TriSpec: Ternary Speculative Decoding via Lightweight Proxy Verification

Haoyun Jiang[1,2] **Junqi He**[1] **Feng Hong**[1] **Xinlong Yang**[2] **Jianwei Zhang**[2]
Zhengyang Zhuge[2] **Zheng Li**[2] **Xiaofeng Cao**[3] **Zhiyong Chen**[1] **Bo Han**[4]
Junyang Lin[2] **Jiangchao Yao**[1*]
[1]CMIC, Shanghai Jiao Tong University [2]Qwen Team, Alibaba Group [3]Tongji University
[4]Hong Kong Baptist University

Abstract

Inference efficiency in Large Language Models (LLMs) is fundamentally limited by their serial, autoregressive generation, especially as reasoning becomes a key capability and response sequences grow longer. Speculative decoding (SD) offers a powerful solution, providing significant speed-ups through its lightweight drafting and parallel verification mechanism. While existing work has nearly saturated improvements in draft effectiveness and efficiency, this paper advances SD from a new yet critical perspective: the verification cost. We propose TriSpec, a novel ternary SD framework that, at its core, introduces a lightweight proxy to significantly reduce computational cost by approving easily verifiable draft sequences and engaging the full target model only when encountering uncertain tokens. TriSpec can be integrated with state-of-the-art SD methods like EAGLE-3 to further reduce verification costs, achieving greater acceleration. Extensive experiments on the Qwen3 and DeepSeek-R1-Distill-Qwen/LLaMA families show that TriSpec achieves up to 35% speedup over standard SD, with up to 50% fewer target model invocations while maintaining comparable accuracy. Code is available at https://github.com/haoyun-jiang/TriSpec.

1 Introduction

Large language models (LLMs) [33, 23, 14] have made remarkable success on complex tasks, with recent advances in reasoning leading to longer and more sophisticated responses. This progress, however, is often hampered by the inherent latency of their serial, autoregressive generation process. Speculative decoding (SD) [18] addresses this efficiency bottleneck with a draft-then-verify paradigm. A small, fast drafter proposes candidate tokens, which the large target model then verifies in a single parallel pass, significantly accelerating inference without compromising output quality.

Generally, the overall speedup in speculative decoding is governed by three key factors: the speed of draft token generation, the acceptance rate of draft tokens, and the speed of verification. Existing work has made substantial progress on the first two. Medusa [4], EAGLE [19] and DFlash [6] propose lightweight drafter architectures, shortening the compute path and making the drafting time almost negligible. Building on this design, subsequent methods further improve acceptance rates by optimizing candidate paths [20, 38], refining training paradigms [37, 21, 23], or relaxing verification constraints [3, 24], pushing this factor close to saturation. In contrast, the *verification speed* remains an equally crucial yet largely overlooked factor. This gap motivates us to explore *whether the verification role of the large target model could be handled by a more efficient, lightweight verifier*.

We answer this question affirmatively. We find that smaller models from the same family as the target model (*e.g.*, Qwen3-1.7B and Qwen3-32B) are promising candidates for lightweight verification, as their token-level predictions are highly consistent with those of the target model (§3.1).

*Correspondence to: Jiangchao Yao (sunarker@sjtu.edu.cn).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

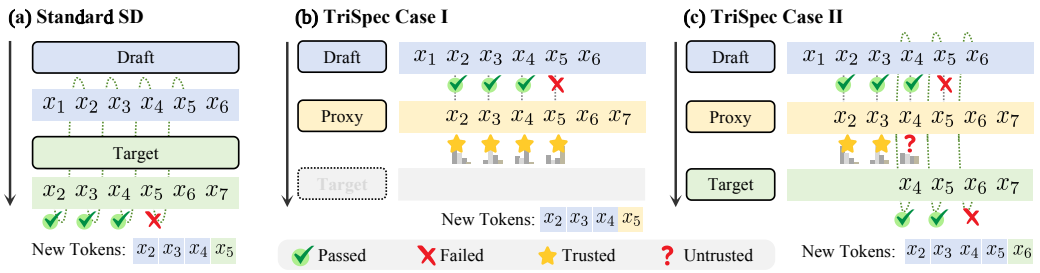


Figure 1: Comparison of (a) standard SD and (b-c) TriSpec. In (b), the proxy remains trusted at the first rejection, enabling local correction without invoking the target. In (c), the proxy becomes untrusted before the first rejection, so the remaining draft tokens are escalated to the target for authoritative verification. A margin-based criterion determines whether proxy verification is trusted.

Consequently, a small model can reliably replace the target model for verifying the majority of tokens, and the challenge lies in a small subset of tokens where their capabilities diverge. Fortunately, we find that a simple margin-based metric can sufficiently characterize this capability boundary, and route the verification task between a large target model and its smaller surrogate (Figure 2(b)).

Based on these insights, we propose TriSpec, a ternary speculative decoding paradigm that coordinates three models with complementary roles. A single-layer drafter provides high-throughput drafting; a lightweight proxy verifier quickly pre-verifies the draft and locally corrects a subset of tokens; and a large verifier supplies authoritative verification when needed (Figure 1). By delegating a substantial portion of verification to the cheaper proxy, TriSpec effectively reduces per-round verification time and thus the overall decoding latency. Extensive experiments on Qwen3 and DeepSeek-R1-Distill-Qwen/LLaMA families demonstrate that TriSpec delivers significant speedups over SOTA speculative decoding methods. In summary, our contributions are as follows:

- We identify the verification cost as a new and critical perspective for accelerating speculative decoding, and show that smaller models within the same family exhibit strong alignment with their larger counterparts, making them effective and lightweight proxy verifiers.
- We introduce TriSpec, a ternary speculative decoding paradigm that jointly orchestrates a drafter, a proxy verifier, and the target model. A margin-based routing criterion adaptively allocates verification to the proxy for confident cases and invokes the expensive target only when necessary, preserving accuracy while substantially reducing verification overhead.
- Extensive experiments on popular models and benchmarks show that TriSpec achieves up to a 35% speedup over standard speculative decoding while reducing target-model invocations by more than half, with an average accuracy degradation of less than 1%.

2 Rethinking Speculative Decoding Speedup

2.1 Speculative Decoding

Standard speculative decoding follows a draft-then-verify paradigm with two models: a draft model $\mathcal{M}_d$ and a target model $\mathcal{M}_t$. Let $\mathcal{V}$ denote a finite vocabulary with size $V := |\mathcal{V}|$, and let the current decoding prefix be $X \in \mathcal{V}^l$, including both the prompt and previously generated tokens.

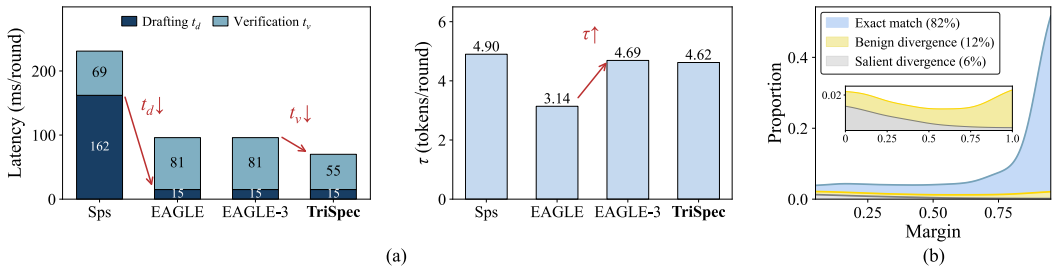
Drafting phase. Conditioned on the current prefix X, the draft model $\mathcal{M}_d$ runs autoregressively for k steps. At each step, it predicts a next-token distribution and samples one draft token:

$$\left(x_1^{(d)}, \mathbf{p}_1^{(d)}\right), \ldots, \left(x_k^{(d)}, \mathbf{p}_k^{(d)}\right) \leftarrow \mathcal{M}_d(X), \tag{1}$$

where $\mathbf{p}_i^{(d)} \in \mathbb{R}^V$ is the draft distribution at position i, and $x_i^{(d)} \in \mathcal{V}$ is the sampled draft token. For any distribution $\mathbf{p}$ over $\mathcal{V}$ and token $x \in \mathcal{V}$, $\mathbf{p}(x)$ denotes the probability assigned to x.

Verification phase. The target model $\mathcal{M}_t$ verifies the k drafted tokens with one parallel forward pass, producing target-side predictions for all draft positions and one bonus position:

$$(x_1^{(t)}, \mathbf{p}_1^{(t)}), (x_2^{(t)}, \mathbf{p}_2^{(t)}), \ldots, (x_{k+1}^{(t)}, \mathbf{p}_{k+1}^{(t)}) \leftarrow \mathcal{M}_t(x_1^{(d)}, x_2^{(d)}, \ldots, x_k^{(d)} \mid X) \tag{2}$$



(a) (b)

Figure 2: (a) Latency decomposition of representative speculative decoding methods, with average per-round times shown on the left and acceptance lengths on the right. EAGLE reduces the drafting cost t_d, EAGLE-3 improves the acceptance length τ, and TriSpec reduces the average verification cost t_v. Measurements use Qwen3-32B on GSM8K with batch size 1, a single A100 GPU, and draft length 6. (b) Token outcomes versus the proxy's top-1–top-2 margin on ShareGPT. Exact matches agree with the target; mismatches passing the target-or-judge criterion (Appendix C.2) are benign; others are salient. Salient divergences cluster at low margins. Stacked areas are globally normalized.

Here, $\mathbf{p}_i^{(t)} \in \mathbb{R}^V$ is the target distribution at position i, and $x_i^{(t)} \in \mathcal{V}$ is the corresponding target token. Each drafted token $x_i^{(d)}$ is accepted with probability $\min(1, \mathbf{p}_i^{(t)}(x_i^{(d)})/\mathbf{p}_i^{(d)}(x_i^{(d)}))$. If any rejection occurs, all drafted tokens to its right are discarded, and a replacement token is sampled from the normalized positive residual $\mathrm{norm}(\max(0, \mathbf{p}_i^{(t)} - \mathbf{p}_i^{(d)}))$ at the first rejection position. Under this acceptance rule, the resulting samples follow exactly the same distribution as naive target-model decoding, which makes standard SD a provably lossless acceleration method [5].

