ShadowSpec: Towards Zero Speculation Overhead for Substitute Speculative Decoding

Anonymous Authors[1]

Abstract

Deploying large language models (LLMs) on memory constrained consumer GPUs requires offloading model parameters to CPU memory, where PCIe bandwidth becomes the dominat inference bottleneck. While speculative decoding (SD) reduces the number of target model invocations, its sequential draft-then-verify pipeline leaves the GPU idle during parameter loading, limiting throughput in offloading environments. We identify that draft model computation and target model parameter loading are heterogeneous, non-contending operations that can be overlapped without resource conflict. Building on this observation, we propose two complementary system-level strategies to overlap draft tree extension with target model parameter loading, converting idle GPU cycles into productive draft computation. Experiments on consumer GPU demonstrate a 17–54% end-to-end speedup over the SubSpec baseline.

1. Introduction

Deploying large language models (LLMs) on consumer-grade hardware presents a significant memory bottleneck. Models exceeding 70 billion parameters routinely surpass the VRAM capacity of standard consumer GPUs, necessitating deployment strategies that trade off inference speed against memory footprint. Two principal approaches have emerged to address this constraint. Quantization reduces model precision to fit within GPU memory, enabling direct on-device inference, but introduces accuracy degradation that can compromise output quality (Dutta et al., 2024; Li et al., 2025c; Ayadi et al., 2026). Parameter offloading, by contrast, preserves full model fidelity by streaming weights from CPU memory to the GPU on demand, yet is fundamentally bottlenecked by PCIe bandwidth–reducing generation throughput to as low as 2.77 tokens per second for 7B-scale models (Wang et al., 2025).

Speculative decoding (SD) (Leviathan et al., 2023; Chen et al., 2023) has emerged as a promising approach to accelerate LLM inference without degrading output quality. SD recasts autoregressive generation into two stages: a draft stage, in which a lightweight model–ranging from smaller standalone network to trained drafting heads (Li et al., 2024a;b; 2025b)–constructs a tree of candidate token sequences (Miao et al., 2024); and a verification stage, in which the target model validates all drafted tokens in a single forward pass. This batched verification amortizes the cost of target model invocation across multiple tokens, yielding substantial throughput gains in standard compute-bound settings.

However, the effectiveness of speculative decoding degrades in memory-constrained offloading environments. In such settings, target model verification is bottlenecked not by computation but by PCIe bandwidth, as model parameters must be streamed layer-by-layer from CPU memory. As a result, the GPU remains idle during weight transfers–an inefficiency that standard SD scheduling does not address. Furthermore, the sequential draft-then-verify pipeline introduces a mutual waiting problem: the draft model sits idle during verification, and the target model waits while drafting completes. In heavily offloaded regimes, this alternating idling dominates end-to-end latency, undermining the throughput gains that SD is designed to provide.

To mitigate this idling, recent work has explored parallelizing the draft and verification stages (Liu et al., 2025a; McDanel et al., 2025; Yin et al., 2025; Li et al., 2025a). However, these systems are designed for compute-bound environments with moderate draft-to-target cost ratios (typically $c \in [5, 10]$), and do not transfer directly to heavily offloaded regimes where cost ratios can exceed 100. In such extreme asymmetry settings, effective pipeline parallelism requires highly optimized tree-based drafting (Liu et al., 2025b) combined with efficient memory transfer scheduling and heterogeneous CPU–GPU coordination (Zhuge et al., 2025; Fan et al., 2025; Zhang et al., 2025; Tang et al., 2025). Despite these efforts, no existing work exploits the struc-

[1]Anonymous 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.

tural asymmetry between GPU-bound draft computation and PCIe-bound parameter loading as a scheduling opportunity.

