

RETHINKING THE HIGH-THROUGHPUT LLM INFERENCE: AN OPPORTUNITY FOR SPECULATIVE DECODING

Anonymous authors

Paper under double-blind review

ABSTRACT

Speculative decoding is a widely adopted method for accelerating autoregressive generation by drafting multiple candidate tokens and verifying them jointly with the target model. While effective in small-batch settings, it has been considered impractical under large-batch inference due to the belief that such regimes are compute-bound. Motivated by recent system-level findings that memory bandwidth, not compute, remains the dominant bottleneck in large-batch inference, we revisit the feasibility of speculative decoding under high-throughput conditions. We introduce γ -tolerance, a latency-based criterion that characterizes when speculative decoding provides tangible speedup, and empirically validate that acceleration remains attainable across practical batch sizes and system configurations. Building on this insight, we derive a revised success condition for speculative decoding and demonstrate that most existing drafter architectures violate it due to poor trade-offs between accuracy and efficiency. To address this, we identify Multi-Token Prediction with Gated LoRA as a promising approach and develop a high-performance implementation. Our system achieves up to $2.37\times$ speedup at batch size 256 without requiring long-context prompts or architectural changes to the target model, demonstrating that speculative decoding can be both feasible and effective in large-batch inference.

1 INTRODUCTION

The recent proliferation of powerful reasoning models, from proprietary systems like GPT-5 (OpenAI, 2025) and Gemini-2.5 (Comanici et al., 2025) to open-source alternatives such as Qwen3 (Yang et al., 2025) and DeepSeek-R1 family (Guo et al., 2025), is significantly expanding the applicability of large language models (LLMs). As these models enable a new class of complex applications, such as autonomous agents and sophisticated code generation (Ferrag et al., 2025; Prabhakar et al., 2025; Wei et al., 2025), the demand for serving infrastructure has grown significantly. Consequently, accelerating LLM inference has become more critical and urgent than ever before, as it is essential for both enhancing user experience and ensuring the economic viability of service providers.

Among various acceleration strategies, speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Li et al., 2024a) stands out as a particularly promising direction. It leverages a lightweight *drafter* model to propose multiple candidate tokens, which are then simultaneously verified by the larger *target* model in a single forward pass. In contrast to compression-based techniques such as quantization or pruning (Frantar et al., 2023; Sun et al., 2024), this method leaves the architecture and parameters of the target model entirely unchanged. As a result, it enables a form of *lossless acceleration* that preserves the model’s full capabilities, which is a critical requirement for high-fidelity applications involving complex inference or problem solving. However, despite this appealing property, speculative decoding has remained limited to small-batch inference (Cai et al., 2024; Li et al., 2024a; Xia et al., 2025), largely due to the prevailing assumption that large-batch settings are compute-bound, where the benefits of parallel verification are presumed negligible.

However, recent system-level analysis (Recasens et al., 2025) challenges the conventional belief that large-batch LLM inference is compute-bound, revealing that memory bandwidth remains the primary bottleneck even under well-optimized inference stacks. This finding undermines the core premise

that has long rendered speculative decoding ineffective in high-throughput scenarios, and reopens a fundamental question:

Can speculative decoding remain effective under high-throughput, large-batch inference?

To address this, we introduce a latency-based metric, γ -tolerance, which quantifies how the target model’s forward latency scales with the number of tokens verified in a single forward pass. This formalism provides a precise and actionable condition for determining when speculative decoding is feasible. Our empirical analysis shows that although speculative decoding remains theoretically feasible in large-batch inference, the available headroom for acceleration is significantly narrower than in small-batch settings.

Recognizing the limited margin for acceleration necessitates re-evaluating the design principles that have guided speculative decoding in small-batch settings. While previous methods relied on generating many candidate tokens at low cost and deferring accuracy to the target model, we demonstrate that such *breadth-based* strategies collapse under large-batch conditions, where the verification cost scales with input length and leads to disproportionately higher latency. Instead, acceleration hinges on *accuracy-driven* drafting, where fewer but more reliable candidates are proposed and efficiently verified.

Guided by this insight, we revisit widely adopted drafter architectures and evaluate their suitability under large-batch constraints. We find that most existing approaches fall short of the accuracy–efficiency trade-off required for effective acceleration in large-batch inference. In response, we identify an alternative strategy based on *Multi-Token Prediction with Gated LoRA* (Samragh et al., 2025). Although their primary objective was not speculative decoding, this method aligns well with the requirements of large-batch speculative decoding due to its high draft accuracy and integrated design. By incorporating it into a high-throughput inference engine, we demonstrate that speculative decoding can achieve substantial acceleration even at large batch sizes, without relying on favorable conditions such as long-context inputs. Our contributions are summarized as follows:

- We introduce γ -tolerance, a latency-based criterion that formalizes the feasibility of speculative decoding in large-batch regimes. Through an empirical study, we validate that non-trivial speedups remain attainable under realistic conditions.
- We derive a revised success formula for achieving acceleration in speculative decoding and demonstrate that most existing drafter architectures violate this condition, highlighting the limitations of breadth-oriented strategies in large-batch settings.
- We identify a promising approach based on Multi-Token Prediction with Gated LoRA, and develop a high-performance implementation that attains up to $2.37\times$ speedup at batch size 256, without requiring favorable conditions such as long-context prompts.

2 RELATED WORK

Speculative decoding. Speculative decoding accelerates autoregressive generation by having a lightweight drafter propose multiple candidate tokens that a target model verifies in a single forward pass. A large body of work has extensively explored a variety of drafter instantiations, including standalone draft models (Leviathan et al., 2023; Chen et al., 2023), predictive-head or multi-branch heads that attach lightweight predictors to intermediate layers (Cai et al., 2024; Ankner et al., 2024; Li et al., 2024a; Zhang et al., 2025; Li et al., 2025a), and self-speculative approaches that reuse pruned, quantized, or partially executed variants of the target (Elhoushi et al., 2024; Zhang et al., 2024; Sathukhan et al., 2025; Xia et al., 2025; Tiwari et al., 2025). Predictive-head methods are often paired with tree drafting (Miao et al., 2024) to compensate for lower draft accuracy (induced by ultra-lightweighting) by proposing many candidates.

More recently, large-batch regimes have been studied. TurboSpec (Liu et al., 2024) introduces a runtime controller for speculative decoding that adapts the number of proposed and verified tokens based on an online goodput objective, and evaluates the approach across various batch sizes and workloads. As such, it is best viewed as a system-level serving optimization rather than a redesign of the algorithm. MagicDec (Sathukhan et al., 2025) analyzes that speculative decoding can provide speedup at high throughput for long-context inference and proposes a drafting strategy that leverages

compressed KV cache. Because its mechanism is built on KV cache compression, the method is primarily effective in long-context regimes, and its gains do not directly generalize to short- or medium-context workloads. Accordingly, its scope is complementary to ours, which targets high-throughput large-batch inference without assuming long contexts.