2.2 Latency Analysis of Speculative Decoding

Latency decomposition for SD. Consider a speculative decoding process that generates N tokens. Let τ denote the average acceptance length, and let t_d, t_v, and t_o denote the average per-round drafting time, verification time, and other overhead, respectively, computed over all rounds required to generate these N tokens. The end-to-end latency can be decomposed as

$$L = \frac{N}{\tau}(t_d + t_v + t_o) \implies L \propto \frac{t_d + t_v}{\tau} \text{ at } t_o \to 0. \tag{3}$$

The proportional form assumes a fixed output length N and omits the minor, stable overhead t_o, which is well motivated in practice under the same evaluation setting.

New perspective. As modern SD methods substantially reduce t_d and increase τ, reducing verification time t_v becomes an important and promising direction for further acceleration.

Drafting has been largely optimized. Recent latency reductions in SD have mainly come from decreasing t_d and increasing τ. Figure 2(a) illustrates how drafting time, verification time, and acceptance length evolve across representative SD methods. Early SD methods [18] often use a separate small model as the drafter, but its multi-layer architecture makes drafting latency-dominant. Single-layer drafter designs, such as Medusa [4] and EAGLE [21], substantially reduce t_d while maintaining a high τ, making them the prevailing choice. Subsequent methods further increase τ through tree-structured candidate paths and enhanced training schemes, as exemplified by EAGLE-3 [21]. With Multi-Token Prediction (MTP) [23] integrating draft-model training into pre-training, the acceptance rate of draft tokens can even reach up to 90%.

Verification remains the dominant bottleneck. In contrast, relatively little progress has been made in directly reducing the per-round verification time t_v. As shown in the left panel of Figure 2(a), verification has become the dominant component of per-round latency in state-of-the-art pipelines such as EAGLE-3, since each target-model invocation must traverse a deep computation path with a large parameter budget. This motivates a lightweight verifier that can offload part of the verification workload from the target, thereby reducing the average verification time across rounds.

3 TriSpec

In this section, we propose TriSpec, a tightly-coupled ternary speculative decoding framework guided by a simple principle: *use the cheapest component to fulfill each role*. Specifically, TriSpec integrates a *single-layer drafter* for low-cost token generation, a *lightweight proxy verifier* to rapidly validate easy tokens and flag uncertain ones, and the *target model* for exact verification of flagged cases to ensure correctness. By properly offloading most verification from the target to the proxy, TriSpec reduces the effective per-round verification time t_v while maintaining accuracy.

3.1 Proxy choice: same-family smaller models

Same-family smaller models are well suited as proxy verifiers due to their lower inference cost and inherent target alignment, as further evidenced by the following three properties.

Strong Alignment. Since speculative decoding verifies draft tokens position-wise, an effective proxy should closely align with the target's token-level decisions. We evaluate this alignment on ShareGPT using Qwen3-family models by categorizing each next-token prediction as an exact match, benign divergence, or salient divergence. Benign divergences are alternative tokens deemed acceptable by the target or a stronger LLM, while the rest are treated as salient, indicating potential semantic deviation. Under this protocol, the proxy achieves an 82% exact-match rate with the target, substantially higher than the EAGLE-3 drafter's 68%, with only 6% salient divergences, suggesting that it can replace the target in selected verification cases. Details are provided in Appendix C.2.

Trustworthy outputs. Beyond token-level alignment, the proxy is reliable at semantic and task levels. Even when its outputs differ from the target in wording or intermediate steps, the proxy can still reach correct final answers, suggesting that a certain degree of divergence does not necessarily imply task-level failure (see Appendix C.1). End-to-end relaxed-verification results further show that limited high-confidence proxy deviations introduce little accuracy degradation (see Appendix C.3).

Clear Separability. A useful proxy verifier should not only be aligned with the target, but also expose a simple signal to identify when its predictions are reliable. We find that the margin between the proxy's top-1 and top-2 probabilities serves this role. As shown in Figure 2(b), this confidence margin separates token outcomes: large-margin predictions are dominated by exact matches and benign divergences, whereas salient divergences concentrate in low-margin regions. This separability provides a practical and robust criterion for deciding when proxy verification can be trusted.

3.2 Ternary speculative decoding

Drafter selection and adapter training. We adopt the EAGLE single-layer architecture as our drafter for its state-of-the-art efficiency. Since EAGLE-style drafters are conditioned on target-model features, TriSpec further requires them to process proxy-side features. As shown in Figure 3(a), we add a one-layer MLP adapter to map proxy features into the drafter's feature space. This design supports two training regimes: (i) *joint training*, where the drafter and adapter are optimized together with target and proxy features; and (ii) *adapter-only finetuning*, where the pretrained drafter is frozen and only the adapter is trained on proxy features. We compare them in Section 4.4.

Proxy pre-verification. Given the drafted pairs $(x_1^{(d)}, \mathbf{p}_1^{(d)}), \ldots, (x_k^{(d)}, \mathbf{p}_k^{(d)})$, we first invoke the proxy verifier $\mathcal{M}_p$ to pre-verify the draft and obtain proxy-side predictions and distributions:

$$\big(x_1^{(p)}, \mathbf{p}_1^{(p)}\big), \big(x_2^{(p)}, \mathbf{p}_2^{(p)}\big), \ldots, \big(x_{k+1}^{(p)}, \mathbf{p}_{k+1}^{(p)}\big) \leftarrow \mathcal{M}_p\big(x_1^{(d)}, x_2^{(d)}, \ldots, x_k^{(d)} \,\big|\, X\big), \tag{4}$$

where $x_i^{(p)} \in \mathcal{V}$ and $\mathbf{p}_i^{(p)} \in \mathbb{R}^V$ denote the proxy token and distribution at position i. Following standard speculative decoding, the proxy accepts each draft token by comparing its probability with the drafter's. Formally, for $i \in \{1, \ldots, k\}$, we define a Bernoulli acceptance variable:

$$s_i \sim \text{Bernoulli}\left(\min\left\{1, \frac{\mathbf{p}_i^{(p)}(x_i^{(d)})}{\mathbf{p}_i^{(d)}(x_i^{(d)})}\right\}\right), \tag{5}$$

which in the greedy case reduces to $s_i = \mathbf{1}\{x_i^{(p)} = x_i^{(d)}\}$. Then, the proxy-side acceptance length is

$$\tau_a \;=\; \min\big(\{\, i \;\big|\; 0 \le i \le k-1,\; s_{i+1} = 0\} \cup \{k\}\big). \tag{6}$$

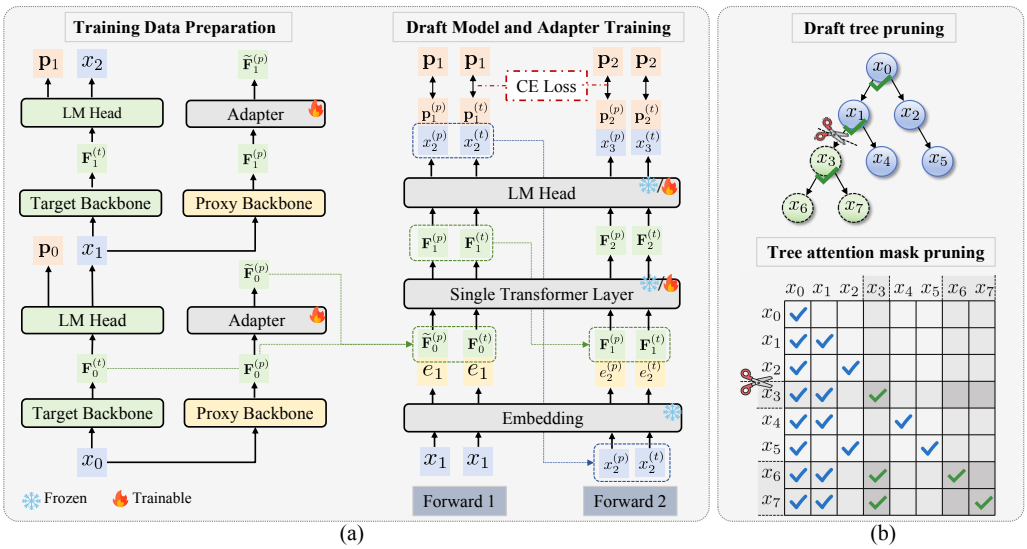


Figure 3: (a) Draft-model and adapter training in TriSpec. The adapter maps proxy features into the drafter's feature space; green, yellow, and gray modules denote target, proxy, and drafter components. (b) Token pruning and tree-attention mask updates after proxy-side verification in TriSpec.

Margin-based Criterion. To avoid unreliable proxy decisions, TriSpec uses the confidence margin between the proxy's top-1 and top-2 probabilities as a reliability signal, as motivated in Sec. 3.1. We define a margin-based predicate $g : \mathbb{R}^V \to \{0, 1\}$ to indicate whether a proxy prediction is trusted:

$$g\big(\mathbf{p}_i^{(p)}\big) = \mathbf{1}\left\{\mathrm{top}_1\left(\mathbf{p}_i^{(p)}\right) - \mathrm{top}_2\left(\mathbf{p}_i^{(p)}\right) \geq \lambda\right\}, \tag{7}$$

where $i \in \{1, \ldots, k+1\}$, $\mathrm{top}_1(x)$ and $\mathrm{top}_2(x)$ denote the largest and second-largest entries of x, and $\lambda \in (0, 1)$ controls the accuracy–latency trade-off. Based on this predicate, we define the proxy-trusted prefix length, which measures how long proxy verification remains reliable:

$$\tau_m = \min\big(\{i \mid 0 \leq i \leq k, g(\mathbf{p}_{i+1}^{(p)}) = 0\} \cup \{k+1\}\big). \tag{8}$$

In particular, if $g(\mathbf{p}_i^{(p)}) = 1$ for all i, then $\tau_m = k+1$, indicating that the proxy's verification is trusted for the entire draft and can reliably provide the $(k+1)$-th token.

Adaptive Verification Routing. Based on the proxy-side acceptance length τ_a and the proxy-trusted prefix length τ_m, TriSpec adaptively routes verification between the proxy and target into two cases:

(i) If $\tau_a < \tau_m$, the proxy remains trusted at the first rejection point and completes the round without invoking the target model (Figure 1(b)). Accordingly, we accept $x_{1:\tau_a}^{(d)}$ and locally correct the next token with $x_{\tau_a+1}^{(p)}$. The generated sequence for this round is $Y = \{x_1^{(d)}, \ldots, x_{\tau_a}^{(d)}, x_{\tau_a+1}^{(p)}\}$.

