
➤ MINEDRAFT: A Framework for Batch Parallel Speculative Decoding

Anonymous Authors¹

Abstract

Speculative decoding (SD) accelerates large language model inference by using a smaller draft model to propose draft tokens that are subsequently verified by a larger target model. However, the performance of standard SD is often limited by the strictly sequential execution of these drafting and verification stages. To address this, this paper proposes MINEDRAFT, a batch parallel speculative decoding (PSD) framework designed to effectively hide drafting latency by overlapping it with verification. Our theoretical analysis shows that PSD is substantially more efficient than standard SD. MINEDRAFT realizes the PSD through a novel batch-parallel design that maintains two batches of requests, overlapping drafting for one batch with verification for the other. Our experimental results show significant improvements of MINEDRAFT in both throughput (up to 75%) and end-to-end latency (up to 39%) over standard SD. Furthermore, we have implemented MINEDRAFT as a plugin for vLLM, demonstrating its practicality for production-ready inference systems.

1. Introduction

Speculative decoding (SD) (Leviathan et al., 2023; Ryu & Kim, 2024; Xia et al., 2024) is a popular method used for accelerating large language model (LLM) inference. Instead of generating tokens sequentially with the target large model, SD uses a typically smaller and faster draft model to autoregressively draft (propose) a short sequence of tokens, which the target model then verifies in parallel. This parallel verification allows SD to achieve higher generation throughput than conventional autoregressive decoding when most drafted tokens are accepted. However, the effectiveness of SD is highly sensitive to the acceptance rate of drafted tokens, as drafting introduces additional computational over-

head, frequent rejections can negate its benefits and even make SD slower than standard autoregressive decoding.

Prior work has improved the verification success rate (VSR) of draft tokens through methods such as specialized drafting models (Cai et al., 2024; Li et al., 2024a), tree-based drafting (Wang et al., 2025), and active draft token selection (Wu et al., 2025). While these methods increase the overall VSR, they often lengthen the drafting stage due to increased model complexity (Cai et al., 2024; Li et al., 2024a) or sampling overhead (Wang et al., 2025; Wu et al., 2025). In standard speculative decoding, drafting and verification are executed *sequentially*, placing drafting directly on the critical path and fundamentally limiting achievable speedups. This observation motivates parallelizing the two phases: since the draft model is typically much smaller than the target model, its computation can be effectively overlapped with the target model’s verification. By parallelizing the drafting and verification stages, a smaller draft model can generate draft tokens without blocking the verification process. When combined with an effective drafting mechanism, parallel speculative decoding hides drafting latency and yields improved overall inference performance.

Since the verification stage inherently depends on the outputs of the drafting stage, implementing parallelism in speculative decoding is nontrivial (Leviathan et al., 2023). Existing methods (Wang et al., 2024; Xiao et al., 2024; Timor et al., 2025) typically break this dependency at the cost of substantial additional resources, such as increased GPU or memory usage (Wang et al., 2024; Timor et al., 2025), or require a dedicated training stage for the draft model (Xiao et al., 2024). While Liu et al. (2025a) proposes a method that parallelizes drafting and verification for a single request using pre-verify and post-verify strategies, scaling this method to batched requests (multi-request) remains challenging. Moreover, its performance degrades when prerequisites for post-verification are not satisfied, a limitation that becomes especially severe in batched request settings.

This paper presents MINEDRAFT, a framework for batch parallel speculative decoding (PSD) that systematically parallelizes the drafting and verification stages to hide drafting latency, thereby accelerating LLM inference. We provide a theoretical justification showing that PSD outperforms standard SD by efficiently utilizing additional computational

¹Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

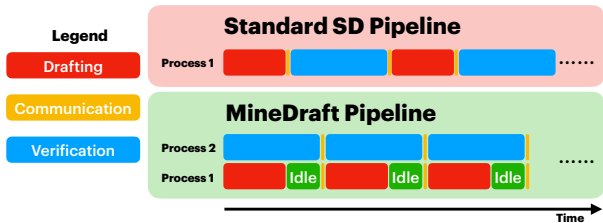


Figure 1. MineDraft parallelizes drafting and verification: a draft model generates tokens while the target model simultaneously verifies the previously generated draft tokens, thereby hiding drafting latency and improving overall inference throughput.

resources through parallel drafting and verification. To efficiently parallelize drafting and verification, our framework uses a novel batch-parallel design that maintains two batches of requests, overlapping drafting for one batch with verification for the other. The name of our framework is inspired by the *Minecraft game engine*, which loads the next ‘chunk’ of blocks (the next set of draft tokens) into the game world while the player is still interacting with the current blocks (verifying the current drafted tokens). This ‘background loading’ is exactly how MINEDRAFT hides drafting latency.

Since MINEDRAFT is agnostic to the strategies for generating and selecting draft tokens, it can be combined with existing SD methods such as EAGLE* (Li et al., 2024a;b; 2025a), PEARL (Liu et al., 2025a), and TETRIS (Wu et al., 2025), and is orthogonal to most LLM inference optimization techniques, including forward-pass modifications, model parallelism, and draft models. Our experimental results show that MINEDRAFT outperforms standard SD by achieving average throughput gains and reducing end-to-end latency at the cost of only 1 additional GPU, highlighting its significant practical impact. Furthermore, MINEDRAFT is implemented as a plugin for the production-ready inference library vLLM (Kwon et al., 2023). The specific contributions of this paper are summarized as follows:

- **Theoretical efficiency of PSD.** In Sec. 4, we first present a theoretical analysis of batch PSD demonstrating that it achieves at least a 37% reduction in end-to-end latency compared to standard SD under mild assumptions about the quality of the draft model.
- **A framework for batch PSD.** In Sec. 5, we propose a novel framework, MINEDRAFT, for batch PSD that maintains two batches of requests, overlapping drafting for one batch with verification for the another. To enable parallel execution, MINEDRAFT runs the draft model on a separate GPU and employs direct GPU-to-GPU communication to transfer draft tokens to the target model, which runs on GPUs separate from the draft model.
- **Experimental results.** In Sec. 6, we evaluate the performance of MINEDRAFT through comprehensive experiments across existing SD methods. Our results show that MINEDRAFT achieves average throughput gains of

up to 75% and reduces end-to-end latency by up to 39% compared to standard SD, at the cost of one extra GPU.

- **MINEDRAFT as a vLLM plugin.** We develop a plug-and-play vLLM plugin that implements MINEDRAFT, encouraging further academic research and experimentation. The plugin also supports continuous batching (iteration-level scheduling) (Yu et al., 2022) and is fully compatible with PagedAttention (Kwon et al., 2023), highlighting its significant practical impact and ease of use.

2. Related Work

Parallel speculative decoding. Recent work has explored reducing drafting overhead through parallelism. Parallel SD methods achieve faster decoding at the cost of substantial GPU resources, such as increased VRAM consumption (Wang et al., 2024; Timor et al., 2025). Another promising direction is parallel decoding with tree attention, which accelerates SD by drafting multiple tokens in parallel (Cai et al., 2024; Li et al., 2024a). In addition, prior work focuses on the single-request setting and modifies the SD process to generate variable-length drafts, thereby reducing synchronization overhead between the drafter and verifier (Liu et al., 2025a). Whereas MINEDRAFT adopts a new parallelism paradigm, which we term *Batch Parallelism*. This paradigm requires only one additional GPU compared to standard SD, balances GPU resource usage and overhead reduction, and avoids the need for a dedicated training stage.

Adaptive drafting. Under many concurrent requests, standard SD can have suboptimal performance when using long speculative lengths, as the increased number of draft tokens places significant pressure on the verification stage. To mitigate this issue, adaptive drafting strategies generate variable-length drafts and selectively verify only the most promising tokens. EAGLE-2 (Li et al., 2024b) adopts this approach by expanding and reranking a context-aware draft tree. Other heuristics include selecting drafts with the highest likelihood of acceptance within a batch (Wu et al., 2025) and choosing among drafts generated by different speculative methods based on their historical acceptance rates (Hou et al., 2025). Additionally, AdaSpec (Huang et al., 2025) dynamically controls the execution of the draft model based on estimated draft quality and generation speed. Despite these advances, existing adaptive drafting methods yield limited performance gains: the number of draft tokens that can be generated remains constrained, as excessive drafting quickly makes verification overhead prohibitive.

3. Preliminaries

Speculative decoding (SD). We consider the standard SD setting, which involves a small draft model (the drafter) and a large target model (the verifier). At each SD step, the drafter first generates a sequence of draft tokens, after which

the verifier evaluates them in parallel to decide whether to accept or reject them (i.e., verification), and then produces the final output. Standard SD does not allow parallelization between drafting and verification, since the verification inherently depends on the prior availability of the draft tokens.

Parallel speculative decoding (PSD). We consider a variant of PSD that systematically parallelizes the drafting and verification stages. By overlapping these stages, the draft model generates tokens for one batch of requests while the target model simultaneously verifies the draft tokens from another batch, thereby effectively hiding drafting latency and providing a significant speedup over standard SD.

Notations. We now introduce the notation used throughout this paper. We consider an LLM inference service provider with a fixed capacity C , representing the maximum number of tokens its computing resources can process in parallel at each step. In practice, this capacity depends on the service provider’s server configuration. At each decoding step, the server processes a batch of m requests r_1, r_2, \dots, r_m . Each request r_i is associated with a partially generated sequence $S_{i,t_i} = (d_{i,1}, \dots, d_{i,t_i})$, where t_i represents the number of tokens generated so far for request r_i . We allow a variable draft window size for each request, i.e., k_i for request r_i . At step s , the draft model \mathcal{S} drafts and selects a set $\mathcal{D}_s := \{(i, j) | i \in [m], j \in [t_i + k_i]\}$, such that the total number of selected tokens satisfies $|\mathcal{D}_s| = \sum_{i=1}^m k_i = C$.

Average throughput. Let $\mathbf{1}_{i,j}$ be an indicator variable that equals 1 if the token $(i, j) \in \mathcal{D}_s$ is verified, 0 otherwise. The average throughput is defined as the total number of verified tokens generated over T steps, divided by the total time τ required to complete these steps:

$$\mathcal{G} := \tau^{-1} \sum_{s=1}^T (\sum_{(i,j) \in \mathcal{D}_s} \mathbf{1}_{i,j} + m).$$

End-to-end latency (E2EL). The end-to-end latency is defined as the average time from request arrival to completion for a set of requests under the SD inference system.

4. Theoretical Efficiency of PSD

To highlight the fundamental differences between SD and PSD, we present an analysis under the constraint that the number of tokens sent for verification at each step is fixed.

Drafting Pareto frontier. Let $f : \mathbb{R}^+ \rightarrow [0, 1]$ denote the *Pareto frontier* of the draft model, which characterizes the maximum achievable verification success rate given a drafting time budget. The function f can be modified by changing the draft model or by applying speculating techniques such as Medusa (Cai et al., 2024), EAGLE (Li et al., 2024a), and TETRIS (Wu et al., 2025). Our goal is to compare the inference time of SD, denoted by T_{SD} , and that of PSD, denoted by T_{PSD} . For simplicity, we ignore the

time required to draft the first batch of requests in the PSD analysis.

Analysis. We assume that the drafting time per step is a constant t . Let R denote the total number of tokens to be generated across all requests. Under standard SD, the expected verification time incurred is

$$T_{\text{SD}} := \min_t \mathbb{E} \left[\sum_{s=1}^T (t + V) \right],$$

where T is a random variable representing the number of steps required to generate responses for all requests such that $\mathbb{E}[T] = f(t) \cdot R$. The term V denotes the verification time per step, which we assume to be a constant since the number of tokens submitted for verification is fixed. In practice, f is typically an increasing function of t . Since $f(\cdot) \in [0, 1]$, it follows that $T_{\text{SD}} \rightarrow \infty$ as $t \rightarrow \infty$. On the other hand, under PSD, the drafting and verification times overlap, resulting in the expected time for a step to become

$$T_{\text{PSD}} := \min_t \mathbb{E} \left[\sum_{s=1}^T \max(V, t) \right] = \min_t \frac{R \max(V, t)}{f(t)}.$$

Under a mild assumption on f , we have the following result.

Theorem 1. *Let W be the Lambert W function and $f(t) = 1 - e^{-\alpha t}$ for some constant $\alpha \in \mathbb{R}^+$. For $\alpha V \geq -W_{-1}(-\frac{1}{2e}) - 1 \approx 1.68$, we have $T_{\text{SD}} > 1.59 T_{\text{PSD}}$.*

This result shows that when either the verification time or the curvature of f (captured by α) is sufficiently large, PSD achieves at least a $(1.59 - 1)/1.59 > 37\%$ reduction in inference time compared to standard SD. Since the larger values of α yield stronger diminishing returns in the verification success rate, the product αV can be interpreted as a proxy for the expected number of accepted tokens per step. From this perspective, any method with an improved Pareto frontier for f leads to more accepted tokens per step and therefore makes PSD more effective relative to SD. Finally, note that the naive upper bound on the improvement is 50%, corresponding to the idealized case in which drafting is perfectly hidden within the verification time.

5. MINE DRAFT: A Framework for Batch Parallel Speculative Decoding

Our parallelism follows a simple principle: at the beginning of each SD step, we regroup requests by filtering out those verified in the previous step, allowing the verifier to continue operating in parallel with the drafter. To fully utilize the verifier, if it processes m requests per step, we maintain $2m$ requests partitioned into two batches and alternate between them, sending m requests to the verifier at each SD step.

In Fig. 2, we illustrate the overall architecture of MINE DRAFT, which consists of a *Batch Manager*, a *Scheduler*, a

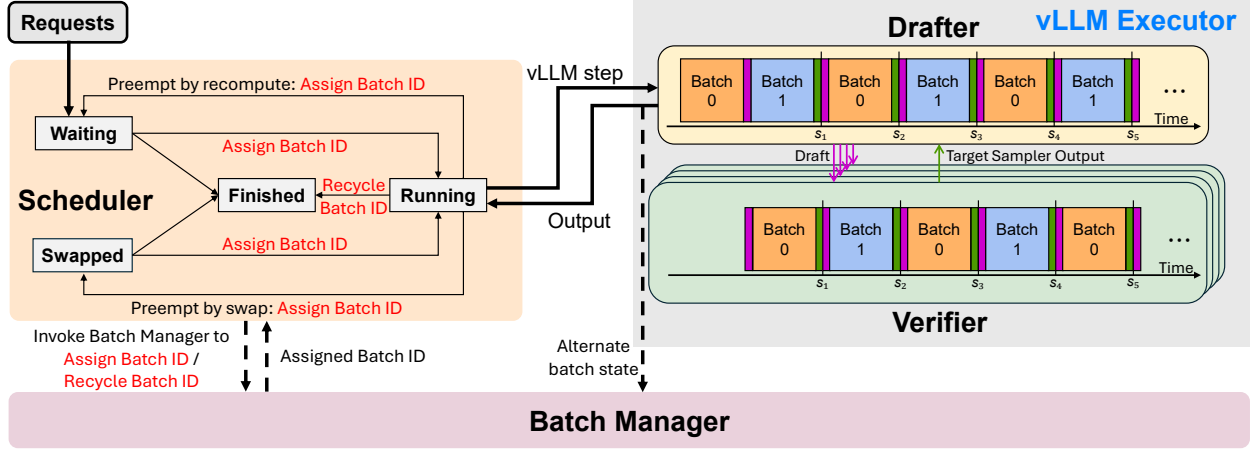


Figure 2. **Architecture overview of MINE-DRAFT.** (Left) The **Scheduler** manages request life-cycles and batch IDs by coordinating with the **Batch Manager**, which maintains two batches to enable parallelism in MINE-DRAFT. (Right) Parallel execution timeline of the **Drafter** and **Verifier** across speculative decoding (SD) steps. Magenta blocks/arrows denote broadcast of drafts from the Drafter to the Verifier, while dark green blocks/arrows denote point-to-point dispatch of target sampler outputs from the Verifier to the Drafter. Ticks indicate synchronization points between SD steps. During the initial SD step (before s_1), the Drafter sequentially drafts Batch 0, broadcasts drafts to the Verifier, then drafts Batch 1, while the Verifier immediately processes Batch 0 upon receipt and returns outputs to the Drafter. In subsequent SD steps, the two batches alternate roles between drafting and verification, enabling overlapped execution. Batch alternation is triggered when the Drafter returns outputs to the Scheduler at the end of each SD step.