LLM serving systems and kernels. High-throughput serving has advanced via continuous batching and memory-aware schedulers (e.g., vLLM, SGLang) (Kwon et al., 2023; Zheng et al., 2024), together with IO-aware attention kernels. FlashAttention and FlashAttention-2 reduce HBM traffic through tiling and fusion and improve throughput via better parallelism and work partitioning (Dao et al., 2022; Dao, 2024). FlashInfer (Ye et al., 2025) targets workloads with a customizable attention engine that supports composable or block-sparse KV formats, JIT-fused kernels, persistent kernels compatible with CUDA Graphs, and load-balanced scheduling; empirically, it reports lower inter-token latency and higher bandwidth utilization than FlashAttention baselines in representative decode workloads.

Recent system profiling revisits the view that large-batch decoding becomes compute-bound. Re-casens et al. (2025) use Nsight-based roofline and stall analysis to demonstrate that attention remains memory-bound at large batch sizes: DRAM bandwidth saturates, and stalls are dominated by memory; furthermore, higher matrix multiplication (MatMul) intensity does not shift the bottleneck. Comparing kernels (e.g., xFormers (Lefaudeux et al., 2022) versus FlashAttention) highlights kernel-dependent behavior: stall mix, cache use, and per-token latency scaling differ across kernels, indicating that kernel design and memory-access patterns materially affect effective bandwidth, even though decoding remains memory-bound at scale.

3 REVISITING THE FEASIBILITY OF SPECULATIVE DECODING FOR LARGE BATCH LLM INFERENCE

This section establishes the analytical and empirical basis for large batch speculative decoding. We first derive the necessary condition for speedup in terms of γ -tolerance, a measure of how target-model latency responds to longer verification inputs. We then present empirical measurements across batch sizes, sequence lengths, and attention kernels to evaluate whether this condition holds in practice.

3.1 THE NECESSARY CONDITION FOR SPECULATIVE DECODING

Speculative decoding accelerates autoregressive generation by employing a lightweight drafter to propose multiple future tokens that the target model verifies in a single pass, a process referred to as *parallel verification*. The efficiency of this mechanism is determined by how latency scales when the target model processes multiple tokens within a single forward pass. If latency remains nearly constant, several draft tokens can be verified with negligible additional cost, thereby reducing the number of sequential target invocations, which yields substantial acceleration. In contrast, if latency increases approximately in proportion to the number of tokens forwarded simultaneously, then verifying γ tokens in one pass incurs nearly the same cost as generating them individually, and the potential gain largely vanishes.

This contrast elucidates why speculative decoding has conventionally been regarded as effective in memory-bound regimes but ineffective in compute-bound regimes. In the former case, latency is dominated by memory transfer (e.g., parameter loading), so extending the verification length introduces minimal marginal cost. In the latter, arithmetic operations dominate, and the computational cost scales almost linearly with the number of tokens, eliminating the benefit of parallel verification. In essence, the decisive factor is not merely whether the system is broadly classified as memory- or compute-bound, but rather the extent to which latency amortization occurs as the verification length increases. Capturing this scaling behavior is therefore essential for determining when speculative decoding can provide practical acceleration.

We now formalize this decisive factor by introducing a latency-based metric, γ -tolerance, which quantifies how latency scales with verification length:

$$\tau(\gamma) := \frac{T_{\mathcal{T}}(1)}{T_{\mathcal{T}}(\gamma)},$$

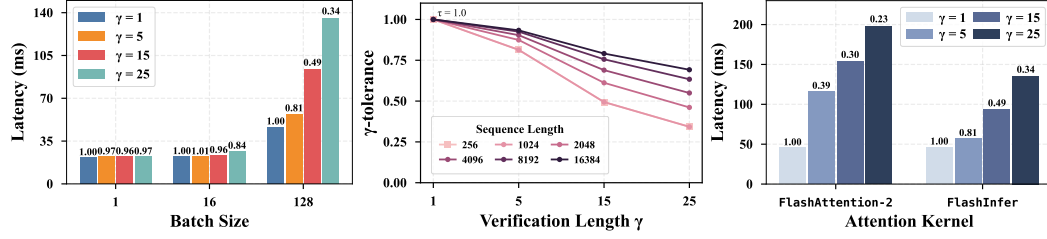


Figure 1: Empirical γ -tolerance across batch sizes, sequence lengths, and kernels. (Left): $\tau(\gamma)$ and forward latency for $\gamma \in \{1, 5, 15, 25\}$ at batch sizes 1, 16, and 128 (sequence length fixed to 1024). (Middle): $\tau(\gamma)$ as a function of past sequence length (KV cache size) at batch size 128. (Right): Comparison between FlashAttention-2 and FlashInfer in terms of latency scaling and $\tau(\gamma)$ at batch size 128 and sequence length 1024. Numbers above bars indicate $\tau(\gamma)$.

where $T_{\mathcal{T}}(\gamma)$ denotes the latency of the target forward pass with verification length γ . Intuitively, $\tau(\gamma)$ measures how slowly latency increases as the verification length grows. A value close to 1 indicates that latency is nearly invariant to input length (highly memory-bound), while a value near $1/\gamma$ corresponds to almost linear scaling (compute-bound). In this sense, $\tau(\gamma)$ provides a concrete, measurable proxy for the classical memory/compute distinction and serves as a direct indicator of speculative decoding’s potential.

With this notion, the condition for speedup can be precisely characterized. Let λ denote the expected number of tokens generated per speculative step (the number of accepted tokens plus the bonus token), and let $T_{\mathcal{D}}$ be the drafter’s per-step latency. Then, one speculative step costs $T_{\mathcal{T}}(\gamma + 1) + \gamma T_{\mathcal{D}}$, whereas sequential generation of λ tokens costs $\lambda T_{\mathcal{T}}(1)$. The resulting speedup is

$$S = \frac{\lambda T_{\mathcal{T}}(1)}{T_{\mathcal{T}}(\gamma + 1) + \gamma T_{\mathcal{D}}} = \frac{\lambda \tau(\gamma + 1)}{1 + \gamma \delta \tau(\gamma + 1)}, \quad (1)$$

where $\delta = T_{\mathcal{D}}/T_{\mathcal{T}}(1)$. In the idealized case $T_{\mathcal{D}} \rightarrow 0$, this simplifies to $S \approx \lambda \tau(\gamma + 1)$, implying that a necessary condition for acceleration is $\tau(\gamma + 1) > 1/\lambda$. As δ grows, the requirement becomes stricter. Thus, $\tau(\gamma)$ bridges system-level intuitions with a formal performance metric: it not only reflects whether a model is operating closer to the memory- or compute-bound regime, but also provides a direct and testable criterion for when speculative decoding can yield tangible acceleration.

3.2 EMPIRICAL VALIDATION OF THE NECESSARY CONDITION

We next evaluate whether the feasibility condition is realized in practice under large-batch inference. Measurements are conducted on Qwen3-8B (Yang et al., 2025) with a single NVIDIA H100 GPU, using 1000 forward passes and 100 warmup runs to remove compilation and CUDA graph effects. Unless otherwise specified, the FlashInfer (Ye et al., 2025) kernel is employed, as it is widely adopted by modern inference engines, including vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), and TensorRT-LLM (NVIDIA, 2025).