In this work, we introduce two system-level strategies, **post-speculation** and **post-verify**, that address this gap by restructuring the SD pipeline for memory-constrained offloading environments. Our contributions are as follows:

- **Latency profiling of offloading-based SD.** We analyze the per-iteration latency composition of SubSpec (Wang et al., 2025) and identify that draft computation and target model parameter loading are heterogeneous, non-contending operations that can be overlapped without resource conflict.
- **Post-speculation.** We propose overlapping draft tree extension with target model parameter loading, converting otherwise idle GPU cycles into productive draft computation and reducing effective drafting overhead.
- **Post-verification.** We propose conditioning post-speculated tokens on the target-verified KV cache, enabling the draft model to identify likely rejection points and improve the acceptance rate of subsequent speculation rounds.
- **Theoretical and empirical analysis.** We extend the standard SD speedup formulation to account for the overlap ratio r and post-speculation success rate s, and validate our framework experimentally, demonstrating a 17–54% end-to-end speedup over SubSpec on consumer GPU hardware.

2. Preliminaries

Speedup Effect of Speculative Decoding The speedup of speculative decoding relative to standard autoregressive (AR) decoding can be represented by:

$$\frac{T_{AR}}{T_{SD}} = \frac{t_{Target}}{\frac{\mathcal{D}\cdot t_{Draft}+\gamma\cdot t_{Target}}{\tau}} = \frac{\tau}{\frac{\mathcal{D}\cdot t_{Draft}}{t_{Target}}+\gamma} \quad (1)$$

where t_{Target} and t_{Draft} denote the the per-step latency of the target model and draft models, respectively. $\mathcal{D}$ is the draft depth, τ is the mean number of tokens accepted per iteration (i.e., average acceptance length) and the factor γ captures the relative overhead of parallel verification compared to a standard AR forward pass of the target model.

Eq. (1) exposes a fundamental trade-off in the draft model selection and the depth configuration. The denominator comprises two competing terms: the normalized drafting overhead $\mathcal{D} \cdot t_{Draft}/t_{Target}$ and the verification cost γ. Increasing $\mathcal{D}$ can improve τ by providing more candidate tokens per iteration of verification, but incurs a proportional increase in drafting overhead. Consequently, effective speculative decoding configurations typically favor lightweight draft models ($t_{Draft} \ll t_{Target}$) operating at moderate depths, though smaller models may exhibit weaker alignment with the target distribution, placing an upper bound on the achievable τ.

SubSpec SubSpec (Wang et al., 2025) is a speculative decoding mechanism to solve the LLM deployment challenge on memory-limited consumer GPUs. The method aligns the draft model by substituting offloaded portions of the target LLM with low-bit layers, while concurrently sharing GPU-resident components with the target model. The method maintains the accuracy of the draft model by sharing the KV cache across both the drafting and verification stages, which eliminates duplicate caching and unnecessary prefilling.

3. Methodology

To further accelerate the LLM inference on consumer-level devices, we analyze the overheads of SD in offloading situation. Based on the observation and analysis, we propose system-level strategies, post-speculate and post-verify, to enhance the speed and accuracy of SD with offloading.

3.1. Latency Analysis of Offloading-Based Speculative Decoding

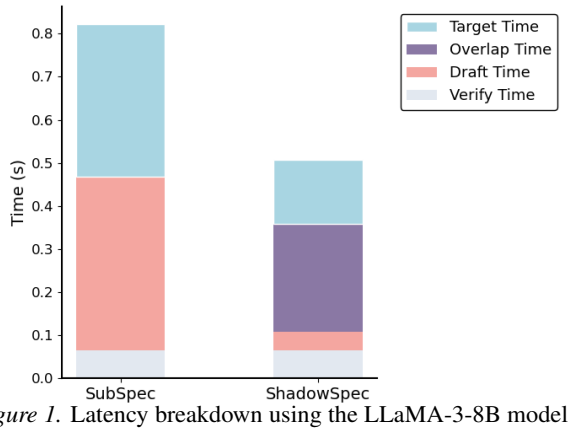

Figure 1. Latency breakdown using the LLaMA-3-8B model on the GSM8K benchmark on a single NVIDIA RTX 5090 GPU constrained to 8GB of VRAM. ShadowSpec reduces end-to-end generation time relative to SubSpec (Wang et al., 2025) baseline by masking draft computation within target model parameter loading, denoted as Overlap Time.

