Faster and Better LLMs via Latency-Aware Test-Time Scaling

Zili Wang[1,2*], **Tianyu Zhang**[3*], **Haoli Bai**[3], **Lu Hou**[3],
Xianzhi Yu[3], **Wulong Liu**[3], **Shiming Xiang**[1,2,†], **Lei Zhu**[3,†]
[1] School of Artificial Intelligence, University of Chinese Academy of Sciences, China
[2] MAIS, Institute of Automation, Chinese Academy of Sciences, China
[3] Huawei Noah's Ark Lab
wangzili2022@ia.ac.cn, {zhangtianyu59,zhulei168}@huawei.com

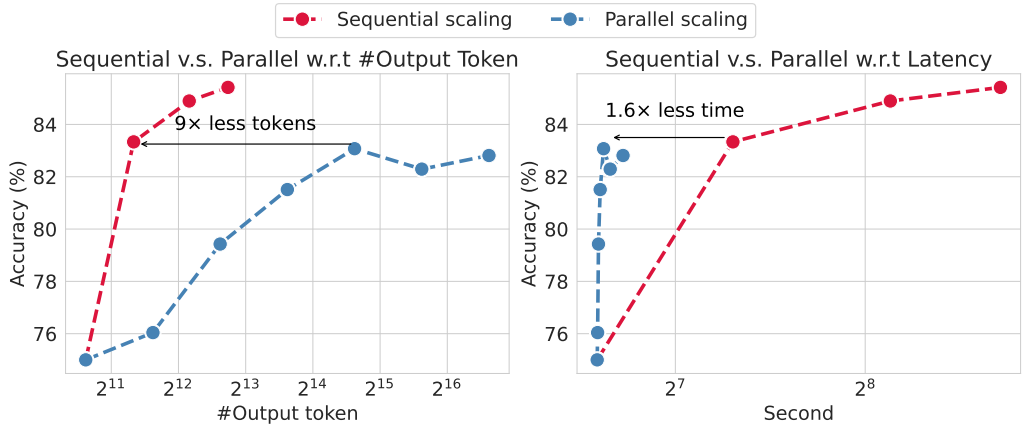

Figure 1: "Compute-optimal" does not necessarily translate to "latency-optimal" for test-time scaling. **Left:** Previous works measure test-time scaling by #token budget, indicating sequential scaling shows superior token efficiency than parallel scaling (majority voting for instance). **Right:** When considering *latency* as budget, parallel scaling can be 1.6x faster to achieve the same accuracy than sequential scaling. Experiments are performed using s1.1-32B (Muennighoff et al., 2025) on MATH-500 (Hendrycks et al., 2021). The red curve corresponds to sequential scaling with sequence length varying in {1024, 2048, 4096, 8192}, via budget forcing. The blue curve represents parallel scaling by majority voting with a fixed sequence length.

Abstract

Test-Time Scaling (TTS) has proven effective in improving the performance of Large Language Models (LLMs) during inference. However, existing research has overlooked the efficiency of TTS from a latency-sensitive perspective. Through a latency-aware evaluation of representative TTS methods, we demonstrate that a compute-optimal TTS does not always result in the lowest latency in scenarios where latency is critical. To address this gap and achieve latency-optimal TTS, we propose two key approaches by optimizing the concurrency configurations: (1) branch-wise parallelism, which leverages multiple concurrent inference branches, and (2) sequence-wise parallelism, enabled by speculative decoding. By integrating these two approaches and allocating computational resources properly to each, our latency-optimal TTS enables a 32B model to reach 82.3% accuracy on MATH-500 within 1 minute and a smaller 3B model to achieve 72.4% within 10 seconds. Our work emphasizes the importance of latency-aware TTS and demonstrates its ability to deliver both speed and accuracy in latency-sensitive scenarios.

1 Introduction

Test-Time Scaling (TTS) is an effective approach to improve the performance of Large Language Models (LLMs) at the cost of additional inference-time computations (Snell et al., 2024; Brown et al., 2024). TTS can be realized by two basic approaches: sequential scaling and parallel scaling. Sequential scaling requires the model to produce an extended reasoning process in a single pass (Muennighoff et al., 2025). In contrast, parallel scaling generates multiple solutions in parallel and selects the final answer, usually through majority voting (Wang et al., 2022; Liu et al., 2025). Hybrid approaches can be constructed on top of the two basic ones (Guan et al., 2025; Wang et al., 2024b).

With the number of generated tokens (#tokens) as budget, many existing studies (Snell et al., 2024; Setlur et al., 2025; Yang et al., 2025; Liu et al., 2025; Shi and Jin, 2025; Zhang et al., 2024) have examined compute-optimal strategies that enhance average performance gain per token, a metric we

[*] Equal Contributions.
[†] Corresponding Author.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 17124–17137
November 4-9, 2025

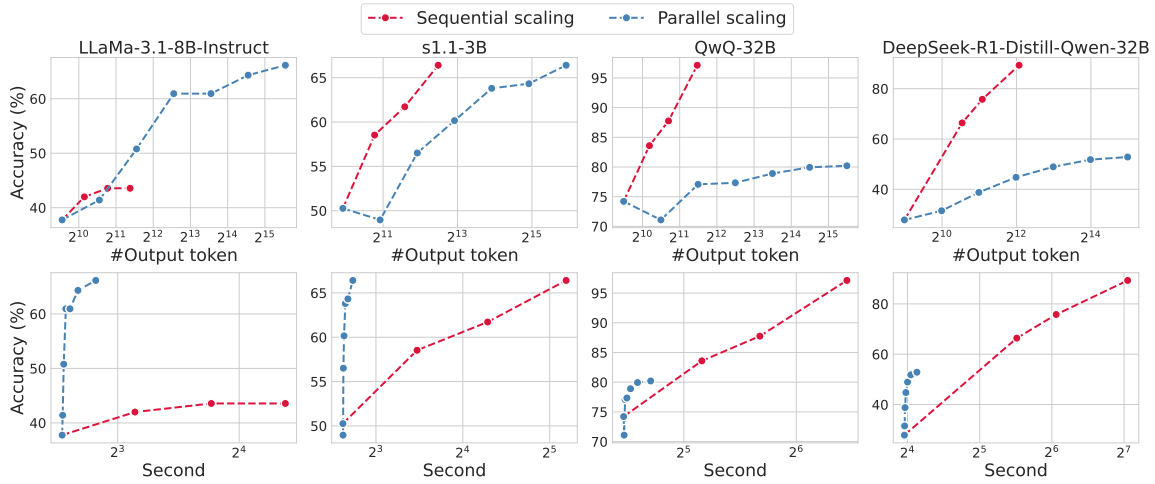

Figure 2: Latency-aware test-time scaling on MATH-500 with different model types, with sequential scaling in red and parallel scaling in blue.

refer to as *token efficiency*. However, in latency-sensitive scenarios where small batch sizes are employed, *e.g.*, personal computer, small-scale commercial deployment and edge device, *"compute-optimal" does not necessarily translate to "latency-optimal"*. This discrepancy is because the performance achieved within a limited time is determined by token efficiency as well as the throughput (*i.e.*, average number of output tokens per second). As shown in Figure 1, for s1.1-32B (Muennighoff et al., 2025) model, although sequential scaling has a better token efficiency, achieving a similar performance with 9× fewer tokens compared to parallel scaling, it turns out that parallel scaling achieves 1.6× lower latency. In fact, under small batch sizes, the time of an LLM autoregressive decoding step is dominated by the memory access of the parameters. Therefore, a moderately increased number of parallel branches incurs little additional latency, allowing a much higher throughput almost for free, as shown in Figure 3.