Scaling with batch size. Fig. 1 (left) shows that speculative decoding remains feasible even under large-batch inference, although the margin for acceleration narrows. For a chain draft of length 4 ($\gamma = 5$), we observe $\tau(5) = 0.81$ at batch size 128, corresponding to an upper bound of $S \approx 0.81\lambda$; with $\lambda \approx 3$ (approximately two accepted tokens plus the bonus token), this yields a $2.43\times$ theoretical upper bound of speedup. By contrast, deeper verification quickly undermines feasibility: at $\gamma = 25$, $\tau(25) = 0.34$, which caps the speedup near $1\times$ for $\lambda = 3$. More generally, as the batch size increases, the marginal latency per additional verified token rises, and the latency- γ curve steepens; consequently, $\tau(\gamma)$ declines with the batch size, shrinking the feasible region. For example, at $\gamma = 5$, τ moves from 0.97 (batch 1) to 0.81 and at $\gamma = 25$, it drops from 0.97 (batch 1) to 0.34 (batch 128). Hence, under large batches, shallow drafts can still be effective, but the necessary condition $\tau > 1/\lambda$ is satisfied only for relatively small γ , and the margin for attainable acceleration contracts as batch size grows.

Scaling with context length. Fig. 1 (middle) quantifies how past context modulates γ -tolerance at batch size 128. In our measurements, $\tau(\gamma)$ generally increases with sequence length and exhibits

sublinear growth that tends to saturate beyond roughly 1k to 4k tokens, consistent with a shift toward memory-dominated execution. This shift materially relaxes the feasibility condition: for shallow verification ($\gamma = 5$), $\tau(5)$ exceeds the $1/\lambda$ threshold for typical acceptance levels (e.g., $\lambda \in [2, 3]$) across much of the measured range, whereas for deep verification ($\gamma = 25$) the threshold is only approached and, under our setup, not surpassed. The ordering $\tau(1) > \tau(5) > \tau(15) > \tau(25)$ is preserved across lengths, but the gaps shrink as context grows, indicating that longer prefixes preferentially enhance larger- γ settings without fully closing the feasibility gap. Practically, workloads with sustained context (e.g., long-context) already operate in the regime where shallow drafts are justified under large-batch inference (Sadhukhan et al., 2025), while very deep drafts remain impractical even at the longest prefixes evaluated.

Kernel dependence. Fig. 1 (right) contrasts latency- γ scaling across kernels at batch size 128 and sequence length 1024. With FlashAttention-2 (Dao, 2024), forward latency rises sharply already between $\gamma=1$ and 5, depressing $\tau(\gamma)$ at small γ and making the condition $\tau > 1/\lambda$ difficult to satisfy. By contrast, FlashInfer exhibits milder growth over the same range, yielding consistently higher $\tau(\gamma)$ and preserving feasibility for shallow verification (e.g., $\gamma=5$). These differences are consistent with implementation-level effects—reduced launch overheads, more aggressive operator fusion, persistent execution, and cache-friendlier memory access patterns—leading to gentler latency- γ scaling under FlashInfer. Consequently, feasibility under large-batch inference is as much a property of the serving stack as of the model.

Importantly, this kernel sensitivity plausibly accounts for prior empirical reports that speculative decoding was ineffective at large batch sizes: when evaluated atop kernels with steeper latency- γ scaling (e.g., FlashAttention-2 in our measurements), $\tau(\gamma)$ at small γ often falls below the necessary threshold, yielding $S \leq 1$ even for shallow drafts; with kernels exhibiting gentler scaling (e.g., FlashInfer), the same configuration can satisfy the feasibility condition and produce speedup. This reconciles seemingly conflicting observations in the literature.

4 FROM POTENTIAL TO PRACTICAL ACCELERATION

Section 3 established that speculative decoding can remain feasible in large-batch LLM inference, but only under much stricter conditions than in small-batch settings. We now turn from feasibility to practicality: *under what circumstances can speculative decoding deliver tangible speedup, and how do existing drafter architectures align with these requirements?* This section refines the success formula for speculative decoding in the large-batch regime, evaluates representative drafter families against the updated criteria, and finally identifies a promising work that reconciles accuracy and efficiency.

4.1 REVISING THE SUCCESS FORMULA FOR LARGE-BATCH INFERENCE

In small-batch inference, the dominant recipe for achieving strong speedups has been to employ an extremely lightweight drafter, generate many candidates per step through tree drafting, and rely on the target model to filter them in a single verification pass. This strategy proved effective because verification cost was nearly invariant with respect to input length; in practice, $\tau(\gamma) \approx 1$ even for large γ , so the target model could verify dozens of draft tokens at almost the same latency as one. From the perspective of γ -tolerance, this explains why even low-accuracy drafters could deliver strong speedups: breadth was inexpensive, and speculative decoding could increase λ without incurring meaningful overhead. Formally, when $\tau(\gamma) \approx 1$, the speedup expression in equation 3.1 simplifies to $S \approx \frac{\lambda}{1+\gamma\delta}$. In the ideal case of $\delta \rightarrow 0$, this reduces to $S \approx \lambda$, meaning that larger trees (higher γ) directly increased speedup by raising λ . This is the basis for the breadth-oriented strategies that have dominated speculative decoding research in small-batch regimes.

However, this success formula does not carry over to large-batch inference. As shown in Section 3, γ -tolerance deteriorates rapidly with verification length: for example, $\tau(5)=0.81$ and $\tau(25)=0.34$ under realistic batch and sequence length configurations. This implies that the latency of the target model grows significantly with verification length, undermining the assumption that verification is nearly cost-free. Substituting such values into equation 3.1 illustrates the collapse: even with $\delta \rightarrow 0$, $\lambda=3$ and $\tau(25)=0.34$ yields $S \approx 1.0$, i.e., at best break-even rather than a gain. In practice, with

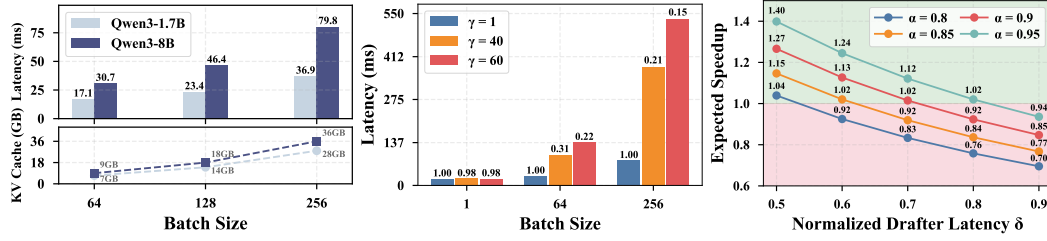


Figure 2: Empirical factors governing drafter viability under large-batch inference. (Left): Forward latency and KV cache footprint of a standalone drafter (Qwen3-1.7B) vs. target (Qwen3-8B) across batch sizes. (Middle): Effect of verification length and batch size on latency. (Right): Expected speedup S as a function of δ for several acceptance rates α (shaded green indicates $S > 1$).

non-negligible δ , the denominator grows further, making dense-tree drafting not just ineffective but counterproductive.

Accordingly, the success formula must be revised for large-batch inference. Feasible acceleration requires keeping γ small so that τ remains high, minimizing δ through efficient drafter design, and increasing λ via accurate, high-confidence predictions rather than by flooding the verifier with options. In other words, speculative decoding must shift from a breadth-oriented paradigm to an accuracy-driven one: real gains come not from generating many speculative paths, but from proposing a compact sequence that is both fast to verify and likely to be accepted.

