Speculative Verification: Exploiting Information Gain to Refine Speculative Decoding

Abstract

LLMs have low GPU efficiency and high latency due to autoregressive decoding. Speculative decoding (SD) mitigates this using a small draft model to speculatively generate multiple tokens, which are then verified in parallel by a target model. However, when speculation accuracy is low, the overhead from rejected tokens can offset the benefits, limiting SD's effectiveness, especially at large batch sizes.

To address this, we propose Speculative Verification (SV), an efficient augmentation to SD that dynamically predicts speculation accuracy and adapts the verification length to maximize throughput. SV introduces a companion model – a small auxiliary model similar in size to the draft model – to estimate the alignment between draft and target model distributions. By maximizing the information gain from quantifying this alignment, SV refines verification decisions, reducing wasted computation on rejected tokens and improving decoding efficiency. Moreover, SV requires no modifications to the draft or target models and is compatible with existing SD variants.

We extensively evaluated SV on publicly available LLMs across three NLP tasks using nine combinations of draft, companion, and target models, including 13B–72B target models and three types of variations: base (no finetuning), instruction-tuned, and task fine-tuned. Across all experiments and batch sizes (4–80), SV consistently outperforms both SD and standard decoding with the target model. It improves SD performance by up to $2\times$, with an average speedup of $1.4\times$ in large-batch settings (batch sizes 32–80). These results demonstrate SV's robustness, scalability, and practical utility for efficient LLM inference.

1 Introduction

Large Language Models (LLMs) are widely used across many application domains, but their size and computational cost make large-scale inference serving a significant challenge. In particular, LLMs rely on autoregressive decoding – generating one token at a time – so producing k tokens requires k sequential steps, leading to GPU underutilization and increased latency.

Speculative Decoding (SD)[1] addresses this problem by using a small draft model to speculatively generate tokens, which are then verified in parallel by a larger target model. Because the draft model is fast and the target model can validate multiple tokens in a single forward pass, overall latency is reduced. However, when drafted tokens are rejected, both their verification and the recomputation incur additional overhead.

SD's effectiveness depends on speculation accuracy, i.e., the fraction of drafted tokens accepted by the target model. Low accuracy negates its benefits especially for scaled inference serving; if most drafted tokens are rejected, or only a portion for large batch sizes, the verification overhead makes SD slower than target decoding. Speculation accuracy depends on the alignment between the draft and target model distributions, which fluctuates due to model capability gaps and input context variations.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Identifying when these distributions align allows for adjusting the speculation length to minimize verification overhead and optimize latency. However, predicting this alignment is challenging due to the complexity of LLM computation. A prior approach proposed tracking acceptance rates via a moving average[2], but our evaluation shows this method is ineffective.

In this paper, we propose speculative verification (SV), an approach to reliably predict speculation accuracy and maintain SD's performance gains. Building on an information-theoretic foundation, SV compares the draft model's output distribution with that of a similarly-sized *companion* model. By quantifying the alignment of their distributions, SV estimates the likelihood that the target model will accept the drafted tokens. Using these estimates, SV dynamically adjusts the verification length, minimizing verification cost for tokens likely to be rejected. This reduces the overhead of misaligned speculation and enables SD to scale effectively for real-world inference serving.

This paper contributes the concept of speculative verification (SV) and an optimized scheduling strategy for SV. Through extensive evaluation with publicly available LLMs across three NLP tasks, we show that our techniques improve performance by up to $2\times$, with an average speedup of $1.4\times$ in large-batch settings (32–80).

The rest of the paper is structured as follows. Section 2 provides background information and related work on SD. Section 3 examines uncertainty in speculation accuracy. Section 4 presents our proposed SV technique, and Section 5 details our optimized scheduling approach. Section 7 evaluates our methods, and Section 8 concludes the paper.

2 Background and Related Work

Speculative decoding (SD) reduces latency by using a small draft model to generate tokens speculatively, which are then verified in parallel by the target model. While SD improves efficiency, misalignment between the models can result in frequent token rejections and wasted computation.

Improving Alignment. To address distribution mismatches between the draft and target models, various alignment techniques have been proposed. DistillSpec[3] applies knowledge distillation to fine-tune draft models to better match the token distribution of target models. HRSS[4] further incorporates context information from the target model during distillation to improve alignment. Eagle[5, 6] trains draft models to detect misalignment and adjust the speculation length accordingly.

Self-speculative models perform speculation using the target model itself, thereby improving alignment. LayerSkip[7] and Kangaroo[8] use only the first few layers for drafting, while Medusa[9] uses the full target model with additional decoding heads to predict multiple tokens in parallel.

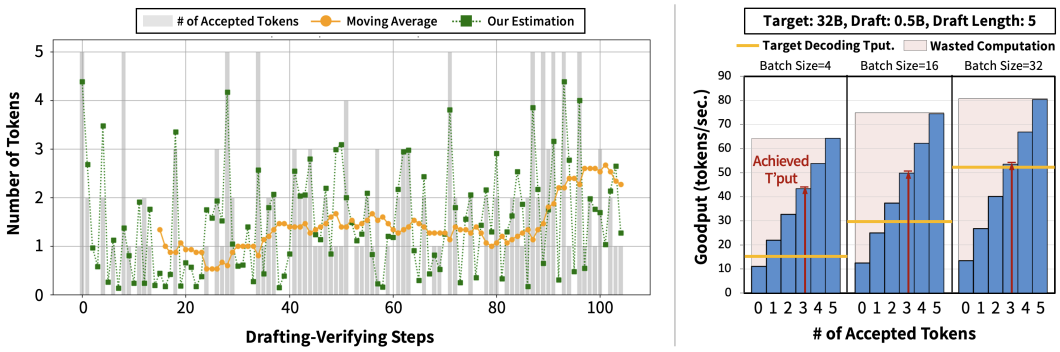
Tiered Speculation and Pipelining. To reduce verification overhead, some approaches introduce mid-sized models to verify drafted tokens before final verification in the target model. Staged spec [10] propose drafting using the n-gram method, verifying the tokens in a small model, and then performing final verification in the target model. HSDDW [11] uses a mid-sized model to decide whether to draft additional tokens before verification in the target model.