We begin by profiling the SubSpec framework (Wang et al., 2025) on an NVIDIA RTX 5090 GPU constrained to 8GB of VRAM to characterize its latency composition. As shown in Fig. 1, the total per-iteration latency is dominated by two roughly equal components: draft model computation and target model execution. The target model execution itself consists of two sequential phases, layer-by-layer parameter loading from CPU memory over the PCIe bus, followed by GPU-side verification computation. Because parameter

loading is memory-bound rather than compute-bound, the GPU remains largely idle during weight transfers, contributing to substantial underutilization.

Critically, draft model computation and target model parameter are *heterogeneous* operations: the former runs on the GPU while the latter occupies the PCIe bus and CPU memory subsystem. This orthogonality means the two operations can proceed concurrently without resource contention, yet the sequential scheduling in SubSpec prevents any such overlap. This observation motivates a system-level redesign that exploits the idle GPU intervals during parameter loading as an opportunity to perform draft computation at effectively zero additional cost.

3.2. Overview of Proposed Method

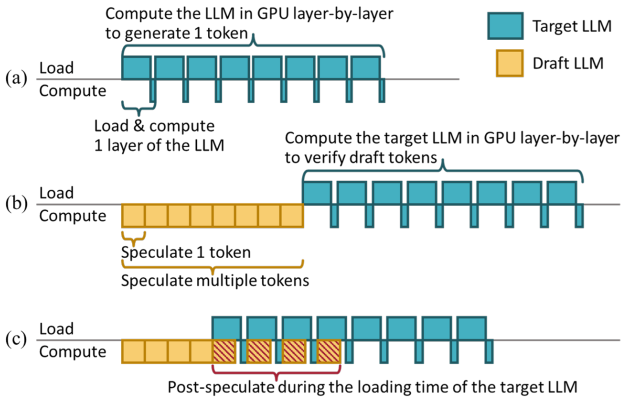

Figure 2. Inference timeline comparison (a) vanilla autoregressive decoding; (b) standard speculative decoding; (c) speculative decoding with our proposed post-speculation and post-verification

Fig. 2 contrasts three inference regimes. In vanilla autoregressive decoding (a), the target model is invoked once per token, making generation strictly memory-bound due to repeated parameter loading. Standard speculative decoding (b) reduces target model invocations by interposing a draft model that proposes multiple candidate tokens verified in a single target forward pass. However, as noted in Section 3.1, the draft model's total computation time grows with speculation depth: SubSpec in particular employs deep draft tree to maximize the average acceptance length, causing draft execution to occupy a duration comparable to that of the target model itself.

Our key observation is that during each target model forward pass, parameter loading accounts for the majority of wall-clock time, and the GPU remains idle throughout these transfers Each individual layer load introduces a brief but consistent idle window–sufficient for the draft model to extend the draft tree by one additional step. By scheduling draft computation to fill these idle windows, drafting overhead can be largely hidden within the unavoidable memory transfer latency, as illustrated in Fig. 2(c).

Building on this principle, we propose two complementary strategies. **Post-speculation** (Section 3.3) overlaps draft tree extension with target model parameter loading, converting otherwise idle GPU cycles into productive draft computation and thereby reducing effective drafting cost. **Post-verification** (Section 3.4) then improves draft quality by conditioning post-speculated tokens on the target verified KV cache, enabling the draft model to anticipate likely rejection points and improve the acceptance rate of subsequent speculation rounds. Together, these strategies reduce both the latency cost of drafting and the frequency of costly re-speculation, yielding consistent end-to-end throughput improvements in memory-constrained offloading environments.

3.3. Post-Speculate: Hiding Speculation Overheads

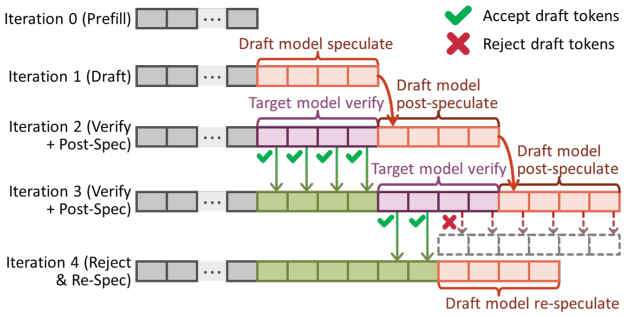

Figure 3. Example of token generation by post-speculation with acceptance in iteration 2 and rejection in iteration 3 (and thus re-speculation in iteration 4)

