Speculative Verification: Exploiting Information Gain for Speculative Decoding

Anonymous ACL submission

Abstract

Speculative decoding (SD) improves LLM inference latency by speculatively generating multiple tokens with a small draft model and verifying them with a larger target model. However, when speculation accuracy is low, the overhead from rejected tokens can negate its benefits, especially at large batch sizes.

We propose Speculative Verification (SV), an efficient augmentation to SD that predicts speculation accuracy and dynamically adapts the verification length to maximize throughput. SV introduces a small companion model, similar in size to draft model, to reduce uncertainty in speculation accuracy. By exploiting the information gain from observing the companion distribution, SV reduces wasted verification on rejected tokens and improves decoding efficiency.

We evaluate SV across publicly available LLMs on seven NLP tasks using over a hundred combinations of draft, companion, and target models, including 13B–72B target models spanning base, instruction-tuned, and task-specific fine-tuned variants. Compared to target-only decoding, standard SD, and state-of-the-art SD variants, SV consistently delivers higher throughput across batch sizes. SV improves SD performance by up to 1.9×, with an average 1.4× speedup at large batch sizes, showing robust and scalable gains for practical 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 resource underutilization and increased latency.

Speculative Decoding (SD) (Leviathan et al., 2023) 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 validates multiple tokens in a resource-efficient manner, overall latency is reduced. However, if 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; if most drafted tokens are rejected, or only a small fraction are rejected at large batch sizes (where SD's gain already drops), the verification overhead can make SD slower than target decoding. Speculation accuracy depends on the alignment between the draft and target models, which fluctuates due to their capability gaps and input context variations.

Identifying when these distributions align makes it possible to adjust the speculation length to minimize verification overhead for rejected tokens. However, predicting this alignment is challenging due to the complexity of LLM inference. Prior approaches attempt to predict acceptance using the draft model's internal signals (Agrawal et al., 2024; Zhang et al., 2025b) or past acceptance history (Liu et al., 2024b), but our evaluation shows that these methods become ineffective as batch size increases.

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 and implementation for SV. Evaluations with publicly available LLMs across seven NLP tasks show up to 1.9× speedup over SD, and an average of 1.4× in large-batch settings (32–64).

The rest of the paper is structured as follows. Section 2 provides background information and related work on SD. Section 3 discusses uncertainty in speculation accuracy. Section 4 presents our proposed SV technique. Section 5 details our optimized scheduling approach and Section 6 describes our prototype implementation. 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 verified in parallel by the target model. Because SD's performance depends on speculation accuracy, prior work has focused on improving this accuracy or predicting it to adapt the speculation length; we review these works below.

Distillation and Self-Speculative Models. To improve draft-target similarity, prior work has applied knowledge distillation, as in DistillSpec (Zhou et al., 2024) and HRSS (Zhang et al., 2025a). Because distillation requires costly fine-tuning, self-speculative models have been proposed to perform speculation mostly using the target model itself, thereby reducing additional training overhead.

LayerSkip (Elhoushi et al., 2024) and Kangaroo (Liu et al., 2024a) use the first few layers of the target model for drafting. Medusa (Cai et al., 2024) employs the full target model with additional decoding heads to predict multiple draft tokens in parallel. Eagle-3 (Li et al., 2025) further uses hidden states from multiple target-model layers in its decoding heads and generates draft token trees instead of linear sequences to better utilize hardware resources, especially for single-batch inference.

Predicting Speculation Accuracy. To predict token acceptance probability, SmartSpec (Liu et al., 2024b) uses a moving average of recent acceptance rates and adjusts the draft length accordingly. Tetris (Wu et al., 2025) and DySpec (Xiong et al., 2025) predict token acceptance using the draft model's probability for sampled tokens and prioritize verification accordingly. However, their approach relies on the strong assumption that the draft model's token probability is well correlated with acceptance probability, which often does not hold in practice.

To address this limitation, SVIP (Zhang et al., 2025b), AdaEDL (Agrawal et al., 2024), and DISCO (Mamou et al., 2024) propose predicting acceptance based on the entropy of the draft model's token distribution. These methods are motivated by the fact that token acceptance can be bounded using the KL divergence between the draft and target distributions. However, this bound is known to be loose (Canonne, 2022; Zhang et al., 2025b), and in practice the target distribution is unavailable during drafting. Hence, these methods estimate the bound using only the draft distribution's entropy, leading to inaccurate acceptance prediction, as we show in Section 7.3.

Staged speculation introduces mid-sized models to verify drafted tokens before final verification by the target model (Spector and Re, 2023; Syu and Lee, 2025). Rather than predicting speculation accuracy, the intermediate model performs speculative sampling on the draft output to reduce the target's verification overhead. However, performance is fundamentally limited by the capability of the intermediate model and can even be worse than target-only decoding, as shown in our evaluation.

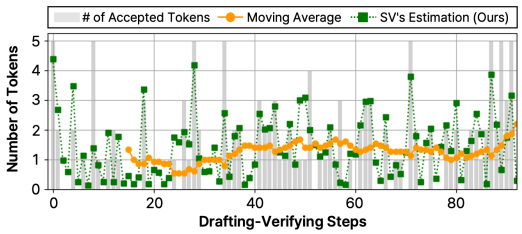

Figure 1: Accepted tokens per SD step: actual vs. estim.

3 Uncertainty in Speculation Accuracy

Since draft models are less capable than target models, their speculation accuracy is often inconsistent and uncertain (Liu et al., 2024b; Chen et al., 2025). Predicting speculation accuracy could improve SD's effectiveness, so we explored whether it can be inferred using only the draft model's information. To investigate this, we ran SD on 128 prompts across two NLP tasks and analyzed the resulting speculation accuracy. Figure 1 shows a subset of the results for one representative query over 100 draft-verification steps (draft length=5). The gray bars (accepted tokens per step) indicate that speculation accuracy fluctuates sharply and unpredictably across steps.

To understand what causes the accuracy changes,

we examined tokens before and after sharp changes in accuracy. However, these tokens do not share any commonality in their embeddings or semantics (based on human interpretation). Moreover, we found that majority of accuracy-changing positions involve high-frequency tokens, such as stop words, which suggests that the observed fluctuations are not likely driven by meaning or context. A prior study proposed predicting speculation accuracy from recent history using a moving average (Liu et al., 2024b), but as shown in Figure 1, this estimation deviates from the actual accuracy. We also tested entropy-based predictions (e.g. SVIP), which showed limited accuracy (see Section 7.3).

This uncertainty leads to a significant waste of verification cost. In our preliminary experiment, we found that over 40% of verification effort was spent on rejected tokens, and that 48% of SD steps were more expensive than decoding directly with the target model. We also observed that at larger batch sizes, SD's already reduced performance gains are further offset by the cost of running the draft model and the additional verification overhead, which can result in overall performance degradation.

Predicting speculation accuracy is difficult due to the inherent complexity of token generation in LLMs. For instance, attention heads across layers serve different roles that vary with context (Clark et al., 2019; Wang et al., 2024). Some compute attention scores globally over many tokens, while others focus locally on a subset. A single attention head can switch unpredictably between global and local computations (Donhauser et al., 2025). Consistent with prior findings, our analysis correlating attention heads between the draft and target models shows that draft-side attention patterns do not provide reliable signals for predicting speculation accuracy.