Using separate GPUs for drafting and verification with pipelining has recently been proposed to improve throughput. PEARL[12] allows drafting to continue speculatively on one GPU while previously drafted tokens are verified on another, assuming all are accepted. SPIN[13] runs multiple draft models in parallel on separate GPUs and selects the best output for verification.

3 Uncertainty in Speculation Accuracy

Since draft models are less capable than target models, their speculation accuracy is often inconsistent and uncertain [2, 13]. Predicting speculation accuracy could improve SD's effectiveness, so we explored whether it can be inferred using only the draft model's internal information. To investigate this, we ran SD on 128 prompts spanning two NLP tasks and analyzed the resulting speculation accuracy. Figure 1(a) shows a subset of the results for one representative query over 100 drafting-verification steps (draft length=5). The gray bars, representing the number of accepted tokens at each step, indicate that speculation accuracy fluctuates sharply and unpredictably across steps.

To understand what causes these accuracy changes, we examined tokens before and after sharp changes in accuracy. However, these tokens do not share any commonality in their embeddings or



(a) Accepted tokens, SV predictions, and moving average estimates.

(b) Goodput by acceptance length (0–5 accepted tokens)

Figure 1: Accepted tokens per SD steps (left) and throughput by acceptance length (right).

semantics (based on human interpretation). Moreover, we found that accuracy-changing positions often involve high-frequency tokens, such as stop words. This suggests that the observed fluctuations are not driven by meaning or context, which may explain why these positions tend to contain high-frequency tokens. A prior study proposed predicting speculation accuracy from recent history using a moving average [2], but as shown in the Figure 1, this estimation deviates significantly from actual speculation accuracy.

This uncertainty in speculation accuracy leads to a significant waste of verification cost. If all drafted tokens are rejected, SD can be more expensive than target decoding with the target model. To quantify this inefficiency, we ran SD with speculation lengths of 5 and batch sizes of 4 to 32 (using the same draft and target model as before) and analyzed the throughput when 0–5 tokens are accepted. We also measured the average number of accepted tokens and the corresponding throughput as shown in Figure 1(b). Overall, more than 40% of verification was spent on rejected tokens, and 48% of SD steps were more expensive than target decoding (not shown in the Figure 1). We also observed that at larger batch sizes, SD's already reduced performance gains are offset by the cost of running the draft model and by this verification overhead, which may result in overall performance degradation.

Predicting speculation accuracy is difficult due to the inherent complexity of token generation in LLMs. Attention heads across layers serve different roles that vary with context [14, 15]. Some compute attention scores globally over many tokens, while others focus locally on a subset. A single attention head is reported to switch unpredictably between global and local computations [16], which we also observed. Furthermore, our analysis showed that mapping attention heads between the draft and target models does not provide useful signals for speculation accuracy – even heads that strongly correlate with those in the target model show no noticeable differences in attention patterns between accepted and rejected tokens (not shown due to space limit).

From these preliminary analyses, we confirmed that predicting speculation accuracy based on the draft model's inference process, attention patterns, and past accuracy is infeasible. This uncertainty in speculation accuracy poses a challenge for scaling up inference serving with SD.

4 Introducing Speculative Verification

To address speculation uncertainty in SD, we propose using additional information, as the draft model alone cannot reliably predict speculation accuracy. We extract this information from another LLM instance of similar size to the draft model. By comparing their token distributions, we aim to reduce uncertainty in token acceptance and enable SD for scalable inference serving.

4.1 Information Gain for Efficient Speculation

SD accelerates inference because the draft model's token probability distribution is reasonably aligned with that of the target model. However, as discussed in Section 3, this alignment is inconsistent,

leading to unpredictable speculation accuracy. If the accuracy could be predicted, we could adjust the speculation length dynamically to minimize wasted verification cost and maximize SD efficiency.

To predict speculation accuracy, we introduce a small auxiliary model, i.e., a companion model, similar in size to or slightly larger than the draft model. We presume that both the draft and companion models are reasonably aligned with the target model – potentially distilled from it. We conjecture that the alignment of the draft and companion models indicates the alignment of the draft and target models. That is, if the probability distribution of the draft model for decoding a token is well aligned with the distribution of the companion model, it is likely that the distribution for the token also aligns well with that of the target model. Thus, by analyzing the alignment between the draft and companion models, we can accurately predict the acceptance of the drafted tokens.

More formally, we exploit the information gain from knowing the distribution similarity between the draft and companion models to reduce the uncertainty in speculation accuracy. If a random variable X represents the speculation accuracy, i.e., the acceptance probability of a token generated by the draft model, and Y denotes the corresponding distribution for the token in the companion model, then the uncertainty of X is measured as the entropy $H(X)$, and the conditional uncertainty is $H(X|Y)$ – representing the remaining uncertainty of X given the value of Y.

We aim to maximize the information gain $I(X;Y) = H(X) - H(X|Y)$, i.e., the amount of uncertainty reduced in speculation accuracy when knowing the companion model's distribution. We carefully choose Y – what to observe in the companion model's distribution (details discussed in the next section) – so that we can accurately predict the acceptance probability of drafted tokens.