Drafter, and a *Verifier*. We describe the role of each component below and provide a concrete example of a MINE-DRAFT execution in Sec. D.3.

5.1. Batch Manager

State variables and initialization. Given a maximum of m requests processed per step, the system supports up to $2m$ concurrent requests, which the Batch Manager organizes into two batches. Each running request is assigned a batch ID, either 0 or 1, indicating membership in Batch 0 or Batch 1, respectively. The imbalance between the two batches is tracked by a state variable `balance`, defined as the difference in their sizes: $\text{balance} = |\text{Batch 1}| - |\text{Batch 0}|$. A positive `balance` indicates that Batch 1 contains more requests, in which case new requests are assigned to Batch 0 for re-balancing; a negative value indicates the opposite. In each SD step, the state variable `skip_batch` is used to identify the batch undergoing parallelized drafting, referred to as the *draft batch*. The remaining batch—whose batch ID differs from `skip_batch`—is the *target batch*, whose draft tokens are verified concurrently in the same SD step. At initialization, `balance` is set to 0, indicating equal batch sizes, and `skip_batch` is set to 1, designating Batch 1 as the initial draft batch.

Batch ID assignment. In the first SD step, the `assign` operation assigns a batch ID to each new running request and serves as a load-balancing mechanism between the two batches. If $\text{balance} \geq 0$, indicating that Batch 1 has at least as many requests as Batch 0, the request is assigned to Batch 0 and `balance` is decremented. Otherwise, if $\text{balance} < 0$, the request is assigned to Batch 1 and

`balance` is incremented. This assignment rule maintains a balanced workload between the two batches.

After the first SD step, assuming both batches have reached their capacity of m requests, new running requests are assigned to the current `skip_batch` to ensure verification occurs only when draft tokens are available. This policy typically preserves balance between the two batches, since after the first SD step, a waiting request can be scheduled only when a request in the target batch finishes. However, exceptional terminations due to preemption or abort can disrupt this balance. Moreover, when chunked prefill is enabled, the Batch Manager may receive fewer than $2m$ requests from the Scheduler in the first SD step, resulting in batch sizes that fall below the capacity m . As subsequent requests are consistently assigned to the draft batch, this initial shortfall can lead to an irrecoverable workload imbalance between the two batches, degrading the expected performance gains of MINE-DRAFT. We further discuss the implications of this limitation and possible mitigation in Sec. F.

Batch ID recycling. The `recycle` operation is invoked when a request is finished, preempted, or aborted. It takes the batch ID b of the terminated request as input, and updates `balance` to reflect the new distribution of requests. If the request belonged to Batch 0 ($b = 0$), `balance` is incremented; if it belonged to Batch 1 ($b = 1$), `balance` is decremented. This update is the inverse of the `assign` operation, preserving consistency of `balance`.

5.2. Drafter, Verifier and Scheduler

The roles and design details of the Drafter, the Verifier, and the Scheduler are provided in Sec. C.

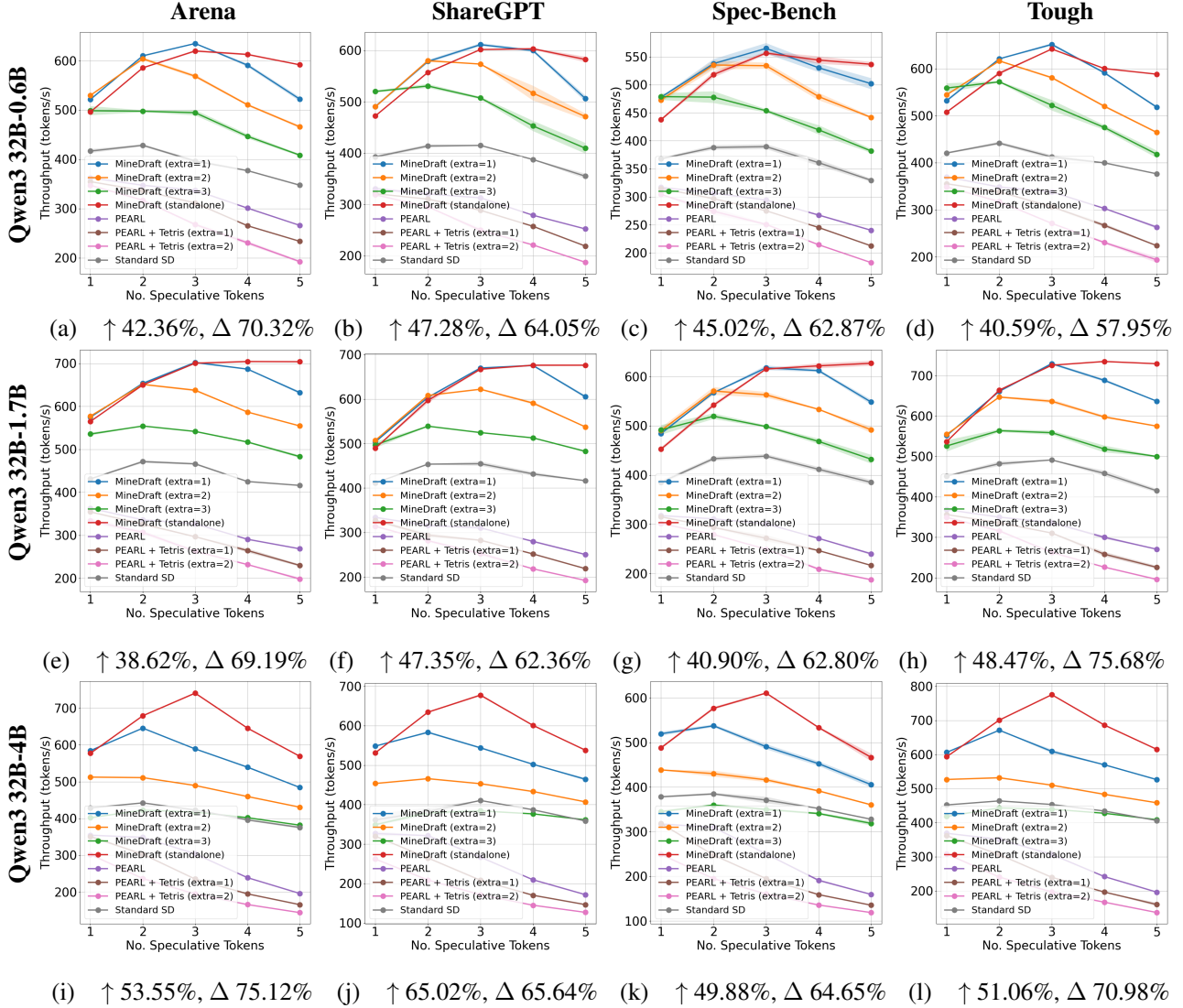


Figure 3. Throughput comparison against baseline methods across model settings 1–3. \uparrow indicates the average improvement over the best baseline method. Δ indicates the maximum average gap between MINEDRAFT and standard SD. More details about \uparrow and Δ are provided in Sec. D.4. MINEDRAFT consistently outperforms baselines, improving average throughput by up to 65.02% over the best-performing baseline and by up to 75.68% over standard SD.

6. Experiments

We evaluate the effectiveness of MINEDRAFT using the performance metrics introduced in Sec. 3, namely average throughput (tokens/second or tokens/s) and end-to-end latency (milliseconds or ms). In addition to demonstrating the performance gains of MINEDRAFT, our empirical results show that MINEDRAFT can further improve speculative decoding by integrating existing drafting techniques.

Model settings. We consider seven target–draft model configurations, listed below. In all settings, the target model is served with tensor parallelism set to four, while a single GPU is dedicated to serving the draft model. Additional setup details are provided in Sec. D.1.

- **Settings 1–4** use Qwen3-32B (Yang et al., 2025) as the target model with different draft models: Qwen3-0.6B (Setting 1), Qwen3-1.7B (Setting 2), Qwen3-4B (Setting 3), Qwen3-8B (Setting 4). All are served on 5 L40 GPUs with $m = 16$.

Datasets. Following the experimental settings of prior work (Hou et al., 2025; Oliaro et al., 2025; Wu et al., 2025), we evaluate MINEDRAFT on ShareGPT (Anon, 2023) (Apache License 2.0), Arena (Zheng et al., 2023) (CC), LLM-Tough-Questions (Tough) (YAV-AI, 2024) (MIT License), and Spec-Bench (Xia et al., 2024) (Apache License 2.0), to robustly assess the applicability of MINEDRAFT across a diverse range of scenarios.

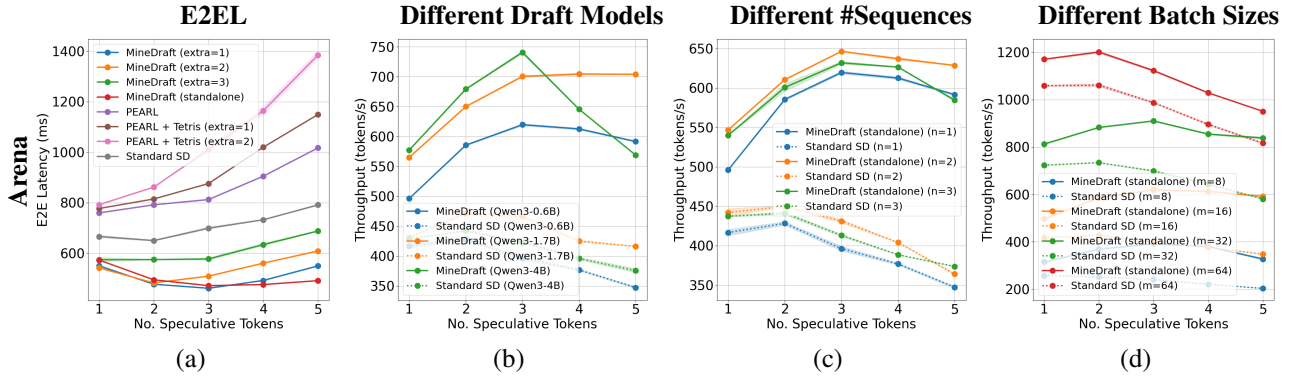


Figure 4. Different ablation studies using the Arena dataset. The comprehensive details and analysis of our ablation studies are included in Sections D.6, D.8 and D.9. Plot (a) shows MINE DRAFT consistently reduces end-to-end latency relative to baselines. Plot (b) indicates choice of draft model can affect MINE DRAFT’s parallelism benefit. Plots (c) and (d) present how MINE DRAFT scales with larger n (number of sequences per request) and m (batch size) values.

6.1. Comparison against Baselines.

We measure the average throughput of MINE DRAFT and established baselines, including PEARL (Liu et al., 2025a) and standard SD. As shown in Fig. 3, MINE DRAFT improves average throughput over the best-performing baseline by 38.62% to 65.02%, with maximum gains over standard SD ranging from 57.95% to 75.68%. These results show MINE DRAFT consistently outperforms all baselines across all model settings and datasets. For all experimental results, we report the mean and standard deviation over three independent trials unless otherwise specified. We omit PEARL from subsequent experiments, as it exhibits substantially lower efficiency than MINE DRAFT in preceding evaluations.

Issue with extreme k value. Notably, increasing the number of draft tokens per step is not always beneficial: excessive drafting can cause the draft stage to dominate verification time, thereby negating the benefit of hiding drafting latency via PSD. This effect is particularly significant for large draft models and is further analyzed in Sec. D.8.

6.2. Ablation Studies

This section presents representative ablation studies on the Arena dataset. Comprehensive results across all selected datasets are provided in Sections D.6, D.8 and D.9. Due to space constraints, analysis of verification success rate of integration with TETRIS and Nsight Systems profiling results are also provided in Sec. D.

End-to-end latency. We measure end-to-end latency of MINE DRAFT and baseline methods under various drafting strategies across the model settings and datasets as described above. Fig. 4 shows representative results for Setting 1 on the Arena dataset, where MINE DRAFT achieves a 26.58% latency reduction relative to the best-performing baseline and a maximum reduction of 37.89% compared to standard SD. A complete results are provided in Sec. D.6.

Draft model sizes. We study the impact of draft model

scale on system performance. Although MINE DRAFT consistently outperforms standard SD across draft model sizes, Fig. 4 shows that increasing the number of draft tokens per step does not monotonically improve average throughput. This degradation is most pronounced for Qwen3-4B: when more than three draft tokens are generated per step, throughput drops below that achieved with only two draft tokens. A detailed analysis of this behavior is provided in Sec. D.8.

Scaling behavior with n and m . We evaluate throughput scaling with respect to the number of sequences per request (n) and batch size (m). The final two plots in Fig. 4 show that, with m held constant, increasing n yields only marginal throughput gains, whereas increasing m consistently improves average throughput. Moreover, when the number of draft tokens per step exceeds 4, the average throughput of MINE DRAFT approaches that of standard SD with doubled batch size, suggesting that MINE DRAFT can complement adaptive drafting strategies, which are often limited by draft length, to enable additional performance gains. More experiments are provided in Sec. D.9.

7. Conclusion

This paper introduces MINE DRAFT, a novel framework that realizes the potential of PSD. MINE DRAFT exploits the latency savings allowed by *Batch Parallelism*, which overlaps drafting and verification in SD by maintaining a target batch and a draft batch. Our theoretical analysis characterizes the expected minimum end-to-end latency gains achievable with PSD. Extensive experiments across diverse model families, model sizes, hyperparameter settings, and datasets further validate the performance advantages of MINE DRAFT in practice. As future work, we will explore extending support to the vLLM v1 engine and the chunked prefill feature (Agrawal et al., 2023). We also plan to study whether the current requirement for an additional GPU in Batch Parallelism can be eliminated by leveraging the weight-padding method described in Appendix F of (Liu et al., 2025a).