4.2 EVALUATING EXISTING DRAFTER ARCHITECTURES

We evaluate three representative drafter families, standalone drafters, predictive-head drafters, and self-speculative methods, under the large-batch constraints identified in Sec. 4.1. Figure 2 summarizes empirical factors that map directly to the quantities in equation 3.1.

Standalone drafters. A smaller, independently deployed model avoids architectural coupling but incurs two penalties at large batch sizes (Fig. 2, left). First, the drafter’s forward latency remains a sizable fraction of the target’s: $\delta = 0.56$ (batch 64), 0.50 (128), and 0.46 (256). These values are far from the near-zero regime required for substantial speedup. Second, maintaining a separate KV cache consumes considerable VRAM that scales linearly with the batch size (e.g., $7 \rightarrow 28$ GB for 1.7B; $9 \rightarrow 36$ GB for 8B), directly constraining the usable context length and batch capacity. The combination of nontrivial δ and additional memory pressure renders the standalone design difficult to justify under large-batch inference, except possibly when pairing an extremely large target (e.g., Llama-3.1-405B (Grattafiori et al., 2024)) with a sufficiently smaller drafter.

Predictive-head drafters. Lightweight heads attached to the target model achieve very low δ , but their reliance on tree drafting is misaligned with large-batch constraints. Figure 2 (middle) shows that $\tau(\gamma)$ collapses as both γ and batch size increase. To mirror common tree budgets in prior predictive-head systems (Li et al., 2024b; 2025b), we evaluate $\gamma = 40$ and $\gamma = 60$. At batch 256, $\tau(40) = 0.21$ and $\tau(60) = 0.15$, versus 0.98 at batch 1. Thus, widening the tree to raise λ rapidly pushes verification into a regime where $S \leq 1$ even under optimistic acceptance. Moreover, extending tree drafting and verification to large-batch operation is non-trivial in practice, as noted by prior systems reports on productionizing speculative decoding and surveys of batching challenges (Tang et al., 2025; Ryu & Kim, 2024; Cai et al., 2024). That said, if predictive-head drafters attain sufficiently high draft accuracy to operate with *chain* drafting (small γ), their very low δ makes them the most promising route to large-batch acceleration.

Self-speculative decoding. Self-speculation reuses a partially executed variant of the target (via layer skipping, early exit, quantization, or KV cache modifications), which typically yields stronger alignment and higher acceptance than lightweight heads (Zhang et al., 2024; Elhoushi et al., 2024; Xia et al., 2025; Sadhukhan et al., 2025; Tiwari et al., 2025). The drawback is that shared computation keeps the δ high: a substantial fraction of the target’s path is still executed, so δ remains large, and even optimistic acceptance cannot compensate. Consistent with our sweep in Fig. 2 (right), when $\delta \in [0.5, 0.9]$ the system is at or near break-even unless acceptance is very high (e.g., at $\delta = 0.7$ one

needs acceptance $\alpha \geq 0.90$ just to reach $S \approx 1.02$), and achieving $S \geq 1.2$ generally requires $\delta \leq 0.6$ together with very high acceptance rate α .

Within this family, layer-skipping approaches reduce cost by selecting a subset of layers. SWIFT (Xia et al., 2025) reports that pushing δ toward ≈ 0.5 relies on *on-the-fly*, per-input layer-set optimization to adapt the skipped-layer configuration during inference, which complicates large-batch operation where homogeneous execution is essential. A separate line compresses or quantizes the draft KV cache (Sadhukhan et al., 2025; Tiwari et al., 2025), which is most effective in long-context regimes where KV traffic dominates. In aggregate, these characteristics make the δ range we evaluate (0.5-0.9) realistic for self-speculation, and they constrain large-batch viability: unless one can simultaneously drive δ low *and* maintain high acceptance without per-request adaptivity, substantial speedups are unlikely.

4.3 A PROMISING CANDIDATE FOR LARGE-BATCH SPECULATIVE DECODING

Motivated by these limitations, we investigate whether any existing mechanisms that were not originally developed for speculative decoding can nonetheless satisfy the revised criteria for large-batch acceleration. Through this investigation, we identify *Multi-Token Prediction with Gated LoRA* (Samragh et al., 2025) as a particularly promising candidate, combining architectural efficiency with high draft accuracy without relying on wide tree structures.

Multi-Token Prediction with Gated LoRA. Samragh et al. (2025) shows that LLMs can forecast a short horizon of future tokens by interleaving *regular* tokens x with special *mask* tokens m , enabling multi-token prediction (MTP) in a single forward pass. We adopt their speculative-decoding view. At each step, the input is formed by grouping every regular (anchor) token with k following masks so that each mask group predicts the next k futures conditioned on its immediately preceding anchor. For instance, with $k=2$ and anchors $(x_0, \hat{x}_1, \hat{x}_2)$, the input is $[x_0, m, m; \hat{x}_1, m, m; \hat{x}_2, m, m]$. Drafting and verification are co-located via *Gated LoRA*: in LoRA-enabled layers a binary gate g is applied per position, $Y = XW + (g \odot X)AB^\top$, so that mask positions ($g=1$) traverse the LoRA path to produce MTP logits, while anchor positions ($g=0$) use the base weights only, preserving verification fidelity to the original target model.

If \hat{x}_1 is accepted and \hat{x}_2 rejected, the verified prefix becomes $[x_0, \hat{x}_1]$ and the base path yields the *bonus* next token x_2 from \hat{x}_1 . The masks following the last accepted anchor (here, after \hat{x}_1) simultaneously provide the next-step drafts (namely, \hat{x}_3, \hat{x}_4), so the following step proceeds with anchors $[x_2, \hat{x}_3, \hat{x}_4]$ and the same layout. In general, each step builds an input of length $(k+1)^2$ (where $k+1$ anchors, each followed by k masks), hence both verification of current drafts and drafting of future proposals happen in a single forward pass.

Why Is It Suitable for Large-Batch Speculative Decoding? This approach exhibits a particularly favorable trade-off for large-batch inference. By collapsing verification and drafting into a single forward pass, it eliminates the need for auxiliary target variants or multiple sequential calls. Unlike conventional self-speculative decoding methods that rely on forwarding target variants multiple times, this method employs a unified model invocation with moderately longer input sequences, resulting in significantly improved drafting efficiency. Moreover, the drafts are generated directly from the full-capacity target model, yielding high acceptance rates and aligning well with the low-tolerance regime of large-batch inference. Taken together, these characteristics position this method as a strong candidate for satisfying the practical success formula of large-batch speculative decoding.

Our Implementation and Adaptation. While the original work demonstrates the conceptual feasibility of multi-token prediction with gated LoRA, it omits many key implementation details and does not release any code implementations. To operationalize this idea for speculative decoding, we develop a complete implementation of Gated LoRA MTP decoding, encompassing (i) a fine-tuning framework for multi-token prediction using gated LoRA, (ii) a verification pipeline aligned with speculative decoding semantics, and (iii) an inference path optimized for high-throughput generation. Our implementation is integrated into a high-performance decoding engine built on FlashInfer, enabling scalable inference with speculative acceleration under large-batch settings. To facilitate reproducibility, the implementation will be released upon acceptance.