With this prediction, we adjust the verification length (details in Section 5) to maximize SD's efficiency. Because the companion model helps determine which tokens to verify or discard, we call this method speculative verification (SV in short). Note that SV differs significantly from staged SD [10], where an intermediate model performs rejection sampling on the draft model's output. The staged approach assumes that the intermediate model is better aligned with the target model, but its effectiveness is inherently limited by the intermediate model's capability.

4.2 Indicators in Companion Model

What should we observe and define as the conditioning random variable to minimize speculation uncertainty? The requirements are: (1) it must be strongly correlated with the acceptance probability of the drafted tokens, and (2) it must be simple to measure and quantify. We propose observing the joint condition of two variables: one (S) measures the distributional similarity between the draft and companion models, and the other (A) measures the draft token's acceptance probability if we apply SD's sampling with the companion model (even though SV does not use SD's sampling mechanism). More formally, we define $S = \sum_{i \in \text{vocab}} \min\left(P_{\text{d}}(t_i), P_{\text{c}}(t_i)\right)$ and $A = \min\left(1, \frac{P_{\text{c}}(t_{\text{d}})}{P_{\text{d}}(t_{\text{d}})}\right)$, where t_{d} is a drafted token, and P_{d} and P_{c} are the token distributions of the draft and companion models, respectively.

The variable S indicates whether the draft and companion models are well aligned during the current decoding step. Our preliminary analysis confirmed that this alignment is strongly correlated with the alignment between the draft and target models. Specifically, we measured the similarity between the draft and companion models' output token distributions (using a natural divergence metric as described in [1] for its computational efficiency) and evaluated its correlation with the similarity between the draft and target models' distributions. Across three different draft-companion-target model groups, we observed strong correlations of 0.75, 0.87, and 0.82 (detailed in the appendix).

However, close alignment of the distributions alone is insufficient to minimize speculation uncertainty, as they may still differ largely at the sampled (drafted) token probability. Thus, we incorporate the drafted token's acceptance probability in the companion model as an additional conditioning variable. If this probability is high and the alignment indicated by S is strong, it indicates that the distributions are not only well aligned overall but also at the specific probability point of the drafted token.

When jointly conditioning on these two continuous variables, we discretize them into sub-ranges using adaptive binning, ensuring each bin contains a similar number of observations. For S (distributional similarity), which ranges from 0 to 1, we divide the interval into 10–30 bins. For A (token acceptance probability in the companion model), we partition the probability range into 10–20 bins. We then perform a profiling run of speculative decoding using these discretized variables and compute the

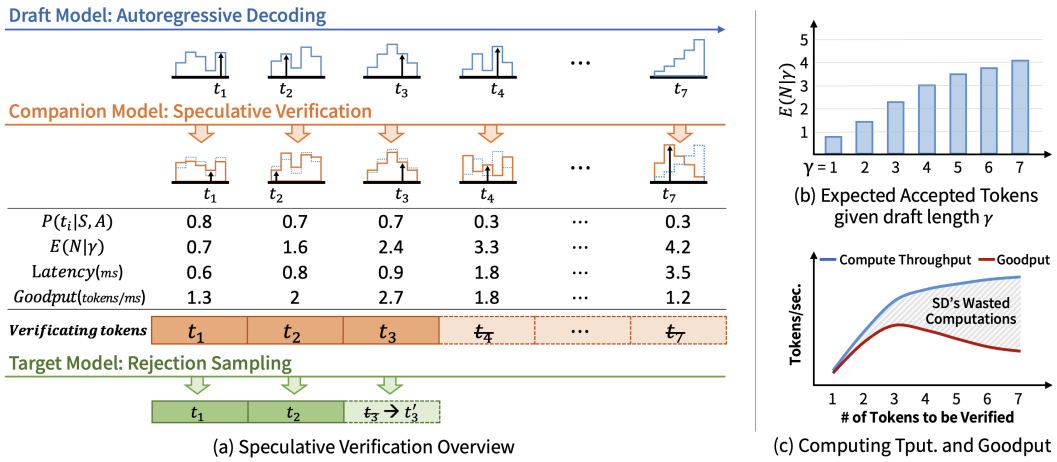


Figure 2: Example: selecting verification length in Speculative Verification.

Table 1: Notations for the random variables and probability distributions

Symbol	Description
S	Random variable for token distribution similarity of draft and companion models
A	Random variable for a token's acceptance probability in the target model
T_i	Random variable for the token at the i'th decoding position in the draft model
$P(T_i\|S,A)$	Conditional probability of a token T_i accepted in the target model
N	Random variable for the number of accepted tokens
γ	Number of drafted tokens that is verified in the target model

average token acceptance probability for each bin combination. Conditioning on these variables to dynamically adjust verification length reduces speculation uncertainty (measured in entropy) by 34% and improves the target model's acceptance rate by 20% (details in the evaluation section).

5 Scheduling for Speculative Verification

This section explains how to effectively utilize information from speculative verification to optimize scheduling and maximize effective throughput in terms of accepted tokens. We refer to this variant of throughput as goodput in this paper – the number of accepted tokens per unit time. To achieve high goodput, we must determine the optimal subset of drafted tokens to verify in the target model, i.e., optimal verification length.

We need to consider two factors to optimize verification length: 1) wasted computation on rejected tokens, and 2) GPU's resource utilization for a given verification length. Our goal is to balance these factors – minimizing wasted computation while maximizing GPU resource utilization – to achieve optimal goodput. We do this by estimating the expected number of accepted tokens for each possible verification length and selecting the length that maximizes estimated goodput (i.e., the expected number of accepted tokens divided by verification latency) as shown in Figure 2(b).