References

- Agrawal, A., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., and Ramjee, R. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv:2308.16369*, 2023.
- Anon, a. Sharegpt dataset. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered, 2023.
- Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proc. ICML*, pp. 5209–5235, 2024.
- Cheng, Y., Zhang, A., Zhang, X., Wang, C., and Wang, Y. Recurrent drafter for fast speculative decoding in large language models. *arXiv:2403.09919*, 2024.
- Dai, D., Deng, C., Zhao, C., Xu, R. X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y. K., Huang, P., Luo, F., Ruan, C., Sui, Z., and Liang, W. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv:2401.06066*, 2024.
- Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.
- Hou, Y., Zhang, F., Du, C., Zhang, X., Pan, J., Pang, T., Du, C., Tan, V. Y. F., and Yang, Z. Banditspec: Adaptive speculative decoding via bandit algorithms. In *Proc. ICML*, 2025.
- Huang, K., Wu, H., Shi, Z., Zou, H., Yu, M., and Shi, Q. Adaspec: Adaptive speculative decoding for fast, slow-aware large language model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing, SoCC '25*, pp. 361–374, 2025.
- Hugging Face. Transformers v4.53.3. <https://github.com/huggingface/transformers/releases/tag/v4.53.3>, 2025. Accessed: 2026-01-09.
- Ibanez, T. ibnztterrell/meta-llama-3.3-70b-instruct-awq-int4. <https://huggingface.co/ibnztterrell/Meta-Llama-3.3-70B-Instruct-AWQ-INT4>, 2024. Accessed: 2026-01-06.
- Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proc. SOSP*, 2023.
- Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *Proc. ICML*, pp. 19274–19286, 2023.
- Li, Y., Wei, F., Zhang, C., and Zhang, H. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proc. ICML*, pp. 28935–28948, 2024a.
- Li, Y., Wei, F., Zhang, C., and Zhang, H. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proc. EMNLP*, pp. 7421–7432, 2024b.
- Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv:2503.01840*, 2025a.
- Li, Z., Chen, Z., Delacourt, R., Oliaro, G., Wang, Z., Chen, Q., Lin, S., Yang, A., Zhang, Z., Chen, Z., Lai, S., Cheng, X., Miao, X., and Jia, Z. Adaserve: Accelerating multi-slo llm serving with slo-customized speculative decoding. *arXiv:2501.12162*, 2025b.
- Liu, T., Li, Y., Lv, Q., Liu, K., Zhu, J., Hu, W., and Sun, X. PEARL: Parallel speculative decoding with adaptive draft length. In *Proc. ICLR*, 2025a.
- Liu, X., Park, J., Hu, L., Kwon, W., Li, Z., Zhang, C., Du, K., Mo, X., You, K., Cheung, A., Deng, Z., Stoica, I., and Zhang, H. Turbospec: Closed-loop speculation control system for optimizing llm serving goodput. *arXiv:2406.14066*, 2025b.
- NVIDIA. NVIDIA Nsight Systems 2024.2. <https://developer.nvidia.com/nsight-systems>, 2024. Accessed: 2026-01-20.
- Oliaro, G., Jia, Z., Campos, D., and Qiao, A. Suffixdecoding: Extreme speculative decoding for emerging ai applications. In *Proc. NeurIPS*, 2025.
- PyTorch. Pytorch 2.7.0. <https://github.com/pytorch/pytorch/releases/tag/v2.7.0>, 2025. Accessed: 2026-01-09.
- Ryu, H. and Kim, E. Closer look at efficient inference methods: A survey of speculative decoding. *arXiv:2411.13157*, 2024.
- Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proc. ICLR*, 2017.
- Timor, N., Mamou, J., Korat, D., Berchansky, M., Pereg, O., Wasserblat, M., Galanti, T., Gordon-Kiwkowitz, M., and Harel, D. Distributed speculative inference (DSI): Speculation parallelism for provably faster lossless language model inference. In *Proc. ICLR*, 2025.

- 385 vLLM. [performance]: vllm eagle performance is
386 worse than expected. [https://github.com/
387 vllm-project/vllm/issues/9565](https://github.com/vllm-project/vllm/issues/9565), 2024. Ac-
388 cessed: 2026-01-09. (Datasets and Benchmarks Track), pp. 46595–46623,
2023.
- 389 vLLM. vllm v0.9.2. [https://github.com/
390 vllm-project/vllm/releases/tag/v0.9.2](https://github.com/vllm-project/vllm/releases/tag/v0.9.2),
391 2025. Accessed: 2026-01-09.
- 392
- 393 Wang, J., Su, Y., Li, J., Xia, Q., Ye, Z., Duan, X., Wang,
394 Z., and Zhang, M. OPT-tree: Speculative decoding with
395 adaptive draft tree structure. *Transactions of the Associa-
396 tion for Computational Linguistics*, 13:188–199, 2025.
- 397
- 398 Wang, S., Yang, H., Wang, X., Liu, T., Wang, P., Liang,
399 X., Ma, K., Feng, T., You, X., Bao, Y., Liu, Y., Luan,
400 Z., and Qian, D. Minions: Accelerating large language
401 model inference with aggregated speculative execution.
402 *arXiv:2402.15678*, 2024.
- 403
- 404 Wu, Z., Zhou, Z., Verma, A., Prakash, A., Rus, D., and Low,
405 B. K. H. TETRIS: Optimal draft token selection for batch
406 speculative decoding. In *Proc. ACL*, pp. 33329–33345,
407 2025.
- 408
- 409 Xia, H., Yang, Z., Dong, Q., Wang, P., Li, Y., Ge, T., Liu, T.,
410 Li, W., and Sui, Z. Unlocking efficiency in large language
411 model inference: A comprehensive survey of speculative
412 decoding. In *Proc. ACL Findings*, pp. 7655–7671, 2024.
- 413
- 414 Xiao, Z., Zhang, H., Ge, T., Ouyang, S., Ordonez, V., and
415 Yu, D. Parallelspec: Parallel drafter for efficient specula-
416 tive decoding. *arXiv:2410.05589*, 2024.
- 417
- 418 Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B.,
419 Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical
420 report. *arXiv:2505.09388*, 2025.
- 421
- 422 YAV-AI. Llm-tough-questions dataset: A collection
423 of complex questions across 100 domains. [https://
424 huggingface.co/datasets/YAV-AI/
425 llm-domain-specific-tough-questions](https://huggingface.co/datasets/YAV-AI/llm-domain-specific-tough-questions),
2024.
- 426
- 427 Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-
428 G. Orca: A distributed serving system for {Transformer-
429 Based} generative models. In *Proc. OSDI*, pp. 521–538,
430 2022.
- 431
- 432 Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G.,
433 and Mehrotra, S. Draft& verify: Lossless large language
434 model acceleration via self-speculative decoding. In *Proc.
435 ACL*, pp. 11263–11282, 2024.
- 436
- 437 Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z.,
438 Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., Zhang, H.,
439 Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge
with mt-bench and chatbot arena. In *Proc. NeurIPS*

A. Leftover Proof

We first derive an upper bound on T_{SD} that does not involve the expectation operator. Under a mild assumption on the form of f , we have the following result.

Lemma 1. *Let $f(t) = 1 - e^{-\alpha t}$ for some constant $\alpha \in \mathbb{R}^+$. We have that*

$$T_{SD} > \frac{\frac{1}{\alpha} \ln(\alpha V + 1) + V}{1 - \frac{1}{2(\alpha V + 1)}} \times R.$$

Proof. First, the form of $f(x)$ is monotone increasing in the domain of α . Hence, the function $g(k) = \frac{k+V}{f(k)}$ has a global minimum, taken at its turning point (critical point theorem). In the following, we derive this turning point $g'(k) = 0$. Taking the derivative with respect to k and substituting the formula of $f(x)$, we have

$$g'(k) = 0 \iff (k+V)f'(k) = f(k) \iff \alpha(k+V)e^{-\alpha k} = 1 - e^{-\alpha k}.$$

Rearrange the last equation:

$$(\alpha(k+V) + 1)e^{-\alpha k} = 1.$$

We use the Lambert W function to solve this equation. Specifically, the Lambert W function is defined as the inverse function of $f(w) = we^w$, i.e.,

$$z = we^w \iff w = W(z).$$

With this, let $u := \alpha(k+V) + 1$. Then, the equation above can be rewritten as $-ue^{-u} = -e^{-(\alpha V + 1)}$. So,

$$u = -W(-e^{-(\alpha V + 1)}),$$

and since $k = \frac{u}{\alpha} - (V + \frac{1}{\alpha})$, we have

$$k^* = -\frac{1}{\alpha} W(-e^{-(\alpha V + 1)}) - (V + \frac{1}{\alpha}).$$

For $z \in (-e^{-1}, 0)$, there are two real branches: $W_0(z) \in (-1, 0)$ and $W_{-1}(z) \in (-\infty, -1]$. Here $z = -e^{-(\alpha V + 1)} \in (-e^{-1}, 0)$ and $u = \alpha(k+V) + 1 \geq 1$. If we took W_0 , we would have $u = -W_0(z) \in (0, 1)$, a contradiction. Therefore, we take W_{-1} . Let $m := \alpha V + 1$. The expansion for the -1 branch as $m \rightarrow \infty$ is

$$W_{-1}(z) = \ln(-z) - \ln(-\ln(-z)) + \frac{\ln(-\ln(-z))}{\ln(-z)} + \dots.$$

Substitute $z = -e^{-m}$, we have

$$W_{-1}(-e^{-m}) = -m - \ln m - \frac{\ln m}{m} + \mathcal{O}\left(\frac{(\ln m)^2}{m^2}\right).$$

Plugging this equation to the equation of k^* , we have

$$\begin{aligned} k^* &= \frac{1}{\alpha} \left[\ln m + \frac{\ln m}{m} + \mathcal{O}\left(\frac{(\ln m)^2}{m^2}\right) \right] \\ &= \frac{1}{\alpha} \ln(\alpha V + 1) + \frac{1}{\alpha} \frac{\ln(\alpha V + 1)}{\alpha V + 1} + \mathcal{O}\left(\frac{(\ln(\alpha V))^2}{\alpha(\alpha V)^2}\right). \end{aligned}$$

With this, we now bound the range of k^* . First, recall that $(\alpha(k+V) + 1)e^{-\alpha k} = 1$. We rearrange the equation of k into

$$\alpha k = \ln(\alpha(k+V) + 1).$$

Let $y := \alpha k$ and define $g(y) := \ln(\alpha V + 1 + y) - y$. Then, $g'(y) = \frac{1}{\alpha V + 1 + y} - 1 < 0$, so g is strictly decreasing and has a unique zero y^* . Define $y_0 := \ln(\alpha V + 1)$ and $y_1 := \ln(2(\alpha V + 1))$. We now show that $y_0 < y^* < y_1$.

At y_0 , we have $g(y_0) = \ln(1 + \frac{\ln(\alpha V + 1)}{\alpha V + 1}) > 0$. This implies $y^* > y_0$.

At y_1 , $g(y_1) = \ln(1/2 + \frac{\ln(2(\alpha V + 1))}{2(\alpha V + 1)})$. For $x \geq 1$, $\ln(2x) \leq x$. Hence, $1/2 + \ln(2x)/(2x) < 1$. Therefore, $g(y_1) < 0$ for $\alpha V > 0$. This implies $y^* < y_1$. Therefore, substitute $k = y/\alpha$, we have

$$\frac{1}{\alpha} \ln(\alpha V + 1) < k^* < \frac{1}{\alpha} \ln(2(\alpha V + 1)).$$

As such, we have a bound for T_{SD} as

$$T_{SD} = \frac{k^* + V}{f(k^*)} > \frac{\frac{1}{\alpha} \ln(\alpha V + 1) + V}{f(\frac{1}{\alpha} \ln(2(\alpha V + 1)))} = \frac{\frac{1}{\alpha} \ln(\alpha V + 1) + V}{1 - \frac{1}{2(\alpha V + 1)}}. \quad \square$$

On the other hand, under PSD, the expected time for verification is

$$\begin{aligned} T_{PSD} &:= \min_t \mathbb{E}[\sum_{s=1}^N \max(V, t)] \\ &= \min_t \mathbb{E}[N] \max(V, t) f(t). \end{aligned}$$

While it is obvious from the two definitions that $T_{PSD} < T_{SD}$, we prove a stronger result:

Lemma 2. *Similarly, let $f(t) = 1 - e^{-\alpha t}$ for some constant $\alpha \in \mathbb{R}^+$. We have that*

$$T_{PSD} = \frac{RV}{1 - e^{-\alpha V}}$$

Proof. since $f(k)$ is increasing, it is easy to observe that $k^* \geq V$. Consider that for $k \geq V$, we have $(\max(V, k))' \geq f'(k)$. Hence, $k^* = V$. Substitute the value, we have

$$T_{PSD} = \frac{\max(V, t)}{f(V)} = \frac{V}{1 - e^{-\alpha V}}. \quad \square$$

Theorem 1. *Let W be the Lambert W function and $f(t) = 1 - e^{-\alpha t}$ for some constant $\alpha \in \mathbb{R}^+$. For $\alpha V \geq -W_{-1}(-\frac{1}{2e}) - 1 \approx 1.68$, we have $T_{SD} > 1.59 T_{PSD}$.*

Proof. Using Lemma 1 and Lemma 2,

$$\frac{T_{SD}}{T_{PSD}} = \frac{\frac{1}{\alpha} \ln(\alpha V + 1) + V}{V} \frac{1 - e^{-\alpha V}}{1 - \frac{1}{2(\alpha V + 1)}}.$$

We are interested in when $\frac{1 - e^{-\alpha V}}{1 - \frac{1}{2(\alpha V + 1)}} \geq 1$, i.e., $e^{-\alpha V} \leq \frac{1}{2(\alpha V + 1)}$. Thus, we solve

$$e^{-\alpha x} = \frac{1}{2(\alpha x + 1)}, \alpha > 0, x > 0.$$

set $t = \alpha x + 1 \implies x = (t - 1)/\alpha$. Then, $e^{-\alpha x} = e^{-(t-1)} = e^{1-t}$. So,

$$2te^{1-t} = 1 \iff te^{-t} = \frac{1}{2e}.$$

Rewrite $(-t)e^{-t} = -\frac{1}{2e}$ and use the definition of Lambert W function, $W(z)e^{W(z)} = z$:

$$-t = W(-\frac{1}{2e}) \implies t = -W(-\frac{1}{2e}).$$

Therefore,

$$x = \frac{-W(-\frac{1}{2e}) - 1}{\alpha}.$$

Therefore, substitute back $V = x$, we have that when $V > \frac{-W(-\frac{1}{2e}) - 1}{\alpha}$, $\frac{1 - e^{-\alpha V}}{1 - \frac{1}{2(\alpha V + 1)}} \geq 1$. Further substitute the numerical solution of $W_{-1}(-\frac{1}{2e}) \approx -2.68$, we have that, when $\alpha V \geq 1.68$, $\frac{T_{SD}}{T_{PSD}} > \frac{\frac{1}{\alpha} \ln(\alpha V + 1) + V}{V} = 1 + \frac{\ln(\alpha V + 1)}{\alpha V}$. Since the function $y = \frac{\ln(x+1)}{x}$ is monotone decreasing, the minimum value is taken at $\alpha V \approx 1.68$, giving $1 + \frac{\ln(\alpha V + 1)}{\alpha V} \approx 1 + 0.59 = 1.59$. \square

B. Additional Related Work

Speculative decoding. SD has been receiving increasing attention as an effective method to accelerate LLM inference by decomposing autoregressive decoding into a drafting phase and a verification phase (Leviathan et al., 2023; Ryu & Kim, 2024; Xia et al., 2024). Prior work has focused on improving SD performance in both single-request settings (Cai et al., 2024; Cheng et al., 2024; Zhang et al., 2024; Li et al., 2025a; Oliaro et al., 2025) and batch-request settings (Li et al., 2025b; Liu et al., 2025b; Wu et al., 2025), primarily by improving drafting efficiency. In contrast, MINEDRAFT explores an orthogonal direction by parallelizing the drafting and verification phases, allowing easy integration with existing methods to further improve SD performance.