Table 1: Acceleration performance results comparing MTP with Gated LoRA against the standard decoding. We report the estimated end-to-end speedup factor and goodput (tokens/s) in parentheses. The maximum sequence lengths are set at 32k for batch sizes 16 to 64, 24k for 128, and 16k for 256. For each configuration, the tensor parallel size is chosen to match VRAM requirements.

Strategy	B	Qwen3-8B			DeepSeek-R1-Distill-Llama-8B		
		AIME2025	CodeForces	GPQA-Diamond	AIME2025	CodeForces	GPQA-Diamond
Greedy	16	$\times 2.61$ (1088.3)	$\times 2.55$ (1057.4)	$\times 2.40$ (1003.3)	$\times 2.22$ (1094.4)	$\times 2.14$ (1056.0)	$\times 1.93$ (954.1)
	32	$\times 2.64$ (2031.2)	$\times 2.52$ (1975.2)	$\times 2.37$ (1865.2)	$\times 2.21$ (2043.6)	$\times 2.11$ (1953.9)	$\times 1.93$ (1796.0)
	64	$\times 2.66$ (3523.5)	$\times 2.51$ (3303.8)	$\times 2.40$ (3182.3)	$\times 2.32$ (3630.9)	$\times 2.21$ (3447.7)	$\times 2.06$ (3218.0)
	128	$\times 2.68$ (4799.0)	$\times 2.55$ (4545.4)	$\times 2.46$ (4427.1)	$\times 2.46$ (5051.4)	$\times 2.29$ (4756.9)	$\times 2.19$ (4573.9)
	256	$\times 2.37$ (6746.2)	$\times 2.22$ (6319.6)	$\times 2.20$ (6224.4)	$\times 2.15$ (7000.3)	$\times 2.02$ (6610.0)	$\times 1.98$ (6503.6)
Sampling	16	$\times 2.26$ (936.3)	$\times 2.11$ (871.1)	$\times 1.98$ (818.8)	$\times 1.98$ (977.8)	$\times 1.80$ (904.0)	$\times 1.68$ (833.4)
	32	$\times 2.31$ (1776.9)	$\times 2.08$ (1610.8)	$\times 1.90$ (1488.4)	$\times 1.99$ (1857.5)	$\times 1.81$ (1675.0)	$\times 1.70$ (1583.2)
	64	$\times 2.32$ (3026.8)	$\times 2.15$ (2809.0)	$\times 2.03$ (2648.4)	$\times 2.07$ (3206.3)	$\times 1.94$ (3004.6)	$\times 1.82$ (2834.7)
	128	$\times 2.38$ (4221.1)	$\times 2.14$ (3790.3)	$\times 2.09$ (3721.1)	$\times 2.22$ (4548.9)	$\times 2.06$ (4264.5)	$\times 2.01$ (4177.1)
	256	$\times 2.03$ (5730.2)	$\times 1.91$ (5367.4)	$\times 1.88$ (5314.6)	$\times 2.01$ (6484.4)	$\times 1.86$ (6064.3)	$\times 1.87$ (6098.0)

5 EVALUATIONS

Setup. We measure the acceleration performance of MTP with Gated LoRA, on Qwen3-8B (Yang et al., 2025) and DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025). Our custom inference engine, built on FlashInfer (Ye et al., 2025) kernels, incorporates standard optimizations such as weight fusion, tensor parallelism, and paged KV cache, using `bfloat16` precision. To assess performance across diverse domains, we use three datasets: AIME2025 (OpenCompass, 2025) (Math), CodeForces (Penedo et al., 2025) (Code Generation), and GPQA-Diamond (Rein et al., 2024) (Science). All measurements exclude prefill and initialization overhead. For speculative decoding, we report goodput (committed tokens/s), which is equivalent to throughput in standard decoding. In our experiments, we exclusively evaluate acceleration performance, since speculative decoding is theoretically guaranteed to preserve the output distribution of the target model.

Evaluation Protocol. To rigorously evaluate decoding throughput for long sequences (e.g., up to 32k tokens), we adopt a protocol designed to mitigate biases arising from variable generation lengths. Since throughput is highly dependent on the KV cache size, simply averaging over entire generations can be misleading. Instead, we measure throughput at various discrete prefix lengths, effectively capturing performance at different stages of the generation process. At each prefix length, decoding continues until 128 tokens are generated per sequence, and the process is repeated 10 times. We then aggregate these length-conditioned measurements to estimate the effective end-to-end throughput for a target sequence length. Full details of this protocol are provided in Appendix B, and full-sequence end-to-end measurements are reported separately in Appendix C.

5.1 MAIN RESULTS

Table 1 presents compelling evidence that speculative decoding with Gated LoRA MTP delivers substantial and consistent acceleration across a wide range of tasks and batch sizes. Throughput improvements persist across both Qwen3 and DeepSeek-R1 models, showing that the benefits are not tied to a particular backbone or implementation detail. This result bridges the gap between theoretical feasibility and practical deployment in real-world serving scenarios.

Closer inspection reveals three key patterns. First, greedy decoding consistently outperforms sampling due to longer average acceptance lengths, as deterministic drafts more closely align with top-1 predictions. Second, speedup varies across tasks; math shows the largest gains, which is likely due to domain alignment with the math-heavy training data, whereas code and science are underrepresented. This suggests that more balanced training or extended finetuning could further improve generalization. Third, speedup increases with batch size up to 128 as larger batches intensify KV cache pressure, amplifying the effectiveness of speculative decoding. The drop at batch size 256 is attributed to the fixed 16k sequence limit, which reduces overall memory traffic rather than exposing a method-level limitation.

5.2 IN-DEPTH ANALYSIS

Figure 3 (left) analyzes how draft length affects speedup across different batch sizes. While longer drafts generally yield better performance at small to medium batch sizes (e.g., batch 16 to 64), an

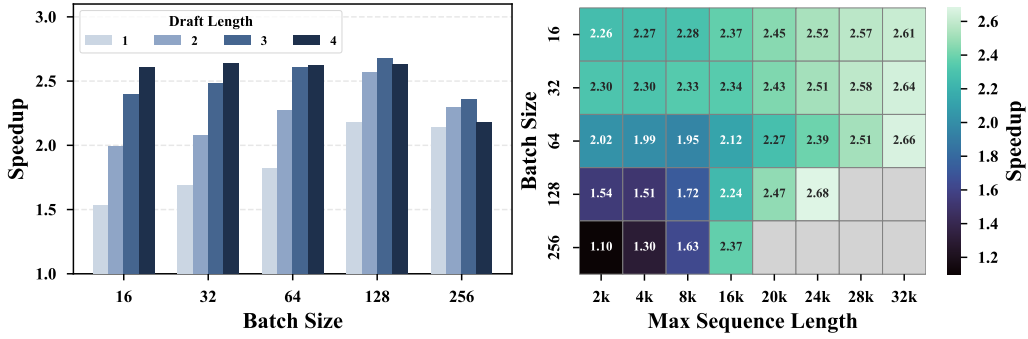


Figure 3: Ablations under greedy decoding for Qwen3-8B on AIME2025. (Left): Speedup vs. draft length $\in \{1, 2, 3, 4\}$ across batch sizes. (Right): Speedup as a function of batch size and maximum sequence length. Gray cells denote configurations omitted due to VRAM limits on H100 \times 8.

inverse pattern emerges at larger batch sizes. At batch 128 and 256, shorter drafts such as $k=2$ or $k=3$ actually outperform longer ones like $k=4$. This shift reflects the diminishing return of increasing draft length when the cost of forward passes grows rapidly with input length. As established in Section 3.1, larger batches significantly amplify the input-length sensitivity of decoding latency, meaning that the extra cost of verifying longer drafts can outweigh the gains from higher acceptance rates. Therefore, in high-throughput regimes, moderate draft lengths strike a better balance between parallel verification and overhead, confirming that optimal draft length is system-dependent and not monotonic with respect to speedup.

