Sample Count Is Not Enough: Candidate-Generation Strategy Shapes the Energy and Performance of LLM Test-Time Scaling

Mobina Kashaniyan and Ali Jannesari
Department of Computer Science
Iowa State University, USA
{mobina, jannesar}@iastate.edu

***Abstract*—Test-time scaling can improve large language model reasoning by generating and combining multiple candidate responses. In sampling-based methods, the inference budget is often described by the number of generated candidates, N. However, N tells us how many candidates are generated, not how they are executed. The same candidate budget can be produced in one batched generation call or split across several sequential calls with smaller batch sizes. We first study the effect of increasing N on reasoning accuracy using Phi-3-mini and Qwen2.5-1.5B on 500 GSM8K prompts. As expected, increasing N from 1 to 8 improves accuracy by 8.4 percentage points for Phi-3-mini and 18.4 points for Qwen2.5-1.5B. However, accuracy alone does not show the systems cost of using a larger candidate budget. We therefore fix $N = 8$ and compare four generation schedules: 1×8, 2×4, 4×2, and 8×1, where $a \times b$ denotes a generation calls with b candidates per call. We measure latency, throughput, GPU-hours, and gross GPU-device energy while keeping the total candidate count fixed. On A100 GPUs, eight serial calls use 4.64–$4.86\times$ as much gross GPU-device energy and have 5.77–$6.12\times$ the P95 latency of one batched call with eight candidates. The same pattern appears across three independently scheduled A100 nodes per model and in short-output SciQ/V100 experiments. These results show that candidate count alone is not enough to describe the systems cost of multi-candidate test-time scaling. When candidates are independent and memory allows it, fewer generation calls with larger batch sizes are more efficient. Evaluations should therefore report not only candidate count and accuracy, but also generation schedule and GPU-level systems metrics.**

***Index Terms*—large language models, test-time scaling, candidate generation, GPU inference, energy measurement, performance, high-performance computing**

I. Introduction

Test-time scaling improves large language model (LLM) reasoning by allocating additional inference compute, often through multiple sampled responses. Self-consistency, for example, generates several reasoning paths and selects the most frequent final answer [1]. In these methods, the inference budget is often summarized by the candidate count, N. However, N tells us how many candidates are generated, not how they are executed. The same candidate budget can be generated in one batched call or split across several sequential calls with smaller batch sizes. Although these schedules use the same number of candidates and the same aggregation rule, they require different numbers of generation calls and use different batch sizes. This can lead to large differences in latency, throughput, GPU-hours, utilization, and energy. We study this execution choice at $N = 8$ using 1×8, 2×4, 4×2, and 8×1 schedules. This issue is especially relevant in HPC settings, where LLM inference may run as finite batch jobs rather than continuous serving workloads. Candidate generation may also be divided across calls for logging, deterministic seeding, control logic, or intermediate analysis. Although serving research has shown that batching and scheduling affect inference efficiency [26]–[28], Table I shows that representative test-time-scaling studies often report candidate budgets without reporting calls per query, candidates per call, or measured energy. This makes systems results harder to reproduce and compare. We address three questions:

1) How do accuracy and systems cost change as batched N increases from 1 to 8?
2) At fixed $N = 8$, how does generation schedule affect latency, throughput, GPU-hours, utilization, and energy?
3) Do these effects remain across different GPU nodes and in a short-output workload?

This paper makes three contributions:

- **Execution schedule and reporting gap.** We formalize the candidate-generation schedule as $S = (b_1, \ldots, b_C)$ and show that candidate count N alone does not fully describe a multi-candidate inference workload. We also show that calls per query and candidates per call are often missing from representative test-time-scaling studies.
- **Fixed-budget systems characterization.** At fixed $N = 8$, we measure the end-to-end effect of 1×8, 2×4, 4×2, and 8×1 schedules on latency, throughput, GPU-hours, utilization, and gross GPU-device energy. On A100 GPUs, serial execution uses 4.64–$4.86\times$ as much energy as a single eight-candidate batched call.
- **Robustness and practical guidance.** We evaluate additional A100 nodes, two models, and a short-output SciQ/V100 case study. The results support a practical guideline: when candidates are independent and memory allows it, fewer generation calls with larger batch sizes are more efficient. We also provide minimum reporting recommendations for multi-candidate inference experiments.

These results show that the system cost of test-time scaling depends not only on how many candidates are generated, but also on how those candidates are grouped into generation calls.

II. Background and Related Work

A. Test-Time Scaling

Test-time scaling improves LLM outputs by using additional compute during inference. Some methods spend this compute on extending or refining a reasoning trajectory, while others generate and combine multiple candidate responses. Prior work shows that additional test-time compute can improve reasoning and can sometimes help smaller models approach the performance of larger ones under similar inference budgets [2]–[4]. However, the benefit depends on factors such as prompt difficulty, reasoning strategy, stopping, and aggregation [5]. Our work focuses on another factor: how a multi-candidate inference budget is executed on the underlying hardware.

B. Multi-Candidate Sampling

Self-consistency samples multiple reasoning paths and selects the most frequent answer [1], while best-of-N selects a candidate using a verifier, reward model, or confidence measure [6]. Prior studies examine larger or adaptive sampling budgets, stopping rules, and different selection methods [7]–[14]. These works mainly study how many candidates to generate or how to select among them. In contrast, we keep the candidate budget fixed and study how executing the same candidates across different numbers of generation calls and batch sizes affects systems cost.