(ii) If $\tau_a \geq \tau_m$, the proxy verification becomes untrusted before reaching the first rejection point, so verification is delegated to the target model (Figure 1(c)). We first accept $x_{1:\tau_m}^{(d)}$, prune these tokens from the draft tree (Figure 3(b)), and pass the remaining branches to the target for validation:

$$(x_{\tau_m+1}^{(t)}, \mathbf{p}_{\tau_m+1}^{(t)}), \ldots, (x_{k+1}^{(t)}, \mathbf{p}_{k+1}^{(t)}) \leftarrow \mathcal{M}_t\big(x_{\tau_m+1}^{(d)}, \ldots, x_k^{(d)} \mid X, x_{1:\tau_m}^{(d)}\big), \tag{9}$$

Applying the standard left-to-right acceptance rule to $\mathbf{p}^{(t)}$ gives the target-verified acceptance length τ_t. The generated sequence for this round is $Y = \{x_1^{(d)}, \ldots, x_{\tau_m+\tau_t}^{(d)}, x_{\tau_m+\tau_t+1}^{(t)}\}$.

3.3 Theoretical Error Analysis of TriSpec

TriSpec allows the proxy verifier to finalize selected token decisions, introducing possible deviations from target-only decoding. We analyze their effects on task-level performance and the sequence-level output distribution.

Task-level Performance Gap. Let $\Delta_A = \mathbb{E}[U_t] - \mathbb{E}[U_A]$ be the signed gap in normalized task score $U_A \in [0, 1]$, with expectations over inputs and decoding randomness. On a common target-only reference trajectory, a position is task-critical if a single-token intervention using a fixed potential proxy token changes the final score under coupled subsequent randomness. Over pooled reference task-critical positions, let F_λ denote proxy finalization and S a salient divergence, a surrogate for potentially harmful decisions. Define $p_\lambda = \Pr(F_\lambda)$, $\mu_0 = \Pr(S)$, and, when $p_\lambda > 0$, $\mu_\lambda = \Pr(S \mid F_\lambda)$. For $p_\lambda > 0$ and $\mu_0 > 0$, assuming comparable average signed contributions from salient divergences and neglecting their interactions and subsequent trajectory changes, we obtain the first-order estimate

$$\Delta_{\text{tri}} \approx p_\lambda \frac{\mu_\lambda}{\mu_0} \Delta_{\text{proxy}}. \tag{10}$$

Under these conditions, the scaling factor is exactly $\rho_\lambda = \Pr(F_\lambda \mid S)$: the fraction of salient divergences finalized by the proxy among reference task-critical positions. The performance-gap relation remains a first-order approximation; when $\Delta_{\text{proxy}} > 0$ and $\rho_\lambda < 1$, it predicts reduced degradation without guaranteeing it. Figure 2(b) provides qualitative support for margin filtering, not estimates of these latent probabilities. Appendix D.1 specifies the reference construction, approximation assumptions, derivation, and boundary cases.

Sequence-level Distributional Deviation. Task-level performance preservation does not imply distributional equivalence. For a fixed prompt and sampling configuration, let $P_\lambda^{(T)}$ and $P_t^{(T)}$ denote the distributions of T-token continuations under TriSpec and target-only decoding. Let $\mathbb{E}[N_F(\lambda)]$ count proxy-finalized tokens in expectation, ε_λ be their average conditional KL divergence from the target, and B_λ be the accumulated target-routed conditional KL contribution, including routing selection effects. The KL chain rule, convexity, and Pinsker's inequality give

$$\begin{aligned} \mathrm{KL}\Big(P_\lambda^{(T)} \Big\| P_t^{(T)}\Big) &\le B_\lambda + \mathbb{E}[N_F(\lambda)]\varepsilon_\lambda, \\ \mathrm{TV}\Big(P_\lambda^{(T)}, P_t^{(T)}\Big) &\le \sqrt{\frac{B_\lambda + \mathbb{E}[N_F(\lambda)]\varepsilon_\lambda}{2}}. \end{aligned} \tag{11}$$

These bounds characterize the sources of deviation; without estimating their terms, they do not establish small actual bias. The simplification $B_\lambda = 0$ additionally requires target-routed conditionals to match the target distribution. Increasing λ does not guarantee a decreasing bound, and KL may be infinite under greedy decoding. Appendix D.2 gives the precise definitions, proof, support conditions, and zero-exposure convention.

3.4 Further Discussion

Unlike Judge Decoding [3] and SpecCascade [24], which increase the average acceptance length τ by accepting misaligned yet high-quality tokens, TriSpec directly reduces verification time t_v by delegating easy cases to the proxy and avoiding unnecessary target calls. This is especially useful for target-feature-dependent drafters such as EAGLE-3 and MTP, where longer draft horizons often degrade prediction quality and limit further gains in τ. By using the proxy to verify and correct draft tokens, TriSpec alleviates this bottleneck and remains compatible with diverse drafter architectures. Moreover, unlike cascaded frameworks such as TriForce [27], TriSpec better coordinates all components within a unified decoding loop to exploit their strengths and reduce redundant calls.

4 Experiments

4.1 Setup

Models and baselines. We evaluate TriSpec on three model families: Qwen3, DeepSeek-R1-Distill-Qwen (DSQ), and DeepSeek-R1-Distill-LLaMA (DSL). Target models (Qwen3-32B, DSQ-32B, DSL-70B) are paired with same-family smaller proxies (1.7B, 1.5B, 8B) and equipped with corresponding HASS/EAGLE3 drafters. Baselines include standard SD and SpecCascade [24] variants. For a fair comparison, all drafters are retrained on the same data and, at inference, use tree depth 6, expansion top-k 10, and a total draft budget of 60 tokens. Unless otherwise stated, TriSpec uses a margin threshold of $\lambda = 0.5$. Further details are provided in Appendix B.1 and B.2.

Table 1: Accuracy and speedup on five reasoning benchmarks at temperature 0. "SC" denotes SpecCascade. In the Avg Acc. column, the small value denotes the change from target-only decoding.

		GSM8K		MATH500		GK23-en		HumanEval		MBPP		Avg	
Model	Method	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup
Qwen3	Single Model (Target,32B)	98%	1.00×	85%	1.00×	75%	1.00×	87%	1.00×	77%	1.00×	84.4%	1.00×
	Single Model (Proxy,1.7B)	86%	2.36×	81%	3.01×	70%	3.03×	81%	2.93×	51%	2.92×	73.8% -10.6%	2.85×
	HASS	98%	2.98×	86%	2.27×	74%	2.30×	87%	2.03×	77%	2.18×	84.4% ±0%	2.35×
	HASS + SC[Chow]	80%	3.33×	55%	2.60×	51%	2.64×	74%	2.23×	56%	2.33×	63.2% -21.2%	2.63×
	HASS + SC[OPT]	76%	3.25×	38%	2.66×	33%	2.65×	49%	2.30×	20%	2.34×	43.2%-41.2%	2.64×
	HASS + SC[Token]	98%	3.10×	86%	2.34×	74%	2.39×	87%	2.15×	74%	2.28×	83.8% -0.6%	2.63×
	HASS + TriSpec	97%	**3.85**×	86%	**3.18**×	75%	**3.26**×	86%	**2.99**×	74%	**2.80**×	83.6% -0.8%	**3.22**×
	EAGLE3	98%	3.25×	87%	2.73×	74%	2.77×	87%	2.71×	76%	2.77×	84.4% ±0%	2.85×
	EAGLE3 + SC[Chow]	95%	3.39×	70%	2.91×	58%	2.94×	76%	2.82×	67%	2.85×	73.2% -11.2%	2.98×
	EAGLE3 + SC[OPT]	93%	3.43×	58%	3.06×	42%	3.01×	61%	2.91×	49%	2.87×	60.6%-23.8%	3.06×
	EAGLE3 + SC[Token]	98%	3.34×	86%	2.90×	74%	2.91×	87%	2.78×	76%	2.85×	84.2% -0.2%	2.96×
	EAGLE3 + TriSpec	98%	**4.18**×	88%	**3.49**×	74%	**3.55**×	87%	**3.27**×	74%	**3.24**×	84.2% -0.2%	**3.55**×
DSQ	Single Model (Target,32B)	93%	1.00×	86%	1.00×	74%	1.00×	85%	1.00×	72%	1.00×	82.0%	1.00×
	Single Model (Proxy,1.5B)	72%	3.09×	76%	3.32×	62%	3.33×	54%	3.22×	38%	3.32×	60.4% -21.6%	3.26×
	HASS	93%	3.28×	88%	2.99×	72%	2.90×	84%	2.78×	73%	3.02×	82.0% ±0%	2.99×
	HASS + SC[Chow]	77%	3.46×	51%	3.22×	44%	3.06×	75%	2.85×	62%	3.10×	61.8% -20.2%	3.14×
	HASS + SC[OPT]	77%	3.59×	39%	3.20×	20%	3.18×	41%	2.88×	18%	3.12×	39.0%-43.0%	3.19×
	HASS + SC[Token]	91%	3.41×	85%	3.13×	69%	2.96×	84%	2.80×	72%	3.04×	80.2% -1.8%	3.07×
	HASS + TriSpec	91%	**3.76**×	87%	**3.41**×	74%	**3.33**×	85%	**2.98**×	71%	**3.24**×	81.6% -0.4%	**3.34**×
	EAGLE3	93%	3.42×	88%	3.19×	74%	3.11×	86%	3.09×	72%	3.27×	82.6% +0.6%	3.22×
	EAGLE3 + SC[Chow]	84%	3.67×	69%	3.47×	51%	3.43×	72%	3.19×	64%	3.33×	68.0% -14.0%	3.42×
	EAGLE3 + SC[OPT]	81%	3.69×	58%	3.51×	45%	3.46×	54%	3.23×	48%	3.33×	57.2%-24.8%	3.44×
	EAGLE3 + SC[Token]	92%	3.54×	82%	3.36×	72%	3.26×	86%	3.18×	70%	3.27×	80.4% -1.6%	3.32×
	EAGLE3 + TriSpec	91%	**4.14**×	88%	**3.69**×	75%	**3.63**×	88%	**3.32**×	69%	**3.57**×	82.2% +0.2%	**3.67**×
DSL	Single Model (Target,70B)	88%	1.00×	90%	1.00×	78%	1.00×	87%	1.00×	79%	1.00×	84.4%	1.00×
	Single Model (Proxy,8B)	77%	3.71×	83%	3.73×	73%	4.01×	80%	3.89×	59%	4.03×	74.4% -10.0%	3.87×
	EAGLE3	88%	2.83×	89%	2.35×	76%	2.31×	84%	2.49×	80%	2.40×	83.4% -1.0%	2.48×
	EAGLE3 + SC[Chow]	82%	2.90×	81%	2.50×	65%	2.61×	74%	2.58×	71%	2.57×	74.6% -9.8%	2.63×
	EAGLE3 + SC[OPT]	78%	3.02×	66%	2.84×	48%	2.93×	60%	2.89×	51%	2.91×	60.6% -23.8%	2.92×
	EAGLE3 + SC[Token]	87%	2.92×	88%	2.43×	75%	2.56×	85%	2.69×	78%	2.65×	82.6% -1.8%	2.65×
	EAGLE3 + TriSpec	86%	**3.74**×	88%	**3.35**×	77%	**3.36**×	87%	**3.07**×	76%	**3.29**×	82.8% -1.6%	**3.36**×