To hide the inference latency induced by sequential drafting and verification, we introduce *post-speculation*. Post-speculation performs drafting phase after the prior iteration of speculation, overlapping with the verification phase. During the verification stage, the parameter shards of the target model are loaded into the GPU one-by-one to verify the current draft sequence; simultaneously, the draft model continues to generate tokens during the memory transfer time of the target model. If the post-speculated token sequence is aligned with the newly accepted sequence, we can splice its novel tokens and proceed directly, avoiding the further drafting time. As the example illustrated in Fig. 3, the draft model continues to generate tokens during the verification time (iteration 3). If a mismatch occurs, we withdraw the post-speculated sequence and re-speculate, constructing a new draft sequence from the last accepted token (see iteration 4 in Fig. 3). The procedure preserves exactness while converting the GPU computation idle time (the parameter loading period of the target model) during the verification stage into meaningful computation when the speculated sequence is accepted by the target.

The step of rejection and re-speculation ensures that valid candidate tokens are available for the next verification step,

and thus prevents the decoding pipeline from stalling. When the post-speculated token(s) is rejected, a mandatory re-speculation round will be triggered to generate a new draft sequence. However, this correction procedure incurs a significant computational cost. Unlike concurrent draft generation, which overlaps with memory loading of the target model, re-speculation cannot be hidden within existing latency windows. It necessitates an additional, serial round of draft model computation, effectively breaking the intended pipeline parallelism between drafting and verification.

3.4. Post-Verify: Retrieving Speculation Accuracy

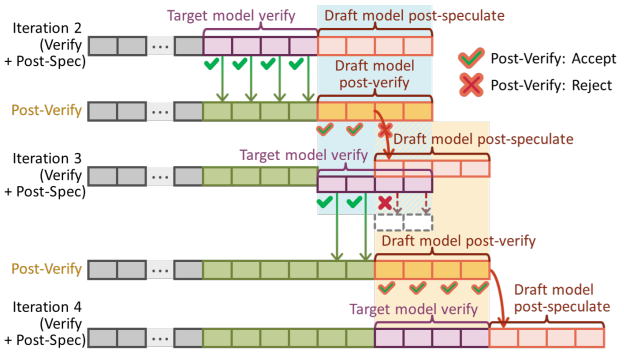

Figure 4. Example of token generation with post-verification being applied after the verification of the target model

We observed that the quality of tokens generated by post-speculation is often lower than that of the original speculation, as the new post-speculation segment corresponds only to the latter half of the original sequence. To address this issue, we propose *post-verification*, a technique designed to restore speculative token quality. Once the target model completes verification and updates the shared KV cache, the draft model uses the same cache, i.e., considers the currently verified tokens and pre-verifies all newly drafted tokens. For example, the draft model considers all the verified (green) tokens after iteration 2 in Fig. 4 to predict the verification result in the next iteration of the target model. This process allows the draft model to identify and speculate the dropped positions again to create new tokens before being rejected by the target model (see iteration 3 in Fig. 4, the draft model starts its post-speculation from the predict-to-be-dropped position). By considering all the verified tokens in real time, our proposed post-verification can improve the generation accuracy of the draft model and reduce the probability of applying re-speculation to generate new tokens without hiding the latency within the loading windows of the target model.

3.5. Theoretical Speedup Analysis

To incorporate the latency savings introduced by our proposed post-speculation, we extend Eq. (1) by adding a latency reduction factor $r \in [0, 1]$, which quantifies the fraction of draft execution time that is effectively hidden by overlapping it with the target model parameter loading during verification:

$$Speedup = \frac{\tau}{\frac{(1-r)\cdot\mathcal{D}\cdot t_{Draft}}{t_{Target}} + \gamma} \tag{2}$$

The factor (1-r) scales down the normalized drafting overhead, reflecting that only the non-overlapped portion of draft computation contributes to the critical path. When $r = 0$, the formulation reduces to the standard SD speedup in Eq. (1); when $r = 1$, drafting is fully masked and imposes no additional latency. In practice, r is bounded by the ratio of parameter loading time to draft execution time, as overlapping is only possible within the duration of target model weight transfers. This formulation shows that post-speculation effectively relaxes the draft depth–speed trade-off: systems with higher r can accommodate larger $\mathcal{D}$ or more expressive draft models without proportionally increasing end-to-end latency.