We then ask *how we can achieve latency-optimal test-time scaling*. One lesson from the observation above is that, in addition to optimizing token efficiency, attention must be given to improving the generation concurrency to chase a throughput. There are two approaches to improving concurrency: (1) branch-wise parallelism, which increases the number of parallel branches B, and (2) sequence-wise parallelism, which generates multiple successive tokens for a sequence in a single forward pass with speculative decoding (Leviathan et al., 2023; Chen et al., 2023). However, current research lacks a thorough analysis of how to allocate computational power to these two resource-competing approaches and to what extent they can be improved.

To bridge the gap, we examine the impact of concurrency configuration for latency-aware test-time scaling in this study. Specifically, we conduct experiments on representative datasets including MATH-500 (Hendrycks et al., 2021), AIME24 (AoPS, 2024), AIME25 (AoPS, 2025), and GPQA-Diamond (Rein et al., 2024) across model sizes from 3B to 32B and under varied concurrency configurations. Revealed by our experiments and analyses, the latency-optimal concurrency depends on the comparative advantage of token efficiency in sequential scaling and parallel scaling. For models with higher token efficiency with parallel scaling (*e.g.*, LLaMa-3.1-8B-Instruct (Meta, 2024)), one should prioritize branch-wise parallelism. Otherwise (*e.g.*, for QwQ-32B (Team, 2025)) sequence-wise parallelism takes priority. To determine the latency-optimal configuration, we propose a simple yet effective greedy search algorithm with less searching steps. Under 1 minute latency constraint, an optimal concurrency configuration can provide up to 7.3% better performance than baselines where only a single parallelism is applied.

Our contributions can be summarized as follows:

1. We introduce latency-aware test-time scaling, which considers the performance scaling under latency constraints. Unlike existing works that prominently only focus on token efficiency, we discover the necessity of taking systemic throughput into consideration for a

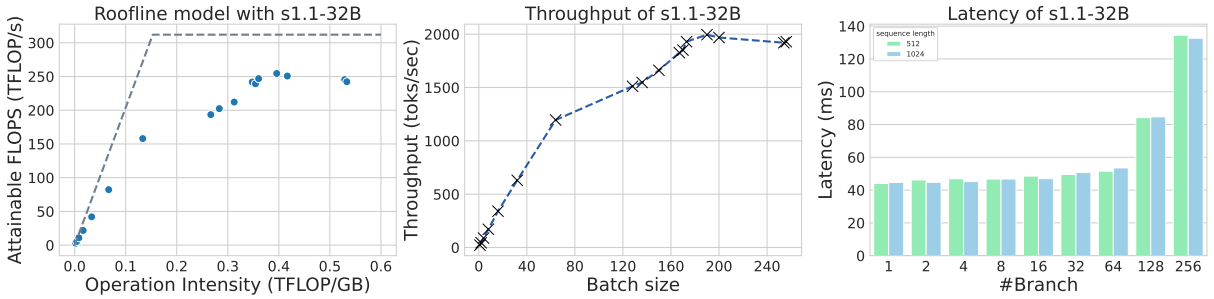

Figure 3: System state, latency, and throughput of s1.1-32B. **Left**: The roofline model with s1.1-32B. Increasing computational demand shifts execution from memory-bound to compute-bound. **Middle**: Throughput scales linearly with batch size before saturating at peak FLOPS. **Right**: The latency per forward pass under varying batch sizes.

better latency-performance trade-off.

2. We provide a unified view for parallel branches and speculative decoding from the perspective of generation concurrency. This allows us to frame latency-optimal test-time scaling as a resource allocation problem. A greedy search algorithm is proposed to search the latency-optimal configuration.

3. Through extensive experiments, we explore the optimal concurrency configuration for latency-aware test-time scaling. Our experiments reveal that for s1.1-32B, an optimal concurrency configuration can improve the accuracy by 7.3% while reducing latency by 1.7×, reaching 82.3% on MATH-500 in 1 minute.

2 Related Work

Test-time scaling Scaling the compute at inference time has been proven to be a prominent approach to improve LLM performance. Generally, test-time scaling methods fall into two main categories: **sequential** and **parallel** scaling. For the former, sequential scaling methods, represented by OpenAI's o1 (OpenAI, September 2024), DeepSeek-R1 (Guo et al., 2025), and Qwen QwQ (Team, 2025), enforce the model to generate longer solutions with a detailed reasoning chain. Such an ability can be incentivized with supervised training (Nye et al., 2021; Lee et al., 2025; Muennighoff et al., 2025) or reinforcement learning (Guo et al., 2025). Another main category is parallel scaling. This method first generates multiple response candidates in parallel, then applies a selection criterion to identify the best output. Recent research primarily uses token count as a metric to measure the budget of test-time scaling (Snell et al., 2024; Setlur et al., 2025; Yang et al., 2025). Recent work (Singhi et al., 2025) shows that majority voting outperforms verifier-based methods under low/medium token budgets. Notably, low latency necessitates limited token usage, where majority voting is superior. Thus, our work employs majority voting for parallel scaling. Different from previous works, our work points out that compute-optimal does not necessarily translate to latency-optimal. We introduce latency-aware test-time scaling and figure out the concurrency configuration to latency-optimal TTS.

Speculative decoding Speculative decoding uses a small draft model to generate several draft tokens autoregressively, and the target model to verify them in parallel while ensuring a lossless acceleration (Leviathan et al., 2023; Chen et al., 2023). Many studies focus on improving acceleration. EAGLE-series (Li et al., 2024b,a, 2025), Medusa (Cai et al., 2024), and Hydra (Ankner et al., 2024) employ features from the target model for better draft acceptance rate. HASS (Zhang et al., 2025) mitigates the inconsistency between training and inference by simulating the multi-step draft generation in the training phase. Hierarchical Drafting (Cho et al., 2025) improves the acceptance rate by regarding the hierarchical drafting strategy based on temporal locality. MoESD (Huang et al., 2025) shows that at medium batch sizes, MoE models with speculative decoding can surpass dense models. Our work demonstrates the feasibility of latency-aware TTS optimization through speculative decoding, presenting it as one viable approach to latency-optimal TTS.

3 Rethinking Test-Time Scaling with Latency

Previous works measure the budget of LLM test-time scaling by #tokens. However, it provides an incomplete evaluation, particularly when considering

real-world deployment constraints. Our investigation reveals the necessity to consider latency-aware TTS in latency-sensitive scenarios.

3.1 Latency-Aware Test-time Scaling

The inference of LLMs on modern accelerators is a memory-bound process, constrained by memory bandwidth. Nowadays, LLMs have billions of parameters (Bai et al., 2023; Yang et al., 2024; Team, 2025; Guo et al., 2025; Meta, 2024). In the autoregressive decoding, parameters (weights & KV caches) are loaded from memory (e.g., HBM on a GPU) into the compute units (e.g., SMs on a GPU) for matrix multiplication and other arithmetic operations. The runtime paid for the iteration is dominated by memory access. This is called the memory-bound nature of LLM. Under memory-bound constraints, slightly increasing the computation overhead does not affect the inference latency, but can increase throughput, as shown in Figure 3.