C. Inference Scheduling and Energy

LLM inference efficiency depends on batching, scheduling, memory management, and resource allocation [16]. ORCA, vLLM, and Sarathi-Serve improve utilization and throughput through different batching and scheduling strategies [26]–[28]. Other work studies heterogeneous scheduling and KV-cache constraints [18], [19]. Energy consumption also depends on model size, hardware, batch size, sequence length, parallelism, and utilization [21]–[25]. Recent work has also compared the performance and energy efficiency of LLM inference across different AI accelerators and batch sizes [17]. These studies show that execution choices can strongly affect inference cost. Our work connects these systems factors to multi-candidate test-time scaling by measuring how generation-call structure affects latency, GPU-hours, utilization, and gross GPU-device energy at fixed N.

D. Reporting Practices in Prior Work

Table I audits representative foundational, adaptive-budget, and repeated-sampling studies. We record a field as reported only when the corresponding execution detail is stated explicitly in the paper or its supplementary material. "NR" denotes not reported. The audit is intended to characterize reporting practices in representative work rather than provide an exhaustive systematic review. Representative studies commonly report candidate budgets but do not fully specify generation-call structure or measured energy cost.

This work reports N, calls per query, candidates per call, inference hardware, and gross GPU-device energy.

TABLE I
Reporting practices in representative multi-candidate and test-time-scaling studies. HW refers to candidate-generation hardware; "NR" means not reported.

Study	N	Calls/ query	Cand./ call	HW	Energy
Self-Consistency [1]	Yes	NR	NR	NR	NR
Adaptive-Consistency [7]	Yes	NR	NR	NR	NR
Universal Self-Consistency [8]	Yes	NR	NR	NR	NR
Large Language Monkeys [38]	Yes	NR	NR	Yes	NR
Scaling Test-Time Compute [2]	Yes	NR	NR	Yes	NR
Difficulty-Adaptive SC [10]	Yes	NR	NR	NR	NR
This work	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

III. Methods

A. Candidate-Generation Strategies

Our goal is to separate how many candidates are generated from how those candidates are executed. Let N be the total number of candidates generated for a prompt. We represent the generation schedule as

$$\mathcal{S} = (b_1, \ldots, b_C), \qquad \sum_{c=1}^{C} b_c = N,$$

where C is the number of generation calls and b_c is the number of candidates generated in call c. Figure 1 shows the workflow for $N = 8$. We compare 1×8, 2×4, 4×2, and 8×1 schedules. In an $a \times b$ schedule, a is the number of generation calls and b is the number of candidates generated in each call. The calls are executed sequentially on the same allocated GPU, while candidates within each call are generated together as a batch. The four schedules therefore correspond to $\mathcal{S} = (8)$, $(4, 4)$, $(2, 2, 2, 2)$, and $(1, 1, 1, 1, 1, 1, 1, 1)$. All schedules use the same

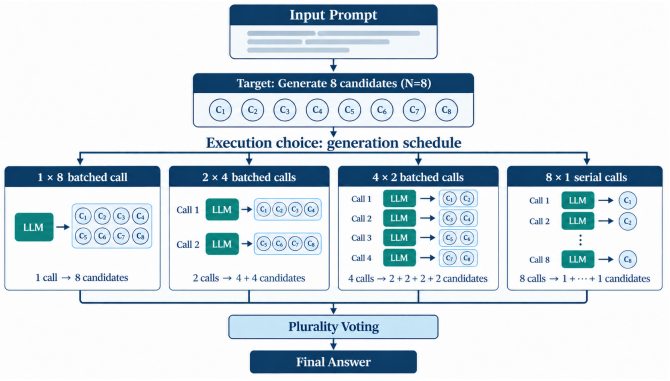

Fig. 1. Generation schedules for a fixed candidate budget of $N = 8$. All schedules use the same prompt, generate eight candidates, and apply the same plurality vote. They differ in the number of sequential generation calls and candidates generated per call. Repeated LLM blocks represent successive calls on the same GPU.

prompts, decoding settings, answer extraction, and voting procedure. Candidate responses are sampled independently across schedules in the systems experiments. We also study how accuracy changes as the candidate budget increases. For each

prompt, we generate eight candidates and compute accuracy for $N \in \{1, 2, 3, 4, 8\}$ using the first N candidates. This gives paired comparisons across candidate counts. These candidates are used only for the accuracy analysis. Systems measurements are collected separately by running each schedule on the GPU.

B. Token Accounting

We track logical token volume to check that differences between schedules are not caused by large differences in generated response length. Let P_q be the prompt length for query q. Since each candidate is generated from the same prompt, the candidate-associated logical input volume is

$$T_{\text{input}}^{\text{logical}}(q) = NP_q.$$

For fixed $N = 8$, this is $8P_q$ for every schedule. Let $L_{q,c,j}$ be the generated length of candidate j in call c. The logical number of generated tokens is

$$T_{\text{gen}}^{\text{logical}}(q, \mathcal{S}) = \sum_{c=1}^{C} \sum_{j=1}^{b_c} L_{q,c,j}.$$

In the fixed-N experiments, the candidate count is identical across schedules and the logical generated-token volume remains closely matched. This lets us compare the systems cost of different generation schedules while keeping the overall candidate budget fixed.