While post-speculation overlaps draft computation with target model parameter loading, it remains subject to the stochastic nature of speculative decoding: when the post-speculated tokens are rejected by the target model, a re-speculation round must be triggered. As described in Sec. 3.3, re-speculation cannot be overlapped with parameter loading and therefore lies on the critical path. To capture this effect, we introduce the post-speculation success rate $s \in [0, 1]$ and incorporate it into the latency reduction factor. The expected normalized drafting overhead marginalizes over the accepted (s) and rejected ($1 - s$) outcomes:

$$\begin{aligned} Speedup &= \frac{\tau}{\frac{s\cdot(1-r)\cdot\mathcal{D}\cdot t_{Draft} + (1-s)\cdot\mathcal{D}\cdot t_{Draft}}{t_{Target}} + \gamma} \\ &= \frac{\tau}{\frac{(1-s\cdot r)\cdot\mathcal{D}\cdot t_{Draft}}{t_{Target}} + \gamma} \end{aligned} \tag{3}$$

When a post-speculated sequence is accepted (s), only the non-overlapped fraction ($1 - r$) of drafting overhead remains on the critical path. When rejected ($1 - s$), the full drafting cost $\mathcal{D} \cdot t_{Draft}$ is incurred due to the mandatory re-speculation. The combined effective overhead is thus governed by the term ($1 - s \cdot r$), indicating that the benefit of overlapping is realized only proportionally to the post-speculation success rate.

Post-verification further improves s at the cost of additional latency. After the target model completes verification and updates the shared KV cache, the draft model performs a lightweight forward pass over the post-speculated tokens conditioned on the newly verified context. This allows the draft model to identify positions likely to be rejected in the next verification round and restart post-speculation from that point, improving draft quality before the subsequent

target model pass. Incorporating this additional step, the speedup becomes:

$$Speedup = \frac{\tau}{\frac{(1-s'\cdot r)\cdot\mathcal{D}\cdot t_{Draft}+t_{post\text{-}verify}}{t_{Target}} + \gamma} \quad (4)$$

where $s' \geq s$ denotes the improved success rate under post-verification pass.

Comparing the Eq. (3) and Eq. (4) against the standard SD formulation in Eq. (1), the improvement brought by our framework depends on two conditions: (1) r must be sufficiently large for post-speculation to meaningfully reduce drafting overhead, and (2) s (or s' with post-verification must be high enough that re-speculation is infrequent. In regimes where parameter loading dominates (the characteristic of heavily offloaded consumer GPU deployments), both conditions tend to hold, yielding substantial end-to-end speedup over standard SD.

4. Experiments

4.1. Evaluation Setup

Evaluation Benchmarks. We evaluated performance across five diverse generative tasks, consistent with the benchmarks from SubSpec. These tasks included multi-turn conversation (MT-Bench (Zheng et al., 2023)), code generation (HumanEval (Chen et al., 2021)), mathematical reasoning (GSM8K (Cobbe et al., 2021)), instruction following (Alpaca (Dubois et al., 2023)), and summarization (CNN/Daily Mail (Nallapati et al., 2016)).

Hardware and Simulated Environments. All experiments were run on a system with an RTX 5090 GPU, an Intel Ultra 7 265K CPU, a PCIe-5.0 x16 bus, and 128GB of DDR5 RAM. GPU memory utilization during evaluations was programmatically restricted to 8GB VRAM capacities to simulate diverse consumer device environments.