To characterize latency-aware test-time scaling under the memory-bound scenario, we explicitly consider two key factors:

Token efficiency is the ratio of task-specific accuracy improvement per token generated by the LLM, measured by %/token. A high token efficiency indicates that each token contributes significantly to achieving the accuracy. Prior work's approach to determining compute-optimal TTS using #token as budget fundamentally represents a search for maximal token efficiency.

Throughput is the number of output tokens generated per wall-clock time, measured by token/s. Note that the throughput may vary as the sequence length grows. A high throughput indicates efficient use of hardware resources and the system's capacity to quickly handle a large volume of tokens.

Consider s1.1-32B (Muennighoff et al., 2025) as an example. Figure 1 compares the accuracy-#token and accuracy-latency curves of sequential and parallel scaling. Sequential scaling achieves higher token efficiency ($83.3\%/2^{11.3}$toks) than parallel scaling ($83.3\%/2^{14.4}$toks), but Figure 3 reveals its throughput is $16\times$ lower. Consequently, sequential scaling requires $1.6\times$ more time to attain comparable accuracy. This suggests that while sequential scaling improves token efficiency, its inferior throughput hinders better accuracy in limited time. Thus, test-time scaling budgets must consider both token efficiency and throughput to achieve high accuracy with low latency.

3.2 How to Improve Latency-Aware TTS? A Concurrency Perspective

To improve latency-aware TTS, the key insight is to improve generation concurrency to increase throughput. To this end, there exist two approaches from the perspective of concurrency:

Branch-wise Parallelism. One approach is employing multiple concurrent branches B for the question, as shown in Figure 4 (b). For instance, when a 2048-token response fails to yield a correct answer, users can infer more branches to generate multiple responses of the same length and determine the final answer through majority voting. This approach harnesses more underutilized memory-bound computational resources, introducing almost no extra latency. Employing multiple branches to explore diverse reasoning paths brings further improvements of TTS performance.

Sequence-wise Parallelism Another effective approach is speculative decoding (Leviathan et al., 2023; Chen et al., 2023), as shown in Figure 4 (c). SD accelerates by verifying multiple draft tokens concurrently, leveraging underutilized memory-bound computational resources to mitigate the memory access burden in sequential generation with lossless performance. With SD, LLM can generate longer responses within a limited time, thereby enhancing TTS performance.

The combination of the two approaches is illustrated in Figure 4 (d). Branch-wise parallelism enhances performance without increasing latency, elevating the scaling curve. Sequence-wise parallelism reduces latency without compromising performance, causing the scaling curve to shift leftward. Their combined effect moves the scaling curve toward the upper-left quadrant.

3.3 Latency-Optimal Test-Time Scaling

The joint application of both approaches increases the overall concurrency and introduces additional computational overhead. When this overhead surpasses memory access overhead, the system transitions into a compute-bound state, leading to more latency. Besides, branch-wise parallelism exhibits diminishing improvement with increasing branches, while sequence-wise parallelism's acceleration reaches a maximum threshold. Consequently, an optimal boundary curve represents the latency-optimal test-time scaling, as shown by the green curve in Figure 4 (d). This curve defines

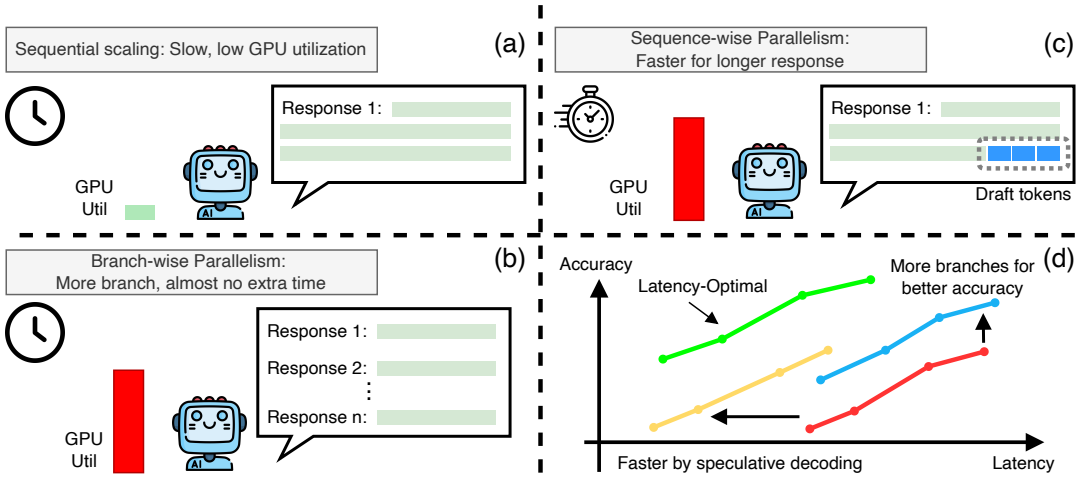

Figure 4: Overview of how to improve TTS with latency budget. (a): default sequential scaling suffers from long latency due to low GPU utilization. (b): Branch-wise parallelism employs more branches to utilize FLOPs, improving accuracy with no extra time. (c): Sequence-wise parallelism utilizes FLOPs with speculative decoding, reaching faster generation and theoretically no performance loss. (d): With speculative decoding, the curve (red) shifts to the left (yellow), indicating reduced latency. With more branches, the curve shifts upward (blue), indicating improved performance. Latency-optimal TTS can be achieved by jointly applying these two optimizations (green).

where neither latency nor accuracy can be further improved without concurrency trade-offs.

We aim to determine the latency-optimal TTS strategy by allocating concurrency resources with parallel branches B and draft length γ. Let $\text{Target}(\theta, T, x)$ be the output distribution over problem x produced by the LLM with test-time compute hyperparameter θ and time limitation T. The settings of branches and draft length are included by $\{B, \gamma\} \subsetneq \theta$. The latency-optimal test-time scaling is given by:

$$\theta^*_{x,y^*(x)}(T) = \arg\max_{\theta} \left(\mathbb{E}_{y \sim \text{Target}(\theta,T,x)} \left[\mathbb{1}_{y=y^*(x)} \right] \right), \quad (1)$$

where $y^*(x)$ indicates the groundtruth of corresponding problem x, and $\theta^*_{x,y^*(x)}(T)$ represents the test-time latency-optimal scaling strategy for problem x with time T. Finding the optimal $\theta^*_{x,y^*(x)}(T)$ is also the way to find optimal branches B^* and draft length γ^*.

Finding (B^*, γ^*) requires to search the configuration space. Although grid search can find the optimal configuration, it brings a significant cost to search the entire configuration space. Therefore, we introduce a simple yet effective greedy search algorithm, as shown by Algorithm 1.

In practice, B takes values in powers of 2 ($B = 2^k$), while γ increments in steps of 1. The basic grid search evaluates every (B, γ), with complexity of $O((\log B_{max} + 1) \times \gamma_{max})$. With greedy search, the complexity reduces to $O((\log B^* + 1) + \gamma^*)$, where B^* and γ^* denote the optimal configuration.