C. Answer Extraction and Voting

After generation, all candidates use the same answer-extraction and plurality-voting procedure. Candidates with the same extracted answer form a group, and the largest group determines the final prediction. We do not use a verifier, reward model, or token-level score. For GSM8K, we extract the numeric final answer by checking the requested final-answer delimiter, boxed expressions, explicit answer statements, trailing numeric expressions, and the last non-empty line. Numeric answers are then normalized to a common form. For SciQ, we extract one of the answer choices A–D case-insensitively. Extraction failures remain possible voting outcomes and are counted as incorrect if selected. If multiple answers receive the same number of votes, we select the answer that appears first among the generated candidates. In the accuracy analysis, this rule can make $N = 2$ identical to $N = 1$ when the first two candidates disagree. We therefore also evaluate $N = 3$ and report a sensitivity analysis using uniform random tie-breaking.

D. Measurement Boundary and Energy

Each measured query begins with GPU synchronization and an initial NVML cumulative-energy reading. The measured interval includes prompt processing, prefill, decoding, all generation calls, answer extraction, and plurality voting. After the final vote, we synchronize the GPU again and record the final cumulative-energy value. Model loading, warm-up, reporting-time token counting, and final grading are excluded. Gross GPU-device energy is computed as

$$E_{\text{gross}} = E_{\text{NVML,end}} - E_{\text{NVML,start}}.$$

Idle energy is not subtracted. The reported values therefore represent gross GPU-device energy over the full measured query interval, not isolated dynamic computation energy or whole-node energy. Average GPU power is computed as gross energy divided by query latency. A schedule can therefore have lower average power but still consume more total energy if it keeps the GPU active for longer.

IV. Experimental Design and Measurement

A. Study Overview

We organize the evaluation around three goals. First, we measure how accuracy changes as the candidate budget increases. Second, we measure how systems cost changes with candidate count and with generation schedule. Third, we check whether the main scheduling effect remains across different GPU nodes and on a short-output workload. Table II summarizes the five experiments used for these goals. For

TABLE II
Summary of the experimental studies.

Study	Workload	Design
Accuracy scaling	GSM8K, 500 prompts	One eight-candidate pool; evaluate $N \in \{1, 2, 3, 4, 8\}$ using prefixes
Batched scaling	GSM8K, 100×3	Batched $N \in \{1, 2, 4, 8\}$ and serial $N = 8$
Schedule sweep	GSM8K, 100×3	Fixed $N = 8$: 1×8, 2×4, 4×2, 8×1
Cross-node check	GSM8K, 100×3	Repeat 1×8 and 8×1 on two additional A100 nodes
Short-output validation	SciQ, 500×3	Complete fixed-$N = 8$ sweep on V100 GPUs

the accuracy study, each model generates eight candidates for 500 GSM8K prompts. Accuracy for smaller values of N is computed using the first N candidates from the same set. This gives paired prompt-level comparisons across candidate budgets. These generations are used only for accuracy analysis; their latency and energy are not assigned to individual values of N. The GSM8K systems experiments use the same 100 prompts over three repetitions. The batched-scaling study measures how systems cost changes as N increases. The main schedule sweep instead fixes $N = 8$ and changes only the number of generation calls and candidates per call. All four schedules are executed within the same A100 job. Two repetitions use the order 1×8, 2×4, 4×2, 8×1, while one uses the reverse order to reduce possible order and thermal effects. To test run-to-run stability, we repeat the 1×8 and 8×1 endpoints on two additional A100 nodes per model. Together with the primary job, this gives three independently scheduled A100 jobs per model. Finally, we repeat the full fixed-$N = 8$ schedule sweep on 500 SciQ prompts over three repetitions using V100 GPUs. SciQ produces much shorter responses than GSM8K, so we use it as a separate short-output validation. It is not intended as a complete study of how output length affects scheduling cost.

B. Prompts, Seeds, and Warm-Up

The GSM8K test split is shuffled with seed 42. The first 100 prompts are used for the systems experiments and are also part of the 500-prompt accuracy study. Prompt order is fixed across schedules, repetitions, jobs, and nodes. We use deterministic method-specific seeds so that each run is reproducible while schedules sample candidates independently. Two warm-up generations are performed after model loading and are excluded from measurement.

C. Models, Workloads, and Hardware

We evaluate Phi-3-mini-4k-instruct [34] and Qwen2.5-1.5B-Instruct [35] on GSM8K [36] and SciQ [37]. Sampling is enabled with temperature 1.0 and top-p 0.95. The batched-scaling experiment runs Phi-3 on a V100 and Qwen on an A100, we interpret each model–GPU pair separately rather than compare their absolute systems values. The primary schedule and cross-node experiments run both models on A100-SXM4 80 GB GPUs with a 500 W power limit and a fixed 1275 MHz graphics clock. The SciQ experiments use V100 PCIe 32 GB GPUs with a 250 W power limit and a fixed 1230 MHz graphics clock. Each job reserves one GPU exclusively.

TABLE III
GSM8K BATCHED-SCALING CONFIGURATIONS.

Configuration	Phi-3	Qwen
GPU	V100 32 GB	A100 80 GB
Power limit	250 W	500 W
Driver / CUDA	580.159.04 / 12.1	580.159.04 / 12.1
PyTorch / Transformers	2.4.1 / 4.57.6	2.4.1 / 4.57.6
Prompts × repetitions	100 × 3	100 × 3
Maximum new tokens	512	512

D. Output-Length Validation

GSM8K uses a maximum output length of 512 tokens. In the 500-prompt accuracy sets, 2.0% of Phi-3 candidates and 5.25% of Qwen candidates reach this limit, with mean response lengths of 251 and 270 tokens, respectively. SciQ uses a 64-token limit. No Qwen candidates and 1.93% of Phi-3 candidates reach the limit, with mean response lengths of 1.71 and 3.46 tokens, respectively. These values confirm that SciQ provides a much shorter-output workload than GSM8K.