We first explain how to compute the expected number of accepted tokens for a given verification length. First, Table 1 defines the notations, where T_i represents the random variable for the i'th token in the draft model, $P(T_i|S,A)$ denotes its conditional acceptance probability in the target model (given information from the companion model), N is the number of accepted tokens in the target model, and γ is the number of drafted tokens verified in the target model.

The probability for N given γ is calculated as:

$$P_\gamma(N) = \begin{cases} P(T_{N+1} \neq t_{N+1}) \prod_{i=1}^{N} P(T_i = t_i) & \text{if } N < \gamma, \\ \prod_{i=1}^{\gamma} P(T_i = t_i) & \text{if } N = \gamma \end{cases}$$

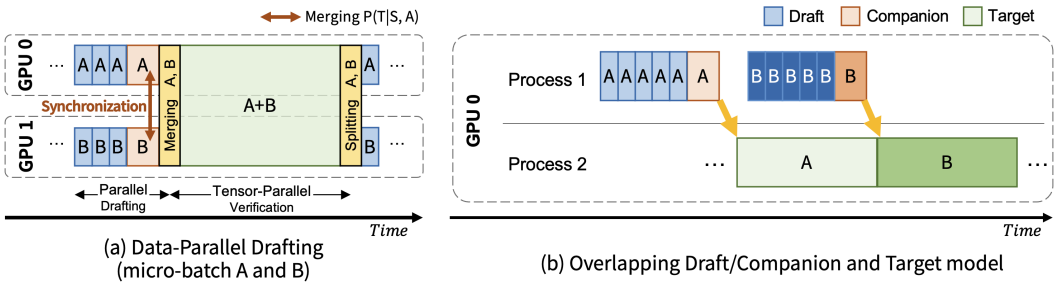


Figure 3: Runtime optimizations: Data-Parallel drafting and overlapping drafting with verification.

Then the expected number of accepted tokens with verification length γ is calculated as follows:

$$E(N|\gamma) = \sum_{i=1}^{\gamma} i \cdot P_\gamma(N = i)$$

Using the expected acceptance length $E(N|\gamma)$, we calculate goodput based on the profiled latency for a given verification length γ. We then vary γ, compute the corresponding goodput, and select the verification length that maximizes it. Figure 2 illustrates this process of computing token acceptance probability, expected acceptance length, and the optimal γ for maximum goodput.

We find the optimal γ by incrementally increasing it while goodput improves; once goodput declines, we revert to the previous γ as the optimal value. This approach works because goodput is concave with respect to the verification length. As we increase the verification length from a small value, latency grows slowly at first, since the GPU's compute resources are not fully utilized. Once the verification length is large enough for full resource utilization, latency increases proportionally, but the expected acceptance length grows more slowly – the cumulative probability of accepted tokens diminishes as more drafted tokens are included.

Scheduling for Batch Execution. To extend our approach to batch-level optimization, we apply the same goodput-based strategy used for single queries.We use a greedy algorithm that starts with an empty verification token sequences V and iteratively adds the token – regardless of which query it belongs to – that yields the greatest increase in expected acceptance length. This process continues as long as goodput improves and stops once it reaches its peak. While this approach may favor some queries over others, it avoids starvation: token acceptance rates drop sharply beyond a certain length, and verification always results in at least one accepted token per query, ensuring forward progress.

6 Implementation

We implemented canonical runtime optimizations for LLM inference in our system based on vLLM. Specifically, we implemented: 1) *input tensor compaction*, which allows verifying different token lengths for queries in the same batch without padding; 2) *data-parallel drafting*, in which multiple GPUs used for the target model's tensor-parallel verification are also fully utilized for drafting and speculative verification as illustrated in Figure 3(a); and 3) *CUDA graph execution*, to minimize kernel launch overhead. While implementing the last one, we fixed a bug in FlashInfer[17] that caused incorrect KV-cache value scores when used with CUDA graph capture and padded inputs.

We further optimize performance by overlapping the target model's verification with the drafting and speculative verification of the next iteration, as illustrated in Figure 3(b). To enable this, we run the target and draft/companion models in separate processes using Nvidia's Multi-Process Service (MPS), which allows them to share GPU resources. We configure MPS so that the draft/companion model uses a small fraction of the resources (30% by default), while the target model uses the remainder – or optionally all resources. Since this optimization involves running two micro-batches concurrently, we dynamically monitor the memory overhead of maintaining their contexts and adjust batch sizes accordingly. To accurately profile verification overhead, we measure it while the draft/companion models are running to capture interference effects, which remain consistent as their workload sizes (i.e., the number of tokens processed by the draft and companion models) are constant.

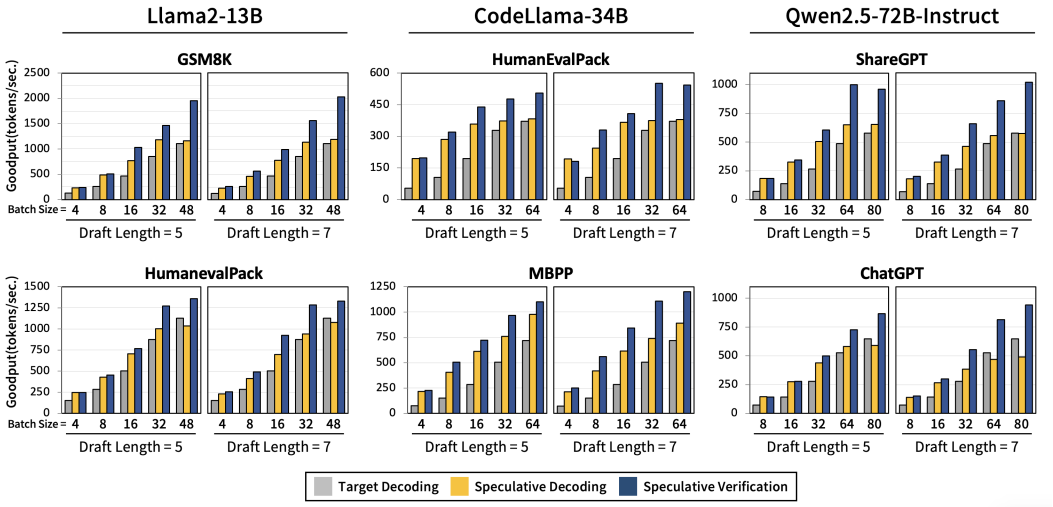


Figure 4: Overall performance: three target models (displayed at top) and six tasks (above each plot).