C. Leftover Details about MINEDRAFT

C.1. Drafter and Verifier

Start-up of parallelism. Batch Parallelism uses a *Drafter* and a *Verifier*. In the first SD step, the Drafter generates drafts for the target batch and sends them to the Verifier. While the Verifier processes the target batch, the Drafter concurrently proposes draft tokens for the draft batch. After verification, the Verifier returns the sampled tokens and probability distributions (target sampler output) to the Drafter.

Batch state alternation. At the end of each SD step, the Verifier sends the target sampler output to the Drafter, which processes it and returns the output to the Scheduler. We refer to the moment when the Drafter returns output to the Scheduler as a *sync point*. At each sync point, `skip_batch` is alternated to the other batch. After the first SD step, the target batch, corresponding to the previous step’s draft batch, is verified by the Verifier, while the Drafter concurrently proposes draft tokens for the new draft batch, which corresponds to the previous step’s target batch.

Fallback mechanism. A corner case arises when one batch becomes empty as requests are finished and no new requests arrive. In this case, Batch Parallelism falls back to standard SD if draft tokens of the target batch are unavailable. This exposes a native limitation of MINEDRAFT: when the two batches cannot be replenished to maintain balance, it eventually leads to an empty batch and degrades performance. We discuss this limitation and a potential mitigation in Sec. F.

C.2. Scheduler

KV block management. In PagedAttention, a *KV block* stores the key–value vectors for a fixed number of tokens, defined by the *KV block size* (B). KV blocks associated with a request need not be contiguous in physical memory, allowing flexible memory management (Kwon et al., 2023). Before token generation, the Scheduler allocates the required KV blocks for each running request.

Over-allocation issue in default scheduler. We observe that during drafting, the Drafter accesses newly allocated KV blocks without populating them, whereas during verification, the Verifier computes and writes KV caches to the allocated blocks. The default vLLM scheduler assumes that all running requests are generated in each step and allocates KV blocks for all of them. Under Batch Parallelism, this causes redundant allocations for requests in the draft batch, which are drafted but not verified, leaving allocated blocks accessed but unwritten. We patch the vLLM scheduler’s KV block allocation logic to maintain compatibility with PagedAttention and avoid unnecessary memory allocation.

Scheduler patch. To implement this patch, we introduce a set `has_deferred` tracking request IDs for which KV block allocation skipping has been deferred. The modified allocation logic proceeds as follows. If Batch 0 or Batch 1 is empty, the Scheduler allocates KV blocks for all running requests, except for the first occurrence of this condition. When both batches are non-empty, KV blocks are allocated for prefill requests to support the next prefill iteration. For requests in the decoding stage, KV blocks are allocated if the request ID is not in `has_deferred` or if the request belongs to the current draft batch; otherwise, allocation is skipped. After this decision, the request ID of each decoding request is added to `has_deferred`. This ensures that KV block allocation is not inadvertently skipped for requests in the target batch during the first SD step.

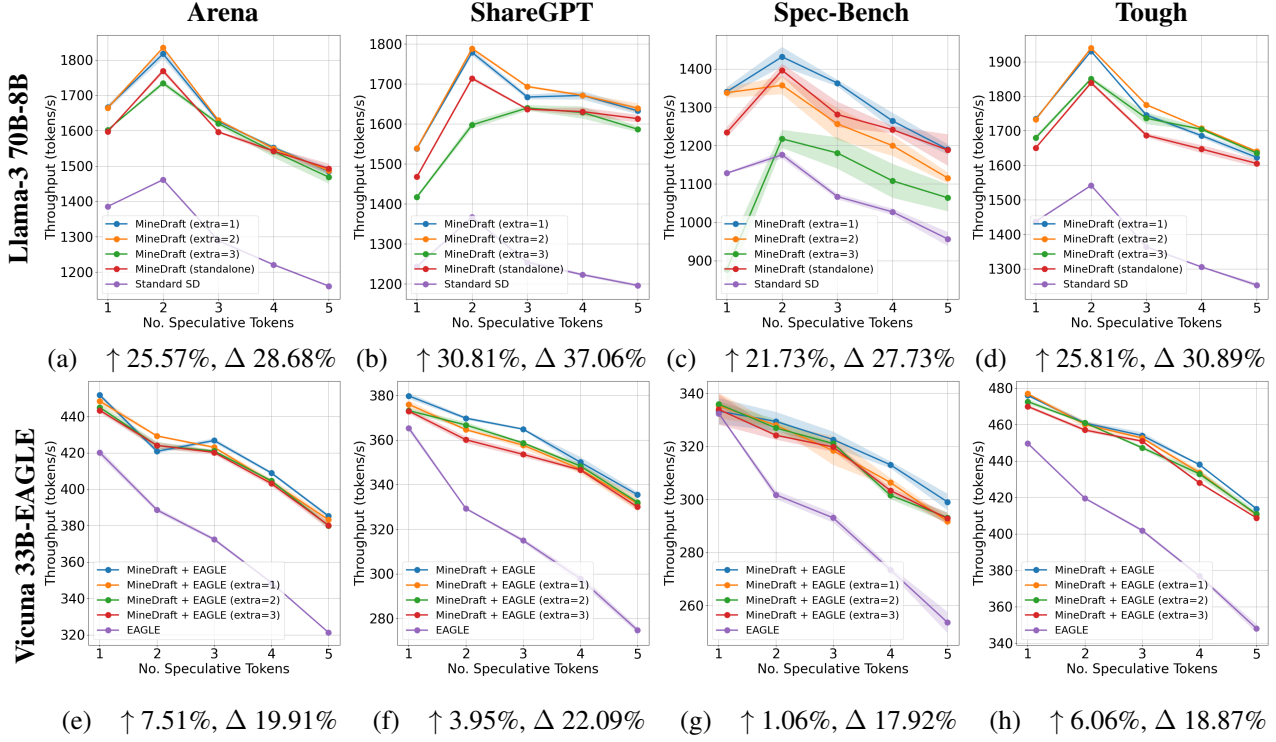


Figure 7. Throughput comparison across model settings 5 and 6. \uparrow indicates the average improvement over the best baseline method. Δ indicates the maximum average gap between MINE DRAFT and standard SD or EAGLE. MINE DRAFT consistently outperforms standalone EAGLE and standard SD, achieving maximum average throughput gains of 37.06% and 22.09%, respectively.

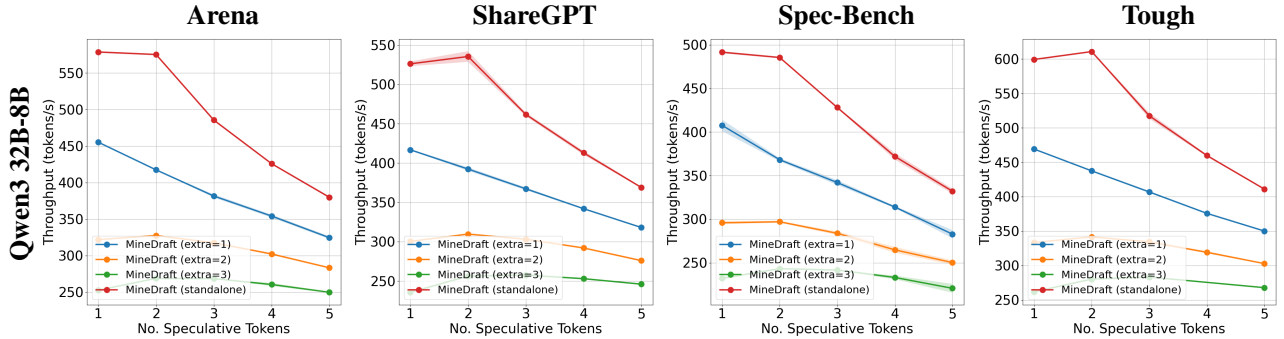


Figure 8. Throughput results on model setting 4. Standard SD experiments failed on this setting due to OOM.

Average improvement over the best baseline. First, for each performance metric p , the optimal baseline draft token count k_p^* is selected as follows:

- If p is **maximized** (higher is better): $k_p^* = \arg \max_k B_p(k)$.
- If p is **minimized** (lower is better): $k_p^* = \arg \min_k B_p(k)$.

Let $B_p^* = B_p(k_p^*)$ be the corresponding baseline performance metric value. Next, among all MINE DRAFT variants, select the best MINE DRAFT performance metric value at k_p^* as $M_p^* = M_p(k_p^*)$. The average improvement of p of MINE DRAFT over the best baseline method \uparrow_p (in percent) is calculated as follows:

- If p is **maximized**: $\uparrow_p = \frac{M_p^* - B_p^*}{B_p^*} \times 100$.

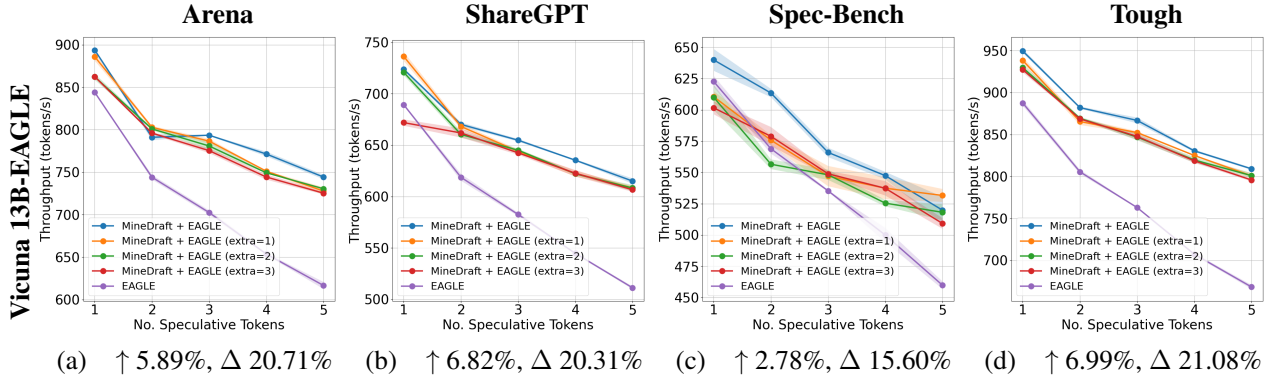


Figure 9. Throughput comparison on model setting 7. \uparrow indicates the average improvement over the best baseline method. Δ indicates the maximum average gap between MINEDRAFT and EAGLE.

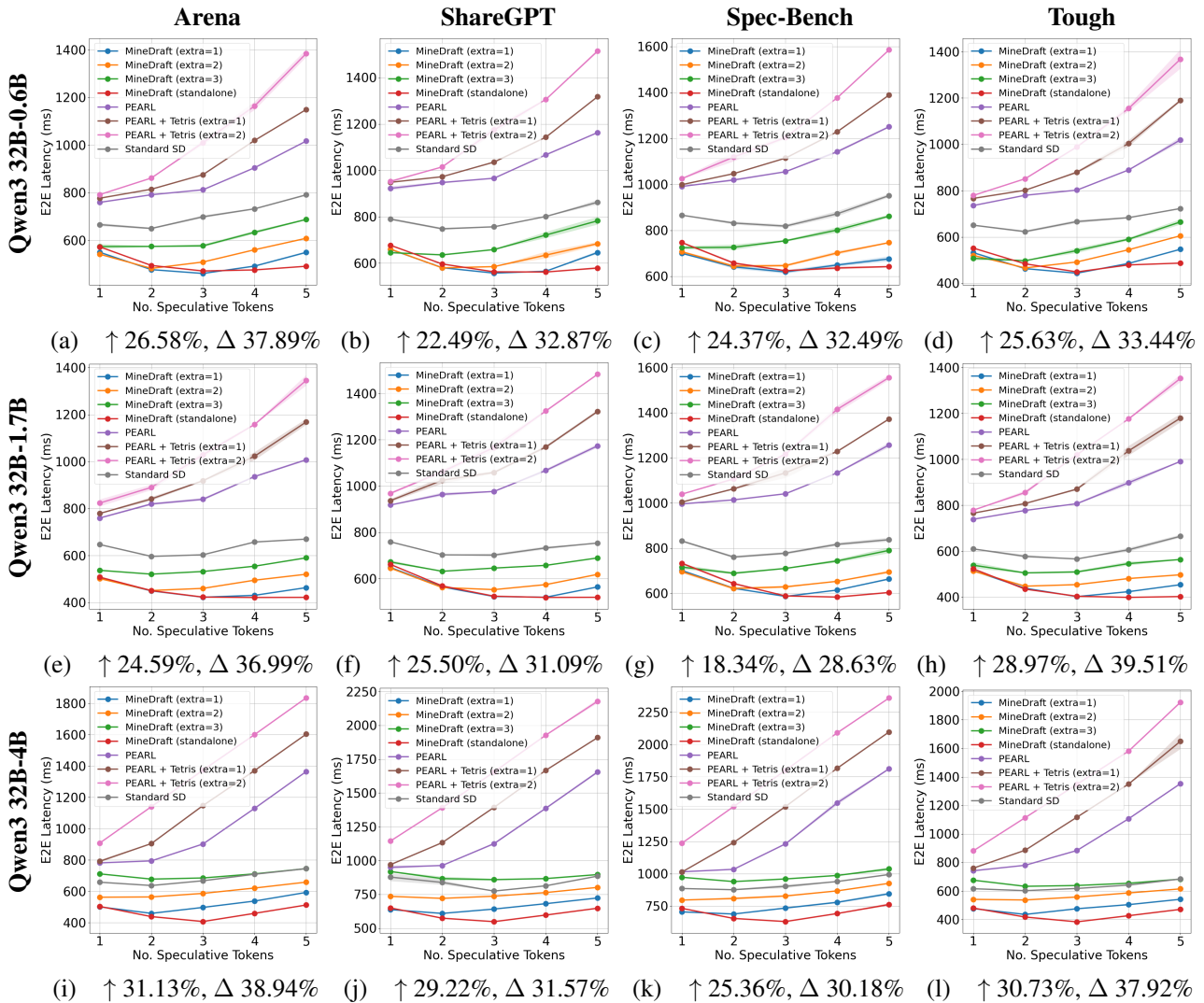


Figure 10. End-to-End Latency (ms) comparison against baseline methods across model settings 1-3. \uparrow indicates the average improvement over the best baseline method. Δ indicates the maximum average gap between MINEDRAFT and standard SD.