E. Statistical Analysis

Systems results are reported as mean±SD across three repetitions. P95 latency is computed within each repetition and then summarized across repetitions. Accuracy confidence intervals use 10,000 prompt-level bootstrap resamples, with paired resampling when comparing candidate budgets. For the fixed-N schedule study, ratios compare 8×1 with 1×8. Confidence intervals use a hierarchical paired bootstrap: repetitions are resampled first, followed by prompts within each selected repetition, while preserving the pairing between schedules. P95 latency is recomputed for each bootstrap sample. These confidence intervals describe variability within the primary jobs. The two additional A100 jobs are analyzed separately, and cross-node results are reported as the observed range across the three jobs.

V. Results

A. Accuracy Gains from Increasing N

Figure 2 shows how GSM8K accuracy changes as the candidate budget increases. As expected, generating more candidates improves accuracy. Phi-3 increases from 81.4% at $N = 1$ to 89.8% at $N = 8$, a gain of 8.4 pp with a 95% paired bootstrap interval of $[5.8, 11.2]$ pp. Qwen increases from 51.4% to 69.8%, a gain of 18.4 pp ($[14.8, 22.0]$ pp).

The identical accuracy at $N = 1$ and $N = 2$ comes from the tie-breaking rule. When the first two candidates disagree, each receives one vote and the first candidate is selected. Accuracy begins to increase at $N = 3$, when a majority can form.

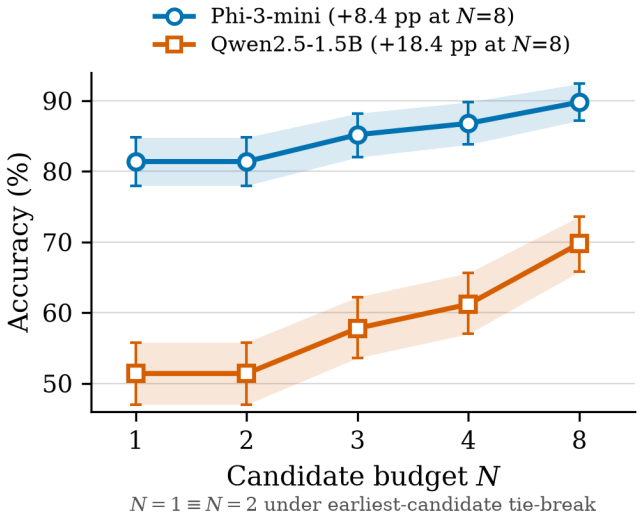

Fig. 2. GSM8K accuracy as the candidate budget increases. Error bars show 95% bootstrap intervals over 500 prompts.

Table IV checks whether tie handling or answer extraction explains the observed gains. Random tie-breaking changes expected accuracy by at most 1.4 pp, while conditioning on successful extraction also produces only small changes. We therefore use the original plurality-voting result as the main accuracy measure.

TABLE IV
TIE AND EXTRACTION DIAGNOSTICS. "RANDOM" USES UNIFORM RANDOM SELECTION AMONG TIED GROUPS. "COND." CONDITIONS ON A SUCCESSFULLY EXTRACTED SELECTED ANSWER.

Model	N	Tie (%)	Primary	Random	Cond.
Phi-3	2	25.0	81.4	81.9	82.1
	4	7.0	86.8	87.3	86.8
	8	3.4	89.8	90.1	89.8
Qwen	2	63.0	51.4	50.5	52.1
	4	31.4	61.2	61.3	61.8
	8	18.2	69.8	71.2	70.5

B. Systems Cost of Increasing Batched N

We next measure what happens to systems cost as the batched candidate budget increases from $N = 1$ to $N = 8$. Phi-3 and Qwen use different GPUs in this experiment, so their absolute values are not compared directly.

Table V highlights the main energy result. For both models, energy per query increases as more candidates are generated, while energy per generated token decreases. For Phi-3, energy increases from 631 to 1286 J/query, while energy per token decreases from 2.634 to 0.655 J. For Qwen, the corresponding values change from 596 to 975 J/query and from 2.222 to 0.459 J/token.

TABLE V
ENERGY COST AS THE BATCHED CANDIDATE BUDGET INCREASES. VALUES ARE MEAN±SD ACROSS THREE REPETITIONS. PHI-3 USES A V100 AND QWEN USES AN A100, SO ABSOLUTE VALUES ARE NOT COMPARED ACROSS MODELS.

N	Phi-3/V100		Qwen/A100	
	J/query	J/token	J/query	J/token
1	631 ± 19	2.634 ± 0.005	596 ± 13	2.222 ± 0.025
2	795 ± 15	1.607 ± 0.005	670 ± 19	1.265 ± 0.013
4	974 ± 36	1.004 ± 0.025	802 ± 6	0.758 ± 0.007
8	1286 ± 49	0.655 ± 0.019	975 ± 20	0.459 ± 0.011

Larger batches also increase token throughput, but total query latency still rises. From $N = 1$ to $N = 8$, mean latency increases from 5.06 to 8.91 s for Phi-3 and from 5.26 to 7.64 s for Qwen. Measured GPU-hours per 1,000 queries also increase from 1.41 to 2.47 and from 1.46 to 2.12, respectively. Thus, batching improves per-token efficiency, but generating more candidates still increases the total cost of a query.