7 Evaluation

7.1 Evaluation Settings

We evaluate SV's performance improvements over both the baseline (decoding with the target model) and standard speculative decoding (SD). For the evaluation, we used two widely adopted public LLM families: Qwen [18] and Llama [19]. We tested base models (no fine-tuning), instruction-tuned models, and task-tuned models across three different target model sizes: 72B, 34B, and 13B. The evaluation covers three task types: conversation, code generation, and math (details in the appendix).

7.2 Overall Performance Evaluation

We first evaluated the token generation throughput of SV, comparing it to SD and target decoding with the target model. Using draft lengths of 5 and 7 tokens, we increased the batch size from 4, doubling up to 48 or 80 – the maximum supported within the GPU's memory limits.

Figure 4 shows results for a subset of models representative of others omitted due to space limits. In all experiments, SV outperforms both SD and target decoding. At the maximum supported batch sizes, SV is on average 1.3× faster than SD, with peak speedups reaching 2×. As batch size increases, SD's performance gains decrease and can even fall below that of target decoding – in such cases, SV outperforms SD by a large margin. Moreover, for difficult tasks, such as GSM8K[20] and ChatGPT[21], where SD is known to perform poorly [11, 13], SV continues to deliver strong performance.

7.3 Information Gain from Observing S and A

In this section, we quantify the information gain in the random variable X, i.e., the acceptance probability of a drafted token in the target model, by comparing the token distributions of the draft and companion models. We observe two variables – S and A: S measures the distributional similarity between the two models, and A estimates the draft token's acceptance probability based on the companion model, assuming SD's sampling rule is applied.

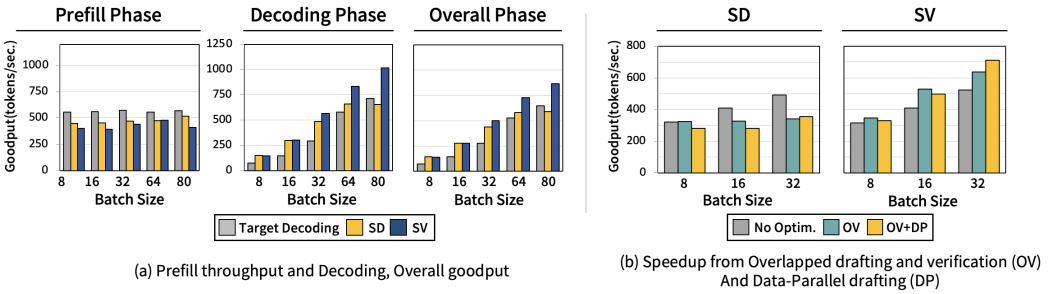
To measure the uncertainty of X (i.e., entropy) and the information gain from observing S and A, we perform inference using the Llama2 13B[22]/160M[23]/68M[23] models, applying SD on the ShareGPT[24] and HumanEvalPack[25] datasets. Specifically, we generate tokens using SD's process with three draft lengths: 3, 5, and 7 tokens. For the drafted tokens, we observe S and A using the companion model, but do not apply SV's optimization of verification lengths. We then use the target model to measure the acceptance probability of the drafted tokens based on SD's acceptance rule.

Table 2: Uncertainty in the acceptance probability of drafted tokens and the information gain from observing S and A (measured with Llama2 13B/160M/68M on ShareGPT and HumanEvalPack).

Observation Resolution	Uncertainty $H(X)$		Conditional Uncertainty $H(X\|S)$		$H(X\|A)$		$H(X\|S,A)$		Information Gain $I(X;S,A)$	
	chat	code	chat	code	chat	code	chat	code	chat	code
5 (5×5 for $\{S,A\}$)	1.38	1.78	1.09	1.55	1.07	1.58	1.01	1.50	0.38	0.28
10 (10×10 for $\{S,A\}$)	1.38	1.78	1.04	1.53	1.05	1.57	0.97	1.48	0.41	0.31
20 (20×20 for $\{S,A\}$)	1.38	1.78	1.03	1.52	1.04	1.57	0.95	1.46	0.43	0.32
Adaptive Binning(272)	1.38	1.78	1.02	1.52	1.04	1.57	0.91	1.42	0.48	0.37

After evaluating acceptance, we repeat the drafting and verification steps following the standard SD procedure.

Table 2 shows the uncertainty of X, the conditional uncertainties when observing S, A, and both, and the information gain from observing both variables. Since S and A are continuous, we discretize their ranges into equal-sized sub-ranges and report the corresponding conditional uncertainties at varying resolution levels (indicated in the leftmost column). We also include results using our adaptive binning approach, which assigns smaller bins where data points are denser (the bottom row). With adaptive binning, observing both S and A yields an information gain of 30–40% of the total entropy, indicating a strong relationship between these variables and X [26, 27, 28].



