40 / 0.00	24.20 / 4.00	18.60 / 16.00	23.93±0.42 / 4.26±1.07	**26.20**±2.46 / 4.88±2.03	**26.20**±0.35 / 2.23±0.90
	POPQA	9.40 / 0.00	21.80 / 4.00	22.20 / 16.00	23.53±0.50 / 3.41±1.64	23.40±0.53 / 3.08±0.37	**24.13**±0.31 / 3.69±0.72
Avg. ΔAcc. (pp) / ΔRounds (%)		13.80 / 0.00 -7.90 / -100.0%	23.87 / 4.00 +2.17 / -75.0%	21.70 / 16.00 reference	23.74±0.10 / 3.47±0.47 +2.04 / -78.4%	24.18±0.75 / 3.60±0.58 +2.48 / -77.5%	**24.31**±0.18 / 3.33±0.85 +2.61 / -79.2%