Algorithm 1 Greedy Search for Latency-Optimal TTS

Require: tts_task: TTS evaluation. T: latency budget. (B, γ): configuration; acc: accuracy

$B \leftarrow 1$
$\gamma \leftarrow 0$
$acc \leftarrow tts\_task(B, \gamma, T)$ ▷ Baseline accuracy
$end \leftarrow$ **False**
while not end **do**
  $acc_b \leftarrow tts\_task(2B, \gamma, T)$ ▷ Expand branch-wise
  $acc_\gamma \leftarrow tts\_task(B, \gamma + 1, T)$ ▷ Expand sequence-wise
  if $acc_b \geq acc_\gamma$ **then**
    $B_{new}, \gamma_{new}, acc_{new} \leftarrow 2B, \gamma, acc_b$
  else
    $B_{new}, \gamma_{new}, acc_{new} \leftarrow B, \gamma + 1, acc_\gamma$
  end if
  if $acc_{new} > acc$ **then**
    $B, \gamma, acc \leftarrow B_{new}, \gamma_{new}, acc_{new}$
  else
    $end \leftarrow$ **True**
  end if
end while
return $(B, \gamma), acc$

4 Experiments and Discussion

4.1 Experimental Setup

Tasks. Representative challenging tasks as benchmarks to measure the scaling property are selected: MATH-500 (Hendrycks et al., 2021) is a popular math benchmark comprising 500 high-school

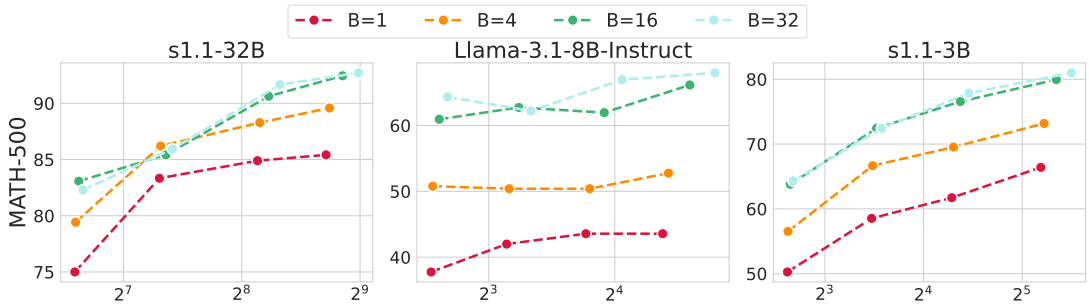

Figure 5: Latency-aware test-time scaling with different branches on s1.1-32B, LLama-3.1-8B-Instruct and s1.1-3B. Speculative decoding is not implemented. Each picture shows the scaling curve with varying branch numbers.

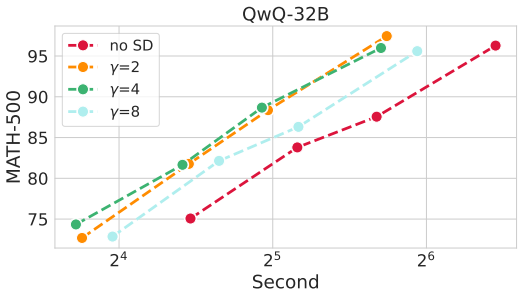

Figure 6: Latency-aware test-time scaling with speculative decoding of different draft length γ on MATH-500. Number of branch is fixed to 1. Draft model: DeepSeek-R1-Distill-Qwen-7B. Target model: QwQ-32B.

competition problems. AIME24 (AoPS, 2024) and AIME25 (AoPS, 2025) each consist of 30 math problems from the 2024 and 2025 American Invitational Mathematics Examination (AIME). GPQA-Diamond (Rein et al., 2024) consists of 198 science QA problems encompassing PhD-level physics, chemistry, and biology.

Models. We choose models with parameter sizes suitable for device-side deployment. Also, to implement speculative decoding, we select model types with draft models available. Therefore, considering its scalability, we employ s1.1-32B as our test-time scaling baseline. s1.1-7B is used as its draft model for speculative decoding. Also, we conduct relevant experiments on LLaMa-3.1-8B-Instruct (Meta, 2024) with Eagle3 (Li et al., 2025) as the draft model. For RL-based thinking model, we employ DeepSeek-R1-Distill-Qwen-32B (Guo et al., 2025) and QwQ-32B (Team, 2025) for their superior reasoning ability. The DeepSeek-R1-Distill-Qwen-7B is used as their draft model. We employ s1.1-3B for small-sized LLM for its outstanding model size and excellent performance with Qwen2.5-0.5B-Instruct (Bai et al., 2023) as the draft model. To better control the output sequence length, we employ budget forcing (Muennighoff et al., 2025) as an appropriate method to enforce the model to generate longer CoT. We use majority voting (Wang et al., 2022) to select the answer. Experiments are conducted on the codebase of OpenR (Wang et al., 2024a).

4.2 Sequential and Parallel Scaling under Latency-Aware TTS

Sequential scaling suffer from low throughput. As shown in Figure 2, sequential scaling achieves superior performance across most model types. However, its impact is less pronounced for models not explicitly trained for long CoT reasoning, such as LLaMa-3.1-8B-Instruct (Meta, 2024). Conversely, Figure 3 reveals that sequential scaling exhibits lower throughput. For instance, the s1.1-32B model processes only 22.7 tokens/s, translating to approximately 1,000 tokens per minute. While sequential scaling demonstrates better token efficiency, its lower throughput makes it less effective than parallel scaling under time constraints. Consequently, sequential scaling struggles to generate sufficient tokens to attain high accuracy.

Parallel scaling improves fast, but limited. In Figure 2, parallel scaling shows lower token efficiency than sequential scaling for most models except LLaMa-3.1-8B-Instruct. However, from Figure 3, increasing the branches of LLM inference hardly requires extra latency. Specifically, when increasing branches from 1 to 16, s1.1-32B gains nearly 10% improvement on MATH-500 at almost no extra latency cost. This can be attributed to the fact that the computation resources are fully utilized. By inference in branches, one autoregressive decoding procedure would generate the branches' tokens, but the latency cost of developing one token by sequential scaling is the same. So parallel

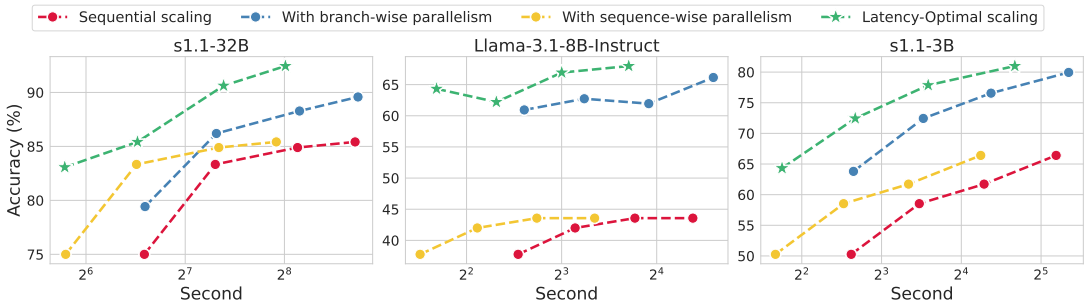

Figure 7: Latency-aware test-time scaling curves on MATH-500 with sequential scaling in red, parallel scaling in blue, and latency-optimal in green (the same to subsequent figures).