(a) Prefill throughput and Decoding, Overall goodput

(b) Speedup from Overlapped drafting and verification (OV) And Data-Parallel drafting (DP)

Figure 5: Performance Breakdown: Prefill vs. Decoding (left) and Runtime Optimizations (right)

7.4 Performance Breakdown

Prefill and Decoding Performance. We separately measured prefill throughput (processed tokens/sec) and decoding goodput (generated tokens/sec) for SD and SV. Figure 5 shows results for Qwen 72B/1.5B/0.5B models. Due to the companion model's overhead, SV's prefill throughput is lower than SD's. We partially mitigate this by skipping logit computations in the draft and companion models during prefill, which reduces overhead by 3–5%. Still, SV's prefill throughput remains about 10% lower than SD's (30% lower than target decoding). However, during the decoding phases, SV outperforms both SD and target decoding, especially at larger batch sizes. At a batch size of 64, SV is 20% faster than SD, and at 80, it is 40% faster. Overall, SV achieves 20–35% higher throughput for batch sizes between 64 and 80.

Runtime Optimization Breakdown. We also evaluated two major runtime optimizations: overlapping decoding with verification (OV) and data-parallel decoding (DP). We applied both optimizations to SD and SV and measured decoding goodput using Qwen 32B/1.5B/0.5B[29]. Figure 5(b) compares the goodput of SD and SV. For SD, applying OV degrades performance due to the high cost of verification; interference between verification and drafting reduces overall throughput.

With SV, applying OV consistently improves performance, as SV reduces the verification cost and thereby mitigates interference between drafting and verification. Applying DP on top of OV provides additional gains only at large batch sizes. At smaller batch sizes, however, the synchronization overhead of the companion model (shown in Figure 3(a)) outweighs the benefits of parallelism.

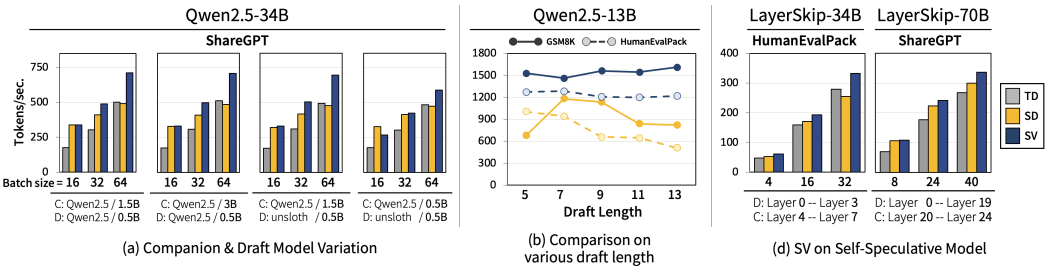


Figure 6: Effectiveness of SV: (a) Impact of draft and companion model selection; (b) sensitivity to draft length and dataset; (c) compatibility with self-speculative decoding models.

7.5 Fairness in Verification Token Selection

SV selects a subset of drafted tokens for verification and discards the rest to maximize goodput. As a result, some queries in a batch may consistently have few or no tokens selected for verification. While the target model's verification guarantees progress by generating at least one token per query—thus preventing starvation—we still evaluate the fairness of SV's verification token selection.

For this analysis, we ran generation with 1024 inputs (draft length=5) and calculated the average number of tokens verified for each sequence. We then examined the five queries with the smallest average verification lengths. The detailed results are presented in the appendix A. Compared to the overall average of 4.1 tokens, these bottom five queries had an average of 2.9 tokens verified. Notably, 39% of their steps involved verifying 4–5 tokens, while 47% involved 1–2 tokens, suggesting that token allocation remains fairly distributed even in these edge cases. This shows that SV's token selection remains reasonably balanced across queries and does not result in substantial unfairness.

7.6 Robustness and Generality of SV

Effect of Draft/Companion Model Selection. Using Qwen2.5 32B as the target, we evaluated SD and SV with different draft/companion pairs: 1.5B/0.5B, 3B/0.5B, and 0.5B/Unsloth 0.5B. As shown in Figure 6(a), SV consistently outperforms SD, with up to 1.4× speedup at batch size 64.

Effect of Draft Length and Datasets. We applied SD and SV with draft lengths from 5 to 13 on ShareGPT and HumanEvalPack, measuring goodput. As shown in Figure 6(b), SD's performance varies significantly with draft length and dataset, while SV consistently delivers better results.

Performance on SD Variants. We applied SV to self-speculative models (LayerSkip-34B/70B[30]), using 4 and 20 layers for drafting and next 4 and 5 layers for the companion. As shown in Figure 6(c), SV reliably outperforms SD, though gains are smaller with larger draft models due to overhead.

8 Conclusion

We proposed Speculative Verification (SV) that improves speculative decoding (SD) by dynamically adjusting verification lengths based on predicted token acceptance. To estimate speculation accuracy without access to the target model, SV introduces a companion model and compares its token distribution with that of the draft model. We show that alignment between the draft and companion models strongly correlates with the draft–target alignment, enabling effective prediction of token acceptance.

Building on this insight, SV adopts an information-theoretic framework to quantify alignment and guide verification decisions. This reduces wasted computation on rejected tokens and improves decoding efficiency, particularly at large batch sizes. Across nine model combinations, three NLP tasks, and batch sizes from 4 to 80, SV consistently outperforms both SD and standard decoding. It achieves up to 2× speedup over SD, with an average 1.4× gain in high-throughput scenarios.

SV maintains fairness in verification across queries, works with various fine-tuning types, and is compatible with self-speculative decoding. These results demonstrate SV's robustness, scalability, and practical utility for efficient LLM inference.

References

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

[2] Xiaoxuan Liu, Cade Daniel, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Xiangxi Mo, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Optimizing speculative decoding for serving large language models using goodput. *arXiv preprint arXiv:2406.14066*, 2024.