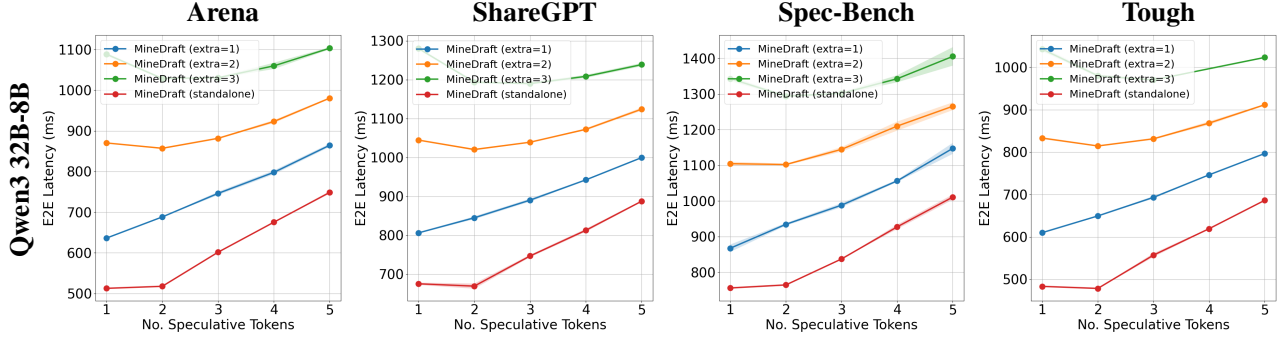


Figure 11. End-to-End Latency (ms) results on model setting 4.

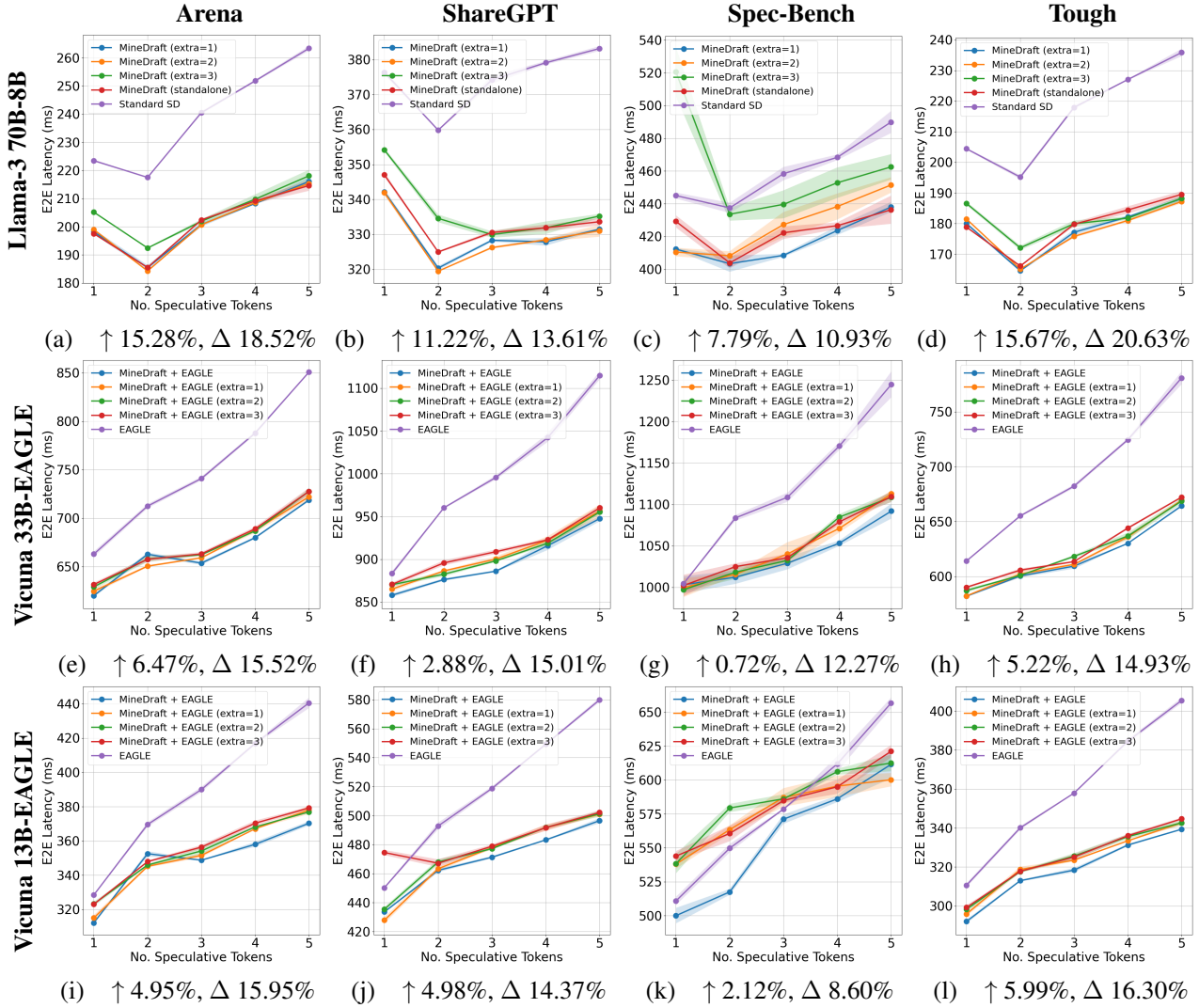


Figure 12. End-to-End Latency (ms) comparison across model settings 5–7. ↑ indicates the average improvement over the best baseline method. Δ indicates the maximum average gap between MINE DRAFT and standard SD or EAGLE*.

• If p is **minimized**: $\uparrow_p = \frac{B_p^* - M_p^*}{B_p^*} \times 100$.

Maximum average improvement. Define $S_p(k)$ as the value of p of standard SD or EAGLE* (Li et al., 2024a;b; 2025a),

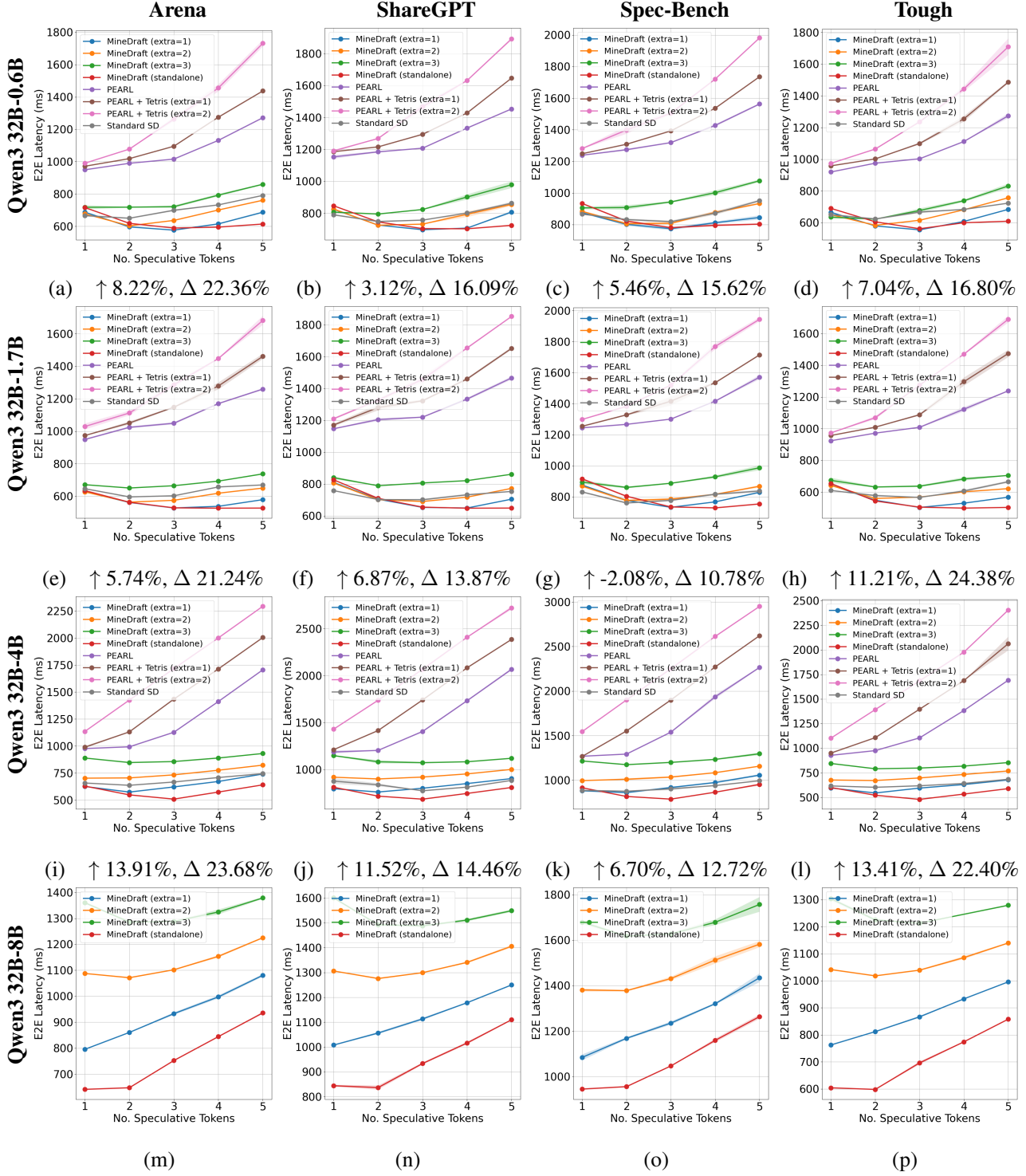


Figure 14. Normalized end-to-end latency comparison against baseline methods across model settings 1–4. \uparrow indicates the average improvement over the best baseline method. Δ indicates the maximum average gap between MINE DRAFT and standard SD. MINE DRAFT outperforms baselines with an exception when k is small (between 1 and 2), where MINE DRAFT may be underperformed compared with standard SD. Besides that, MINE DRAFT improves normalized latency by up to 13.91% over the best-performing baseline and by up to 24.38% over standard SD.

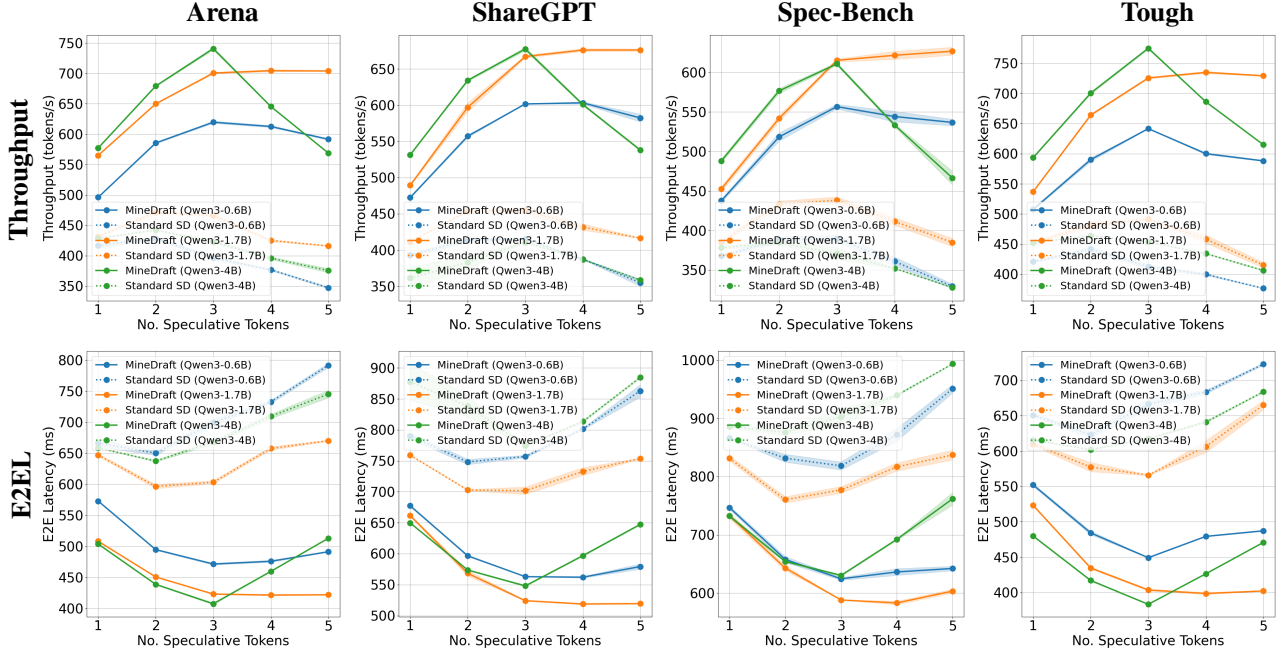


Figure 15. Comparisons for various draft models paired with Qwen3-32B using MINE DRAFT.

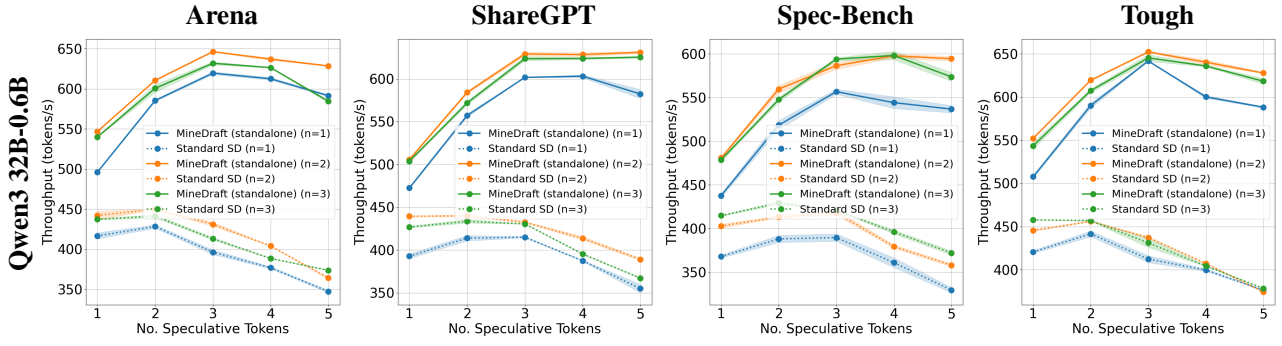


Figure 16. Throughput comparison against standard SD across various n on model setting 1.