Benchmarks. We primarily evaluate our method on five reasoning benchmarks: three mathematical reasoning datasets (GSM8K [8], MATH500 [22], Gaokao-2023-EN) and two code-generation datasets (HumanEval [7], MBPP [2]). To further assess performance across diverse domains and more challenging settings, we also include SpecBench [31], HotpotQA [35], Polymath [30], AIME24&25 [39], and GPQA-Diamond [26]. To prevent excessive GPU memory usage, we set the maximum generation length to 8192 tokens. Consistent with prior work [4, 21] on speculative decoding, we fix the batch size to 1. Detailed benchmark settings and analysis are in Appendix B.4.

Metrics. We report pass@1 accuracy for tasks with verifiable answers and LLM-judge scores for SpecBench. Efficiency is measured by throughput (tokens/s) and speedup relative to target-only decoding. Additionally, we define the target-invocation ratio (r_t) as the percentage of target model calls per generated token to quantify reliance on the expensive target model.

4.2 Effectiveness of TriSpec

Table 1 reports the accuracy and efficiency of TriSpec at temperature 0. Across all benchmarks, TriSpec preserves accuracy with less than 1% average degradation while substantially improving throughput over standard SD. For Qwen3, it increases the average speedup of HASS from 2.35× to 3.22× and EAGLE3 from 2.85× to 3.55×, yielding relative gains of approximately 25%–37%, with consistent improvements observed on DSQ and DSL. Compared with lossy SpecCascade variants, TriSpec achieves a better accuracy–speed trade-off. Notably, on Qwen3, EAGLE3 + TriSpec achieves the same average accuracy of 84.2% as EAGLE3 + SC[Token], while improving the average speedup from 2.96× to 3.55×. These results demonstrate that TriSpec provides substantial additional acceleration with minimal task-level performance degradation.

Results at temperature 1.0 (Table 2) further demonstrate the effectiveness of TriSpec under stochastic decoding. With EAGLE-3, TriSpec increases the average speedup from 2.59× to 3.08× on Qwen3,

Table 2: Accuracy and speedup of different methods on five reasoning benchmarks, evaluated with a sampling temperature of 1.

		GSM8K		MATH500		GK23-en		HumanEval		MBPP		Avg	
Model	Method	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup
Qwen3	Single Model (Target,32B)	98%	1.00×	85%	1.00×	76%	1.00×	87%	1.00×	76%	1.00×	84.4%	1.00×
	Single Model (Proxy,1.7B)	88%	2.62×	80%	2.84×	73%	2.90×	76%	2.90×	61%	2.90×	75.6% -8.8%	2.83×
	HASS	96%	2.61×	86%	2.05×	75%	2.05×	88%	1.96×	74%	1.94×	83.8% -0.6%	2.12×
	HASS + TriSpec	95%	**3.01**×	84%	**2.51**×	77%	**2.54**×	89%	**2.31**×	72%	**2.25**×	83.4% -1.0%	**2.52**×
	EAGLE3	98%	2.78×	84%	2.55×	77%	2.52×	86%	2.58×	74%	2.52×	83.8% -0.6%	2.59×
	EAGLE3 + TriSpec	96%	**3.33**×	86%	**3.06**×	76%	**3.13**×	88%	**2.94**×	73%	**2.93**×	83.8% -0.6%	**3.08**×
DSQ	Single Model (Target,32B)	92%	1.00×	88%	1.00×	77%	1.00×	88%	1.00×	76%	1.00×	84.2%	1.00×
	Single Model (Proxy,1.5B)	71%	2.99×	78%	2.91×	66%	3.15×	62%	3.12×	47%	3.11×	64.8% -19.4%	3.06×
	HASS	90%	2.97×	87%	2.19×	76%	2.33×	90%	1.98×	75%	2.04×	83.6% -0.6%	2.30×
	HASS + TriSpec	89%	**3.27**×	87%	**2.47**×	77%	**2.74**×	87%	**2.12**×	73%	**2.18**×	82.6% -1.6%	**2.56**×
	EAGLE3	92%	3.14×	85%	2.40×	77%	2.53×	89%	2.23×	76%	2.29×	83.8% -0.4%	2.52×
	EAGLE3 + TriSpec	88%	**3.40**×	87%	**2.79**×	79%	**2.92**×	88%	**2.36**×	72%	**2.42**×	82.8% -1.4%	**2.78**×
DSL	Single Model (Target,70B)	87%	1.00×	88%	1.00×	75%	1.00×	87%	1.00×	80%	1.00×	83.4%	1.00×
	Single Model (Proxy,8B)	72%	3.80×	81%	3.92×	70%	3.95×	79%	3.93×	60%	3.96×	72.4% -11.0%	3.91×
	EAGLE3	85%	2.79×	88%	1.98×	76%	2.05×	88%	2.01×	81%	1.96×	83.2% -0.2%	2.16×
	EAGLE3 + TriSpec	86%	**3.93**×	89%	**2.74**×	78%	**3.06**×	89%	**2.32**×	78%	**2.39**×	84.0% +0.6%	**2.89**×

Table 3: Performance and speedup on general-domain and more challenging reasoning benchmarks.

		SpecBench		HotpotQA		Polymath		AIME24		AIME25		GPQA-Diamond	
	Method	Score	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup
Qwen3	Single Model (Target, 32B)	8.82	1.00×	93.0%	1.00×	27.8%	1.00×	24/30	1.00×	22/30	1.00×	67.2%	1.00×
	Single Model (Proxy, 1.7B)	7.33	2.72×	84.2%	2.96×	9.4%	3.35×	16/30	3.60×	12/30	3.54×	44.9%	3.33×
	EAGLE3	8.77	2.80×	92.8%	2.77×	27.2%	2.34×	22/30	2.52×	21/30	2.31×	68.2%	2.97×
	EAGLE3+TriSpec	8.62	**3.17**×	92.8%	**3.58**×	26.1%	**3.11**×	23/30	**3.37**×	21/30	**3.30**×	66.2%	**4.39**×

from 2.52× to 2.78× on DSQ, and from 2.16× to 2.89× on DSL, while maintaining average accuracy within 1.4 percentage points of target-only decoding across these model families.

TriSpec remains effective beyond standard reasoning tasks, generalizing to broader domains and more challenging reasoning settings (Table 3). On HotpotQA, TriSpec improves EAGLE-3 speedup from 2.77× to 3.58× while maintaining the same 92.8% accuracy. It also achieves substantial gains on challenging benchmarks such as AIME24 and GPQA-Diamond, increasing speedup from 2.52× to 3.37× and from 2.97× to 4.39×, respectively, with only minor accuracy variations. On SpecBench, TriSpec maintains competitive performance while improving speedup across subtasks, as shown in Figure 4(a). Further details are provided in Appendix B.5.

Table 4 breaks down the verification efficiency of TriSpec on MATH500 and HumanEval. Compared with standard SD, TriSpec consistently reduces the target-invocation ratio (r_t), average per-round verification time (t_v), and end-to-end latency (L), while maintaining comparable acceptance lengths (τ). For example, on Qwen3, it nearly halves r_t and reduces t_v from 0.094s to 0.070s. This indicates that the savings from fewer target invocations outweigh the additional proxy overhead, translating into end-to-end acceleration without relying on longer draft acceptance. Similar trends across DSQ and DSL further demonstrate the effectiveness of proxy verification across model families.

We further evaluate the additional memory overhead introduced by TriSpec's proxy verifier. In our Qwen3-series experiments, the proxy requires substantially less memory than the target model across all context lengths. It accounts for only about 6.4% of the total memory at 8k context and remains below 15% even at 128k, indicating that proxy verification introduces only modest additional memory cost. Detailed measurements are provided in Appendix B.3.

4.3 Routing Strategy

Criterion Selection. We compare rule-based and training-based routing criteria, all using proxy-verifier features to avoid potential semantic bias from the limited-capacity drafter. Rule-based crite-

Table 4: Per-sample averages of methods on MATH500 and HumanEval: Target-invocation ratio (r_t), average per-round verification time (t_v), average acceptance length (τ), Latency(L), and Speed.