Figure 3 (right) explores how speculative decoding benefits scale with both batch size and maximum sequence length. Across all batch sizes, speedup improves steadily with longer generations, rising from 2.26 to 2.61 at batch size 16 and from 1.10 to 2.37 at batch size 256. This pattern highlights the role of KV cache pressure in inducing memory-bound behavior, making speculative decoding more effective for long-form generation tasks. Moreover, the benefits of increasing sequence length are amplified at higher batch sizes: at a batch size of 64, a 24k context is required to achieve a $2.39\times$ speedup, whereas a batch size of 256 achieves a comparable $2.37\times$ at just 16k. This suggests that larger batches naturally accumulate higher KV cache memory pressure, allowing them to enjoy the benefits of speculative decoding at shorter context lengths. These results not only validate our theoretical analysis but also offer actionable insights for practitioners: high speedup is most reliably obtained in regimes with large batch sizes and moderate-to-long generation lengths, especially when paired with appropriately tuned draft lengths.

6 CONCLUSION AND LIMITATIONS

In this work, we have revisited speculative decoding for high-throughput, large-batch LLM inference through the lens of γ -tolerance, a practical metric capturing how verification length affects target model latency. Our analysis showed that meaningful acceleration remains attainable when verification is shallow, drafter overhead δ is small, and the expected number of accepted tokens λ increases through accuracy rather than breadth. Guided by this understanding, we adopted Multi-Token Prediction with Gated LoRA, a method that aligns with large-batch constraints and delivered up to $2.37\times$ end-to-end speedup at batch size 256. This result provides a concrete path toward scalable speculative decoding, connecting system behavior with algorithmic design.

While encouraging, this work has several limitations. First, the evaluation focuses on reasoning-oriented models. Although the underlying principles are generally applicable, further validation on broader model families, including general-purpose LLMs, remains an open direction. Second, the method has not been evaluated under very short sequence lengths (e.g., under 1k tokens), which are less common in modern workloads and often do not require acceleration. Third, while kernel and batch configurations were explored, closer integration with runtime components such as schedulers and memory management could further improve $\tau(\gamma)$ and enhance overall performance. We hope these insights offer a foundation for future advances in scalable, high-throughput LLM inference.

ACKNOWLEDGEMENT ON LLM USAGE

The authors acknowledge the use of LLM for linguistic assistance, specifically for editing and clarifying the phrasing of sentences and paragraphs. LLM was not involved in any aspect of the research content, including conceptual development, methodological design, or experimental design.

REFERENCES

- Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=FbhjirzvJG>.
- Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *International Conference on Machine Learning*, pp. 5209–5235. PMLR, 2024.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.
- Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.
- Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=mZn2Xyh9Ec>.
- Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.
- Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.
- Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*, 2025.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=tcbBPnfwxS>.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL <https://arxiv.org/abs/2506.04178>.

- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeeFYf9>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.
- Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. <https://github.com/facebookresearch/xformers>, 2022.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/leviathan23a.html>.
- Jinze Li, Yixing Xu, Haiduo Huang, Xuanwu Yin, Dong Li, Edith C. H. Ngai, and Emad Barsoum. Gumiho: A hybrid architecture to prioritize early tokens in speculative decoding. In *Forty-second International Conference on Machine Learning*, 2025a. URL <https://openreview.net/forum?id=00bGn4e1IS>.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, pp. 28935–28948. PMLR, 2024a.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, 2024b.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025b.
- Xiaoxuan Liu, Cade Daniel, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Xiangxi Mo, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Optimizing speculative decoding for serving large language models using goodput. *arXiv preprint arXiv:2406.14066*, 2024.
- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>.
- Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS ’24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi: 10.1145/3620666.3651335. URL <https://doi.org/10.1145/3620666.3651335>.
- NVIDIA. Tensorrt-llm. <https://github.com/NVIDIA/TensorRT-LLM>, 2025. URL <https://github.com/NVIDIA/TensorRT-LLM>. Version v0.21.0. Accessed: September, 2025.

- OpenAI. Introducing gpt-5. <https://openai.com/index/introducing-gpt-5/>, 2025. Accessed: September, 2025.
- OpenCompass. Aime2025. <https://huggingface.co/datasets/opencompass/AIME2025>, 2025.
- Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. <https://huggingface.co/datasets/open-rl/codeforces>, 2025.
- Vignesh Prabhakar, Md Amirul Islam, Adam Atanas, Yao-Ting Wang, Joah Han, Aastha Jhunjhunwala, Rucha Apte, Robert Clark, Kang Xu, Zihan Wang, et al. Omniscience: A domain-specialized llm for scientific reasoning and discovery. *arXiv preprint arXiv:2503.17604*, 2025.
- Pol G Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll Berral. Mind the memory gap: Unveiling gpu bottlenecks in large-batch llm inference. In *IEEE International Conference on Cloud Computing*, 2025.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=Ti67584b98>.
- Hyun Ryu and Eric Kim. Closer look at efficient inference methods: A survey of speculative decoding. *arXiv preprint arXiv:2411.13157*, 2024.
- Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=CS2JWaziYr>.
- Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025.
- Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=PxoFut3dWW>.
- Bangsheng Tang, Carl Chengyan Fu, Fei Kou, Grigory Sizov, Haoci Zhang, Jason Park, Jiawen Liu, Jie You, Qirui Yang, Sachin Mehta, et al. Efficient speculative decoding for llama at scale: Challenges and solutions. *arXiv preprint arXiv:2508.08192*, 2025.
- Xiaoyu Tian, Yunjie Ji, Haotian Wang, Shuaiting Chen, Sitong Zhao, Yiping Peng, Han Zhao, and Xiangang Li. Not all correct answers are equal: Why your distillation source matters, 2025. URL <https://arxiv.org/abs/2505.14464>.
- Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Quantspec: Self-speculative decoding with hierarchical quantized KV cache. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=7SHbJENGHX>.
- Minnan Wei, Ziming Li, Xiang Chen, Menglin Zheng, Ziyang Qu, Cheng Yu, Siyu Chen, and Xiaolin Ju. Evaluating and improving large language models for competitive program generation. *arXiv preprint arXiv:2506.22954*, 2025.
- Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=EKJhH5D5wA>.

- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for LLM inference serving. In *Eighth Conference on Machine Learning and Systems*, 2025. URL <https://openreview.net/forum?id=RXPoFAsL8F>.
- Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.
- Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=T9u56s7mbk>.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

APPENDIX

A EXPERIMENTAL DETAILS

A.1 TRAINING

Backbones. We use two representative 8B-scale reasoning models: Qwen3-8B (Yang et al., 2025) and DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025).