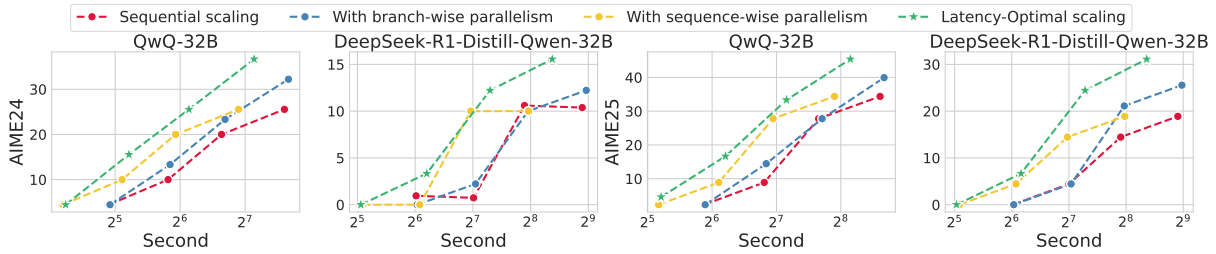

Figure 8: Latency-aware test-time scaling curves of QwQ-32B and DeepSeek-R1-Distill-Qwen-32B under latency on AIME24 (left two panels) and AIME25 (right two panels).

scaling can reach a larger #token faster. When the branches are increased to 64, the performance gain becomes slight, and the latency slightly grows because of the large amount of KV cache. Overall, parallel scaling maximizes hardware parallelism and scales output tokens simultaneously. But, parallel scaling often yields suboptimal accuracy due to branch redundancy and limited scalability.

Mostly, parallel scaling can surpass sequential scaling within limited time period. As shown in Figure 2, merely employing parallel scaling can surpass sequential scaling within a shorter, limited time. However, the extent of performance improvement varies depending on the model type. Specifically, for reasoning models like QwQ-32B, parallel scaling reaches 80.1% accuracy on the MATH-500 dataset within 30 seconds, but sequential scaling requires $1.4\times$ more time to achieve the comparable performance. Sequential scaling shows a slight performance gain for LLaMa-3.1-8B-Instruct, which is not designed for long CoT reasoning. While parallel scaling is still effective on MATH-500 since it explores more diverse solution paths, it shows obvious token efficiency, and sequential scaling cannot surpass it. Overall, parallel scaling can surpass sequential scaling w.r.t. latency, reaching a comparable accuracy within a relatively short time.

4.3 The Impact of Branch-wise Parallelism for Latency-Aware TTS

We conduct experiments on sequential scaling with varying branch sizes. In these configurations, we implement sequential scaling in parallel, with all parallel branches aggregated through majority voting. As shown in Figure 5, s1.1-32B demonstrates an initial upward trend in the TTS curve as the number of branches increases, indicating effective performance improvements. However, when the branch size grows excessively, the performance gains diminish, and latency increases slightly due to the non-negligible overhead of KV cache. In contrast, LLaMa-3.1-8B-Instruct exhibits minimal sequential scaling effects, resulting in a flatter curve compared to s1.1-32B and QwQ. Nevertheless, parallel scaling proves impactful, yielding an accuracy improvement that elevates the curve.

4.4 The Impact of Sequence-wise Parallelism for Latency-Aware TTS

Speculative decoding can accelerate model inference to some extent while preserving accuracy. As shown in Figure 6, most scaling curves with speculative decoding are on the left side of the baseline curve. As the speed-up ratio grows, the left-shifted trend becomes more obvious. However, when the draft length γ becomes large, the speed-up ratio

Strategy	1024		2048		4096		8192	
	Lat. (s)	Acc. (%)	Lat. (s)	Acc. (%)	Lat. (s)	Acc. (%)	Lat. (s)	Acc. (%)
Baseline	96.2±0.2	75.0±0.6	157.9±0.5	83.4±1.2	280.6±0.5	84.9±1.8	419.5±0.7	85.4±0.9
Bnh-wise	96.7±0.2	79.4±0.7	159.2±0.8	86.2±1.1	284.5±0.3	88.3±1.5	428.0±0.6	89.6±1.0
Seq-wise	55.5±0.4	75.0±0.3	91.1±0.3	83.4±1.9	161.9±0.3	84.9±1.4	241.9±0.8	85.4±0.1
Lat-Opt.	57.2±0.4	82.3±1.8	95.9±0.3	85.9±0.7	179.1±0.4	91.7±1.3	283.1±0.6	92.7±0.3
Improve	1.7×	7.3+	1.6×	2.5+	1.6×	6.8+	1.5×	7.3+

Table 1: Results of baseline, branch-wise parallelism, sequence-wise parallelism and latency-optimal TTS with different sequence lengths on MATH-500 of s1.1-32B. Lat.: Latency. Acc.: Accuracy. **Bnh-wise**: Branch-wise parallelism. **Seq-wise**: Sequence-wise parallelism. **Lat-Opt.**: Latency-Optimal. **Improve** is reported between **Baseline** and **Lat-Opt.** Results are obtained from 3 repeated experiments with mean and standard deviation reported.

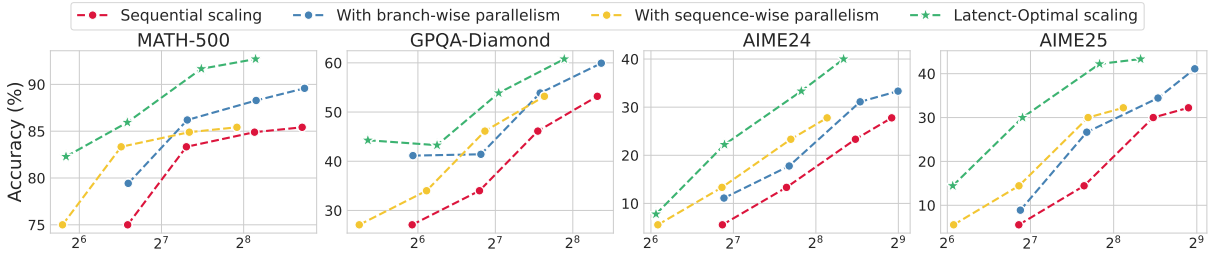

Figure 9: Latency-aware TTS curves of s1.1-32B on MATH-500, AIME24, AIME25 and GPQA-Diamond.

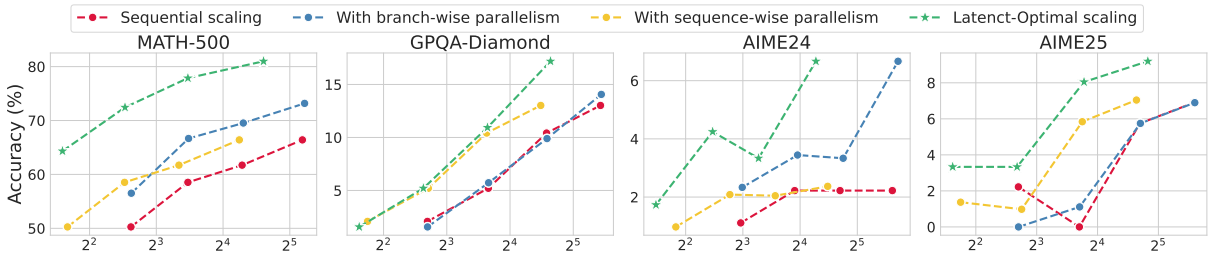

Figure 10: Latency-aware TTS curves of s1.1-3B on MATH-500, AIME24, AIME25 and GPQA-Diamond.

would decay, slowing down the overall speed. In our experiments, QwQ-32B is the target model, while DeepSeek-R1-Distill-Qwen-7B as the draft model, achieving a maximum speed-up ratio of 1.64×. Therefore, the scaling curve moves to the left first and then to the right. Sequential scaling with speculative decoding can achieve 7.5% higher accuracy than the baseline curve for a limited time. These results demonstrate that employing a proper draft length γ can push the inference speed to an optimal stage, enabling further sequential scaling within a limited time.