From these preliminary analyses, we found that predicting speculation accuracy based on the draft model's token distribution (or its entropy), attention patterns, or past accuracy is infeasible. This uncertainty in speculation accuracy poses a major challenge to scaling 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 while preserving the target model's output distribution.

4.1 Information Gain for Efficient Speculation

SD accelerates inference because the draft model's token distribution is reasonably aligned with that of the target model. However, this alignment is inconsistent, leading to unpredictable speculation accuracy, as discussed in Section 3. If the accuracy could be predicted, the speculation length could be adjusted dynamically to minimize wasted verification cost and maximize SD efficiency.

To predict speculation accuracy, we introduce a small auxiliary model, namely a *companion* model, similar in size to the draft model. We assume that comparing the token distributions of the draft and companion models provides positive information for predicting token acceptance by the target model; for example, similarity between the draft and companion distributions can indicate alignment with the target model. This assumption holds as long as the two models are not statistically independent. Since modern LLMs are typically trained on partially shared datasets (e.g., Wikipedia and C4), this condition holds in practice. We further demonstrate in our evaluation that, across all examined combinations of draft, companion, and target models (trained by the same or different entities), SV yields consistently positive information gain.

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 (see Section 5) to maximize SD efficiency.

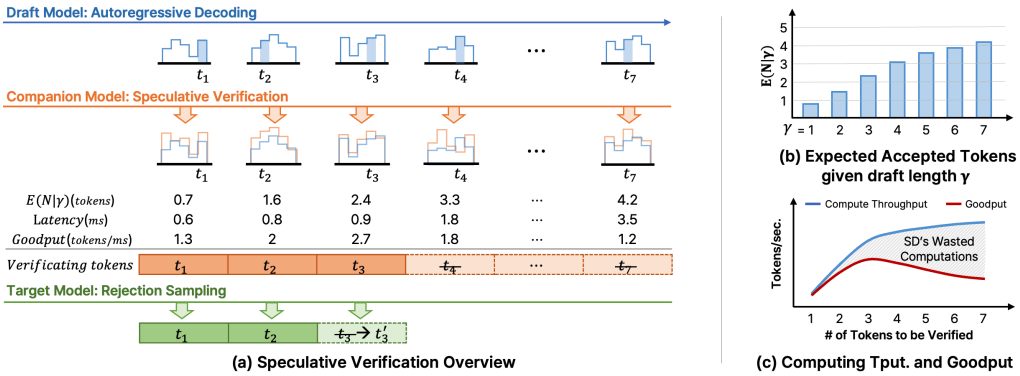

Figure 2: Example of selecting the verification length to maximize goodput (accepted tokens/sec) in SV.

Because the companion model helps determine which tokens to verify or discard, we call this method speculative verification (SV). Note that SV differs significantly from staged SD, where an intermediate model performs speculative sampling on the draft output. Their approach assumes that the intermediate model is better aligned with the target model, but its efficiency is limited by the intermediate model's capability, as shown in our evaluation.

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 under speculative sampling in the companion model (although this sampling is not actually performed). More formally, S is a natural divergence in SD, i.e., $S = \sum_{i \in \text{vocab}} \min\left(P_{\text{d}}(t_i), P_{\text{c}}(t_i)\right)$ 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.

S denotes the similarity of the draft and companion token distributions at the current decoding step. When these distributions are similar – i.e., the draft and companion models agree on likely next tokens – the target model's distribution is also likely similar, as the current context is relatively easy to decode even for less-capable models. Thus, S serves as an indicator of draft–target distribution similarity.

We empirically validated this indicator by measuring the similarity between the draft-companion token distributions and correlating it with the similarity between the draft-target distributions. Across three draft-companion-target model groups, we observed strong correlations of 0.75, 0.87, and 0.82 (see appendix B.1 for details).

However, distribution similarity alone is not sufficient to minimize speculation uncertainty, as the specific drafted token also has a significant impact on its acceptance probability. To account for this, we incorporate the drafted token's acceptance probability under the companion model as an additional conditioning variable A. Together, this probability and the distribution similarity S yield a reliable predictor of token acceptance, substantially reducing speculation uncertainty.

We validated experimentally that these two variables significantly reduce speculation uncertainty. The details are in Section 7.4, but using S and A reduces speculation uncertainty by 34% and improves the target model's acceptance rate by 20%.

5 Scheduling for Speculative Verification

We now explain how to take advantage of the predicted acceptance probability to 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 estimat-

ing 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) and 2(c).

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

The probability for N given γ is calculated as:

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

where we use $P(T_i|S, A)$ instead of $P(T_i)$ for a more accurate acceptance probability, or its empirical estimate $\hat{P}(T_i|S, A)$, which is obtained from sampled data. Then, the expected number of accepted tokens conditioned on γ is $E(N|\gamma) = \sum_{i=1}^{\gamma} i \cdot P_\gamma(N = i)$.

Using the expected acceptance length $E(N|\gamma)$, or its estimate $\hat{E}(N|\gamma)$, 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 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 and iteratively adds the token that yields the highest increase in expected acceptance length, regardless of which query it belongs to. This process continues as long as goodput improves and stops once it reaches its peak. While this strategy may prioritize some queries over others, it does not cause starvation. In practice, token acceptance rates drop sharply beyond a certain length, and verification always yields at least one accepted token per query, ensuring forward progress (see the fairness study in our evaluation).

6 Implementation

We implemented standard LLM inference optimizations in our vLLM-based system, including input tensor compaction (to avoid padding for variable-length verification), data-parallel drafting (reusing multiple GPUs allocated for tensor-parallel verification for drafting), and CUDA graph execution (to reduce kernel launch overhead; applied to all baselines – SD, SV, and other SD variants). In doing so, we also fixed a FlashInfer bug that miscomputed attention scores when CUDA graph capture was used with padded inputs (Ye et al., 2025).

We further optimize performance by overlapping the target model's verification with the drafting and speculative verification of the next iteration. 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 uses a small fraction (30%) of the resources, while the target uses the remainder – or optionally all resources. Since this optimization involves running two micro-batches concurrently, we 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 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.