Adapters. For LoRA (Hu et al., 2022) finetuning, we set the rank to 16 and α to 32. The Multi-token Prediction (MTP) head was trained with a default draft length of 4.

Dataset. We utilized two datasets for finetuning:

- *AM-Qwen3-Distilled* (Tian et al., 2025): From the total 1.89M samples, we created a 120k-sample subset by random sampling that preserves the original data distribution.
- *OpenThought-114k* (Guha et al., 2025): We used the full dataset.

Response Collection for Training. We used the prompts from the training datasets as seeds to generate responses, which then served as the training targets. Responses were collected via vLLM (Kwon et al., 2023) using the default sampling parameters for each model: temperature 0.6 and top- p 0.95. For Qwen3-8B, top- k of 20 was additionally applied.

Optimization. Models were trained for 30k steps using the AdamW (Loshchilov & Hutter, 2019) optimizer with a constant learning rate of 2×10^{-4} and a linear warmup over the first 5k steps. We used a batch size of 4 and `bfloat16` precision. The entire training process for one model took approximately 3 days on $4 \times \text{H100}$ GPUs.

A.2 INFERENCE ENGINE

To benchmark performance, we built a custom inference engine based on FlashInfer (Ye et al., 2025) kernels for attention, RoPE, and sampling. The engine incorporates several standard optimizations:

- **Weight Fusion:** Applied to the Q,K,V projections and the MLP gate/up projections.
- **Tensor Parallelism:** Enabled to distribute the model across multiple GPUs.
- **Paged KV Cache:** Utilized for efficient memory management of the KV cache.
- **Precision:** `bfloat16` was used, consistent with training.

Engine build flags, kernel versions, and fusion scripts are fixed across all runs and will be released with the code.

B DETAILS ON EVALUATION PROTOCOL

B.1 MOTIVATION AND RATIONALE FOR PROTOCOL DESIGN

Our primary goal is to measure the acceleration in the **decoding phase** of inference. Therefore, all measurements deliberately exclude the prefill and initialization stages. To create a controlled environment for this, we disabled dynamic batching techniques like continuous batching. Instead, we adopted a **Batch Halting Rule**: the entire batch’s generation process is terminated as soon as any single sequence within it emits an `[EOS]` token.

Why Batch Halting? This rule is crucial for speculative decoding. If a sequence continues to generate tokens post-`[EOS]`, it produces unnatural token sequences. These can arbitrarily inflate or deflate the acceptance rate of speculative drafts, introducing significant noise and confounding the performance measurement. The batch halting rule prevents this contamination.

However, this controlled setup introduced two significant measurement challenges:

1. **Length-Induced Bias.** The generation length for problems across our datasets (AIME, Code-Forces, GPQA) varies dramatically. Some runs might terminate in under 1000 tokens, while

others could extend much longer. In LLM inference, throughput is highly sensitive to the KV cache length—shorter sequences yield much higher throughput. If one method (e.g., standard decoding) happens to be tested on more batches with short-sequence problems than another (e.g., speculative decoding), its average throughput will be artificially inflated. This creates a statistical bias that makes a fair comparison impossible. This issue was more pronounced with larger batch sizes, where the probability of including at least one short-sequence problem increases.

2. **Infeasibility of Reaching Long Contexts.** We aimed to evaluate performance on sequences up to 32k tokens. However, due to the length variance and our batch halting rule, virtually no batch ever completed generation up to 32k tokens naturally. Direct end-to-end measurement for very long sequences was therefore not feasible.

To overcome these challenges, we designed our main evaluation protocol, **Long-Sequence Throughput Estimation**, to explicitly factorize the effect of KV cache length on throughput, allowing for a robust and unbiased estimation for long sequences. We complement this with a more traditional **End-to-End Generation** benchmark for shorter sequences.

B.2 MAIN PROTOCOL: LONG-SEQUENCE THROUGHPUT ESTIMATION

Goal. To obtain a stable and unbiased estimate of the effective throughput for long sequences (e.g., 24k/32k) by simulating the generation process.

Data Preparation. For each dataset, we first randomly sampled 1000 seed prompts. We then used vLLM to pre-generate a full-length response for each prompt, creating a large pool of (prompt, response) pairs.

Procedure.

1. We measure the decoding throughput $r(L)$ at a set of discrete prefix lengths L .
2. For each L , batches are assembled from the pre-generated response pool. If a sequence was shorter than the target prefix length L , it was left-padded, and an `[EOS]` token was inserted into the padding to prevent it from affecting the generation dynamics of other sequences.
3. From the prefix of length L , we run the decoding process until the batch collectively commits a fixed number of new tokens ($128 \times \text{bsz}$). We record the throughput $r(L)$ for this segment. This process is repeated for various prefix lengths L as detailed in Appendix B.4.

Throughput Estimation. With the measured per-prefix throughputs $r(L_i)$, we estimate the total time to generate a sequence of length T by summing the time spent in each length bin:

$$\hat{T}_{\text{decode}} = \sum_i \frac{\Delta n_i}{r(L_i)}, \quad \text{Throughput}(T) = \frac{T - L_0}{\hat{T}_{\text{decode}}},$$

where $\Delta n_i = \min(L_{i+1}, T) - L_i$ and L_0 is the initial prompt length. This method effectively calculates a weighted harmonic mean of throughputs, providing a robust estimate of performance over long generation horizons.

B.3 SUPPLEMENTAL PROTOCOL: END-TO-END GENERATION

Goal. To provide a more conventional benchmark on shorter, fixed-length generation tasks.

Setup. We fix the prompt length to 1024 tokens and the maximum generation length to 4096 tokens. The same **Batch Halting Rule** described above is applied.

Dataset Composition. To minimize the length variance challenge even in this setup, we adjusted the number of problems per prompt to normalize the context length:

- **AIME2025 & GPQA-Diamond:** 4 problems per prompt.
- **CodeForces:** 1 problem per prompt. This was necessary because CodeForces problem statements are very long, and including multiple problems would create excessively long prompts compared to the other datasets.

B.4 CONFIGURATIONS

Aggregation. For each configuration, we perform 11 runs. The first is discarded as a warm-up to eliminate compilation overhead. The final metric is the mean and standard deviation of the remaining 10 runs.

Rationale for Tensor Parallelism (TP) Sizing. The VRAM required for inference is dominated by the KV cache, which scales with both batch size and sequence length. To accommodate this, we adjusted the tensor parallelism (TP) degree. For each experimental setting, we determined the minimum TP size necessary to prevent out-of-memory errors, ensuring an efficient use of hardware resources. As shown in Tables 2 and 3, larger batch sizes and longer context lengths demanded higher TP degrees.

Rationale for Draft Length Selection. The optimal draft length in speculative decoding is determined by a trade-off between the potential throughput gains from longer drafts and the increasing cost of the parallel verification step. This verification overhead is particularly sensitive to the batch size; as the batch size grows, the computational cost of verifying a long draft across all sequences increases substantially. Consequently, shorter draft lengths often become more performant for larger batches. To ensure we report the best possible performance for our method in every scenario, we benchmarked a range of candidate draft lengths (e.g., 3, 4) for each configuration. The results presented in our paper reflect the performance of the optimal draft length identified for that specific setting.

Table 2: Configurations for the **Long-Sequence Throughput Estimation** protocol. The TP size was increased to accommodate the VRAM required by the maximum prefix length in each setting. For each batch size, various draft lengths were tested to find the optimal configuration.