4.5 Latency-Optimal TTS with Branch-wise and Sequence-wise Parallelism

We conduct comprehensive experiments with various parallel branches and draft lengths to identify the concurrency configuration to achieve latency-optimal TTS. The results are shown in Figure 7 and 8. For LLMs that benefit primarily from sequential scaling, speculative decoding emerges as the dominant factor in the latency-optimal configuration. Conversely, for LLMs hat exhibit improvements from branches, increasing the number of branches yields huge performance gains. This difference stems from the different token efficiency. For reasoning models like QwQ, accuracy improvements are achieved through long CoT, making better SD acceleration more advantageous within limited time. In contrast, models like LLaMA-3.1-8B-Instruct, which do not benefit from long CoT, using more branches to expand the search paths is more efficient. See appendix for detailed configurations.

5 Results under Latency-Optimal TTS

5.1 Can LLM Solve the Problem in 1 minute?

Achieving as high accuracy as possible within a limited time holds significant practical value. We

Model	Grid Search			Greedy Search		
	(B^*, γ^*)	Accuracy	Steps	(B^*, γ^*)	Accuracy	Steps
s1.1-32B	$(16, 5)$	82.3%	56	$(16, 5)$	82.3%	10
s1.1-3B	$(32, 5)$	72.4%	56	$(32, 5)$	72.4%	11
DS-32B	$(16, 4)$	62.2%	56	$(16, 4)$	62.2%	9
QwQ-32B	$(4, 5)$	94.5%	56	$(4, 5)$	94.5%	8

Table 2: Comparison of Grid Search and Greedy Search for Different Models. DS-32B: DeepSeek-Distill-32B.

aim to find out how TTS can improve the accuracy individually on the device side within a relatively short time limitation, like 1 minute. To this end, we comprehensively evaluate s1.1-32B and s1.1-3B w.r.t. latency and record their inference latency.

From Figures 9 and 10, we derive the following observations: (1) Large model like s1.1-32B, achieves relatively high accuracy within just 1 minute, which is unattainable by the baseline. (2) Smaller model (s1.1-3B) attains notable accuracy in merely 10 seconds, showing significant potential. This trend persists across datasets spanning diverse domains, suggesting that popular models can be optimized using the latency-optimal TTS strategy.

5.2 How Can Latency-Optimal Improve Compared with Baseline?

Based on previous findings of latency-optimal TTS with different model types, branches, and draft lengths, we summarize the results in Table 1. For s1.1-32B on MATH-500, we find that latency-optimal TTS can achieve 6% accuracy improvement on average than merely using sequence-wise parallelism, and $1.6\times$ on average faster than branch-wise parallelism. Latency-optimal TTS outperforms the baseline in both accuracy and latency to a noticeable degree. However, increased branch and draft length causes LLMs to enter a compute-bound regime, where computational overhead exceeds memory access overhead. This eliminates the benefits of parallelism, leading to increased latency and sometimes even worse than the baseline. This suggests that a latency-optimal TTS strategy requires extremely fine-grained parameter tuning.

5.3 Effectiveness of Greedy Search

To validate the effectiveness of our proposed greedy search algorithm, extensive experiments are conducted across various models on MATH-500. The results in Table 2 show that our greedy search achieves the same optimal configuration as grid search, while significantly reducing searching steps (8–10 vs 56). These results show the practical efficiency of our greedy search algorithm.

Aggregation Method	1024	2048	4096	8192
	Acc. (%)	Acc. (%)	Acc. (%)	Acc. (%)
Majority Voting	82.3$_{\pm 1.8}$	85.9$_{\pm 0.7}$	91.7$_{\pm 1.3}$	92.7$_{\pm 0.3}$
Confidence min-max	79.2$_{\pm 1.6}$	82.2$_{\pm 1.5}$	88.6$_{\pm 1.4}$	90.1$_{\pm 1.3}$
Confidence avg-max	81.2$_{\pm 1.4}$	83.2$_{\pm 1.4}$	90.6$_{\pm 0.3}$	91.1$_{\pm 1.2}$
Confidence min-vote	82.4$_{\pm 1.3}$	85.2$_{\pm 1.5}$	91.5$_{\pm 1.6}$	92.7$_{\pm 0.4}$
Confidence avg-vote	**83.0**$_{\pm 1.2}$	**86.3**$_{\pm 1.3}$	**92.3**$_{\pm 1.2}$	**93.5**$_{\pm 0.8}$

Table 3: Performance comparison of majority voting and different confidence-based strategies. Acc.: Accuracy. Results are obtained from 3 repeated experiments with mean and standard deviation reported.

5.4 Different Aggregation Strategies

We conduct additional ablation experiments with s1.1-32B on MATH-500 comparing majority voting (self-consistency) with **confidence-based aggregation strategies** (Liu et al., 2025): first weight the answers by minimum (min) or average (avg) confidence, then select the answer with highest confidence (max) or accumulates the scores of all identical answers and then selects the answer with the highest score (vote). Results are shown in Table 3. Note that confidence-based strategies do not affect the latency. The results show that using confidence scores can indeed slightly improve output quality over simple majority voting.

6 Conclusion

In this paper, we propose to rethink test-time scaling in latency-sensitive scenarios. We show that compute-optimal does not always result in latency-optimal under such conditions due to memory-bound constraint. To address this, we propose to improve TTS on generation concurrency to maximize throughput from a unified view. Specifically, we present two approaches: (1) branch-wise parallelism via multiple branches and (2) sequence-wise parallelism by speculative decoding, along with their combinations. Furthermore, we investigate the concurrency allocation strategy to balance these approaches for latency-optimal TTS. A simple yet effective greedy search algorithm is proposed to determine the optimal configuration. Experimental results show that latency-optimal TTS enables 32B model to achieve 82.3% accuracy on MATH-500 within 1 minute, while 3B model attains notable accuracy within just 10 seconds, showing significant improvements in both speed and accuracy.

7 Limitations

Workload The basic concept of this paper is that the inference of LLM is a memory-bound process. However, this concept holds on small (like mobile phones, personal computers) or medium (like work stations) scale hardware. For large-scale servers, which deals with hundreds or thousands of requests at one time, the main bottleneck of inference is computation, and the budget of inference can be measured by #tokens. However, the studies on small and medium scale hardware still hold significant meaning, as the practical budget measured on these platforms is often under memory-bound scenarios.

8 Potential Risks

While our work shows test-time scaling under latency budget and our methods can significantly reduce inference latency, it introduces several risks. For instance, TTS could sometimes degrade performance on underrepresented data domains, exacerbating fairness issues. Scaled models may become more susceptible to adversarial prompts that exploit simplified decision pathways.