7 Evaluation

7.1 Evaluation Settings

We evaluate SV against target-only decoding, standard SD, and five SD variants (including SVIP and Eagle-3). Our evaluation uses two widely adopted LLM families, Qwen (v2.5, v3) and Llama (v2, v3), with 104 draft/companion/target combinations and five target sizes (13B – 72B), covering base,

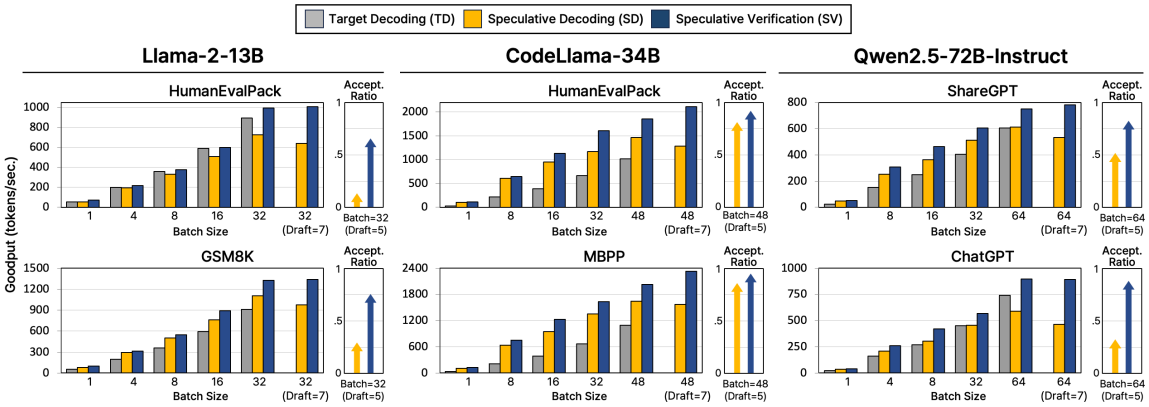

Figure 3: Goodput (accepted tokens/sec) of target-only decoding, SD, and SV across three target models and six tasks.

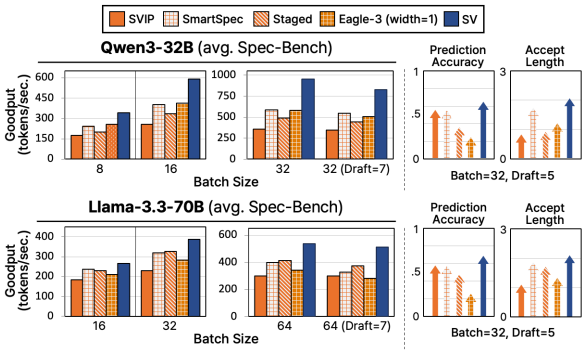

Figure 4: Goodput of SOTA SD variants and SV.

instruction-tuned, and task-tuned models across seven tasks. Detailed settings are in the appendix.

7.2 Overall Performance Evaluation

We evaluated the token generation throughput of SV, comparing it to target-only decoding and SD. Using draft lengths of 5 and 7 tokens, we increased the batch size from 1, doubling up to 32–64, the maximum supported by the GPU's memory.

Figure 3 shows results for a representative subset of models and tasks (other results omitted due to space limits) with draft length of 5. In all experiments, SV consistently outperforms both SD and target decoding. At the maximum supported batch sizes, SV is on average 1.4× faster than SD, with peak speedups reaching 1.9×; at the same batch sizes, SV also achieves up to 4.5× higher token accept rate (indicated by the arrow-shaped bars). 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 and ChatGPT, where SD is known to perform poorly (Syu and Lee, 2025; Chen et al., 2025), SV continues to deliver strong performance.

7.3 Comparison to SOTA SD Variants

We compare SV with other state-of-the-art (SOTA) SD variants, including SVIP, SmartSpec, Staged SD, and Eagle-3. We use the same previous evaluation settings. For Eagle-3, we use the authors' official models and set the tree width to 1 for batch sizes>1 following the authors' setting. For the other variants, whose implementations are not publicly available, we implemented them following the descriptions in their papers. We verified that our implementations match the reported single-batch performance; multi-batch performance is not reported for most prior methods.

Figure 4 shows results for two target models on Spec-Bench, with full per-subtask results reported in the appendix. Across all batch sizes and models, SV consistently outperforms the other SD variants. Compared to the best SD variant, SV is up to 1.61× faster for Qwen(batch=32, draft=5) and 1.37× faster for Llama(batch=64, draft=7). At the maximum batch sizes, SV is 1.3–2.64× faster than the SD variants.

At large batch sizes, the efficiency of these variants degrades for different reasons. For SVIP and SmartSpec, draft lengths can vary widely across queries in the same batch, causing a small number of long drafts to incur large drafting overhead. For Staged SD, performance is limited by the capability of the intermediate model, which becomes significant as batch size grows. For Eagle-3, speculation accuracy is low; while token-tree generation mitigates this issue for single-batch inference, its overhead becomes dominant at larger batch sizes.

7.4 Information Gain from 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

Resolution	Uncertainty $H(X)$		Cond. Uncertainty $H(X\|S,A)$		Info. Gain $I(X;S,A)$	
	chat	code	chat	code	chat	code
5×5	1.38	1.78	1.01	1.50	0.38	0.28
10×10	1.38	1.78	0.97	1.48	0.41	0.31
20×20	1.38	1.78	0.95	1.46	0.43	0.32
272	1.38	1.78	0.91	1.42	0.48	0.37

Table 1: Uncertainty in the token acceptance probability and the information gain from observing S and A.

Target Model	# of D-C pairs	Info. Gain
Llama-2-13B-chat	30	0.07 – 0.59
Qwen2.5-14B-Instruct	30	0.03 – 0.60
Llama-3.3-70B-Instruct	30	0.03 – 0.40

Table 2: Information gain across 90 draft-companion-target (D-C-T) models (full table in appendix B.3).

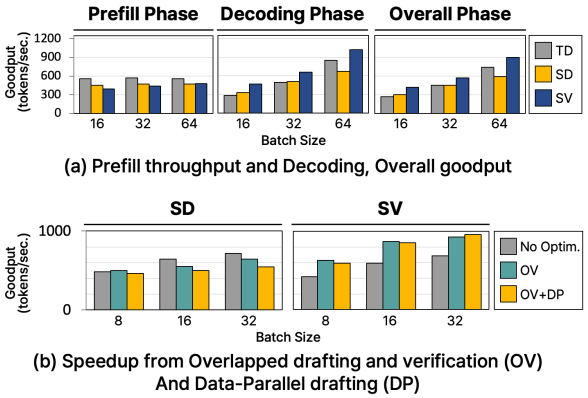

Figure 5: Performance Breakdown: Prefill vs. Decoding (top) and Runtime Optimizations (below)

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 is 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/160M/68M models, applying SD on the ShareGPT and HumanEvalPack datasets. Specifically, we generate tokens using SD's process with two draft lengths (5 and 7). 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. After evaluating acceptance, we repeat the SD's standard drafting and verification steps.

Table 1 shows the uncertainty of X, the conditional uncertainties when observing S and A, and the information gain from observing the two 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, which assigns smaller bins where data samples are denser (the bottom row). With adaptive binning, observing S and A gives an information gain of 30–40% of the total entropy, indicating a strong relationship between these variables and X (Panzeri and Treves, 1996; Quinlan, 1996).

Furthermore, we measured information gain more extensively across 90 publicly available draft–companion–target (D–C–T) model combinations using the Spec-Bench dataset. Table 2 summarizes the observed ranges of information gain across these combinations, which cover nearly all publicly available configurations for the three target models (Llama-2-13B, Qwen2.5-14B, and Llama-3.3-70B). Across all 90 combinations, we confirmed that SV's information gain is consistently positive (see appendix B.3 for details).