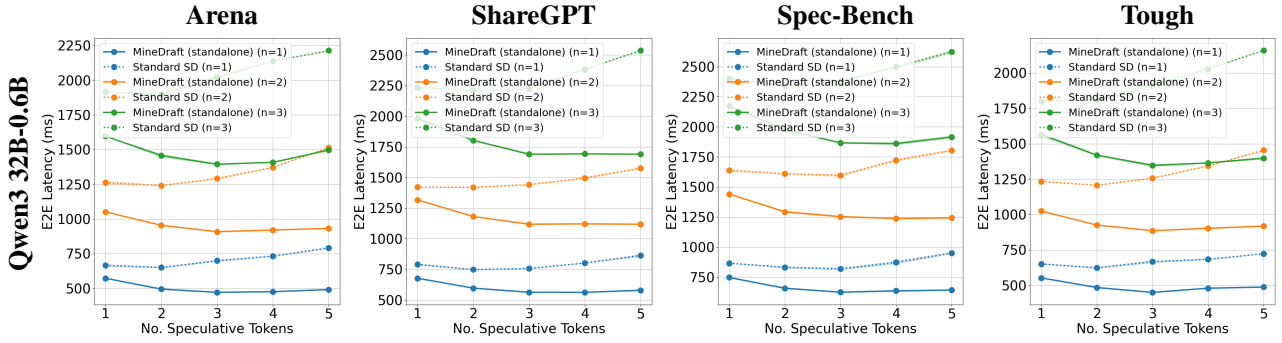


Figure 17. End-to-End Latency (ms) comparison against standard SD across various n on model setting 1.

at k ($\Delta_p(k)$) is calculated as:

- If p is maximized: $\Delta_p(k) = \frac{M_p(k) - S_p(k)}{S_p(k)} \times 100$.

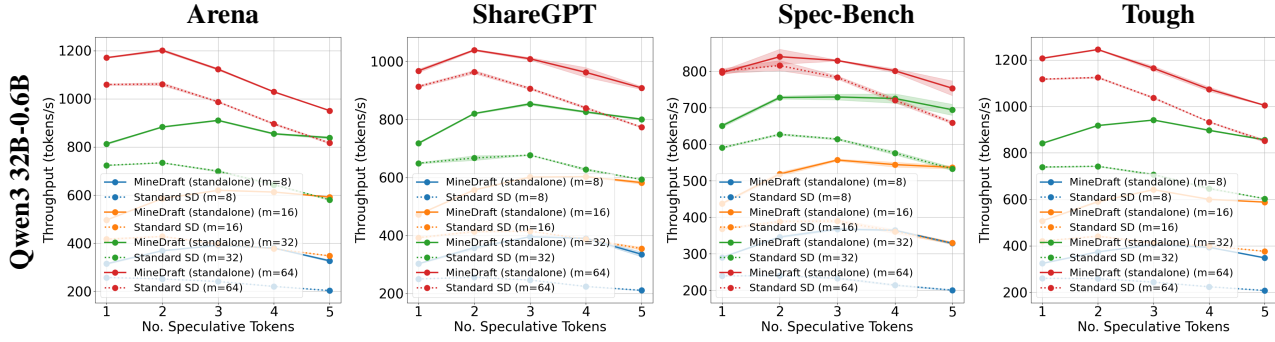


Figure 18. Throughput comparison against standard SD across various m (batch size) on model setting 1.

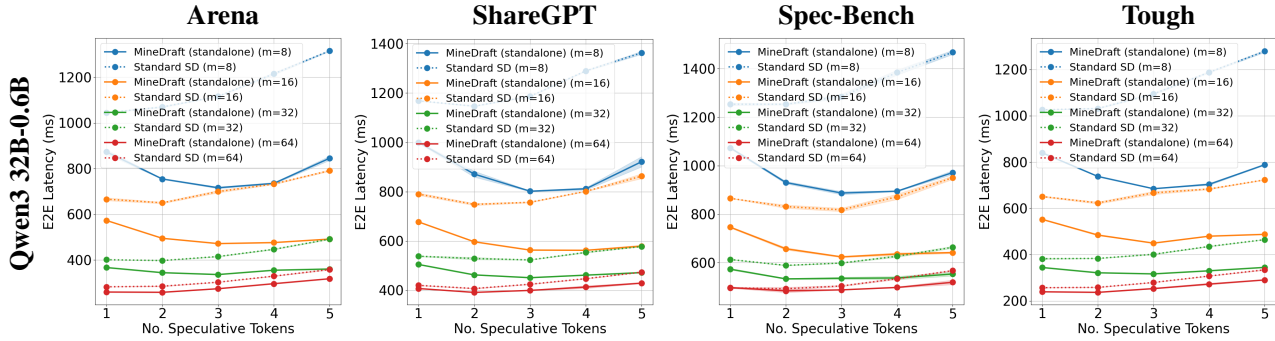


Figure 19. End-to-End Latency (ms) comparison against standard SD across various m (batch size) on model setting 1.

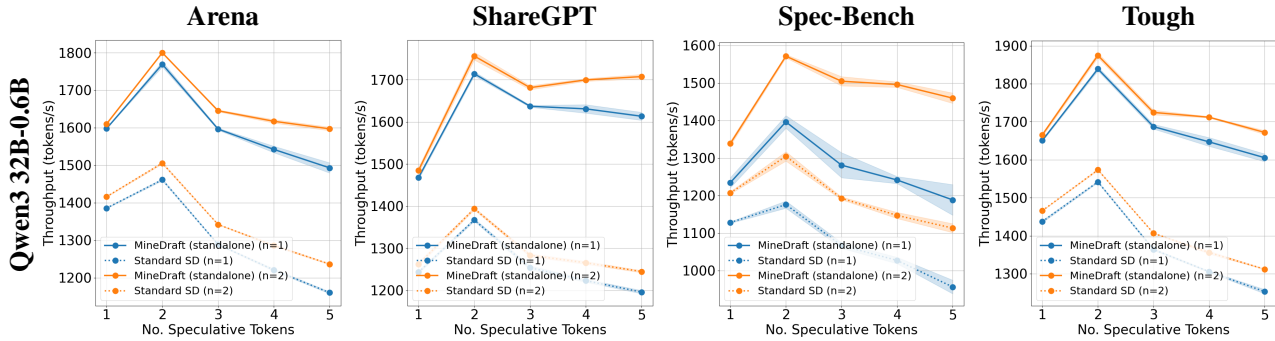


Figure 20. Throughput comparison against standard SD across various n on model setting 5.

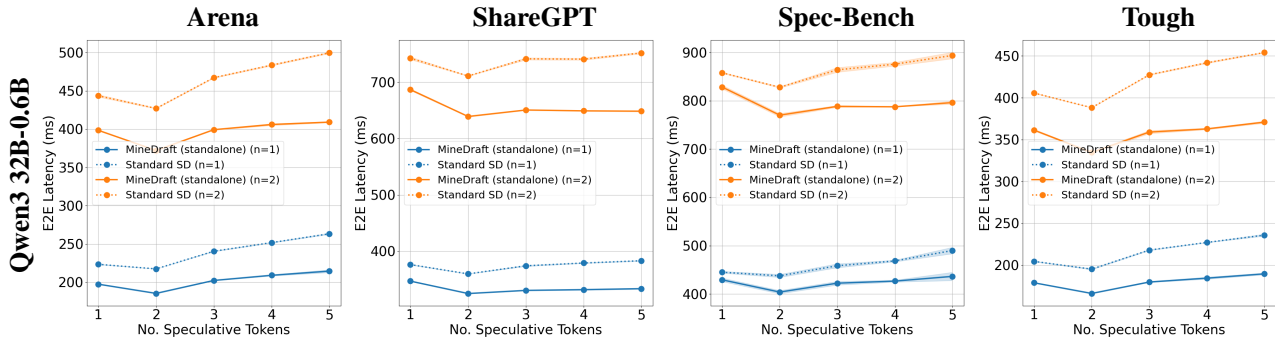


Figure 21. End-to-End Latency (ms) comparison against standard SD across various n on model setting 5.

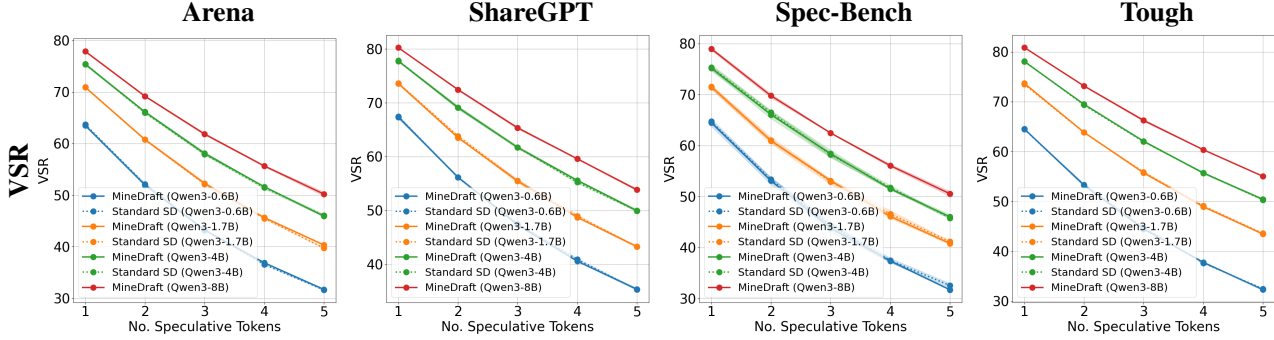


Figure 22. VSR comparisons for various draft models paired with Qwen3-32B between MINEDRAFT and standard SD. Larger draft models generally imply higher VSR.

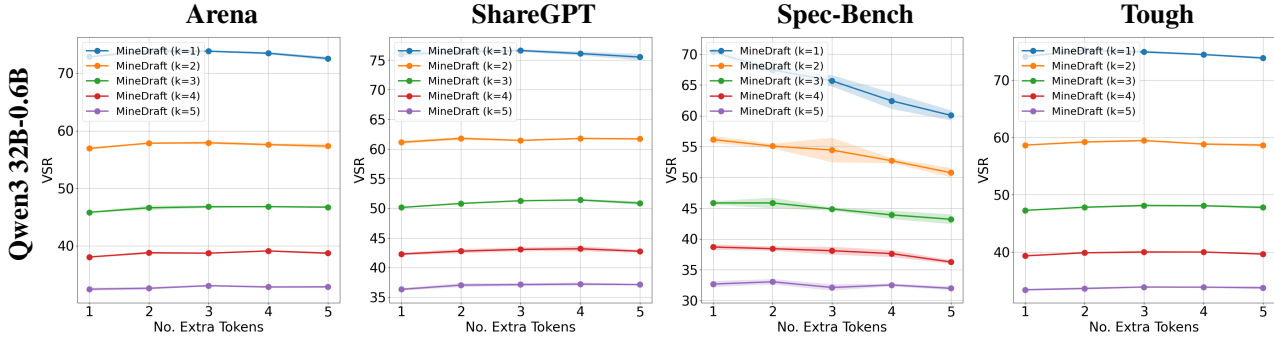


Figure 23. VSR comparison for MINEDRAFT with TETRIS across extra tokens on model setting 1.

- If p is minimized: $\Delta_p(k) = \frac{S_p(k) - M_p(k)}{S_p(k)} \times 100$.

The maximum average percentage improvement of p of MINEDRAFT over standard SD or EAGLE (Δ_p) is the maximum of $\Delta_p(k)$ over all k values: $\Delta_p = \max_k \Delta_p(k)$.

D.5. Results with Different Drafting Strategies

We further evaluate MINEDRAFT integrated with other drafting strategies, including EAGLE and TETRIS, on the other model settings. As shown in Fig. 7, integrating MINEDRAFT with TETRIS and/or EAGLE consistently outperforms standalone EAGLE and standard SD across all datasets. In particular, MINEDRAFT improves average throughput by up to 30.81% and 7.51% over the best results of standard SD and EAGLE, respectively. The maximum average improvements over standard SD and EAGLE reach 37.06% and 22.09%, respectively.

Memory contention in standard SD. As shown in Fig. 8, standard SD can encounter memory limitations when deploying larger draft models, such as Qwen3-8B, due to the coexistence of the draft and target models within the same GPU memory, which leads to VRAM contention. In contrast, the MINEDRAFT architecture places the draft model on a dedicated GPU, enabling parallel drafting without competing for memory with the target model. This design alleviates memory bottlenecks and promotes a more balanced utilization of available GPU resources. Similar memory constraints arise for standard SD when deploying large target models, such as Qwen3-235B-A22B-Instruct-2507-FP8 (Yang et al., 2025), as detailed in Sec. D.11.

EAGLE’s subpar performance. We observe that EAGLE’s performance degrades as the number of draft tokens per step increases, potentially due to the current vLLM implementation of EAGLE-based speculators, which is under investigation by the vLLM team (vLLM, 2024).