		MATH500					HumanEval				
Model	Method	r_t	t_v(s)	τ	L(s)	Speed(Tok/s)	r_t	t_v(s)	τ	L(s)	Speed(Tok/s)
Qwen3	Single Model (Target,32B)	100.00%	/	/	258.76	14.24	100.00%	/	/	234.27	14.65
	HASS	25.01%	0.094	4.21	115.04	32.45	27.14%	0.103	3.93	119.32	29.80
	HASS + TriSpec	**10.24%**	**0.070**	**4.39**	**83.03**	**45.26**	**11.61%**	**0.077**	**4.41**	**78.74**	**43.38**
	EAGLE3	21.38%	0.094	**4.72**	97.52	38.96	22.05%	0.103	4.60	86.62	39.76
	EAGLE3 + TriSpec	**9.95%**	**0.071**	4.59	**74.73**	**49.74**	**11.15%**	**0.080**	**4.76**	**71.83**	**48.00**
DSQ	Single Model (Target,32B)	100.00%	/	/	156.82	16.82	100.00%	/	/	163.68	17.35
	HASS	22.59%	0.068	**4.50**	47.92	50.29	23.55%	0.072	**4.16**	56.75	48.40
	HASS + TriSpec	**11.10%**	**0.054**	4.31	**43.25**	**57.30**	**14.20%**	**0.061**	4.05	**52.90**	**51.15**
	EAGLE3	20.67%	0.068	**4.79**	46.13	53.77	21.45%	0.072	**4.54**	51.80	53.66
	EAGLE3 + TriSpec	**10.63%**	**0.055**	4.78	**42.06**	**60.03**	**13.25%**	**0.064**	**4.54**	**48.92**	**57.02**
DSL	Single Model (Target,70B)	100.00%	/	/	256.79	8.22	100.00%	/	/	254.18	8.37
	EAGLE3	28.70%	0.144	3.48	108.89	19.33	26.88%	0.147	**3.66**	107.92	20.84
	EAGLE3 + TriSpec	**11.60%**	**0.093**	**3.64**	**78.32**	**27.54**	**15.18%**	**0.114**	**3.59**	**87.51**	**25.70**

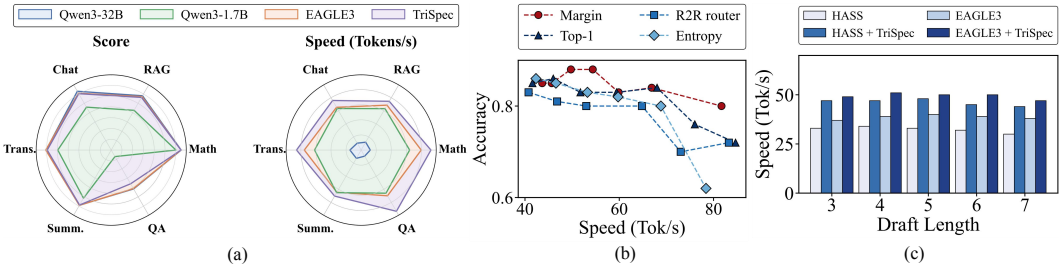


Figure 4: (a) Radar chart illustrating score and speed across all SpecBench subtasks. All experiments are conducted using models from the Qwen3 family. (b) Accuracy–speed trade-offs of different routing strategies. (c) Decoding speed under varying draft lengths for standard speculative decoding and TriSpec. All experiments are conducted on the MATH500 and using Qwen3-32B as the target.

ria include top-1 probability, composite entropy [29], and the top-1–top-2 probability margin, while the training-based baseline follows R2R [13] with a lightweight router for cross-model coordination. As shown in Figure 4(b), although all criteria can preserve accuracy after threshold calibration, margin-based routing achieves the best accuracy–speed trade-off and is used as TriSpec's default.

Threshold Sensitivity. We further examine the impact of the margin threshold λ, which controls when TriSpec escalates proxy verification to the target model. As shown in Figure 5, larger λ values induce more conservative routing, gradually reducing speed while stabilizing accuracy by filtering low-confidence proxy decisions. This consistent trend across models and tasks shows that λ provides an effective control knob for navigating the efficiency–accuracy trade-off. Empirically, $\lambda = 0.5$ is a robust general default, while task-specific calibration can further improve the trade-off.

4.4 Ablation Studies

Draft length. Per-round draft length affects both routing decisions and attainable speedup. Since proxy-target mismatches and token pruning change the verification dynamics, TriSpec's optimal length may differ from standard SD. Figure 4(c) compares throughput across draft lengths using Qwen3 on the MATH500 benchmark. TriSpec outperforms standard SD at all lengths, with a modest shift in the optimum: from 4 to 5 for HASS-trained drafters and from 5 to 4 for EAGLE3.

Draft Training Strategies. TriSpec supports two training paradigms for EAGLE-family drafters to balance performance and training cost. Joint training from scratch optimizes the drafter backbone and adapter together, using 1.77B trainable parameters, while adapter finetuning freezes the pretrained drafter and updates only the 0.42B-parameter adapter. As shown in the left part of Table 5, joint training yields slightly longer acceptance lengths and higher speedups, but adapter finetuning remains close, offering a cost-effective alternative when pretrained weights are available.

Table 5: Ablations on the Qwen3 family. Left: comparing draft-model training strategies, with TP denoting trainable parameters in billions. Right: average performance with/without token pruning.

		MATH500			HumanEval		
Training Strategy	TP	τ	r_t	Speedup	τ	r_t	Speedup
joint from scratch	1.77	4.59	9.95%	3.49×	4.76	11.1%	3.27×
adapter finetuning	0.42	4.52	9.99%	3.44×	4.65	11.4%	3.21×

	five datasets (avg)		
Verification strategy	τ	Acc.	Speedup
w/ Token pruning	4.71	84.0%	3.55×
w/o Token pruning	4.58	84.2%	3.46×

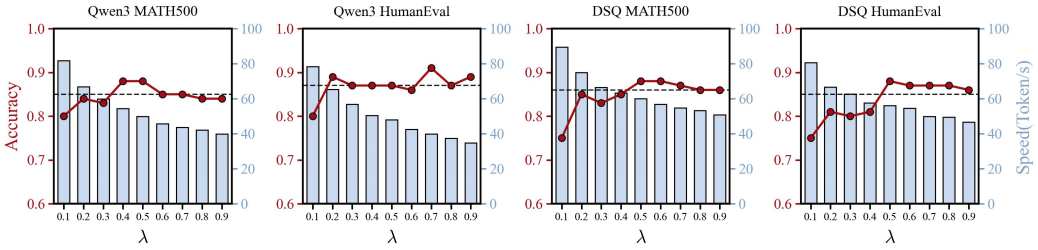


Figure 5: Accuracy–speed trade-off of TriSpec under different margin thresholds λ, where the dashed line denotes the target-only baseline accuracy.

Token Pruning Strategies. In TriSpec's Case II, the target model is invoked for verification. It can either re-verify the entire draft or trust the proxy-verified prefix and process only remaining tokens, which we call token pruning. As shown in the right part of Table 5, token pruning reduces target-side computation and extends the accepted prefix, improving speedup with only a slight accuracy drop. The drop is consistent with Section 3.3: pruning increases proxy-finalized decisions and may induce a small task-level gap. Thus, it provides an optional efficiency–accuracy trade-off for deployment.

5 Related Work

Beyond the approaches discussed earlier, several lines of work accelerate SD in specific settings. On the drafting side, retrieval-based methods like REST [16] and LLMA [34] match relevant input segments to propose draft tokens; cache-reuse methods, including GLIDE [10] and MoA [42], exploit target KV caches for fast drafting; and layer-skipping methods [11, 36] build simplified draft models by partially reusing target parameters. AdaEAGLE [38] and SpecDec++ [17] further dynamically adjust draft length to reduce unnecessary computation. On the verification side, Judge Decoding [3] trains a small judge to accept correct but misaligned drafts, while Cactus [15] improves draft acceptance by adjusting the verifier distribution. SPRINTER [41] uses a lightweight classifier for sequential approximate verification, whereas VIA-SD [32] employs a target-derived slim verifier for hierarchical verification. For long-context acceleration, SpecPV [28] uses partial KV states for efficient verification, while TriForce [27] combines retrieval-based drafting with hierarchical speculation. Despite these advances, most methods relax acceptance or approximate verification, without fully exploiting the inherent alignment between same-family models to facilitate verification.

Small-large model collaboration has also been studied for acceleration. Query-level routing methods [25, 12] select an SLM or LLM per request, R2R [13] learns token-level routing to correct divergent SLM tokens, and SpecCascade [24] combines cascaded deferral with speculative execution. In contrast, TriSpec extends this principle to speculative decoding, where an efficient SLM proxy coordinates with the target and drafter for fine-grained verification and fewer target invocations.

6 Conclusion

In this work, we introduced a new perspective for accelerating speculative decoding by reducing verification time and proposed TriSpec, a ternary framework that combines a lightweight proxy verifier with the drafter and target model. By leveraging same-family smaller models as proxies, TriSpec exploits their strong alignment and reliable outputs to offload a substantial fraction of verification from the target, while a margin-based routing mechanism ensures accuracy is preserved through selective escalation. Extensive experiments show that TriSpec achieves up to 35% additional speedup over standard speculative decoding and sustains accuracy with minimal loss. These results highlight the potential of proxy verification as a new and effective direction for speculative decoding.

Acknowledgments and Disclosure of Funding

HYJ and JCY are supported by National Natural Science Foundation of China (No. 62676245), the Lingang Laboratory (Grant No. LGL-8889-01), STCSM (No. 22DZ2229005), and the 111 Plan (No. BP0719010). XFC is supported by National Natural Science Foundation of China (No. 62476109). BH was supported by RGC Research Fellow Scheme No. RFS2627-2S02, RGC General Research Fund No. 12202026, RGC Young Collaborative Research Grant No. C2005-24Y, RGC General Research Fund No. 12200725, NSFC Major Research Plan No. 92570109, NSFC General Program No. 62376235, Guangdong and Hong Kong Universities "1+1+1" Joint Research Collaboration Scheme No. JRCS-25-26-COMP-05, HKBU Faculty Niche Research Areas No. RC-FNRA-IG/25-26/SCI/05, and HKBU CSD Departmental Incentive Scheme.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] 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.

[3] Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. *arXiv preprint arXiv:2501.19309*, 2025.

[4] 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. *arXiv preprint arXiv:2401.10774*, 2024.

[5] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[6] Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

[9] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*, 2023.

[10] Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. Glide with a cape: A low-hassle method to accelerate speculative decoding. *arXiv preprint arXiv:2402.02082*, 2024.

[11] 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.

[12] Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834*, 2024.

[13] Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang Yuan, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. R2r: Efficiently navigating divergent reasoning paths with small-large model token routing. *arXiv preprint arXiv:2505.21600*, 2025.

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

[15] Yongchang Hao and Lili Mou. Cactus: Accelerating auto-regressive decoding with constrained acceptance speculative sampling. *arXiv preprint arXiv:2604.04987*, 2026.

[16] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*, 2023.

[17] Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.

[18] 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.

[19] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

[20] 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.

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

[22] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[23] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[24] Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. *arXiv preprint arXiv:2405.19261*, 2024.

[25] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

[26] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[27] Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

[28] Zhendong Tan, Xingjun Zhang, Chaoyi Hu, Junjie Peng, and Kun Xia. Specpv: Improving self-speculative decoding for long-context generation via partial verification. *Neurocomputing*, page 135419, 2026.

[29] Jikai Wang, Zhenxu Tian, Juntao Li, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. Alignment-augmented speculative decoding with alignment sampling and conditional verification. *arXiv preprint arXiv:2505.13204*, 2025.