7.5 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 32, SV is 25% faster than SD, and at 64, it is 52% faster. Overall, SV achieves higher throughput consistently across all batch sizes.

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. Figure 5(b) compares the goodput of SD and SV. For SD, applying OV degrades performance due to the high cost of verification; that is, interference between verifica-

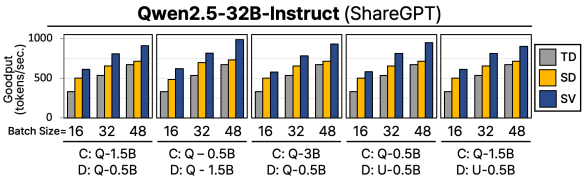

Figure 6: SV speedup across five draft-companion pairs of different model sizes and training entities (Q:Qwen2.5, U:Unsloth; see appendix A.1 for details).

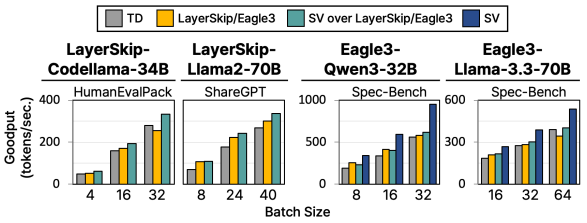

Figure 7: Performance of SV over LayerSkip/Eagle3.

tion 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 outweighs the benefits of parallelism.

7.6 Fairness in Verification Token Selection

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

For this analysis, we ran generation with 1024 inputs and calculated the average number of tokens verified for each sequence. We then examined the five queries with the smallest average verification lengths. Compared to the overall average of 4.1 tokens, these bottom five queries had an average of 2.9 tokens verified; 39% of their steps involved verifying 4–5 tokens, while 47% involved 1–2 tokens, suggesting that their token allocation is fairly distributed. The full results are shown in the appendix, but SV's token selection is reasonably balanced and does not result in substantial unfairness.

7.7 Robustness and Generality of SV

Effect of Draft/Companion Selection. Using Qwen2.5-32B as the target model, we evaluated SV with five draft/companion pairs spanning different model sizes and training entities (0.5B, 1.5B, and 3B models trained by Alibaba and Unsloth; see appendix A.1). Figure 6 shows that SV's performance is robust to draft/companion choice, remaining consistent across all evaluated combinations.

SV for Self-Speculation. We applied SV to two self-speculative models, LayerSkip and Eagle-3. For LayerSkip, we evaluated CodeLlama-34B and Llama2-70B (the drafting and companion layers specified in the appendix). For Eagle-3, we evaluated Qwen3-32B and Llama3.3-70B, using Qwen3-0.6B and Llama-3.2-1B as companion models.

Figure 7 shows the performance impact of applying SV to these models. For LayerSkip, SV yields substantial and consistent improvements, increasing performance at maximum batch sizes by 30% for CodeLlama and 12% for Llama-2. For Eagle-3, applying SV yields more modest gains and may degrade performance slightly at small batch sizes. This is due to the limited capability of Eagle-3's draft model, which achieves low acceptance rates of only 26% and 27% for drafted tokens. Using a stronger draft model would likely improve the synergy with SV. When we replaced the draft with a separate, more capable model, performance increased by up to 56%, as indicated by the SV results in Figure 7.

8 Conclusion

We presented Speculative Verification (SV), a technique that improves speculative decoding (SD) by adapting verification lengths based on predicted token acceptance. SV introduces a lightweight companion model and uses the alignment between its token distribution and that of the draft model to estimate draft–target alignment, which enables reliable prediction of speculation accuracy.

SV uses an information-theoretic framework to guide verification decisions, reducing wasted computation on rejected tokens and improving decoding efficiency, particularly at large batch sizes. Extensive evaluation across public LLMs, multiple NLP tasks, diverse draft–companion–target combinations, and batch sizes up to 64 shows that SV consistently outperforms target decoding, standard SD, and state-of-the-art SD variants, achieving up to 1.9× speedup over SD with an average 1.4× gain in high-throughput settings.

9 Limitations

Public Model Coverage. Our evaluation is conducted using publicly available models, which may introduce model-specific biases. However, within this scope, we perform extensive evaluation: we report goodput improvements across 14 draft–companion–target model pairs from the HuggingFace model hub and also analyze speculation uncertainty reduction across more than 90 model pairs.

Limited Variance Analysis. We do not repeat the full generation workload multiple times per evaluation setting to report variance or confidence intervals. Each evaluation run processes multiple requests over approximately 10,000 decoding iterations, and we report the resulting average goodput. While we do not measure variance across repeated runs, the reported results are aggregated over a large number of decoding steps within each run.

References

Sudhanshu Agrawal, Wonseok Jeon, and Mingu Lee. 2024. Adaedl: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. *arXiv preprint arXiv:2410.18351*.

AMD. 2024. AMD-Llama-135m: A 135M Parameter Language Model. https://huggingface.co/amd/AMD-Llama-135m.

anon8231489123. 2023. Sharegpt vicuna unfiltered dataset. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Clément L Canonne. 2022. A short note on an inequality between kl and tv. *arXiv preprint arXiv:2202.07198*.

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

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

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. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

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. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. In *ACL (1)*, pages 12622–12642.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Red Hat. 2025. Speculators: A unified library for speculative decoding algorithms in llm serving. https://github.com/vllm-project/speculators.

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

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

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

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

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. 2024. Dynamic speculation lookahead accelerates speculative decoding of large language models. *arXiv preprint arXiv:2405.04304*.

Meta. 2024a. Layerskip codellama 34b. https://huggingface.co/facebook/layerskip-codellama-34B.

Meta. 2024b. Layerskip llama 2 70b. https://huggingface.co/facebook/layerskip-llama2-70B.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, and 1 others. 2024. 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.

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

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

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

Mohamed Rashad. 2023. Chatgpt prompts dataset. https://huggingface.co/datasets/MohamedRashad/ChatGPT-prompts.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

safeailab. 2025. Eagle official implementation. https://github.com/SafeAILab/EAGLE.

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

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Unsloth. 2024. Qwen2.5-0.5b. https://huggingface.co/unsloth/Qwen2.5-0.5B.

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

Zhaoxuan Wu, Zijian Zhou, Arun Verma, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. 2025. Tetris: Optimal draft token selection for batch speculative decoding. *arXiv preprint arXiv:2502.15197*.

Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, and Lei Zou. 2025. Dyspec: Faster speculative decoding with dynamic token tree structure. *World Wide Web*, 28(3):36.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

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

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

Ziyin Zhang, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Rui Wang, and Zhaopeng Tu. 2025b. Draft model knows when to stop: Self-verification speculative decoding for long-form generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16696–16708.

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

A Evaluation Settings

A.1 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.