MINEDRAFT’s as a batch PSD framework. When integrated with TETRIS (denoted by “extra=” in the legends), MINEDRAFT outperforms its standalone variant under certain settings, highlighting its potential for further inference

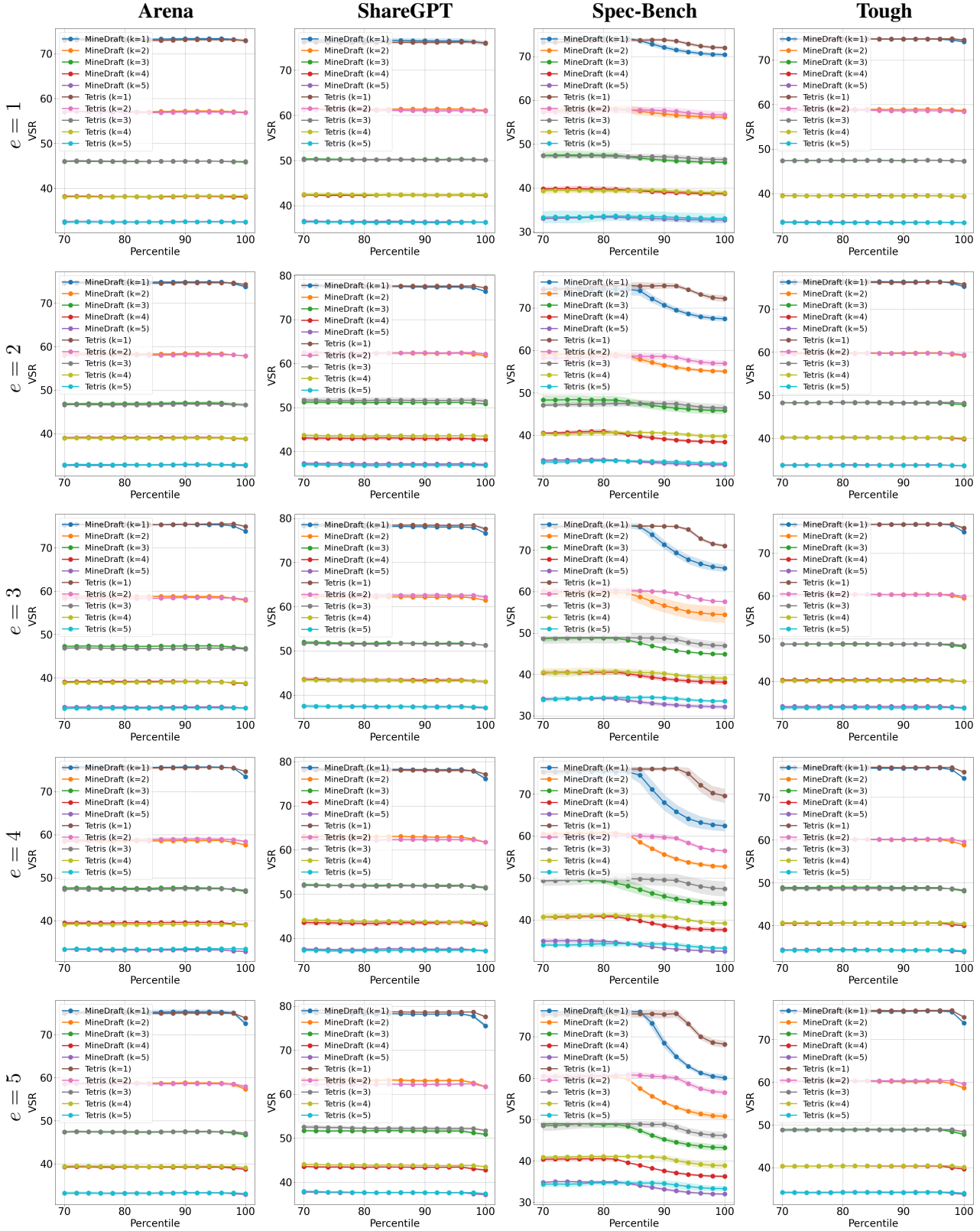


Figure 24. VSR comparison for MINE DRAFT with TETRIS and standalone TETRIS across trimming percentiles on model setting 1.

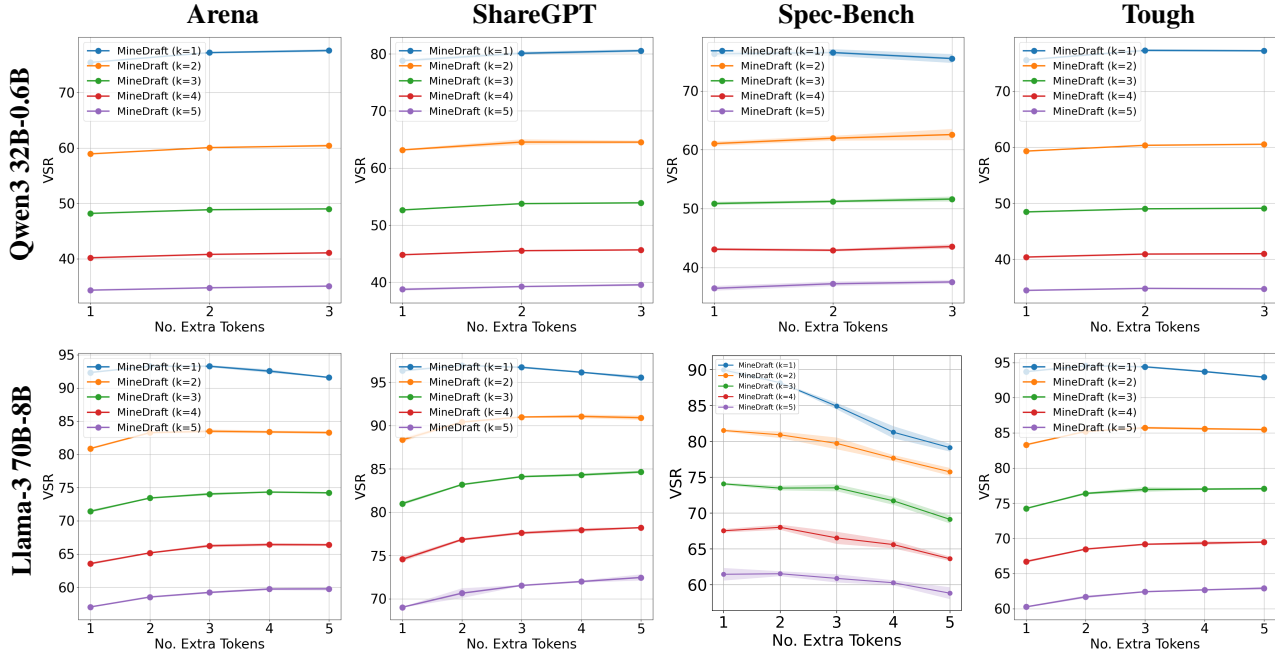


Figure 25. VSR comparison for MINE DRAFT with TETRIS across model settings 1 and 5 on $n = 2$ and temperature = 0.8.

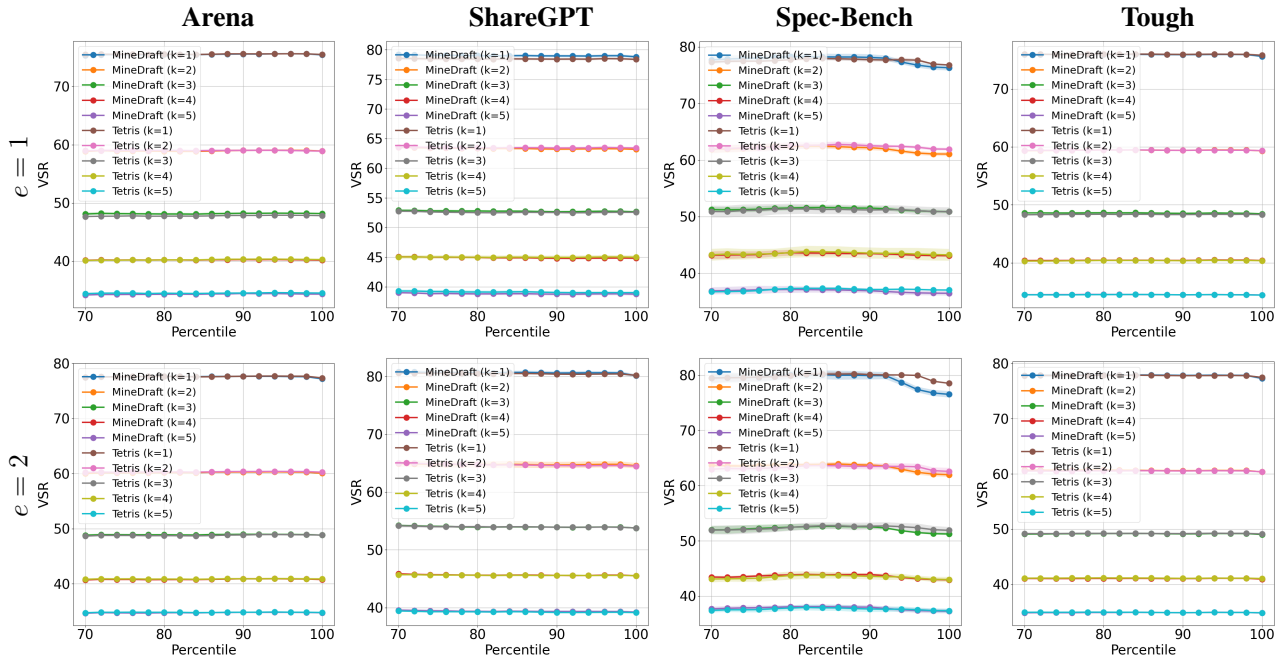


Figure 26. VSR comparison for MINE DRAFT with TETRIS and standalone TETRIS across trimming percentiles on model setting 1, $n = 2$ and temperature = 0.8.

optimization when paired with effective drafting strategies. A more detailed breakdown of performance differences between integrations with drafting strategies and the standalone MINE DRAFT implementation is left for future work.

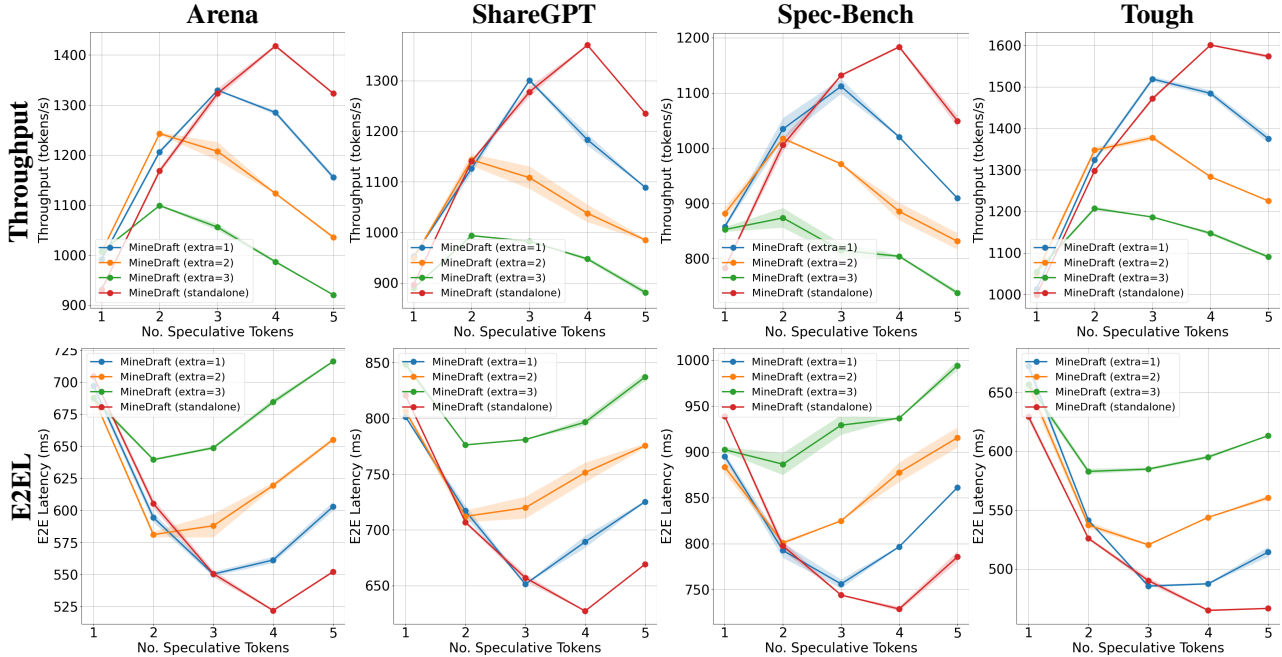


Figure 27. Minedraft Throughput and End-to-End Latency (ms) results on Qwen3-235B-A22B-Instruct-2507-FP8 paired with Qwen3-13B, $n = 2$ and temperature = 0.8. The reported numbers reflect the mean and standard deviation over 2 independent trials.

D.6. End-to-End Latency Results

Comparison with baselines. Fig. 10 shows that Minedraft reduces end-to-end latency by up to 31.13% relative to the best-performing baseline. The maximum reduction compared to standard SD reaches 39.51%, consistent with the theoretical analysis in Sec. 4, which predicts a minimum improvement of 37%.

Results with larger draft model. As shown in Fig. 11, under model setting 4 with Qwen3-8B as the draft model, Minedraft achieves its lowest end-to-end latency when generating 1–2 draft tokens per step. Drafting more than two tokens per step leads to degraded performance. In this setting, integrating TETRIS does not yield additional reductions in end-to-end latency, as the computational cost of drafting already dominates that of verification. Standard SD fails to execute on this setting due to VRAM constraints. The effect of draft model size on performance is further examined in Sec. D.8.

Integration with different drafting strategies. Fig. 12 shows that Minedraft achieves a 15.67% reduction in end-to-end latency relative to the best standard SD result on Setting 5, with a maximum improvement of 20.63%. On the selected EAGLE model settings, Minedraft outperforms the best standalone EAGLE result by up to 6.47% and attains a maximum latency reduction of 16.30%.

D.7. Normalized Results w.r.t GPU Count

Given that we evaluate parallel methods like Minedraft and PEARL with 5 GPUs and standard SD with 4 GPUs, we provide a fairer comparison by strictly aligning the number of GPUs used during the two sets of experiments. We normalize average throughput by dividing the metric values of Minedraft and PEARL by 5 and those of standard SD by 4, and normalize end-to-end latencies by multiplying the latencies of Minedraft and PEARL by 1.25. Figs. 13 and 14 show that Minedraft outperforms standard SD and PEARL except when k is small (between 1 and 2), where Minedraft may underperform compared to standard SD. Overall, Minedraft improves normalized average throughput by maximally 40.55% and end-to-end latency by maximally 24.38% over standard SD.

D.8. Ablation on Draft Model Size

We fix the target model to Qwen3-32B and evaluate Minedraft using three draft models: Qwen3-0.6B, Qwen3-1.7B, and Qwen3-4B. As shown in Fig. 15, performance degrades noticeably when the number of draft tokens per step exceeds 3 for

the Qwen3-4B and Qwen3-0.6B draft models, whereas the medium-sized Qwen3-1.7B model is less affected.

These results highlight the need to balance draft quality and speed in MINEDRAFT. As shown in Fig. 22, an overly small draft model (e.g., Qwen3-0.6B) may fail to generate sufficiently accurate drafts, leading to a low verification success rate and degraded performance. Conversely, an overly large draft model (e.g., Qwen3-4B), while capable of producing high-quality drafts, may generate them too slowly to benefit from PSD, which requires drafting to be faster than verification. As shown in Fig. 15, Qwen3-1.7B strikes this balance by producing accurate drafts at sufficient speed to exploit parallelized drafting in MINEDRAFT. This observation is further supported by Nsight Systems (NVIDIA, 2024) profiling with Spec-Bench in Sec. D.12.

D.9. Empirical Results with Varied Sequences per Request and Batch Sizes.

Using Setting 1, we evaluate MINEDRAFT across varying numbers of sequences per request (n) and batch sizes (m). As shown in Figs. 16 to 19, MINEDRAFT consistently improves both average throughput and end-to-end latency over standard SD across all tested configurations of n and m . Specifically, varying n , MINEDRAFT achieves up to 72.64% improvement in throughput and 38.41% reduction in latency; varying m , the corresponding improvements reach 75.81% and 40.79%.

As shown in Figs. 18 and 19, when the number of draft tokens per step reaches five, the performance of MINEDRAFT becomes comparable to that of standard SD with a doubled batch size ($2m$). This result highlights the effectiveness of MINEDRAFT as a batch PSD framework: with appropriate adaptive drafting strategies and model configurations, MINEDRAFT can further optimize performance even when a large number of draft tokens are generated per step.