C. Effect of Generation Schedule at Fixed $N = 8$

The previous experiment changes the candidate count. We now keep the candidate budget fixed at eight and change only how those candidates are grouped into generation calls. Mean logical generated-token volume varies by only 0.8% across Phi-3 schedules and 1.0% across Qwen schedules.

Table VI shows a clear trend. Splitting the same eight candidates across more calls increases both energy and P95 latency for both models. The intermediate schedules follow the same pattern, showing that the effect is gradual rather than appearing only at the fully serial endpoint.

TABLE VI
FIXED-$N = 8$ SCHEDULE SWEEP. VALUES ARE NORMALIZED TO THE 1×8 SCHEDULE. LOWER VALUES ARE BETTER.

	Relative energy		Relative P95 latency	
Schedule	Phi-3	Qwen	Phi-3	Qwen
1×8	1.00	1.00	1.00	1.00
2×4	1.63	1.66	1.81	1.97
4×2	2.71	2.86	3.21	3.57
8×1	4.64	4.86	5.77	6.12

At the fully serial endpoint, eight single-candidate calls use $4.64\times$ as much gross GPU-device energy as one eight-candidate call for Phi-3 (95% CI: $[4.48, 4.79]$) and $4.86\times$ as much for Qwen ($[4.71, 5.02]$). P95 latency reaches $5.77\times$ ($[5.38, 5.99]$) and $6.12\times$ ($[5.70, 6.75]$), while throughput falls to 16.7% and 17.9% of the batched baseline.

The practical cost is also visible in GPU time. For 1,000 queries, measured GPU time increases from 2.09 to 12.49 GPU-hours for Phi-3 and from 2.13 to 11.76 GPU-hours for Qwen. Lower average power does not remove this penalty. For example, Phi-3 mean power decreases from 177.8 W to 139.8 W, but mean latency increases by $5.97\times$, so total energy still increases.

These results give a simple practical guideline for the settings studied here: when candidates are independent and memory allows it, fewer generation calls with larger batch sizes are more efficient.

D. Cross-Node Robustness

We repeat the 1×8 and 8×1 endpoints in three independently scheduled A100 jobs per model. Table VII shows that the main effect remains stable across these jobs.

TABLE VII
OBSERVED $8 \times 1/1 \times 8$ RATIO RANGES ACROSS THREE INDEPENDENT A100 JOBS PER MODEL.

Metric	Phi-3	Qwen
Gross J/query ↓	4.43–4.64	4.85–4.88
Mean latency ↓	5.85–5.97	5.50–5.53
P95 latency ↓	5.53–5.77	6.06–6.12
Throughput retained ↑	16.7–17.1%	17.9–18.0%

The observed ranges are narrow compared with the size of the scheduling effect. Serial throughput remains about 17–18% of batched throughput in every job. The similar results across the three A100 jobs show that the scheduling effect is consistent across the tested nodes. However, we do not assume that the exact ratios will remain the same on other GPU architectures, clusters, or software environments.

E. Length-Stratified GSM8K Analysis

We next check whether the scheduling effect appears only for prompts that produce long responses. The 100 GSM8K systems prompts are divided into four groups using mean candidate length from the separate accuracy generation.

For Phi-3, Figure 3 shows energy ratios between 4.40 and 4.77 and latency ratios between 5.45 and 6.16. The ratios are not monotonic with response length.

Qwen shows the same general behavior. Figure 4 shows energy ratios between 4.51 and 5.09 and latency ratios between 5.15 and 5.88.

The scheduling penalty therefore appears across all four response-length groups rather than only for the longest outputs. This analysis is descriptive because each group contains only 25 prompts and the serial and batched schedules independently sample their candidates.

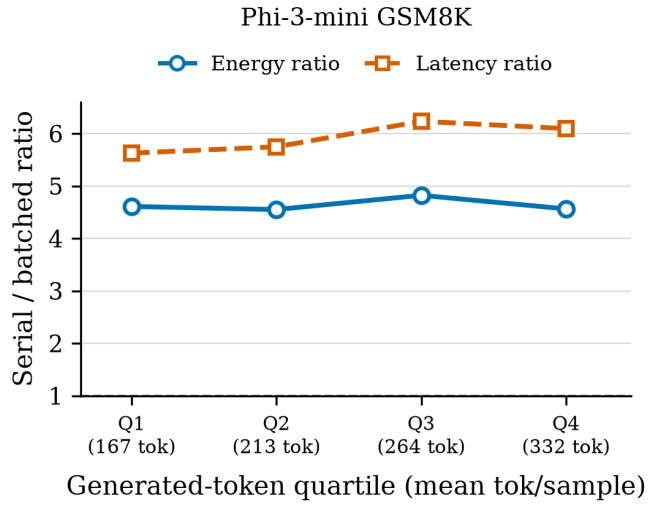

Fig. 3. Phi-3 serial-to-batched energy and latency ratios across GSM8K response-length groups. Each group contains 25 prompts.