Comparative Methodology and Parameters. We compared the end-to-end speedup of ShadowSpec against SubSpec and baseline models (offloading with no SD) from the Qwen3 (8B) and Llama3.1 (8B). For fair comparison, all methods used an identical context-aware dynamic draft tree algorithm without additional tree pruning techniques. A portion of the target model decoder layers was kept resident on the GPU within the VRAM limits. All SD methods were evaluated on identical samples, randomly selected from each dataset. The baseline (offloading with no SD) used the initial three samples due to its significantly longer runtime.

Table 1. Performance of speculative decoding across varying maximum draft tree depths

Max depth	Throughput (tokens/s)	τ	Speculation time (s)	Verification time (s)	Tree accepted rate
24	34.50	20.22	0.156	0.377	0.778
32	36.66	23.30	0.20	0.39	0.622
48	39.34	28.97	0.30	0.419	0.448

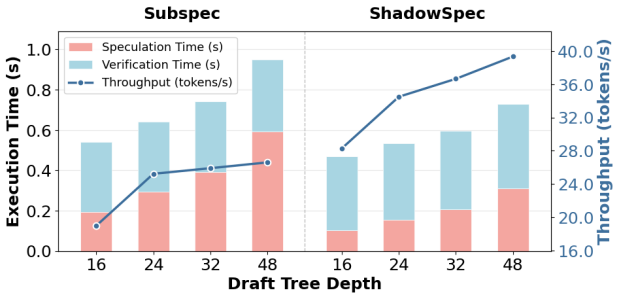

Figure 5. Execution time breakdown (speculation vs. verification) and throughput of SubSpec and ShadowSpec Waccelerating Llama3.1-8B under an 8GB VRAM limit.

4.2. Impact of Draft Tree Depth on Speculation Efficiency

Table 1 demonstrates the impact of the maximum depth parameter on speculative decoding performance while maintaining a constant draft temperature and top-k length. The empirical results illustrate the trade-off inherent in constructing deeper context-aware draft trees. As the maximum depth parameter increases, the system achieves higher throughput and captures significantly more accepted tokens per step. However, this scaling simultaneously doubles the computational overhead required for draft generation, with speculation time climbing from 0.156 to 0.309 seconds. Furthermore, extending the tree depth causes the previous path acceptance rate to drop sharply from 0.778 to 0.448.

In Fig. 5, the results demonstrate ShadowSpec significantly reduces overhead associated with the speculation phase compared to the SubSpec baseline. In the baseline configuration, as draft tree depth increases, speculation time scales disproportionately and eventually dominates total execution time, reaching approximately 0.6 seconds out of a total 0.95 seconds at depth of 48. This scaling behavior indicates a severe efficiency bottleneck in the drafting phase for deeper trees. Conversely, ShadowSpec substantially mitigates this growth. Even at the maximum tested depth of 48, speculation time remains constrained to approximately 0.3 seconds, maintaining a significantly lower total execution time than Subspec. Because verification time remains roughly comparable across both methods, the data confirms the primary efficiency gains of the ShadowSpec architecture stem directly from an optimized speculation process.

Table 2. Throughput (tokens/s) and average acceptance lengths τ by using different draft models. L31 represents Llama-3.1-8B-Instruct, Q represents Qwen3-8B. None represents vanilla autoregressive decoding is used. The number in each parentheses below the target model name denotes the restricted VRAM limit. Speedup is calculated relative to the vanilla autoregressive decoding (None) baseline.