Section	Dataset	GPU	Target Model	Companion Model	Draft Model
Sec. 7.2	ChatGPT ShareGPT	A100 ×4	Qwen/ Qwen2.5-72B-Instruct	Qwen/ Qwen2.5-1.5B-Instruct	Qwen/ Qwen2.5-0.5B-Instruct
	HumanEvalPack MBPP	A100 ×2	meta-llama/ CodeLlama-34b-hf	TinyLlama/ TinyLlama_v1.1_math_code	amd/ AMD-Llama-135m-code
	HumanEvalPack GSM8K	A100 ×2	meta-llama/ Llama-2-13b-hf	JackFram/ llama-160m	JackFram/ llama-68m
Sec. 7.3 Sec. 7.7	Spec-Bench	A100 ×4	meta-llama/ Llama-3.3-70B-Instruct	meta-llama/ Llama-3.2-3B-Instruct	meta-llama/ Llama-3.2-1B-Instruct
				meta-llama/ Llama-3.2-1B-Instruct	yuhuili/ EAGLE3-LLaMA3.3-Instruct-70B
		A100 ×2	Qwen/ Qwen3-32B	Qwen/ Qwen3-1.7B	Qwen/ Qwen3-0.6B
				Qwen/ Qwen3-0.6B	RedHatAI/ Qwen3-32B-speculator.eagle3
Sec. 7.7	ShareGPT	A40 ×4	facebook layerskip-llama2-70B	facebook layerskip-llama2-70B(5)‡	facebook layerskip-llama2-70B(20)‡
	Humaneval	A40 ×2	facebook layerskip-codellama-34B	facebook layerskip-codellama-34B(4)‡	facebook layerskip-codellama-34B(4)‡
Sec. 7.7	ShareGPT	A100 ×2	Qwen/ Qwen2.5-32B-Instruct	Qwen/ Qwen2.5-3B-Instruct	Qwen/ Qwen2.5-0.5B-Instruct
				Qwen/ Qwen2.5-1.5B-Instruct	Qwen/ Qwen2.5-0.5B-Instruct
					unsloth/ Qwen2.5-0.5B-Instruct
				Qwen/ Qwen2.5-0.5B-Instruct	Qwen/ Qwen2.5-1.5B-Instruct
					unsloth/ Qwen2.5-0.5B-Instruct

‡ Layerskip with (N) layers

Table 3: Model Pair and Hardware Settings used in Evaluations.

Our experimental design encompasses three dimensions of variation per family: base models (no fine-tuning), instruction-tuned models, and task-tuned models.

Models used in overall evaluations

- **Large-sized models:** We employed the Qwen2.5-Instruct family (Yang et al., 2024) 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 (Roziere et al., 2023) as target model for mid-scale experiments. and TinyLlama_v1.1_math_code (1.2B) (Zhang et al., 2024) as the *companion*, and AMD-Llama-135M (AMD, 2024) as the *draft*.
- **Small-sized models:** To cover smaller models, we included the Llama2-14B variant (Touvron et al., 2023) as the *target* model, paired with a JackFram_llama-160m (Miao et al., 2024) as the *companion*, and JackFram_llama-68m (Miao et al., 2024) as the *draft*, which was specifically trained for speculation tasks with a reduced parameter size.

Models used in SD variants evaluations

- **Large-sized models:** We adopted Llama-3.3-70B-Instruct (Grattafiori et al., 2024) as the *target* model, paired with Qwen3-1.7B (Yang et al., 2025) as the *companion* model, and Qwen3-0.6B (Yang et al., 2025) as the *draft* model.
- **Mid-sized models:** We used Qwen3-32B (Yang et al., 2025) as the *target* model, together with Llama3.2-3B-Instruct (Grattafiori et al., 2024) as the *companion* model, and Llama3.2-1B-Instruct (Grattafiori et al., 2024) as the *draft* model.
- **LayerSkip:** 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 (Meta, 2024b,a).

- **Eagle-3:** We evaluated EAGLE-3 (Li et al., 2025) by pairing each backbone (verifier) with its corresponding Eagle-3 speculator checkpoint, following the draft model checkpoint recommended in the official implementation (safeailab, 2025; Hat, 2025). When adopting SV on EAGLE-3 (Li et al., 2025), we used smaller companion models for speculation: Llama-3.2-1B-Instruct for the large-sized setting and Qwen3-0.6B for the mid-sized setting.

Models used in other evaluations and analysis For additional analytical experiments (Section 7.5, 7.6, and 7.7), 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 fine-tuned version of Qwen2.5-Instruct models by unsloth (Unsloth, 2024).

Sampling Parameters For all models, we adhered to the sampling hyperparameters recommended in their respective HuggingFace repositories or official GitHub documentation. Specific values are shown in Table 4. While many prior SD works report greedy decoding evaluations (or both greedy and sampling), we exclude greedy decoding because it often produces lower-quality outputs (e.g., repetitive tokens) and is less representative of typical LLM serving settings.

Model series	Top-K	Top-P	Temp.	Repetition Penelty
Qwen2.5 based	20	0.8	0.7	1.05
Qwen3 based	20	0.6	0.95	—
Llama-2 based	—	0.9	0.6	—
Llama-3 based	—	0.9	0.6	—

Table 4: Sampling hyperparameters for each model series.

A.2 Dataset and Pre-Processing

We evaluated our approach using seven distinct task categories: mathematical reasoning, code generation, multi-turn conversation, translation, summarization, QA, and RAGFor all experiments, we maintained consistency by using identical randomly sampled subsets across different evaluation scenarios.

Dataset	Task
GSM8K	Mathematical Reasoning
HumanEvalPack	Code generation
MBPP	Code generation
ShareGPT	Multi-turn conversation
ChatGPT	Multi-turn conversation
Spec-Bench	Translation, Summarization, QA, Mathematical Reasoning, RAG, Multi-turn Conversation

Table 5: Datasets and corresponding tasks used in our evaluation.

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 (anon8231489123, 2023), a collection of human-assistant conversations extracted from various online sources, and ChatGPT Dataset (Rashad, 2023), consisting of diverse dialogue prompts and responses.

Our code generation evaluation encompassed six programming languages using the HumanEvalPack (Muennighoff et al., 2023) 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 (Austin et al., 2021), which consists of approximately 1,000 crowd-sourced Python programming problems.

To assess mathematical reasoning capabilities, we employed the GSM8K (Cobbe et al., 2021) dataset, which contains grade school math word problems that require multi-step reasoning to solve.