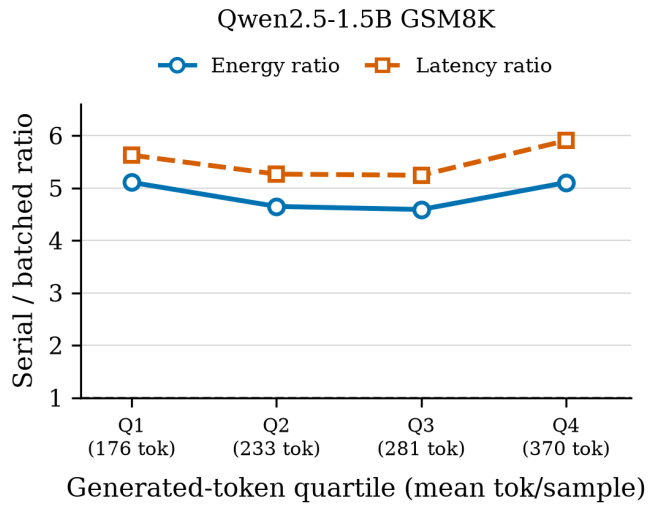

Fig. 4. Qwen serial-to-batched energy and latency ratios across GSM8K response-length groups. Each group contains 25 prompts.

F. Short-Output Validation on SciQ

GSM8K still contains reasoning-style outputs even in its shortest group. We therefore repeat the complete fixed-$N = 8$ schedule sweep on SciQ, where mean responses contain only a few generated tokens.

Table VIII shows the relative systems cost. The same trend appears for both models: energy and latency increase as the candidate budget is divided across more calls, while throughput decreases. From 1×8 to 8×1, gross energy increases by $2.57\times$ for Phi-3 and $3.34\times$ for Qwen. Mean latency increases by $2.88\times$ and $4.42\times$, while throughput falls to 36% and 23% of the batched baseline. Because SciQ uses V100 GPUs while the primary GSM8K study uses A100 GPUs, we do not claim that the difference in ratio size is caused only by output length. The supported conclusion is narrower: the fixed-N scheduling effect appears in both the longer-output GSM8K/A100 setting and the short-output SciQ/V100 setting.

TABLE VIII
SciQ fixed-$N = 8$ schedule sweep on V100 GPUs. Values are normalized to 1×8 within each model.

Model	Schedule	Rel. energy	Rel. latency	Rel. throughput
Phi-3	1×8	1.00	1.00	1.00
	2×4	1.29	1.37	0.74
	4×2	1.75	1.97	0.51
	8×1	2.57	2.88	0.36
Qwen	1×8	1.00	1.00	1.00
	2×4	1.38	1.59	0.63
	4×2	2.04	2.59	0.39
	8×1	3.34	4.42	0.23

VI. Discussion

Our results show that candidate count alone does not fully describe the systems cost of multi-candidate inference. For the same $N = 8$ budget, changing how candidates are grouped into generation calls produces large differences in latency, throughput, GPU-hours, and gross GPU-device energy. Across both models, dividing the candidate budget across more calls consistently increases systems cost. The same pattern remains across independently scheduled A100 jobs and also appears in the short-output SciQ/V100 setting. These results have a direct practical implication: when candidates are independent and memory allows it, fewer generation calls with larger batches are more efficient. This does not mean that $1 \times N$ is always optimal. Memory limits, request dependencies, continuous batching, and serving constraints may require a different schedule.

The effect can become substantial at scale. Under the configurations studied here, applying the measured per-query difference to one million queries would add approximately 1.37 MWh of gross GPU-device energy and 10,401 measured GPU-hours for Phi-3 when moving from 1×8 to 8×1. For Qwen, the corresponding differences are approximately 1.00 MWh and 9,631 GPU-hours. These values are linear illustrations of the measured configurations, not projections to other deployments.

Multi-candidate evaluations should therefore report not only N, but also the number of generation calls, candidates per call, batching mode, latency, throughput, GPU-hours, and the energy-measurement boundary. Two experiments with the same model, dataset, decoding settings, and candidate count can otherwise have substantially different systems costs simply because their candidate-generation schedules differ. Reporting these details makes systems results easier to reproduce and compare.

A. Why Generation Schedule Changes Systems Cost

Keeping N fixed keeps the candidate budget unchanged, but it does not keep the execution pattern unchanged. In a larger batch, several candidates can make progress within the same generation call. When the same candidates are split across smaller calls, less work is exposed to the GPU at the same time, and the query must pass through more generation

TABLE IX
PRACTICAL GUIDANCE FOR SELECTING A CANDIDATE-GENERATION SCHEDULE.

Situation	Suggested approach
All candidates fit in memory	Use one generation call with batch size N
Memory limits batch size	Use the largest feasible batch and the fewest calls
Candidates depend on previous outputs	Sequential calls may be required
Continuous-serving environment	Coordinate with the serving scheduler
Additional latency or resource limits	Choose a feasible schedule; among feasible options, prefer fewer calls

calls before completion. Several factors can contribute to the additional cost. More calls can add per-call framework and synchronization overhead and can repeat work associated with processing the prompt and starting generation. Smaller calls can also provide less parallel work to the GPU. Together, these effects can increase execution time even though the total number of requested candidates remains the same. The energy results also show why average power alone is not enough to judge efficiency. A smaller or more serial workload may draw less power at a given moment, but it can keep the GPU active for much longer. In our Phi-3/A100 measurements, mean power decreases from 177.8 W for 1×8 to 139.8 W for 8×1, while mean latency increases by $5.97\times$. The longer execution time outweighs the lower average power, leading to substantially higher total energy. These measurements capture the combined end-to-end effect of the generation schedule. They do not isolate how much of the difference comes from repeated prompt processing, call overhead, synchronization, GPU utilization, or other low-level effects. Separating these mechanisms would require more detailed profiling and is an important direction for future work.