[30] Yiming Wang, Pei Zhang, Jialong Tang, Haoran Wei, Baosong Yang, Rui Wang, Chenshu Sun, Feitong Sun, Jiran Zhang, Junxuan Wu, et al. Polymath: Evaluating mathematical reasoning in multilingual contexts. *arXiv preprint arXiv:2504.18428*, 2025.

[31] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*, 2024.

[32] Yuchen Xian, Yang He, Yunqiu Xu, and Yi Yang. Via-sd: Verification via intra-model routing for speculative decoding. *arXiv preprint arXiv:2606.12243*, 2026.

[33] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[34] Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*, 2023.

[35] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

[36] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*, 2023.

[37] Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. *arXiv preprint arXiv:2408.15766*, 2024.

[38] Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. *arXiv preprint arXiv:2412.18910*, 2024.

[39] Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

[40] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[41] Meiyu Zhong, Noel Teku, and Ravi Tandon. Speeding up speculative decoding via sequential approximate verification. *arXiv preprint arXiv:2502.04557*, 2025.

[42] Matthieu Zimmer, Milan Gritta, Gerasimos Lampouras, Haitham Bou Ammar, and Jun Wang. Mixture of attentions for speculative decoding. *arXiv preprint arXiv:2410.03804*, 2024.

A Algorithm

Algorithm 1 presents the formal pseudo-code for a single generation round of TriSpec, detailing the autoregressive drafting process, proxy-based pre-verification, and margin-based routing mechanism for conditional escalation to the target model.

Algorithm 1 TriSpec in a single round

Inputs: Draft model $\mathcal{M}_d$, Proxy verifier $\mathcal{M}_p$, Target model $\mathcal{M}_t$, input prefix X, margin-based threshold λ

 # Sample k draft tokens from $\mathcal{M}_d$ autoregressively

for $i = 1$ to k **do**
  $\mathbf{p}_i^{(d)} \leftarrow \mathcal{M}_d(X + [x_1, \ldots, x_{i-1}])$
  $x_i \sim \mathbf{p}_i^{(d)}$
end for

 # Pre-verification by $\mathcal{M}_p$

$\tau_a,\ [\mathbf{p}_1^{(p)}, \ldots, \mathbf{p}_{k+1}^{(p)}] \leftarrow \text{Verify}(\mathcal{M}_p, X + [x_1, \ldots, x_k])$

 # Determine proxy-trusted length via margin-based rule

$g(\mathbf{p}_i^{(p)}) \leftarrow \mathbf{1}\Big\{\text{top}_1(\mathbf{p}_i^{(p)}) - \text{top}_2(\mathbf{p}_i^{(p)}) \geq \lambda\Big\},\ i = 1, \ldots, k+1$

$\tau_m \leftarrow \min(\{\, i \mid 0 \leq i \leq k,\ g(\mathbf{p}_{i+1}^{(p)}) = 0\} \cup \{k+1\})$

 # Conditional routing to target based on proxy reliability

if $\tau_a < \tau_m$ **then**
  $x_{\text{bonus}} \sim \mathbf{p}_{\tau_a+1}^{(p)}$
  return $X + [x_1, \ldots, x_{\tau_a}, x_{\text{bonus}}]$
else
  $\tau_t,\ [\mathbf{p}_{\tau_m+1}^{(t)}, \ldots, \mathbf{p}_{k+1}^{(t)}] \leftarrow \text{Verify}(\mathcal{M}_t, X + [x_1, \ldots, x_k])$
  $x_{\text{bonus}} \sim \mathbf{p}_{\tau_m+\tau_t+1}^{(t)}$
  return $X + [x_1, \ldots, x_{\tau_m+\tau_t}, x_{\text{bonus}}]$
end if

B Experimental Details and Additional Experimental Results

B.1 Draft Model Training and Inference Details for TriSpec

We train the draft models following the official HASS/EAGLE-3 codebases and the guidelines in their respective papers. In TriSpec, the drafter backbone and adapter are jointly trained from scratch by default. Specifically, HASS is trained on 68K ShareGPT samples, while EAGLE-3 additionally incorporates UltraChat-200K [9]. We first generate responses autoregressively using the target model, and then compute token-level hidden features offline from both the proxy and target models to supervise baseline drafter training and TriSpec adapter learning.

Training is conducted on 8× NVIDIA A100 80GB GPUs, using the official default hyperparameters unless otherwise specified. During EAGLE-3 training, we retain the full vocabulary and set the batch size to 1 to avoid out-of-memory errors. For a 32B target model, HASS/EAGLE-3 training typically takes 3–4 days, whereas TriSpec requires approximately 4–5 days due to the additional proxy-feature extraction and adapter training. Detailed training costs are reported in Table 6.

During inference, we follow the official EAGLE implementation scripts. All benchmark inference experiments are conducted on a single NVIDIA A100 80GB GPU. For KV-cache management, TriSpec retains the target cache across decoding rounds and, when target verification resumes, processes any uncached accepted tokens together with the current drafts in a single forward pass. This avoids recomputing the cached prefix, and the cache-update cost is included in the reported t_v.

B.2 Baseline Implementation Details: SpecCascade

We adopt SpecCascade as a main baseline, which introduces several rule-based strategies for relaxed verification in speculative decoding. In our experiments, we implement three representative *deferral rules*:

Table 6: Training cost of EAGLE3 and TriSpec on Qwen3 family

Method	Training Strategy	Data Size	Seq Len	TP(B)	Epochs	Training Time(h)
EAGLE3	/	518K	2048	1.35	6	78
EAGLE3 + TriSpec	joint-training from scratch	518K	2048	1.77	6	115
EAGLE3 + TriSpec	adapter-only finetune	518K	2048	0.42	3	44

- **Chow rule**: This rule accepts a candidate token when the draft model assigns it a probability above a fixed threshold (see Table 1 in the original SPECCASCADE paper). Our *confidence-filter* method in Appendix C.2 is equivalent to this rule.
- **Optimal deferral rule**: This rule compares the draft-model probability of the candidate token with the maximum probability under the target-model distribution. Its rejection threshold is determined by the distance between the draft and target distributions (see Eq. (10) in the original paper).
- **Token-specific deferral rule**: This rule accepts a candidate token by comparing its probability under the target-model distribution with the target model's maximum probability, without using the draft-model distribution (see Eq. (14) in the original paper).

We set $\alpha = 0.05$ for the Chow and Optimal deferral rules, both of which rely on the draft-model distribution, and use a larger value of $\alpha = 0.2$ for the token-specific deferral rule. As shown in Table 1, when the drafter is a *small single-layer* model, deferral rules based on draft-model probabilities substantially degrade accuracy: even with a small α, Chow and Optimal deferral still incur notable performance drops, consistent with our token-level analysis in Appendix C.2. By contrast, the token-specific deferral rule better preserves accuracy, since its acceptance decision is governed by the target-model distribution and therefore favors tokens that are more consistent with the target model's preferences. However, because a small singlelayer drafter can generate only a limited number of high-quality candidates, the additional speedup obtained by this rule remains modest.

B.3 Memory Usage

We measure the GPU memory usage of the target model, proxy verifier, and drafter on the Qwen3-series models with batch size 1. As shown in Table 7, the additional memory overhead introduced by the proxy verifier is small relative to the overall footprint. At an 8k context length, the proxy accounts for only 6.4% of total memory, and the complete system still fits within a single A100 GPU. Even at a 128k context length, the proxy accounts for only 14.8% of total memory, a setting where the acceleration benefit of speculative decoding is already substantially constrained by long-context verification and attention overhead. These results indicate that the proxy verifier introduces only modest memory overhead in practical settings.

Table 7: GPU memory usage (GB) of Qwen3-series models under different context lengths.

Component	**2k**	**4k**	**8k**	**16k**	**32k**	**64k**	**128k**
Target	66.01	66.50	67.47	69.42	73.32	81.12	96.72
Proxy	4.27	4.49	4.91	5.76	7.46	10.86	17.66
Drafter	4.16	4.16	4.18	4.21	4.28	4.41	4.67
Total	74.44	75.15	76.56	79.39	85.06	96.39	119.05
Proxy / Total	5.7%	6.0%	6.4%	7.3%	8.8%	11.3%	14.8%

B.4 Standard Reasoning Benchmark Details and Statistical Robustness Analysis

Due to the substantial inference cost of large-scale reasoning models, we primarily evaluate on randomly sampled subsets of 100 examples from each reasoning benchmark. These subsets are used for all experimental results reported in the main text, enabling broad and computationally feasible comparisons across multiple model families and diverse tasks. For the Qwen3 family, we further conduct evaluations on the full benchmark datasets to provide a more comprehensive performance

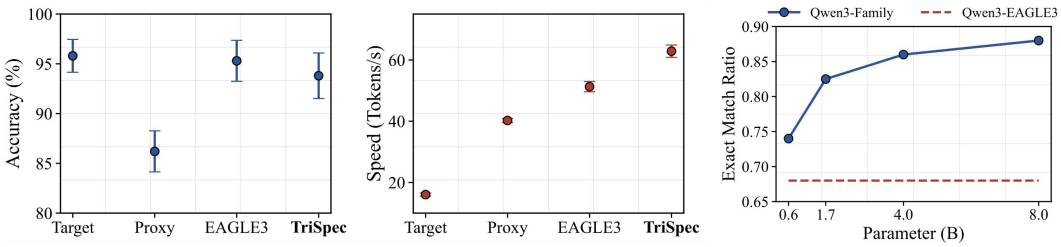


Figure 6: (a–b) Error bars indicate one standard deviation of accuracy and speed across repeated random subset evaluations for four generation methods. (c) Token-level exact-match ratios of Qwen3-family models at different parameter scales and the EAGLE-3 drafter, using Qwen3-32B predictions as the reference.

profile. The complete-test-set results are reported in Table 8, showing that the performance and efficiency trends observed on the 100-sample subsets remain consistent at full scale.

We further assess the statistical robustness of the subset-based evaluation on GSM8K by drawing eight independent random subsets of 100 examples. As shown in Figure 6(a–b), we report the mean accuracy and speed, with error bars indicating one standard deviation across the repeated subset samples. The narrow error bars show that the variation introduced by random sampling is small. Together with the full-benchmark results in Table 8, these findings confirm that TriSpec consistently improves generation speed over standard speculative decoding while incurring only minor accuracy degradation, and that our conclusions are not an artifact of a particular subset selection.