Batch Size	TP Size	Prefix Lengths(L) Swept	Candidate Draft Lengths
16	2	{1k, 2k, 4k, 8k, 12k, 16k, 20k, 24k, 28k}	3,4
32	4	{1k, 2k, 4k, 8k, 12k, 16k, 20k, 24k, 28k}	3,4
64	8	{1k, 2k, 4k, 8k, 12k, 16k, 20k, 24k, 28k}	2,3,4
128	8	{1k, 2k, 4k, 8k, 12k, 16k, 20k}	2,3,4
256	8	{1k, 2k, 4k, 8k, 12k}	1,2,3

Table 3: Configurations for the **End-to-End Generation** protocol. Prompt length was fixed at 1024 and maximum generation length at 4096. TP sizes were selected based on VRAM usage for the total sequence length of 5120.

Batch Size	TP Size	Candidate Draft Lengths	
		Greedy	Sampling
16	1	3, 4	3, 4
32	1	3, 4	3, 4
64	2	3, 4	2, 3
128	2	2, 3	2, 3
256	4	2, 3	1, 2

C END-TO-END GENERATION RESULTS

Table 4: End-to-End generation results on Qwen3-8B with evaluation protocol B.3. Throughput (tok/s) is reported with standard deviation, and speedup relative to standard decoding/sampling is highlighted in bold.

Batch	AIME2025		CodeForces		GPQA-Diamond	
	Standard	MTP	Standard	MTP	Standard	MTP
Greedy Decoding						
16	603.6 \pm 0.9	1352.7 \pm 0.2 (\times 2.24)	572.7 \pm 2.6	1259.9 \pm 15.9 (\times 2.20)	594.7 \pm 1.2	1295.9 \pm 18.8 (\times 2.18)
32	917.8 \pm 0.1	1959.6 \pm 57.5 (\times 2.14)	890.2 \pm 10.4	1795.0 \pm 32.3 (\times 2.02)	907.0 \pm 2.3	1907.2 \pm 30.6 (\times 2.10)
64	1135.0 \pm 1.1	2376.4 \pm 0.8 (\times 2.09)	1134.4 \pm 21.2	2228.6 \pm 44.4 (\times 1.96)	1135.3 \pm 2.4	2382.1 \pm 5.0 (\times 2.10)
128	1944.6 \pm 0.9	4119.2 \pm 1.9 (\times 2.12)	1964.1 \pm 13.0	3842.2 \pm 32.7 (\times 1.96)	1969.6 \pm 2.8	4011.5 \pm 6.9 (\times 2.04)
256	3470.5 \pm 0.9	6543.9 \pm 32.1 (\times 1.89)	3531.3 \pm 25.1	6079.1 \pm 29.4 (\times 1.72)	3516.6 \pm 1.3	6402.5 \pm 37.0 (\times 1.82)
Random Sampling						
16	597.1 \pm 0.8	1065.8 \pm 10.1 (\times 1.78)	584.9 \pm 3.3	950.7 \pm 24.9 (\times 1.63)	590.2 \pm 1.7	955.6 \pm 18.8 (\times 1.62)
32	903.7 \pm 2.7	1585.9 \pm 13.3 (\times 1.75)	892.8 \pm 10.6	1381.5 \pm 37.5 (\times 1.55)	909.0 \pm 1.4	1442.2 \pm 8.3 (\times 1.59)
64	1117.2 \pm 1.0	2098.2 \pm 9.7 (\times 1.88)	1120.4 \pm 20.1	1799.2 \pm 35.7 (\times 1.61)	1125.8 \pm 2.7	1689.5 \pm 15.0 (\times 1.50)
128	1919.0 \pm 0.5	3665.3 \pm 8.2 (\times 1.91)	1924.8 \pm 12.5	3204.1 \pm 33.6 (\times 1.67)	1939.1 \pm 2.6	3427.0 \pm 19.4 (\times 1.77)
256	3435.0 \pm 0.6	5577.0 \pm 18.8 (\times 1.62)	3490.5 \pm 24.4	5474.5 \pm 24.6 (\times 1.57)	3462.5 \pm 5.5	5577.0 \pm 18.8 (\times 1.61)

Table 5: End-to-End generation results on DeepSeek-R1-Distill-Llama-8B with evaluation protocol B.3. Throughput (tok/s) is reported with standard deviation, and speedup relative to standard decoding/sampling is highlighted in bold.

Batch	AIME2025		CodeForces		GPQA-Diamond	
	Standard	MTP	Standard	MTP	Standard	MTP
Greedy Decoding						
16	591.4 \pm 1.0	1245.3 \pm 0.7 (\times 2.11)	562.5 \pm 2.9	1188.0 \pm 12.2 (\times 2.11)	579.8 \pm 1.4	1215.4 \pm 10.4 (\times 2.10)
32	902.3 \pm 0.6	1850.9 \pm 21.5 (\times 2.05)	875.1 \pm 9.8	1762.2 \pm 24.3 (\times 2.01)	889.7 \pm 2.1	1812.6 \pm 19.6 (\times 2.04)
64	1112.5 \pm 1.3	2244.7 \pm 6.9 (\times 2.02)	1105.6 \pm 19.7	2140.3 \pm 33.1 (\times 1.94)	1109.3 \pm 3.0	2201.8 \pm 11.8 (\times 1.98)
128	1901.7 \pm 1.0	3787.4 \pm 5.2 (\times 1.99)	1910.4 \pm 12.1	3640.2 \pm 26.8 (\times 1.90)	1907.2 \pm 2.7	3712.0 \pm 14.9 (\times 1.95)
256	3390.6 \pm 0.8	6195.0 \pm 27.3 (\times 1.83)	3428.7 \pm 23.4	5902.5 \pm 31.2 (\times 1.72)	3411.4 \pm 4.6	6071.2 \pm 22.7 (\times 1.78)
Random Sampling						
16	584.0 \pm 0.9	985.2 \pm 9.7 (\times 1.69)	571.2 \pm 3.1	902.1 \pm 19.8 (\times 1.58)	577.1 \pm 1.6	943.6 \pm 15.2 (\times 1.64)
32	889.6 \pm 2.5	1452.4 \pm 11.9 (\times 1.63)	878.2 \pm 9.5	1341.7 \pm 22.7 (\times 1.53)	883.5 \pm 1.8	1397.0 \pm 10.3 (\times 1.58)
64	1099.2 \pm 1.2	1925.6 \pm 7.4 (\times 1.75)	1102.8 \pm 18.6	1704.3 \pm 28.9 (\times 1.54)	1101.0 \pm 2.5	1810.2 \pm 13.5 (\times 1.64)
128	1890.3 \pm 0.6	3360.7 \pm 8.5 (\times 1.78)	1896.5 \pm 11.2	3070.1 \pm 29.6 (\times 1.62)	1893.4 \pm 2.4	3215.4 \pm 18.7 (\times 1.70)
256	3381.8 \pm 0.7	5261.4 \pm 19.4 (\times 1.56)	3420.6 \pm 22.1	4988.3 \pm 26.1 (\times 1.46)	3400.7 \pm 4.1	5124.9 \pm 17.2 (\times 1.51)