For Spec-Bench evaluation, we performed a task-wise split to support both probability profile construction and evaluation. Specifically, for each task category in Spec-Bench, we randomly sampled 10% of the queries (8 queries per task) to construct the probability profile, and used the remaining 90% for evaluation (72 queries per task). The same ran-

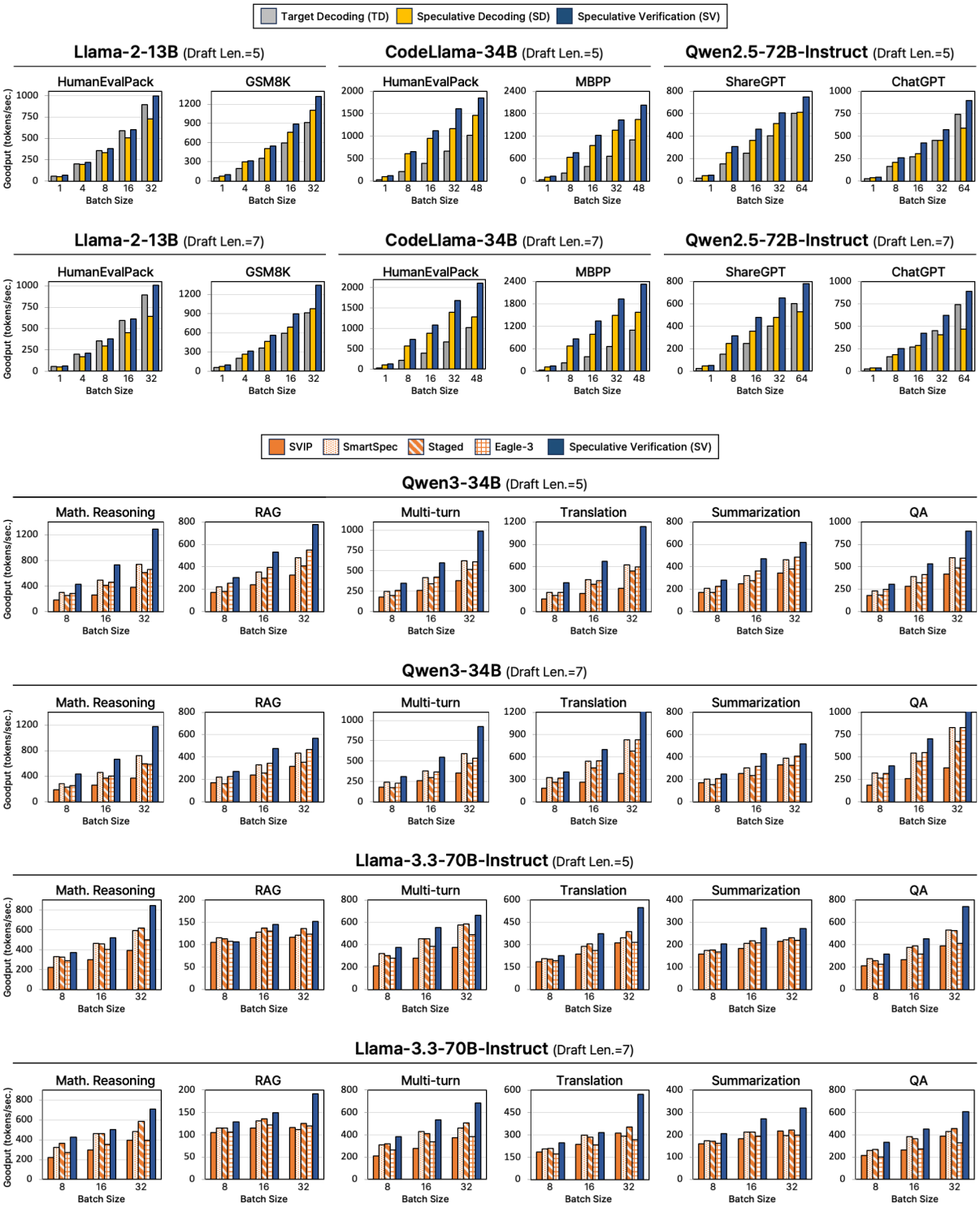

Figure 8: Performance Evaluation Across Model Pairs, Datasets, and Baselines.

dom partition was reused across all experimental settings to ensure consistency and fair comparison.

A.3 Hardware Settings

All experiments were conducted in multiple computing environments to meet the heterogeneous compute and memory demands of the evaluated models.

Azure cloud instances. For large-sized models, we used an Azure VM of type Standard NC96ads A100 v4. For mid- and small-sized models, we used Standard NC48ads A100 v4 instances. Both VMs are equipped with an AMD EPYC 7V13 64-core CPU and 2TB of RAM.

Target Model	Symbol	Companion Model *(Training Entity)*	Information Gain (Draft Model: Gain)
meta-llama/ Llama-2-13b-chat-hf	**A**	Llama-2-7b-hf (*Meta*)	**B**: 0.47, **C**: 0.52, **D**: 0.39, **E**: 0.57, **F**: 0.56
	B	TinyLlama_v1.1 (*TinyLlama*)	**A**: 0.07, **C**: 0.39, **D**: 0.40, **E**: 0.33, **F**: 0.29
	C	Llama-160M-Chat-v1 (*Felladrin*)	**A**: 0.10, **B**: 0.13, **D**: 0.44, **E**: 0.40, **F**: 0.40
	D	llama-160m (*JackFram*)	**A**: 0.15, **B**: 0.05, **C**: 0.28, **E**: 0.59, **F**: 0.49
	E	llama-135m (*AMD*)	**A**: 0.11, **B**: 0.11, **C**: 0.32, **D**: 0.45, **F**: 0.42
	F	llama-68m (*JackFram*)	**A**: 0.11, **B**: 0.31, **C**: 0.40, **D**: 0.46, **E**: 0.37
Qwen/ Qwen2.5-14B-Instruct	**A**	Qwen2.5-1.5B-Instruct (*Qwen*)	**B**: 0.36, **C**: 0.22, **D**: 0.18, **E**: 0.18, **F**: 0.22
	B	Qwen2.5-1.5B-Instruct (*Unsloth*)	**A**: 0.36, **C**: 0.22, **D**: 0.18, **E**: 0.18, **F**: 0.22
	C	TinySwallow-1.5B-Instruct (*SakanaAI*)	**A**: 0.12, **B**: 0.12, **D**: 0.12, **E**: 0.12, **F**: 0.21
	D	Qwen2.5-0.5B-Instruct (*Qwen*)	**A**: 0.03, **B**: 0.60, **C**: 0.06, **E**: 0.60, **F**: 0.54
	E	Qwen2.5-0.5B-Instruct (*Unsloth*)	**A**: 0.60, **B**: 0.03, **C**: 0.06, **D**: 0.03, **F**: 0.54
	F	Qwen2.5-0.5B-Instruct-ITA (*ReDiX*)	**A**: 0.33, **B**: 0.33, **C**: 0.11, **D**: 0.33, **E**: 0.33
meta-llama/ Llama-3.3-70B-Instruct	**A**	Llama-3.2-3B-Instruct-pythonic (*Baseten*)	**B**: 0.23, **C**: 0.31, **D**: 0.09, **E**: 0.22, **F**: 0.25
	B	Llama-SmolTalk-3.2-1B-Instruct (*prithivMLmods*)	**A**: 0.04, **C**: 0.05, **D**: 0.04, **E**: 0.33, **F**: 0.39
	C	DeepSeek-R1-Distill-Llama-3B (*Suayptalha*)	**A**: 0.13, **B**: 0.17, **D**: 0.18, **E**: 0.13, **F**: 0.24
	D	Llama-3.2-1B-Instruct (*Unsloth*)	**A**: 0.05, **B**: 0.40, **C**: 0.04, **E**: 0.05, **F**: 0.35
	E	Llama-3.2-3B-Instruct (*Unsloth*)	**A**: 0.09, **B**: 0.23, **C**: 0.31, **D**: 0.22, **E**: 0.25
	F	Vikhr-Llama-3.2-1B-Instruct (*Vikhrmodels*)	**A**: 0.03, **B**: 0.30, **C**: 0.06, **D**: 0.03, **E**: 0.26

Table 6: Uncertainty in token acceptance probability and information gain from observing S and A. Symbols **A–F** denote the corresponding draft (or companion) models.