		MT-Bench		HumanEval		GSM8K		Alpaca		CNN/DM		Mean		
Target	Draft	tokens/s	τ	tokens/s	τ	tokens/s	τ	tokens/s	τ	tokens/s	τ	tokens/s	τ	Speedup
Temperature = 0.0														
L31 8B (8GB)	None	3.65	1	3.63	1	3.55	1	3.66	1	3.58	1	3.61	1	1.00×
	SubSpec	29.11	23.22	27.40	26.26	27.82	25.40	27.46	21.13	28.20	25.14	28.00	24.23	7.74×
	ShadowSpec	**36.09**	**26.50**	**39.93**	**33.22**	**39.35**	**28.97**	**33.63**	**24.13**	**37.96**	**29.07**	**37.39**	**28.38**	**10.34 ×**
Q 8B (8GB)	None	3.54	1	3.54	1	3.54	1	3.54	1	3.5	1	3.53	1	1.00×
	SubSpec	17.67	21.23	25.08	23.89	28.42	26.67	27.86	24.78	27.51	24.72	25.31	24.26	7.16×
	ShadowSpec	**23.81**	**22.96**	**33.05**	**33.48**	**36.70**	**35.70**	**30.70**	**30.24**	**28.57**	**27.66**	**30.57**	**30.01**	**8.65×**
Temperature = 0.6														
L31 8B (8GB)	None	3.65	1	3.63	1	3.55	1	3.66	1	3.58	1	3.61	1	1.00×
	SubSpec	19.02	15.63	17.04	15.53	18.52	16.82	13.71	11.05	15.39	13.79	16.73	14.56	4.63×
	ShadowSpec	**21.31**	**17.07**	**23.07**	**18.21**	**20.84**	**16.42**	**15.83**	**12.01**	**16.89**	**13.77**	**19.59**	**15.49**	**5.42 ×**
Q 8B (8GB)	None	3.54	1	3.54	1	3.54	1	3.54	1	3.5	1	3.53	1	1.00×
	SubSpec	9.691	13.49	15.01	21.60	14.31	20.41	10.53	14.60	11.10	15.69	12.13	17.16	3.43×
	ShadowSpec	**18.31**	**18.35**	**17.99**	**18.90**	**25.11**	**26.20**	**15.91**	**16.03**	**16.41**	**16.61**	**18.74**	**19.22**	**5.30×**

4.3. End-to-end Performance

The original Subspec method noted that PCIe bandwidth is the primary latency source when loading parameters to the GPU. However, when upgrading hardware to the RTX 5090 and PCIe 5.0, the target model's verification time (t_{Target}) is significantly shortened due to the doubled bandwidth. This ironically makes the draft model's generation overhead ($\mathcal{D} \cdot t_{Draft}$) the new performance bottleneck.

Table 2 presents a performance comparison of different draft methods, specifically SubSpec and ShadowSpec, against a vanilla autoregressive decoding baseline ("None") under a restricted 8GB VRAM limit. The results indicate that the ShadowSpec framework consistently outperforms both the baseline and Subspec method on multiple benchmarks for both Llama-3.1-8B-Instruct (L31) and Qwen3-8B (Q) target architectures. At a temperature of 0.0, our new pipeline achieves a speedup of 10.34× for L31 and 8.65× for Q, yielding significantly higher throughput (tokens/s) and extended average acceptance lengths (τ). Although overall throughput and speedup ratios naturally decrease at a higher temperature of 0.6, the proposed method maintains a substantial performance advantage, delivering mean speedups exceeding 5× for both target configurations. Overall, the data highlights the robust efficiency of ShadowSpec in accelerating large language model inference under strict memory constraints.

Beyond outperforming the baseline, the data highlights a leap achieved by ShadowSpec over the Subspec method. An analysis of normalized speedup ratios reveals the updated framework consistently extracts significant residual efficiency from the SubSpec. Specifically, for the L31 model utilizing a greedy decoding setting (Temperature = 0.0), ShadowSpec elevates the speedup from 7.74× to 10.34×. This yields an additional 33.6% relative performance gain over Subspec. This proportional enhancement remains consistent across configurations; for the Q model at a temperature of 0.6, ShadowSpec achieves a 54.5% relative improvement over SubSpec, increasing the speedup from 3.43× to 5.30×.

ShadowSpec consistently achieves higher average acceptance lengths (τ) than the SubSpec baseline. These results indicate the proposed post-verification mechanism successfully restores speculative token quality. By leveraging the updated shared KV cache of the target model to verify tokens, the framework improves the generation accuracy of the draft model and prevents costly re-speculations, ultimately minimizing overall inference latency.

5. Conclusion

We presented ShadowSpec, including post-speculation and post-verification, two system-level strategies that restructure the speculative decoding pipeline for offloading-based LLM inference. By overlapping GPU-bound draft computation with PCIe-bound parameter loading, post-speculation converts unavoidable memory transfer latency into productive draft generation at zero additional cost. Post-verification further reduces re-speculation frequency by conditioning the draft model on the target-verified KV cache. Together, the two strategies yield a 17–54% end-to-end speedup over SubSpec on consumer GPU hardware, without modifying the target model or compromising output correctness.