[3] Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*, 2024.

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

[5] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning*, 2024.

[6] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024.

[7] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *ACL (1)*, pages 12622–12642, 2024.

[8] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[9] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning*, 2024.

[10] Benjamin Frederick Spector and Christopher Re. Accelerating LLM inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023.

[11] Shensian Syu and Hung-yi Lee. Hierarchical speculative decoding with dynamic window. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8260–8273, 2025.

[12] Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*, 2025.

[13] Fahao Chen, Peng Li, Tom H Luan, Zhou Su, and Jing Deng. Spin: Accelerating large language model inference with heterogeneous speculative models. *arXiv preprint arXiv:2503.15921*, 2025.

[14] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[15] George Wang, Jesse Hoogland, Stan van Wingerden, Zach Furman, and Daniel Murfet. Differentiation and specialization of attention heads via the refined local learning coefficient. *arXiv preprint arXiv:2410.02984*, 2024.

[16] Konstantin Donhauser, Charles Arnal, Mohammad Pezeshki, Vivien Cabannes, David Lopez-Paz, and Kartik Ahuja. Unveiling simplicities of attention: Adaptive long-context head identification. *arXiv preprint arXiv:2502.09647*, 2025.

[17] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for LLM inference serving. In *Eighth Conference on Machine Learning and Systems*, 2025.

[18] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[19] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[20] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[21] Mohamed Rashad. Chatgpt prompts dataset. `https://huggingface.co/datasets/MohamedRashad/ChatGPT-prompts`, 2023.

[22] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[23] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, 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*, pages 932–949, 2024.

[24] anon8231489123. Sharegpt vicuna unfiltered dataset. `https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered`, 2023.

[25] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. *arXiv preprint arXiv:2308.07124*, 2023.

[26] Stefano Panzeri and Alessandro Treves. Analytical estimates of limited sampling biases in different information measures. *Network: Computation in neural systems*, 7(1):87, 1996.

[27] J Ross Quinlan. Improved use of continuous attributes in c4. 5. *Journal of artificial intelligence research*, 4:77–90, 1996.

[28] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[29] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[30] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[31] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[32] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[33] AMD. AMD-Llama-135m: A 135M Parameter Language Model. `https://huggingface.co/amd/AMD-Llama-135m`, 2024.

[34] Unsloth. Qwen2.5-0.5b. `https://huggingface.co/unsloth/Qwen2.5-0.5B`, 2024.

[35] Meta AI (Facebook). Layerskip llama 2 70b. `https://huggingface.co/facebook/layerskip-llama2-70B`, 2024.

[36] Meta AI (Facebook). Layerskip codellama 34b. `https://huggingface.co/facebook/layerskip-codellama-34B`, 2024.

[37] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

A Additional Experimental Data

The detailed experimental results for Section 7.5 are shown in Table 3.

Table 3: Number of Steps by Verification Length from Worst/Best Case Requests

Cases	Request ID	Avg. Veri. Length	γ=1	γ=2	γ=3	γ=4	γ=5
Worst Cases	125	2.85	6	8	3	4	6
	390	2.88	5	3	4	1	5
	141	2.90	3	3	0	2	3
	121	3.00	1	1	1	1	1
	441	3.11	5	8	4	1	10
Best Cases	274	4.96	0	0	0	2	43
	44	4.95	0	0	0	1	19
	211	4.94	0	0	1	1	47
	419	4.91	0	1	0	0	43
	62	4.89	0	0	2	1	43

B Limitations and Future Work

Although we extensively evaluated our proposed methods, the experiments were limited to publicly available models, which may introduce model-specific biases that affect the results. In addition, we did not include experiments on reasoning tasks due to limited public availability. Public reasoning models with multiple size variants – required for speculative verification – were only released at the end of April 2025. We plan to evaluate SV on reasoning tasks and report the results in the final version of the paper.

C Evaluation Settings in Detail

Table 4: Hardware settings and model assignments for all tasks

Task	**Dataset**	**GPU**	**Architecture**	**Size (T/C/D)**[†]
Chat*	ChatGPT, ShareGPT	A100×4	Qwen2.5-Instruct	72B / 1.5B / 0.5B
Chat*	ShareGPT	A100×2	Qwen2.5-Instruct	32B / 1.5B / 0.5B
Chat*	ShareGPT	A100×2	Qwen2.5-Instruct	32B / 3B / 0.5B
Chat*	ShareGPT	A100×2	Qwen2.5-Instruct	32B / 1.5B / 0.5B**
Chat*	ShareGPT	A100×2	Qwen2.5-Instruct	32B / 0.5B** / 0.5B
Code*	Humaneval, MBBP	A100×2	CodeLlama / TinyLlama / AMDLlama	34B / 1.1B / 135M
Code, Math	Humaneval, GSM8K	A100×1	Llama2	13B / 160M / 68M
Chat	ShareGPT	A40×4	LayerSkip-Llama	70B / 70B(5)[‡] / 70B(20)[‡]
Code*	Humaneval	A40×2	LayerSkip-CodeLlama	34B / 34B(4)[‡] / 34B(4)[‡]

* Finetuned ** unsloth/Qwen2.5 [†] (Target, Companion, Draft) [‡] Layerskip with (N) layers

C.1 Hardware Environment

All experiments were conducted across three distinct computing environments to accommodate the varying computational requirements of different models. For the largest models (Qwen2.5-72B-Instruct as target model), we utilized an Azure Cloud VM equipped with an AMD EPYC 7V13 (Milan) 64-core processor and four NVIDIA A100 PCIe GPUs, each with 80GB VRAM. This system was configured with 2TB of RAM to handle the substantial memory requirements of these parameter-dense models. Medium-sized models (Qwen2.5 32B/1.5B/0.5B, CodeLlama 34B variants, and Llama2 13B variants) were deployed on a Runpod Cloud instance featuring an AMD EPYC 7352 24-core processor paired with two NVIDIA A100 PCIe GPUs, each with 80GB VRAM, and 200GB of system RAM. For the Layerskip experiments on Layerskip-Llama2-70B and Layerskip-CodeLlama-34B models, we employed a private computing resource with an AMD EPYC 7313 16-core processor, four NVIDIA A40 GPUs, each with 48GB VRAM, and 500GB of system RAM. All systems ran Ubuntu 22.04 LTS with CUDA 12.4 to maintain environmental consistency across experimental platforms.