Table 8: Accuracy and speedup results on the full benchmark suite for Qwen3 family

	GSM8K		MATH500		Gaokao2023en		Humaneval		MBPP	
Method	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup
Single Model (Target, 32B)	94.9%	1.00×	86.8%	1.00×	78.2%	1.00×	86.6%	1.00×	74.8%	1.00×
Single Model (Proxy, 1.7B)	86.0%	2.52×	78.2%	2.87×	69.9%	2.76×	77.4%	2.87×	52.8%	2.93×
EAGLE3	94.3%	3.29×	88.0%	2.77×	77.1%	2.77×	87.2%	2.72×	74.2%	2.71×
EAGLE3 + TriSpec	92.9%	**4.08**×	87.4%	**3.48**×	77.4%	**3.39**×	86.0%	**3.23**×	71.4%	**3.33**×

B.5 Implementation Details for General-Domain and Challenging Reasoning Benchmarks

This section describes the benchmarks and implementation protocols used to evaluate TriSpec on general-domain tasks and frontier-level reasoning challenges, including SpecBench, HotpotQA, Polymath, AIME 2024, AIME 2025, and GPQA-Diamond.

SpecBench contains 480 samples across six categories: multi-turn conversation, translation, summarization, question answering (QA), mathematical reasoning, and retrieval-augmented generation (RAG). Its multi-turn conversation category further covers diverse open-ended tasks such as writing, roleplay, extraction, and humanities, providing a broad evaluation of model performance across domains. HotpotQA is a natural-language multi-hop QA benchmark with explicit supporting-fact supervision, offering a QA format distinct from SpecBench; we evaluate on a randomly selected subset of 500 samples. Polymath is a multilingual mathematical reasoning benchmark covering 18 languages and four difficulty levels. To evaluate performance in multilingual and highly challenging settings, we sample 20 high-difficulty instances from each language, resulting in 360 test samples.

AIME 2024 and AIME 2025 are challenging mathematical reasoning benchmarks drawn from the American Invitational Mathematics Examination, each consisting of 30 competition-level problems. GPQA-Diamond is the high-quality subset of GPQA, comprising 198 expert-written multiple-choice questions across biology, physics, and chemistry. For AIME 2024, AIME 2025, and GPQA-Diamond, we evaluate on the full test sets to assess robustness under extremely challenging settings.

For SpecBench, since most tasks are open-ended, we adopt the LLM-judge protocol from FastChat [40], using GPT-4.1 [1] as the judge model to score responses from 0 to 10 on a per-turn basis. For the remaining benchmarks, we report accuracy.

B.6 Experiments on Non-Reasoning Models

We further evaluate TriSpec on non-reasoning models, using LLaMA-3.1-8B-Instruct as the target model and LLaMA-3.2-1B-Instruct as the proxy verifier. As shown in Table 9, TriSpec remains effective in this setting, maintaining accuracy close to target-only decoding while achieving faster verification than standard speculative decoding.

Table 9: Accuracy and speedup results for original LLaMA3 family

	GSM8K		MATH500		Gaokao2023en		Humaneval		MBPP	
Method	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup	Acc.	Speedup
Single Model (Target, 8B)	64%	1.00×	34%	1.00×	18%	1.00×	66%	1.00×	57%	1.00×
Single Model (Proxy, 1B)	34%	2.54×	26%	2.53×	10%	2.50×	46%	2.58×	29%	2.53×
HASS	64%	2.93×	34%	3.27×	18%	3.23×	66%	3.52×	57%	3.46×
HASS + TriSpec	62%	**3.17**×	34%	**3.73**×	17%	**3.66**×	66%	**3.81**×	54%	**3.76**×

B.7 Proxy selection in TriSpec

Figure 6(c) compares the token-level exact match ratio between Qwen3 models of varying parameter counts and the Qwen3-32B target model. Match ratios increase with model size, but the marginal gains diminish beyond the 1.7B scale. Since token-level alignment directly impacts a proxy's ability to assist the target in verification, this trend is a key factor in proxy selection. In particular, the 0.6B model surpasses the single-layer EAGLE3 drafter by only about 6%, yielding relatively few local corrections. Conversely, the 4B model offers merely a 2.5% increase over the 1.7B model, yet incurs substantially higher inference overhead. Balancing speed and alignment effectiveness, we adopt the 1.7B model as the proxy in our main experiments.

C Further Analysis of the Three Models in TriSpec

C.1 Standalone examples

Figure 7 presents a standalone inference example from the three models used in TriSpec within the Qwen3 family. The same-family smaller model (Qwen3-1.7B, proxy verifier) exhibits implicit alignment with the target model (Qwen3-32B) as reflected by their similar response format and language style. Meanwhile, the proxy also demonstrates reasonable standalone capability. Although its output contains token-level differences from the target, it still solves the problem correctly, suggesting that some proxy-target divergences are semantically tolerable and do not necessarily affect the final outcome.

In contrast, the single-layer drafter (EAGLE-3) is trained to mimic the target model but has much weaker contextual modeling capacity due to its substantially reduced attention computation. Without periodic target-model verification during generation, its standalone outputs can easily become illogical or inconsistent.

C.2 Token-level alignment experiments

Experiment details To evaluate token-level alignment between the proxy verifier and the target model, we use ShareGPT as the reference dataset. We first generate autoregressive responses with Qwen3-32B as the target model. We then prefill the target completions with Qwen3-1.7B, the proxy verifier, and record the proxy's next-token prediction at each position. A token is counted as an exact match when the proxy and target predictions are identical. For non-matching positions, we further determine whether the proxy token can be categorized as a benign divergence. Specifically, we first check whether the proxy token lies within the target model's top-k candidates or top-p nucleus at the same position, with $k = 3$ and $p = 0.9$. If not, we consult Qwen3-235B as a stronger judge

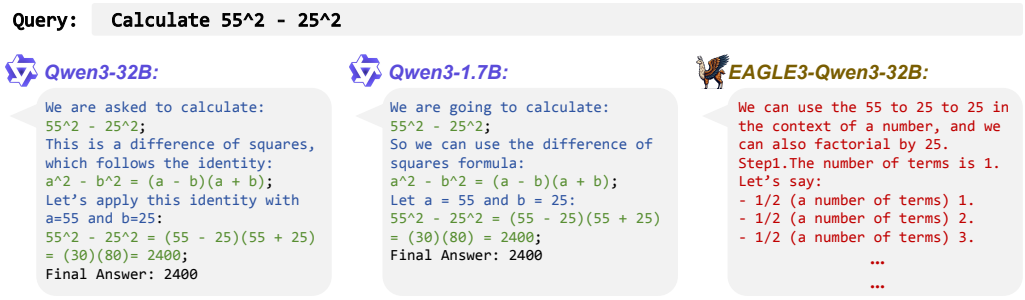


Figure 7: An example of standalone inference for the three models in TriSpec.

using the same top-k and top-p settings, mark the proxy token as benign if it appears in the resulting candidate set. All remaining mismatches are labeled as salient divergences. For both the proxy and target models, we use temperature $= 0$ to ensure deterministic outputs.

Case analysis. We further compare token-level alignment across the same-family small model and the single-layer EAGLE-3 drafter. Figure 8 presents a representative case, showing token-level outputs from the three models together with their confidence scores. The proxy verifier typically assigns higher confidence to exact-match tokens and lower confidence to divergent tokens, indicating that its confidence provides an informative signal for prediction reliability. Overall, the proxy verifier shows stronger alignment with the target than the EAGLE-3 drafter, achieving an exact-match ratio of 82% compared with 68%. This advantage is particularly evident for decision-critical tokens, such as "*" and "78", where the proxy can locally correct errors made by the single-layer drafter. By contrast, due to its limited capacity, the single-layer drafter can be overconfident in erroneous predictions, making its confidence scores unreliable for deciding whether its outputs should be accepted.

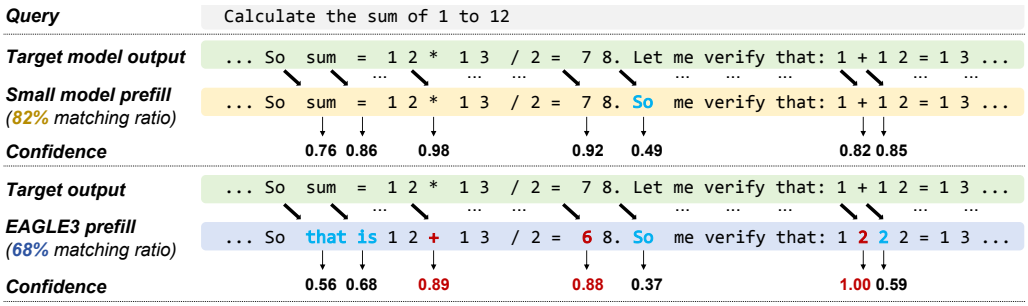


Figure 8: Token-level comparison of the three models in TriSpec. Blue highlights mark benign divergences, and red highlights mark salient divergences. The reported matching ratios are measured over all ShareGPT samples.

C.3 Attempts at relaxing verification

This work focuses on the verification side of speculative decoding and aims to accelerate generation by reducing the cost of strict target-model verification. Besides the proxy-based verification adopted in TriSpec, another natural direction is to relax verification constraints so as to increase the acceptance length. We examine two common relaxation strategies in the SD pipeline: (i) confidence filtering, which accepts a drafter token and skips target verification when the drafter confidence score, $\max p_i$, exceeds a threshold; and (ii) top-k verification, which accepts a drafter token if it appears among the target model's top-k candidates. We evaluate these strategies using both single-layer drafters, i.e., HASS/EAGLE3 weights, and same-family smaller models as draft models. In these experiments, we use a chain-based draft structure with draft length set to 3.

Single-layer drafters are difficult to relax. As shown in Figure 9(b), single-layer drafters suffer rapid accuracy degradation under both relaxation strategies. Confidence filtering may admit

high-confidence but incorrect drafts, because capacity-limited single-layer drafters are often overconfident. Top-k verification is also brittle, as it may accept near-optimal yet incorrect tokens that single-layer drafters frequently produce at decision-critical positions. As a result, relaxed verification is fragile for single-layer drafters: even mild relaxation can induce noticeable accuracy loss, limiting the attainable speedup. This trend is further confirmed by our benchmark results in Table 1, where confidence filtering with a high threshold of 0.95 still struggles to preserve accuracy and often leads to repeated outputs, which we attribute to the single-layer drafter's unwarranted overconfidence discussed in Appendix C.2. This motivates us to avoid drafter-side confidence signals when designing the routing strategy.