B. Practical Schedule Selection

Our results suggest that candidate generation can be treated as a scheduling problem rather than specified only by the candidate count N. For independent candidates, the main execution choice is how many candidates to place in each generation call. A simple heuristic is to use the largest feasible batch size. Let $b_{\max}$ be the largest number of candidates that fits the available GPU memory and other system constraints. We select

$$b = \min(N, b_{\max}), \qquad C = \left\lceil \frac{N}{b} \right\rceil,$$

where b is the maximum number of candidates per call and C is the number of generation calls. If N is not divisible by b, the final call contains the remaining candidates. This policy minimizes the number of calls while keeping the candidate budget fixed. For example, if all eight candidates fit in memory, our results favor 1×8. If only four candidates fit at once, 2×4 is preferred over schedules with more, smaller calls. The same rule naturally extends to larger candidate budgets.

This heuristic is a starting point rather than a universal optimizer. The best schedule may change with model size, sequence length, available memory, concurrent traffic, and the inference engine. A more general scheduler could predict the energy, latency, and memory cost of candidate schedules and select the lowest-cost configuration that satisfies the system constraints.

VII. LIMITATIONS AND FUTURE WORK

Our experiments cover two models, two GPU architectures, and batch-scheduled Hugging Face generation. The measured ratios should therefore not be assumed to hold unchanged for larger models, other GPU generations, continuous batching systems, or multi-GPU inference. The fixed-N schedules use independently sampled candidates. Although their candidate counts are identical and their logical generated-token volumes are closely matched, we do not isolate the contribution of individual mechanisms such as repeated call overhead, prompt processing, synchronization, or batching effects. The results should therefore be interpreted as the end-to-end cost of each generation schedule. Energy measurements represent gross GPU-device energy over the measured query interval. They do not include whole-node energy and do not subtract idle GPU power. Future work can extend the study to larger models, continuous batching, quantization, speculative decoding, and multi-GPU execution.

VIII. CONCLUSION

Candidate count N tells us how many responses are generated, but not how they are executed. Our experiments show that this execution choice can substantially change the systems cost of multi-candidate inference. At fixed $N = 8$, splitting candidates across more generation calls consistently increases latency, GPU-hours, and gross GPU-device energy while reducing throughput. On A100 GPUs, 8×1 uses $4.64\times$ as much gross GPU-device energy as 1×8 for Phi-3 and $4.86\times$ as much for Qwen. The same scheduling pattern remains across independent A100 jobs and also appears in the short-output SciQ/V100 setting. For independent candidates, our results support a practical rule: when memory allows it, use fewer generation calls with larger batch sizes. Multi-candidate evaluations should report generation schedule and systems metrics alongside candidate count so that their cost can be reproduced and compared.

ACKNOWLEDGMENT

This project was supported by the National Science Foundation under grant #2211982. This work utilized the Nova high-performance computing cluster at Iowa State University and the Delta system at the National Center for Supercomputing Applications (NCSA) through allocation CIS240855. We acknowledge computational support from the Iowa State University Research IT Unit. Some of the Nova HPC equipment was purchased through funding provided by the National Science Foundation under MRI grant 2018594. We also acknowledge support for ACCESS through U.S. National Science Foundation grants 2138259, 2138286, 2138307, 2137603, and 2138296.

References

[1] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain-of-thought reasoning in language models," in *Proc. International Conference on Learning Representations*, 2023.

[2] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM test-time compute optimally can be more effective than scaling model parameters," arXiv preprint arXiv:2408.03314, 2024.

[3] Q. Zhang, F. Lyu, Z. Sun, L. Wang, W. Zhang, W. Hua, H. Wu, Z. Guo, Y. Wang, N. Muennighoff, and I. King, "A survey on test-time scaling in large language models: What, how, where, and how well?" arXiv preprint arXiv:2503.24235, 2025.

[4] N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. B. Hashimoto, "s1: Simple test-time scaling," in *Proc. Conference on Empirical Methods in Natural Language Processing*, 2025, pp. 20286–20332.

[5] S. S. Ghosal, S. Chakraborty, A. Reddy, Y. Lu, M. Wang, D. Manocha, F. Huang, M. Ghavamzadeh, and A. S. Bedi, "Does thinking more always help? Mirage of test-time scaling in reasoning models," in *Advances in Neural Information Processing Systems*, vol. 38, pp. 172664–172691, 2026.

[6] Z. Kang, X. Zhao, and D. Song, "Scalable best-of-N selection for large language models via self-certainty," in *Advances in Neural Information Processing Systems*, vol. 38, pp. 19720–19745, 2026.

[7] P. Aggarwal, A. Madaan, Y. Yang, and Mausam, "Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs," in *Proc. Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 12375–12396.

[8] X. Chen, R. Aksitov, U. Alon, J. Ren, K. Xiao, P. Yin, S. Prakash, C. Sutton, X. Wang, and D. Zhou, "Universal self-consistency for large language model generation," arXiv preprint arXiv:2311.17311, 2023.

[9] L. Chen, J. Davis, B. Hanin, P. Bailis, I. Stoica, M. Zaharia, and J. Zou, "Are more LLM calls all you need? Towards the scaling properties of compound AI systems," in *Advances in Neural Information Processing Systems*, vol. 37, pp. 45767–45790, 2024.

[10] X. Wang, S. Feng, Y. Li, P. Yuan, Y. Zhang, C. Tan, B. Pan, Y. Hu, and K. 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.