D.10. Analysis of Verification Success Rate

The verification success rate (VSR) is defined as follows:

$$VSR = \frac{\text{Accepted tokens}}{\text{Tokens sent for verification}},$$

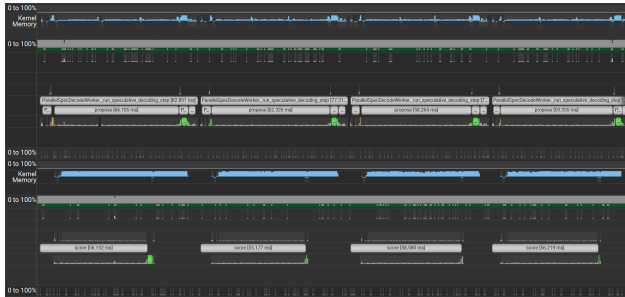
which is the proportion of tokens successfully verified out of the total number of tokens submitted for verification. VSR measures the quality of the selected draft tokens. We analyze the VSR of MINEDRAFT integrated with TETRIS under Settings 1 and 5. As shown in Figs. 23 and 25, VSR generally increases with the number of extra tokens, except on the Spec-Bench dataset, where VSR decreases more frequently than on the other datasets.

We attribute this anomaly to the combined effects of Spec-Bench’s smaller sample size and TETRIS’s behavior on non-full batches. Because Spec-Bench contains fewer conversations than the other datasets, its requests are exhausted more quickly, causing both batches to fall below their nominal capacity m earlier in the lifespan of MINEDRAFT. When TETRIS operates on non-full batches, the reduced pool of candidate draft tokens increases the likelihood of selecting lower-quality drafts, which, in turn, raises the target model’s rejection rate. As a result, VSR decreases during non-full-batch operation; since a larger fraction of SD steps occur under this regime, the aggregate VSR is negatively affected, effectively diluting the high VSR observed during the initial, fully saturated steps.

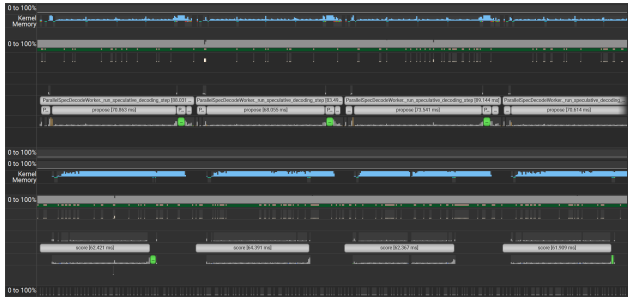
To validate this explanation for the VSR degradation observed on Spec-Bench, we compute VSR over truncated portions of the execution, considering only the initial 70%–100% or 80%–100% of SD steps. As shown in Figs. 24 and 26, on Spec-Bench, the VSR of MINEDRAFT integrated with TETRIS remains comparable to or higher than that of standalone TETRIS until approximately 82–86% of SD steps for $n = 1$, and until approximately 92% for $n = 2$. In contrast, on the Arena, ShareGPT, and Tough datasets, the VSR of MINEDRAFT with TETRIS matches that of standalone TETRIS over the first 98% of SD steps for $n = 1$, and throughout the entire evaluation for $n = 2$.

D.11. Additional Experiment Results on Larger Models

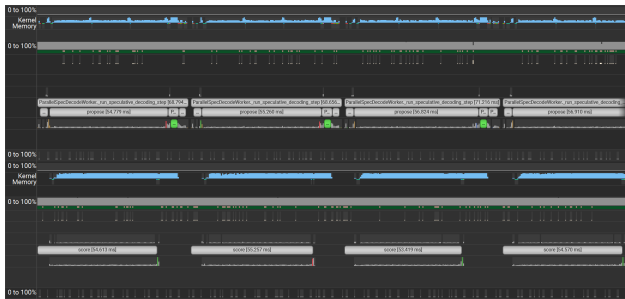
We conduct additional experiments on a large Mixture-of-Experts (MoE) target model (Shazeer et al., 2017; Dai et al., 2024; Yang et al., 2025), Qwen3-235B-A22B-Instruct-2507-FP8 (Yang et al., 2025), to evaluate the compatibility of MINEDRAFT with modern MoE architectures. We use Qwen3-14B (Yang et al., 2025) as the draft model with $m = 64$, $n = 2$, and the temperature set to 0.8, served on five H100 GPUs. Under this configuration, standard SD is infeasible due to VRAM constraints. As shown in Fig. 27, MINEDRAFT achieves its best performance when four draft tokens are generated per step, reaching an average throughput of approximately 1200–1600 tokens per second.



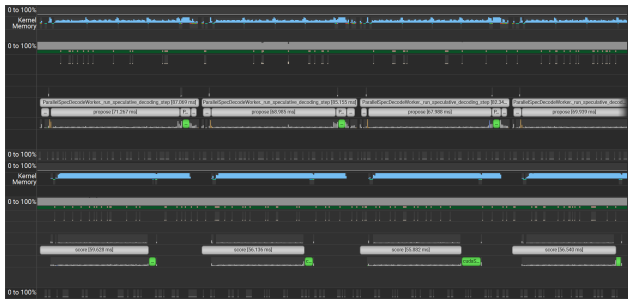
(a) Sample segment of Nsight Systems profiling results with model setting 1, Spec-Bench dataset, and $k = 3$. Median drafting and verification times are 49.60 ms and 48.62 ms, respectively, suggesting MINEDRAFT almost perfectly overlaps drafting with verification under this configuration.



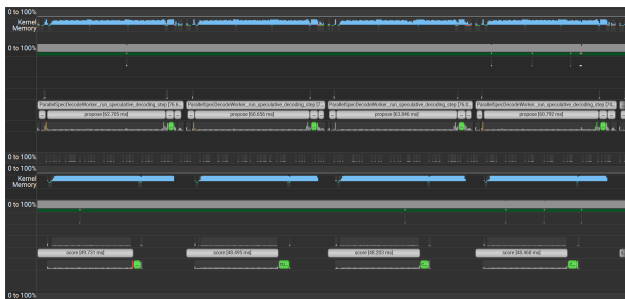
(b) Sample segment of Nsight Systems profiling results with model setting 1, Spec-Bench dataset, and $k = 4$. Median drafting and verification times are 64.06 ms and 51.06 ms, respectively, suggesting drafting time exceeds verification time by 25.48%, and the performance of MINEDRAFT may degrade under this configuration.



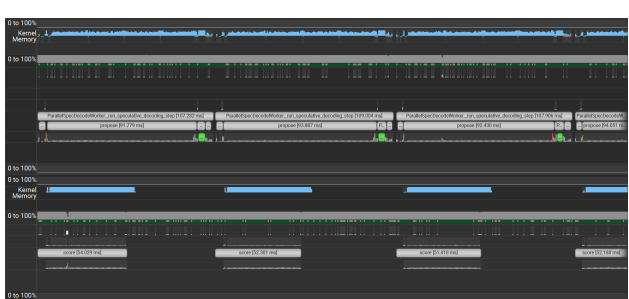
(c) Sample segment of Nsight Systems profiling results with model setting 2, Spec-Bench dataset, and $k = 3$. Median drafting and verification times are 49.53 ms and 49.30 ms, respectively, suggesting MINEDRAFT almost perfectly overlaps drafting with verification under this configuration.



(d) Sample segment of Nsight Systems profiling results with model setting 2, Spec-Bench dataset, and $k = 4$. Median drafting and verification times are 65.75 ms and 50.70 ms, respectively, suggesting drafting time exceeds verification time by 29.68%, and the performance of MINEDRAFT may degrade under this configuration.



(e) Sample segment of Nsight Systems profiling results with model setting 3, Spec-Bench dataset, and $k = 3$. Median drafting and verification times are 61.83 ms and 47.62 ms, respectively, suggesting drafting time exceeds verification time by 29.86%, and the performance of MINEDRAFT may degrade under this configuration.



(f) Sample segment of Nsight Systems profiling results with model setting 3, Spec-Bench dataset, and $k = 4$. Median drafting and verification times are 81.00 ms and 48.93 ms, respectively, suggesting drafting time exceeds verification time by 65.52%, and the performance of MINEDRAFT is highly likely to degrade under this configuration.

Figure 28. Comparison of Nsight Systems profiling across 6 different configurations.

D.12. Nsight Systems Profiling Results

Nsight Systems profiling on Settings 1–3 using the Spec-Bench dataset shows that Qwen3-1.7B and Qwen3-0.6B exhibit similar drafting times, whereas Qwen3-4B has the longest drafting time among the three draft models. As illustrated in Fig. 28, Qwen3-4B spends significantly more time on drafting than the verification time, limiting its ability to benefit from PSD. In contrast, Qwen3-1.7B, while slightly slower than Qwen3-0.6B, has greater model capacity and is thus more effective at generating draft tokens that pass verification. This observation supports the conclusion in Sec. D.8 that Qwen3-1.7B achieves the optimal balance between drafting speed and draft quality.

Table 1. Percentage of total steps when standalone MineDraft operates in parallel vs. sequential mode across different settings and datasets. Reported numbers are averaged from three independent runs. MineDraft achieves an average parallelization percentage of 99.19%.

Setting	k	Arena	ShareGPT	Spec-Bench	Tough
Qwen3 32B-0.6B (Setting 1)	1	99.84	99.96	94.32	99.88
	2	99.78	99.83	98.10	99.85
	3	99.86	99.84	98.61	99.76
	4	99.80	99.81	98.85	99.93
	5	99.86	99.61	98.79	99.87
Qwen3 32B-1.7B (Setting 2)	1	99.90	99.75	93.61	99.94
	2	99.94	99.77	97.16	99.92
	3	99.72	99.79	98.13	99.81
	4	99.60	99.75	98.82	99.86
	5	99.78	99.66	98.02	99.86
Qwen3 32B-4B (Setting 3)	1	99.96	99.86	93.00	99.90
	2	99.90	99.83	95.66	99.96
	3	99.89	99.81	97.52	99.88
	4	99.83	99.65	98.51	99.84
	5	99.83	99.71	99.31	99.73
Qwen3 32B-8B (Setting 4)	1	99.93	99.95	93.48	99.94
	2	99.91	99.88	95.87	99.94
	3	99.95	99.70	98.91	99.81
	4	99.83	99.88	97.12	99.85
	5	99.65	99.83	98.16	99.74
Llama-3 70B-8B (Setting 5)	1	99.77	99.88	99.59	99.81
	2	99.68	99.71	98.01	99.80
	3	99.64	99.60	97.02	99.81
	4	99.50	99.19	98.05	99.73
	5	98.86	99.53	95.62	99.63
Vicuna 33B-EAGLE (Setting 6)	1	99.89	99.81	98.59	99.76
	2	99.78	99.69	97.63	99.91
	3	99.89	99.63	96.36	99.82
	4	99.86	99.72	98.58	99.71
	5	99.88	99.76	99.56	99.82
Vicuna 13B-EAGLE (Setting 7)	1	99.89	99.91	96.01	99.92
	2	99.88	99.74	97.31	99.87
	3	99.75	99.87	99.57	99.83
	4	99.73	99.88	97.52	99.83
	5	99.76	99.68	98.18	99.75

D.13. Parallelization Coverage

We validate the tail effect of MINEDRAFT discussed in Sec. F by calculating the percentage of decoding steps operating in the parallel mode. Tab. 1 includes the percentage numbers across different experiment settings and datasets and shows that MINEDRAFT achieves an overall average parallelization percentage of 99.19%, which highlights that the limitation brought by tail effect is extremely limited.

E. Frequently Asked Questions

Question 1. Why does MINEDRAFT not attain the maximum theoretical latency improvement of 50% in the experiments?

Answer. The theoretical upper bound of a 50% latency reduction assumes an idealized setting in which drafting and verification times are perfectly matched throughout the execution of MINEDRAFT. In practice, both stages exhibit stochastic variation due to system overheads, resource contention, and time multiplexing, making the precise temporal alignment required to attain this bound infeasible in real-world deployments.

Question 2. Why does MINEDRAFT maintain two batches instead of more? Why is the “Batch Parallelism size” fixed to 2?

Answer. The design of MINEDRAFT is intrinsically tied to the two-stage structure of speculative decoding (SD): drafting and verification. Compared to autoregressive decoding, the additional drafting stage introduces extra latency, which can become prohibitive when either the number of draft tokens per step is large or the draft model is computationally expensive. MINEDRAFT mitigates this overhead by overlapping the drafting of one set of requests with the verification of another. This design naturally requires exactly two batches: one for drafting and one for verification. Introducing additional batches would not produce further parallelism benefits, as there are no additional stages to overlap, and would instead incur unnecessary scheduling and management overhead.

Question 3. Why does MINEDRAFT not incorporate prefill requests in the parallelism?

Answer. The current design of MINEDRAFT assumes that chunked prefill is disabled. Under this configuration, vLLM completes the prefill stage for new requests in a dedicated step before transitioning them to decoding in subsequent steps, isolating prefill requests from the speculative decoding steps in which MINEDRAFT operates. As support for chunked prefill is a planned extension, future versions of MINEDRAFT will integrate prefill requests into its parallel execution workflow.

F. Limitations

Irrecoverable from biased batch workload. The workload balance between batches can become irrecoverably disrupted under certain conditions. When a request is terminated due to user abort or preemption, MINEDRAFT may fail to recover from the resulting imbalance because new replacement running requests are consistently assigned to the draft batch. While this policy ensures that draft tokens are available for verification, it prevents the load-balancing mechanism (the `assign` operation) from correcting the induced batch imbalance. Furthermore, when chunked prefill is enabled, the Scheduler may initialize fewer than $2m$ running requests in the first SD step. In this case, MINEDRAFT must continue to apply the load-balancing `assign` operation for all subsequent batch ID assignments, rather than switching to the simplified post-first-step policy. Otherwise, an irrecoverable workload imbalance between the two batches can arise. However, continuously applying the `assign` operation introduces a minor overhead. If a new running request is assigned to the target batch, the Verifier can sample only a single token because there are no draft tokens, resulting in one less efficient “idle” SD step for that request. Since this cost is incurred at most once per request, it is deemed an acceptable trade-off for the long-term performance benefits of maintaining batch balance. Accordingly, we plan to adopt this revised batch ID assignment strategy in a future version of MINEDRAFT that supports the vLLM v1 engine and chunked prefill.

Batch exhaustion and tail effect. The Batch Parallelism scheme, as specified in Sec. 5, includes a fallback mechanism: when one batch becomes empty and the other batch contains no verifiable drafts, the system reverts to standard SD, which inevitably curtails the duration of parallel execution. Although Batch Parallelism successfully parallelizes the substantial majority of SD steps and yields significant performance gains per our empirical results, the tail effect circumscribes its maximum potential improvement. A possible approach to offset this limitation involves adopting a modified PEARL parallelism. When one batch becomes empty, the system instructs the Drafter to generate draft tokens for the remaining batch, including those undergoing verification, without confirming their success. In the next step, instead of verifying only previously successful requests as in standard PEARL, the system re-drafts requests that failed verification before submitting the batch to the Verifier. This re-drafting mechanism for failed verification attempts is intended for partially overlapping drafting time, optimizing performance to the greatest extent possible.