C.2 LLM Models and hyperparameters

To demonstrate the versatility and broad applicability of SV, we selected two widely-used open-source LLM families: the Qwen and Llama series. Our experimental design encompasses three dimensions of variation per family: base models (no fine-tuning), instruction-tuned models, and task-tuned models.

Sampling Parameters For all models, we adhered to the sampling hyperparameters recommended in their respective Hugging Face repositories or official GitHub documentation. Specific values are shown in Table 5:

Table 5: Sampling hyperparameters for each model series.

Model series	top_k	top_p	temperature	repetition_penalty
Qwen2.5/Qwen2.5 based models	20	0.8	0.7	1.05
Llama/Layerskip/Llama based models	—	0.9	0.6	—

Models used in overall evaluations

- **Large-sized models:** We employed the Qwen2.5-Instruct family [29] for large-sized evaluation in main experiments, we used the 72B variant as the *target* model, the 1.5B variant as the *companion* model, and the 0.5B variant as the *draft* model.
- **Mid-sized models:** We selected the CodeLlama-34B [31] as target model for mid-scale experiments. and TinyLlama_v1.1_math_code (1.2B) [32] as the *companion*, and AMD-Llama-135M [33] as the *draft*.
- **Small-sized models:** To cover smaller models, we included the Llama2-14B variant [22] as the *target* model, paired with a JackFram_llama-160m [23] as the *companion*, and JackFram_llama-68m [23] as the *draft*, which was specifically trained for speculation tasks with a reduced parameter size.

Models used in other evaluations and analysis For additional analytical experiments in (7.4, 7.5, 7.6), we utilized the 32B/1.5B/0.5B model sizes of Qwen2.5-Instruct family. To test robustness across fine-tuned variants, we incorporated the unsloth-fine-tuned version of Qwen2.5-Instruct models [34].

We also evaluated Layerskip-Llama2-70B and Layerskip-CodeLlama-34B to assess performance of SV adopted on self-speculation techniques. The number of drafting layers was set to the default values specified by model providers in huggingface repository [35, 36].

C.3 Dataset and pre-processing

We evaluated our approach using three distinct task categories: dialogue, code generation, and mathematical reasoning. For all experiments, we maintained consistency by using identical randomly sampled subsets across different evaluation scenarios.

For probability profile construction, we extracted 512 samples from training sets where available. For datasets without explicit train-test splits, we randomly sampled 512 instances. For goodput evaluation, we randomly selected between 128 and 256 samples from evaluation/test sets, carefully excluding any samples that appeared in the probability profile to prevent data leakage. These randomly sampled datasets remained constant across all experimental conditions to ensure fair comparisons.

For dialogue evaluation, we utilized two comprehensive datasets: ShareGPT [24], a collection of human-assistant conversations extracted from various online sources, and ChatGPT Dataset [21], consisting of diverse dialogue prompts and responses.

Our code generation evaluation encompassed six programming languages using the HumanEvalPack [25] benchmark, which includes Python, C++, Java, JavaScript, Rust, and Go. We constructed a balanced subset by randomly sampling tasks across all languages to ensure comprehensive coverage of different programming paradigms and syntactic structures. We also incorporated MBPP [37], which consists of approximately 1,000 crowd-sourced Python programming problems.

To assess mathematical reasoning capabilities, we employed the GSM8K [20] dataset, which contains grade school math word problems that require multi-step reasoning to solve.

These three task categories—dialogue, code generation, and mathematical reasoning—were selected to provide a thorough evaluation of model performance across domains requiring different cognitive abilities and knowledge representations.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In the abstract and Section 1, we clearly demonstrate the contribution and scope of this paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of this work in B in appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: In Section 4, 5, we have detailed the motivation and proofs of the formulas we presented.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Based on the formulas, algorithms, descriptions, datasets, and experimental settings presented in this paper, one can implement the code and reproduce the experimental results as in this paper. Additionally, if the paper is accepted, we will consider releasing the code for the benefit of other researchers.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: For security reasons, we plan to postpone releasing the code until the paper is officially published. Once the paper is accepted, we will script every step we used —including data pre-processing and experiment execution— and make it publicly available so that other researchers can run it immediately.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: In appendix C, we have provided detailed descriptions of the dataset sources, preprocessing methods, and sampling hyperparameters used for LLM inference, and in the appendix we specify which references were consulted to determine these values.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Due to the high cost of produce experimental results across various settings, we do not repeat the same experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: In appendix C, we provide the details of computer resources for reproducing the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We affirm that the research presented in this paper fully complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

Justification: : This paper aims to improve the inference efficiency of large launguage model serving system, without any negative societal impacts.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper does not present any such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We have duly credited the creators or original owners of all assets used in this paper—including code, data, and models—and have clearly stated and respected their licenses and terms of use.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not include crowdsourcing experiments nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve any crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [No]

Justification: In this work, any method development does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.