Private GPU server (LayerSkip experiments). LayerSkip experiments were performed on a private GPU server equipped with an AMD EPYC 7313 16-core CPU, four NVIDIA A40 GPUs (48 GB VRAM each), and 500GB of RAM.

B Full Evaluation Results

B.1 Additional Evaluation Results for Section 4.2 (Correlation of S)

Dataset	Models (D/C/T)	Corr.
sharegpt	AMD-Llama-135m TinyLlama_v1.1 Llama-2-13b-hf	0.8706
humanevalpack	AMD-Llama-135m-code TinyLlama_v1.1_math_code CodeLlama-34b-hf	0.8215
chatgpt	Qwen2.5-0.5B-Instruct Qwen2.5-1.5B-Instruct Qwen2.5-72B-Instruct	0.7249

Table 7: Correlation results by dataset and model pairs (Target–Draft–Companion).

B.2 Additional Evaluation Results for Section 7.2 (Overall Performance) and 7.3 (Comparison to SD Variants)

Across both evaluations, SV consistently improves goodput over TD and standard SD. The gains typically grow with batch size, indicating that SV better amortizes verification overhead. Notably, as draft length increases, SD often incurs higher verification cost (more drafted tokens to validate per step), which can reduce goodput; in contrast, SV more effectively controls this overhead by reducing uncertainty in token acceptance and thereby preserving its advantage across draft length 5, 7.

Cases	Request ID	Avg. Veri. Length	γ 1	2	3	4	5
Bottom 5 reqs (lowest avg. veri. length)	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
Top 5 reqs (highest avg. veri. length)	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

Table 8: Top/bottom 5 requests with highest/lowest average values of verification length γ and its selected counts while scheduling

B.3 Additional Evaluation Results for Section 7.4 (Information Gain)

To examine whether SV consistently provides positive information gain across diverse draft–companion–target (D–C–T) configurations, we evaluated 90 distinct model triplets. Specifically, we selected three target models and, for each target, paired it with six smaller models that can serve as either the draft or the companion, resulting in $3 \times (6 \times 5) = 90$ ordered (D, C, T) combinations. The selected triplets cover both regimes where the parameter gap between the draft and companion is

small and where it is large.

For each triplet, we measured the target model's acceptance probabilities with and without SV, and quantified how our two variables (S and A) reduce uncertainty in acceptance. We compute the uncertainty reduction as the information gain in predicting acceptance when conditioning on (S, A), compared to the baseline without SV. Across all configurations, we consistently observed positive information gain, indicating that SV reliably decreases uncertainty about acceptance over a wide range of model combinations.

B.4 Additional Evaluation Results for Section 7.6 (Scheduling Fairness)

The detailed experimental results for Section 7.6 are shown in Table 8.

C Optimality of SV's Verification Length Selection

Definition C.1. *Let S be a random variable representing natural divergence of P_d and P_c:*

$$S = \sum_{j \in vocab} \min(P_d(t_j), P_c(t_j)) \tag{1}$$

where P_d and P_c are token probability distributions of the draft and companion model, respectively.

Definition C.2. *Let A be a random variable for t_d's acceptance probability in the companion model:*

$$A = \min\left(1, \frac{P_c(t_d)}{P_d(t_d)}\right) \tag{2}$$

where t_d is the token generated by the draft model, and P_d and P_c are defined as in A.1.

Definition C.3. *Let T_i be a random variable following the probability distribution of i'th draft token and $P(T_i)$ is its acceptance probability in the target model:*

- *$P(T_i = t)$ is the probability that a generated token t is accepted by the target model and*
- *$P(T_i = t|S, A)$ is its conditional probability when S and A are given.*

Definition C.4. *Let N be a random variable for the number of tokens accepted by the target model.*

The probability for N given γ is calculated as:

$$P_\gamma(N) = \begin{cases} P(T_{N+1} \neq t_{N+1}) \prod_{i=1}^{N} P(T_i = t_i) & \text{if } N < \gamma, \\ \prod_{i=1}^{\gamma} P(T_i = t_i) & \text{if } N = \gamma \end{cases} \tag{3}$$

The expected number of accepted tokens when verifying γ tokens is calculated as:

$$E(N|\gamma) = \sum_{i=1}^{\gamma} i \cdot P_\gamma(N = i) \tag{4}$$

where $P_\gamma(N = i)$ is the probability of accepting exactly i tokens when verifying γ tokens.

Definition C.5. *Let goodput(γ) be the goodput (i.e., the number of accepted tokens per unit time) when verifying γ number of tokens, which is calculated as:*

$$Goodput(\gamma) = \frac{N}{Latency(\gamma)} \tag{5}$$

where N is the number of tokens accepted when verifying γ tokens.

To estimate goodput, we use the profiled latency and expected number of accepted tokens as following:

$$\hat{Goodput}(\gamma) = \frac{E(N|\gamma)}{Latency_{profiled}(\gamma)} \tag{6}$$

Assumption C.6. *The draft and companion models are reasonably aligned with the target model.*

Definition C.7. *We define $\hat{P}(T_i)$, i.e., an estimator of $P(T_i)$, to be $P(T_i|S, A)$.*

Definition C.8. *We obtain $P(T_i|S, A)$ by observing sample data and grouping (i.e., binning) acceptance probability according to the values of S and A as following:*

$$P(T_i|S, A) = \begin{cases} E(P(T_i|s_0 \leq S < s_1,\, a_0 \leq A < a_1)) \\ E(P(T_i|s_1 \leq S < s_2,\, a_1 \leq A < a_2)) \\ \dots \end{cases} \tag{7}$$

We can reduce the variance of the observed acceptance probabilities by controlling the bin sizes of S and A. Reducing this variance subsequently decreases the uncertainty associated with the acceptance probabilities.

Assumption C.9. *The observed probability in Definition C.8 constitutes an unbiased estimator of $P(T_i)$, as long as the observed distributions of S, A, and P coincide with those encountered at inference time.*

Assumption C.10. *Let $Latency(\gamma)$ be the end-to-end wall-clock latency required to process γ tokens. and its finite difference is*

$$\Delta Latency(\gamma) = Latency(\gamma{+}1) - Latency(\gamma), \quad \gamma \geq 1. \tag{8}$$

We assume the following properties.