9 License For Artifacts

The artifacts utilized in this study, including datasets, codebase, and pre-trained models, are sourced from publicly available repositories under permissive licenses. All datasets adhere to open-access licenses (MIT License), ensuring compliance with redistribution and modification terms. Codebase adheres to open-access licenses (MIT License). For pre-trained models, we verify compatibility with Apache License 2.0. This alignment guarantees ethical reuse while maintaining transparency in our methodology.

10 Information About Use of AI Assistants

In the preparation of this work, we employ AI assistants to assist with refining academic language, and debugging code segments. The AI tools were used solely for improving clarity, grammatical correctness, and syntactic efficiency—tasks analogous to those performed by a human editor or linter. All conceptual contributions, technical claims, and critical analysis remain the authors' own.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. *Preprint*, arXiv:2402.05109.

AoPS. 2024. Aime 2024 dataset.

AoPS. 2025. Aime 2025 dataset.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv: 2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Sukmin Cho, Sangjin Choi, Taeho Hwang, Jeongyeon Seo, Soyeong Jeong, Huije Lee, Hoyun Song, Jong C Park, and Youngjin Kwon. 2025. Lossless acceleration of large language models with hierarchical drafting based on temporal locality in speculative decoding. *arXiv preprint arXiv:2502.05609*.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. 2025. Moesd: Unveil speculative decoding's potential for accelerating sparse moe. *arXiv preprint arXiv:2505.19645*.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. 2025. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Empirical Methods in Natural Language Processing*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *Preprint*, arXiv:2503.01840.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*.

Meta. 2024. Introducing llama 3.1: Our most capable models to date.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, and 1 others. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

OpenAI. September 2024. Learning to reason with llms.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. 2025. Scaling test-time compute without verification or rl is suboptimal. *arXiv preprint arXiv:2502.12118*.

Wenlei Shi and Xing Jin. 2025. Heimdall: test-time scaling on the generative verification. *arXiv preprint arXiv:2504.10337*.

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. 2025. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning. *arXiv preprint arXiv:2504.01005*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, and 1 others. 2024a. Openr: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024b. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. 2025. Towards thinking-optimal scaling of test-time compute for llm reasoning. *arXiv preprint arXiv:2502.18080*.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2025. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*.

A Appendix

A.1 Why does the parallel scaling curve exhibit a steeper slope than the sequential scaling?

This result can be attributed to the heavy model weights of LLM under the measurement of memory access. For instance, Qwen2.5-32B-Instruct has 32 billion parameters, so the model weights is 64GB. While the total KV cache at sequence length 1024 and batch size 1 is 0.25GB. Under this condition, KV cache only accounts for a small portion of memory access. When parallel scaling can easily extend branches with a small budget by 0.25GB per branch with 1024 length, sequential scaling tries hard to load the whole model weights of the entire 64GB just for 1 more extended token. This extreme imbalance on memory access makes parallel scaling significantly advantageous regarding memory access. In contrast, sequential scaling suffers from high memory access costs.

A.2 Additional Results

We present additional experimental results. The results on the influence of branch-wise parallelism is shown in Figure 11. The results of the influence of branch-wise parallelism under different speculative decoding configurations is shown in Figure 12. The results of sequence-wise parallelism under different branch counts are shown in Figure 13..

The acceptance rate α of the speculative decoding we employ is reported below: Target: s1.1-32B, draft: s1.1-7B, α: 0.831. Target: DeepSeek-R1-Distill-Qwen-32B, draft: DeepSeek-R1-Distill-Qwen-7B, α: 0.897. Target: QwQ-32B, draft: DeepSeek-R1-Distill-Qwen-7B, α: 0.781. Target: LLaMa-3.1-8B-Instruct, draft: Eagle3, α: 0.904. Target: s1.1-3B, draft: Qwen2.5-0.5B-Instruct, α: 0.701.

The latency-optimal configurations shown in Figure 7, 8, 9 and 10 are reported below: Figure 7: Left: B=16, γ=5. Right: B=32, γ=3. Right: B=32, γ=5. Figure 8: AIME24 with QwQ-32B: B=16, γ=4. AIME24 with DeepSeek-R1-Distill-Qwen-32B: B=32, γ=4. AIME25 with QwQ-32B: B=16, γ=4. AIME24 with DeepSeek-R1-Distill-Qwen-32B: B=32, γ=5. Figure 9: MATH-500: B=16, γ=5. GPQA-Diamond: B=32, γ=5. AIME24: B=32, γ=4. AIME25: B=32, γ=5. Figure 10: MATH-500: B=32, γ=5. GPQA-Diamond: B=16, γ=5. AIME24: B=8, γ=5. AIME25: B=16, γ=5.

A.3 Discussion on prefill phase

The whole inference time of LLM is composed of prefilling and decoding. While prefill time is indeed an important overhead of the overall inference, it is not the dominant factor in our case. For example, we have analyzed MATH-500 and find that the question length distribution is highly skewed towards shorter inputs: 88% of questions contain fewer than 360 tokens, and 99% contain fewer than 800 tokens. In contrast, our model generates responses up to 8,192 tokens in length. On modern GPUs, the prefill phase is significantly faster than the autoregressive decoding phase. Given the ratio of input and output lengths in our dataset, the prefill time is negligible of the total latency.

Therefore, while prefill time is technically part of the inference process, its contribution is small. Excluding it does not affect the conclusions.

A.4 Discussion on large batch, multi-requests scenario

We show the experiments as number of branches grows in the table below. The first column: model configuration. The second column: Accuracy. #Req: number of requests. Each cell contains the latency. With more requests, the workload transitions to a compute-bound state. At this point, latency depends on the FLOPs, which is directly proportional to the token count. Consequently, for our latency-aware strategy, once the workload transitions from memory-bound to compute-bound, the trade-off strategies can be directly informed by previous token count strategy.

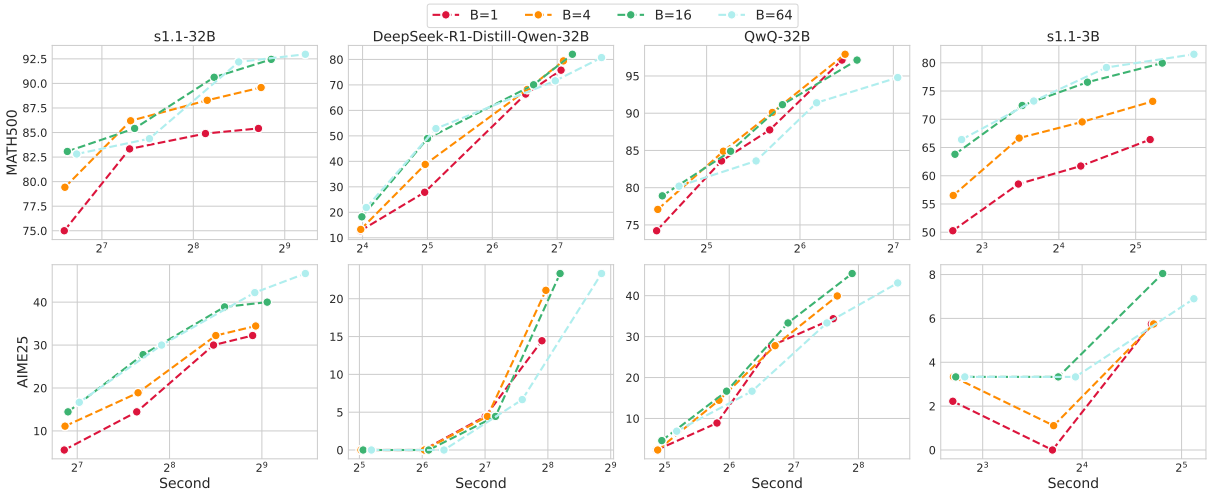

Figure 11: More results of the influence of the number of branches from branch-wise parallelism.