References

Ayadi, S., Daróczi, S., Günnemann, S., and Charpentier, B. Reliability scaling laws for quantized large language

models, 2026. URL https://openreview.net/forum?id=QhkW8xPH1v.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dubois, Y., Li, C. X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P. S., and Hashimoto, T. B. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36:30039–30069, 2023.

Dutta, A., Krishnan, S., Kwatra, N., and Ramjee, R. Accuracy is not all you need. *Advances in Neural Information Processing Systems*, 37:124347–124390, 2024.

Fan, J., Zhang, Y., Li, X., and Nikolopoulos, D. S. Apex: Asynchronous parallel cpu-gpu execution for online llm inference on constrained gpus. *arXiv preprint arXiv:2506.03296*, 2025.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, R., Liu, Z., Shi, Y., Shao, J., Zhang, C., and Li, X. Pipeline parallelism is all you need for optimized early-exit based self-speculative decoding. *arXiv preprint arXiv:2509.19368*, 2025a.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a.

Li, Y., Wei, F., Zhang, C., and Zhang, H. 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.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025b.

Li, Z., Su, Y., Wang, S., Yang, R., Xie, C., Liu, A., Li, M., Cao, J., Xie, Y., Wong, N., et al. Quantization meets reasoning: Exploring and mitigating degradation of low-bit llms in mathematical reasoning. *arXiv preprint arXiv:2505.11574*, 2025c.

Liu, T., Li, Y., Lv, Q., Liu, K., Zhu, J., Hu, W., and Sun, X. Pearl: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=QOXrVMiHGK.

Liu, X., Luo, L., Tang, M., Huang, C., and Chen, X. Flowspec: Continuous pipelined speculative decoding for efficient distributed llm inference. *arXiv preprint arXiv:2507.02620*, 2025b.

McDanel, B., Zhang, S. Q., Hu, Y., and Liu, Z. Pipespec: Breaking stage dependencies in hierarchical llm decoding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 12909–12920, 2025.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., et al. 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*, pp. 932–949, 2024.

Nallapati, R., Zhou, B., Dos Santos, C., Gulçehre, Ç., and Xiang, B. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pp. 280–290, 2016.

Tang, L., Zhang, D., Chen, J., Huang, P., Jin, F., Xu, C., Chen, Y., Sun, F., and Chen, G. Multipath transfer engine: Breaking gpu and host-memory bandwidth bottlenecks in llm services. *arXiv preprint arXiv:2512.16056*, 2025.

Wang, P.-S., Chen, J.-J., Yang, C.-C., Chang, C.-C., Huang, N.-C., Abdelfattah, M. S., and Wu, K.-C. Speculate deep and accurate: Lossless and training-free acceleration for offloaded llms via substitute speculative decoding. In *Advances in Neural Information Processing Systems*, 2025.

Yin, H., Xiao, M., Li, T., Zhang, X., Yu, D., and Zhang, G. Specpipe: Accelerating pipeline parallelism-based llm inference with speculative decoding. *arXiv preprint arXiv:2504.04104*, 2025.

Zhang, L., Zhang, Z., Li, R., Tian, Z., Mei, S., Li, D., et al. Dovetail: A cpu/gpu heterogeneous speculative decoding for llm inference. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17393–17406, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36: 46595–46623, 2023.

Zhuge, X., Shen, X., Wang, Z., Dang, F., Ding, X., Li, D., Han, Y., Hao, T., and Yang, Z. Specoffload: Unlocking latent gpu capacity for llm inference on resource-constrained devices. *arXiv preprint arXiv:2505.10259*, 2025.

A. You *can* have an appendix here.

You can have as much text here as you want. The main body must be at most 8 pages long. For the final version, one more page can be added. If you want, you can use an appendix like this one.

The `\onecolumn` command above can be kept in place if you prefer a one-column appendix, or can be removed if you prefer a two-column appendix. Apart from this possible change, the style (font size, spacing, margins, page numbering, etc.) should be kept the same as the main body.