1. *There exists a threshold γ_0 such that $\Delta Latency(\gamma)$ is a constant value for all $\gamma \geq \gamma_0$:*

$$\Delta Latency(\gamma) = \Delta Latency(\gamma_0) \text{ for } \gamma \geq \gamma_0$$

2. *$\Delta Latency(\gamma)$ is an increasing function for γ smaller than the threshold γ_0:*

$$\Delta Latency(\gamma{+}1) \;>\; \Delta Latency(\gamma) \;\; \textit{for}\; 1 \leq \gamma < \gamma_0$$

From the above two properties, $\Delta Latency(\gamma)$ is a non-decreasing function for all $\gamma > 0$:

$$\Delta Latency(\gamma+1) \;\geq\; \Delta Latency(\gamma), \;\; \textit{for}\; \gamma > 0 \tag{9}$$

Lemma C.11. *The expected number of accepted tokens $E(N|\gamma)$ is increasing and concave with respect to γ:*

$$\frac{d}{d\gamma}E(N|\gamma) \geq 0 \tag{10}$$

$$\frac{d^2}{d\gamma^2}E(N|\gamma) \leq 0 \tag{11}$$

Proof. The first and second derivatives are computed as discrete differences, represented as Δ:

$$\frac{d}{d\gamma}E(N|\gamma) = \Delta^1 E(N|\gamma) = E(N|\gamma+1) - E(N|\gamma) \tag{12}$$

$$\begin{aligned}\frac{d^2}{d\gamma^2}E(N|\gamma) &= \Delta^2 E(N|\gamma) \\ &= \Delta^1 E(N|\gamma+1) - \Delta^1 E(N|\gamma) \\ &= E(N|\gamma+2) - 2E(N|\gamma+1) + E(N|\gamma).\end{aligned} \tag{13}$$

Part 1: Proof of $\Delta^1 E(N|\gamma) \geq 0$. Increasing γ can only increase the number of accepted tokens, thus $\Delta^1 E(N|\gamma) \geq 0$.

Part 2: Proof of $\Delta^2 E(N|\gamma) \leq 0$.

$$\Delta^1 E(N|\gamma) = E(N|\gamma+1) - E(N|\gamma) \tag{14}$$

$$= \sum_{k=1}^{\gamma+1} kP_{\gamma+1}(k) - \sum_{k=1}^{\gamma} kP_{\gamma}(k) \tag{15}$$

$$= \underbrace{\sum_{k=1}^{\gamma-1} k(P_{\gamma+1}(k) - P_{\gamma}(k))}_{\alpha} \tag{16}$$

$$+ \underbrace{\gamma P_{\gamma+1}(\gamma) + (\gamma+1)P_{\gamma+1}(\gamma+1) - \gamma P_{\gamma}(\gamma)}_{\beta} \tag{17}$$

$$\Delta^2 E(N|\gamma) = \Delta^1 E(\gamma+1) - \Delta^1 E(N|\gamma) \tag{18}$$

$$= \sum_{k=1}^{\gamma-1} k\,\underbrace{[P_{\gamma+2}(k) - 2P_{\gamma+1}(k) + P_{\gamma}(k)]}_{(a)} \tag{19}$$

$$+ \gamma\,\underbrace{[P_{\gamma+2}(\gamma) - 2P_{\gamma+1}(\gamma) + P_{\gamma}(\gamma)]}_{(b_1)} \tag{20}$$

$$+ (\gamma+1)P_{\gamma+2}(\gamma+1) - 2(\gamma+1)P_{\gamma+1}(\gamma+1) \tag{21}$$

$$+ (\gamma+2)P_{\gamma+2}(\gamma+2) \quad : \text{Term } (b_2) \tag{22}$$

$\Delta^2 E(N|\gamma)$ is ≤ 0 because $(a) = 0$ and $(b_1) + (b_2) \leq 0$.

- **Term $(a) = 0$:** The probability $P(N = k)$ for $k < \gamma$ is independent of the total verification length if it's already $> k$.
- **Terms $(b_1) + (b_2) \leq 0$:** These represent the diminishing returns in expected tokens as γ increases.

Proof of terms (b) + (c) $\leq$ 0: Applying Def. C.4, we express probabilities in terms of $P_\gamma(\gamma)$:

$$\begin{aligned}P_{\gamma+1}(\gamma) &= P_\gamma(\gamma)P(T_{\gamma+1} \neq t_{\gamma+1}) \\ P_{\gamma+1}(\gamma+1) &= P_\gamma(\gamma)P(T_{\gamma+1} = t_{\gamma+1}) \\ P_{\gamma+2}(\gamma) &= P_\gamma(\gamma)P(T_{\gamma+1} \neq t_{\gamma+1}) \\ P_{\gamma+2}(\gamma+1) &= P_\gamma(\gamma)P(T_{\gamma+1} = t_{\gamma+1})P(T_{\gamma+2} \neq t_{\gamma+2}) \\ P_{\gamma+2}(\gamma+2) &= P_\gamma(\gamma)P(T_{\gamma+1} = t_{\gamma+1})P(T_{\gamma+2} = t_{\gamma+2}).\end{aligned} \tag{23}$$

Then, substituting these:

$$\begin{aligned}(b_0) + (b_1) = &- P_\gamma(\gamma)[\gamma P(T_{\gamma+1} \neq t_{\gamma+1}) \\ &+ P(T_{\gamma+1} = t_{\gamma+1})P(T_{\gamma+2} \neq t_{\gamma+2})] \leq 0\end{aligned} \tag{24}$$

Thus, $\Delta^2 E(N|\gamma) \leq 0$. □

Lemma C.12. *The Goodput function is concave with respect to γ:*

$$\Delta^2 Goodput(\gamma) \leq 0 \textit{ for all } \gamma \tag{25}$$

Proof.

$$\begin{aligned}\Delta\text{goodput}(\gamma) &= \frac{E(N|\gamma+1)}{L(\gamma+1)} - \frac{E(N|\gamma)}{L(\gamma)} \\ &= \frac{L(\gamma)E(N|\gamma+1) - E(N|\gamma)L(\gamma+1)}{L(\gamma+1)L(\gamma)} \\ &= \frac{L(\gamma)\Delta E(N|\gamma) - E(N|\gamma)\Delta L(\gamma)}{L(\gamma+1)L(\gamma)}\end{aligned} \tag{26}$$

Let $\Psi(\gamma) = L(\gamma)\Delta E(N|\gamma) - E(N|\gamma)\Delta L(\gamma)$, then:

$$\begin{aligned}\Delta\Psi(\gamma) &= \Psi(\gamma+1) - \Psi(\gamma) \\ &= \underbrace{L(\gamma)\Delta^2 E(N|\gamma)}_{(a)\leq 0} + \underbrace{(-E(N|\gamma)\Delta^2 L(\gamma))}_{(b)\leq 0} \\ &\quad + \underbrace{\Delta L(\gamma)\Delta E(N|\gamma+1) - \Delta L(\gamma+1)\Delta E(N|\gamma)}_{(c)\leq 0}\end{aligned} \tag{27}$$

All terms $(a), (b), (c)$ are ≤ 0 based on Lemma C.11 and Assumption C.10. Thus $\Delta\Psi(\gamma) < 0$, meaning $Goodput(\gamma)$ is concave.

Optionally, with conditions:

$$\Delta Goodput(\gamma_0) > 0, \quad \Delta Goodput(\gamma_1) < 0 \tag{28}$$

we can find an optimal γ^* that maximizes Goodput. Our evaluations indicate these conditions generally hold true. □