Models	Configuration	Accuracy	#Req		
			1	**4**	**16**
s1.1-32B	$B = 1, \gamma = 5$	82.3%	90.5s	91.2s	92.9s
s1.1-32B	$B = 4, \gamma = 5$	85.6%	91.3s	93.1s	156.8s
s1.1-32B	$B = 16, \gamma = 5$	83.8%	90.5s	142.7s	576.8s
s1.1-32B	$B = 64, \gamma = 5$	83.0%	159.8s	602.8s	2621.8s
Llama-3.1-8B-Instruct	$B = 1, \gamma = 4$	42.3%	7.6s	7.6s	8.2s
Llama-3.1-8B-Instruct	$B = 4, \gamma = 4$	47.6%	7.6s	8.3s	26.4s
Llama-3.1-8B-Instruct	$B = 16, \gamma = 4$	63.1%	8.1s	25.8s	95.8s
Llama-3.1-8B-Instruct	$B = 64, \gamma = 4$	64.9%	24.4s	91.2s	374.5s

Table 4: Latency under different configurations and number of requests (#Req). Memory-bound workloads are marked with gray. As the number of computed tokens per forward grows, the system transits from memory-bound to compute-bound. The latency of the latter can be measured by #tokens.

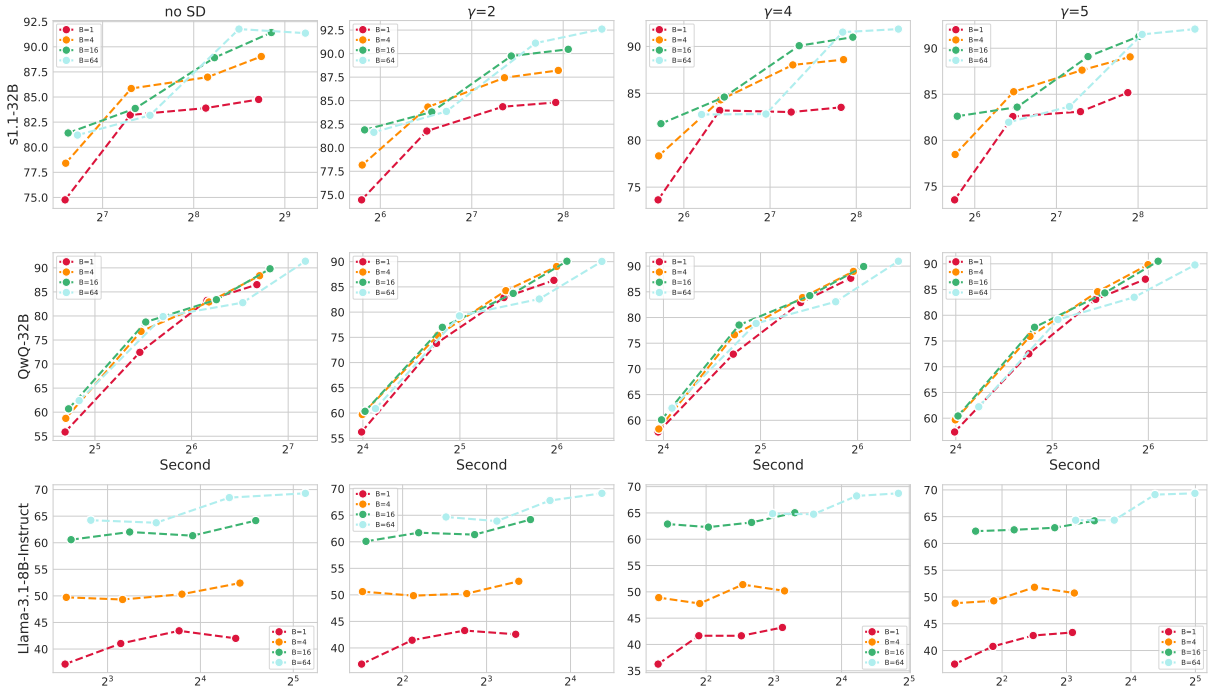

Figure 12: More results of the influence of the number of branches from branch-wise parallelism under different sequence-wise configurations. The draft lengths of speculative decoding varies among {2, 4, 5}.

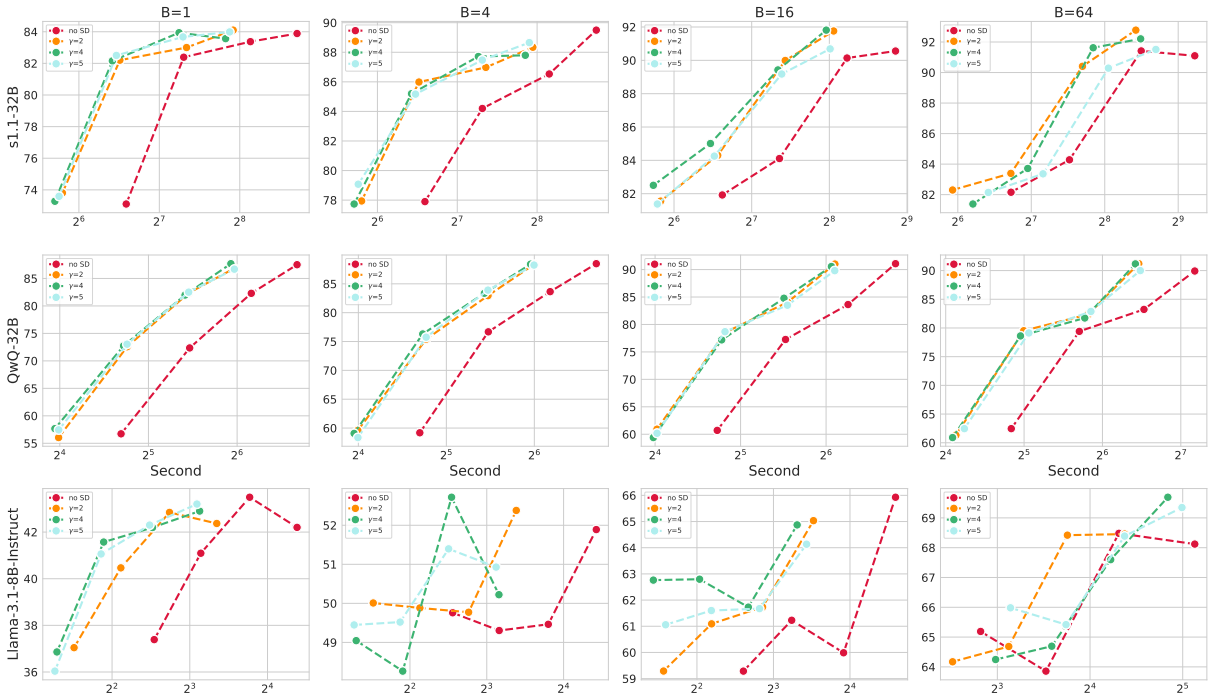

Figure 13: More results of the influence of draft length from sequence-wise parallelism under different branch-wise configurations. The number of branches varies among {1, 4, 16, 64}.