[11] D. Ding, A. Mallick, S. Zhang, C. Wang, D. Madrigal, M. D. Garcia, M. Xia, L. V. Lakshmanan, Q. Wu, and V. Rühle, "BEST-Route: Adaptive LLM routing with test-time optimal compute," arXiv preprint arXiv:2506.22716, 2025.

[12] J. Kim, N. Yang, K. Min, and K. 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.

[13] N. Iwase, Y. Ichihara, M. A. Quamar, and J. Komiyama, "Reliable chain-of-thought via prefix consistency," arXiv preprint arXiv:2605.07654, 2026.

[14] M. Kashaniyan and A. Jannesari, "Interpretable adaptive sampling for LLM test-time scaling," arXiv preprint arXiv:2608.03961, 2026.

[15] Z. Zheng, X. Ren, F. Xue, Y. Luo, X. Jiang, and Y. You, "Response length perception and sequence scheduling: An LLM-empowered LLM inference pipeline," in *Advances in Neural Information Processing Systems*, vol. 36, pp. 65517–65530, 2023.

[16] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari, "LLM inference serving: Survey of recent advances and opportunities," in *Proc. IEEE High Performance Extreme Computing Conference*, 2024, pp. 1–8.

[17] G. Brunetta, V. Sastry, X. Wu, V. Taylor, M. Papka, and Z. Lan, "Beyond Throughput: Performance and Energy Insights of LLM Inference Across AI Accelerators," in *Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 970–982, 2026. doi: 10.1109/IPDPS65963.2026.00083.

[18] Z. Yang, Y. Yang, C. Zhao, Q. Guo, W. He, and W. Ji, "PerLLM: Personalized inference scheduling with edge-cloud collaboration for diverse LLM services," arXiv preprint arXiv:2405.14636, 2024.

[19] P. Jaillet, J. Jiang, K. Mellou, M. Molinaro, C. Podimata, and Z. Zhou, "Online scheduling for LLM inference with KV-cache constraints," arXiv preprint arXiv:2502.07115, 2025.

[20] J. Stojkovic, C. Zhang, I. Goiri, J. Torrellas, and E. Choukse, "DynaLLM: Designing LLM inference clusters for performance and energy efficiency," in *Proc. IEEE International Symposium on High Performance Computer Architecture*, 2025, pp. 1348–1362.

[21] N. Jegham, M. Abdelatti, C. Y. Koh, L. Elmoubarki, and A. Hendawi, "How hungry is AI? Benchmarking energy, water, and carbon footprint of LLM inference," arXiv preprint arXiv:2505.09598, 2025.

[22] G. Wilkins, S. Keshav, and R. Mortier, "Offline energy-optimal LLM serving: Workload-based energy models for LLM inference on heterogeneous systems," *ACM SIGENERGY Energy Informatics Review*, vol. 4, no. 5, pp. 113–119, 2024.

[23] M. Özcan, P. Wiesner, P. Weiß, and O. Kao, "Quantifying the energy consumption and carbon emissions of LLM inference via simulations," arXiv preprint arXiv:2507.11417, 2025.

[24] J. Stojkovic, E. Choukse, C. Zhang, I. Goiri, and J. Torrellas, "Towards greener LLMs: Bringing energy efficiency to the forefront of LLM inference," arXiv preprint arXiv:2403.20306, 2024.

[25] Y. Ding and T. Shi, "Sustainable LLM serving: Environmental implications, challenges, and opportunities," in *Proc. IEEE International Green and Sustainable Computing Conference*, 2024, pp. 37–38, doi: 10.1109/IGSC64514.2024.00016.

[26] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for transformer-based generative models," in *Proc. 16th USENIX Symposium on Operating Systems Design and Implementation*, 2022, pp. 521–538.

[27] W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in *Proc. ACM Symposium on Operating Systems Principles*, 2023.

[28] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. Gulavani, A. Tumanov, and R. Ramjee, "Taming the throughput–latency trade-off in LLM inference with Sarathi-Serve," in *Proc. 18th USENIX Symposium on Operating Systems Design and Implementation*, 2024, pp. 117–134.

[29] J. Fernandez, C. Na, V. Tiwari, Y. Bisk, S. Luccioni, and E. Strubell, "Energy considerations of large language model inference and efficiency optimizations," arXiv preprint arXiv:2504.17674, 2025.

[30] J. Delavande, R. Pierrard, and S. Luccioni, "Understanding efficiency: Quantization, batching, and serving strategies in LLM energy use," arXiv preprint arXiv:2601.22362, 2026.

[31] M. F. Argerich, J. Fürst, and M. Patiño-Martínez, "Watt Counts: An energy-aware benchmark for sustainable LLM inference on heterogeneous GPU architectures," arXiv preprint arXiv:2604.09048, 2026.

[32] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.

[33] NVIDIA, "NVIDIA Management Library API reference," 2026.

[34] M. Abdin et al., "Phi-3 technical report: A highly capable language model locally on your phone," arXiv preprint arXiv:2404.14219, 2024.

[35] A. Yang et al., "Qwen2.5 technical report," arXiv preprint arXiv:2412.15115, 2024.

[36] K. Cobbe et al., "Training verifiers to solve math word problems," arXiv preprint arXiv:2110.14168, 2021.

[37] J. Welbl, N. F. Liu, and M. Gardner, "Crowdsourcing multiple-choice science questions," in *Proc. 3rd Workshop on Noisy User-Generated Text*, 2017, pp. 94–106.

[38] B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini, "Large language monkeys: Scaling inference compute with repeated sampling," arXiv preprint arXiv:2407.21787, 2024.