High-confidence divergences in same-family smaller models can be tolerated. For same-family smaller models, relaxing verification with an appropriate threshold does not cause significant accuracy loss. This end-to-end experiment shows that high-confidence divergences from smaller models are often tolerable, further supporting their use as proxy verifiers. A similar observation is also reported in Judge Decoding [3].

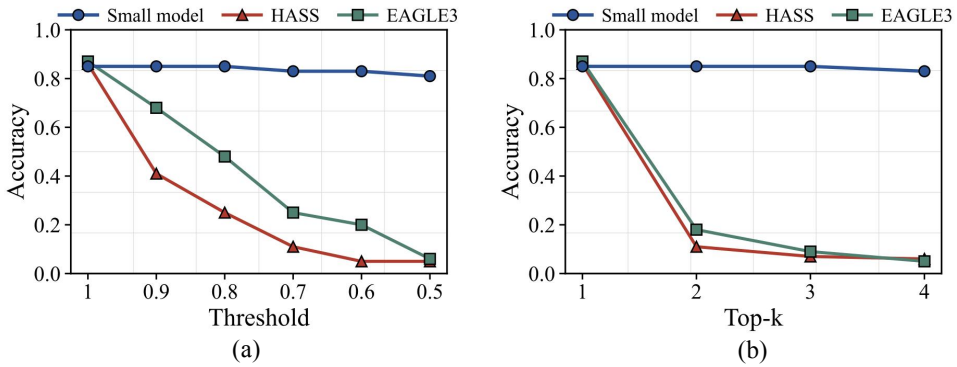


Figure 9: Verification relaxation in the Qwen3 family on MATH500. (a) Confidence filtering under different confidence thresholds. (b) Top-k verification under different values of k.

D Detailed Error Analysis

D.1 First-order Task-level Performance-gap Decomposition

Task metric and common reference space. For an input $q \sim \mathcal{D}$, let $U_A(q, \xi_A) \in [0, 1]$ denote method A's score under a common normalized task metric, where ξ_A contains its decoding randomness. Binary correctness for pass@1 is one example. We define $\Delta_A = \mathbb{E}[U_t] - \mathbb{E}[U_A]$, with expectations over inputs and the respective decoding randomness. This is a signed gap and can be negative.

We use a target-only trajectory as a common reference. A reference realization ω comprises the prompt, the finite target-only response, coupled random choices, and aligned auxiliary draft/proxy states supplying a potential proxy token at each reference history. The potential proxy token is a fixed reference candidate supplied by the proxy-side token decision rule on these aligned states, including speculative acceptance of a draft token or local correction, as applicable; it is not simply the proxy top-1 prediction. The reference histories, auxiliary states, random choices, and resulting candidates are fixed before varying λ, independently of whether proxy finalization occurs. Actual proxy-finalized outputs may differ across thresholds; identifying them with these fixed candidates is part of the first-order approximation, not an assertion that the routed output is invariant to λ. At each reference position, replace only the target-selected token with its potential proxy token, then continue target decoding using the same subsequent random choices and termination rule as in the unmodified reference. A position is task-critical precisely when this paired intervention changes the final task score. Once ω is fixed, this is a definite comparison, not an existential statement over alternative future randomness.

Let $\mathcal{C}(\omega)$ be the resulting set of task-critical positions and let $c = \mathbb{E}_\omega[|\mathcal{C}(\omega)|]$, assuming $0 < c < \infty$. We use the pooled-position probability measure

$$\Pr_{\text{ref}}(A) = \frac{\mathbb{E}_\omega\left[\sum_{i\in\mathcal{C}(\omega)} \mathbf{1}\{A(\omega, i)\}\right]}{c}.$$

Thus, positions are pooled with their input probabilities; per-input event fractions are not averaged with equal weight. On this same space, S denotes a salient divergence between the potential proxy token and the target token. The event F_λ is the reference-model flag that the routing rule, evaluated on the aligned reference draft/proxy states, would finalize that decision through the proxy without target verification. All probabilities below are under $\Pr_{\text{ref}}$.

Task-critical positions are latent analytical objects, not token classes identified during inference. Actual proxy-only and TriSpec histories can depart from the target reference, so mapping their decisions and routing events onto these aligned states is itself a first-order modeling approximation. In particular, F_λ is not an exact statistic of actual TriSpec trajectories. The candidate-set criterion in Appendix C.2 makes S a surrogate for potentially harmful decisions: it does not establish that every benign divergence is harmless or every salient divergence changes the task outcome.

First-order derivation. Define $p_\lambda = \Pr(F_\lambda)$ and $\mu_0 = \Pr(S)$; when $p_\lambda > 0$, define $\mu_\lambda = \Pr(S \mid F_\lambda)$. By the pooled-position definition, the expected reference counts of salient divergences and proxy-finalized salient divergences are exactly $c\mu_0$ and $c\Pr(F_\lambda \cap S) = cp_\lambda\mu_\lambda$, respectively, with the latter factorization used only when $p_\lambda > 0$. These are counts in the reference model, not measured counts on the methods' diverging generation paths.

To relate these counts to task scores, we approximate the gap by the sum of signed contributions from salient divergences, neglecting contributions not captured by this surrogate, interactions among interventions, and changes to subsequent trajectories. We further assume a comparable mean signed contribution κ for all reference salient divergences and the subset selected by F_λ. This comparability need not hold merely because routing uses a confidence margin. For $p_\lambda > 0$ and $\mu_0 > 0$, the model gives

$$\Delta_{\text{proxy}} \approx c\mu_0\kappa, \qquad \kappa \approx \frac{\Delta_{\text{proxy}}}{c\mu_0}, \qquad \Delta_{\text{tri}} \approx cp_\lambda\mu_\lambda\kappa \approx p_\lambda\frac{\mu_\lambda}{\mu_0}\Delta_{\text{proxy}},$$

recovering Eq. (10). Here κ is a modeling coefficient for signed score differences, not a necessarily positive penalty. The derivation is an explanatory approximation, not a performance bound.

Probability interpretation and boundary cases. For $p_\lambda > 0$ and $\mu_0 > 0$, the following identity is exact on the common reference space:

$$\rho_\lambda = p_\lambda\frac{\mu_\lambda}{\mu_0} = \frac{\Pr(F_\lambda \cap S)}{\Pr(S)} = \Pr(F_\lambda \mid S). \tag{12}$$

Thus, $\rho_\lambda \in [0, 1]$ is the conditional fraction of salient-divergence positions among the reference task-critical positions that remain proxy-finalized. It combines exposure and conditional divergence in one quantity; no additional filtering factor should be multiplied into it. The identity is exact, whereas $\Delta_{\text{tri}} \approx \rho_\lambda\Delta_{\text{proxy}}$ still depends on the first-order assumptions above. Only when $\Delta_{\text{proxy}} > 0$ and $\rho_\lambda < 1$ does the model predict smaller task-level degradation. It does not guarantee that the actual signed gap contracts by this factor.

If $p_\lambda = 0$ and $\mu_0 > 0$, define $\rho_\lambda = \Pr(F_\lambda \cap S)/\Pr(S) = 0$ without defining $\Pr(S \mid F_\lambda)$; the reference proxy-finalized salient count is zero. Eq. (10) must not be evaluated through the undefined conditional probability in this case. If $\mu_0 = 0$, the ratio and $\Pr(F_\lambda \mid S)$ are undefined; if $c = 0$, the pooled reference measure is unavailable. We do not apply the decomposition in these cases or infer that the actual performance gap vanishes. Finally, Figure 2(b) describes global token-level margin/divergence statistics and supplies qualitative evidence for filtering, not an estimate of ρ_λ or the latent task-critical probabilities.

D.2 Sequence-level Distributional Bound

Sequence distributions and token-level routing. Fix a common prompt, a sampling configuration (including temperature and any top-k/top-p truncation), and a horizon T. We retain the first

T generated tokens, padding terminated responses with an absorbing EOS symbol. Let $P_\lambda^{(T)}$ and $P_t^{(T)}$ be the resulting TriSpec and target-only distributions, and $H_i = Y_{<i}$ the generated history before position i. All distributions in this subsection are conditional on the fixed prompt. Unlike Appendix D.1, this analysis uses the actual generation law, not the target-reference approximation. KL uses natural logarithms, and $\mathrm{TV}(P, Q) = \frac{1}{2}\sum_y |P(y) - Q(y)|$.

Under TriSpec, let $Z_i = 1$ if token i is finalized through the proxy and $Z_i = 0$ if it is finalized through the target. This distinction concerns the token's final decision source, not whether its decoding round invokes the target. In particular, proxy-finalized prefix tokens pruned before target escalation have $Z_i = 1$. At each output history h with positive probability under TriSpec, define

$$\begin{aligned}\alpha_{\lambda,i}(h) &= \Pr_\lambda(Z_i = 1 \mid H_i = h),\\ r_{\lambda,i}(\cdot \mid h) &= \Pr_\lambda(Y_i \in \cdot \mid H_i = h, Z_i = 1),\\ s_{\lambda,i}(\cdot \mid h) &= \Pr_\lambda(Y_i \in \cdot \mid H_i = h, Z_i = 0),\end{aligned}$$

and let $p_i^{(t)}(\cdot \mid h)$ be the target sampling kernel under the same configuration, evaluated at that history. This kernel is specified by the target decoding policy even at histories unreachable under target-only generation; on histories with positive target probability, it is the target-only conditional. Internal drafts, cache states, and algorithm randomness are marginalized conditional on the displayed events. Consequently, $r_{\lambda,i}$ includes acceptance, correction, and routing selection, and is not generally the proxy's raw softmax distribution. For deterministic EOS padding, set $Z_i = 0$ and use the EOS point mass for the output conditional. Conditionals on zero-probability routes may be assigned arbitrarily, and their weighted KL contributions are defined as zero.

Writing $q_{\lambda,i}$ for the actual TriSpec next-token conditional, the law of total probability gives

$$q_{\lambda,i}(\cdot \mid h) = (1 - \alpha_{\lambda,i}(h))s_{\lambda,i}(\cdot \mid h) + \alpha_{\lambda,i}(h)r_{\lambda,i}(\cdot \mid h). \tag{13}$$

Routing depends on draft/proxy outcomes, so conditioning on the target route can alter its output law. Standard speculative verification alone therefore does not establish $s_{\lambda,i} = p_i^{(t)}$ after this selection.

KL decomposition and TV bound. Expectations are under TriSpec's joint generation law; history-only expectations are written $\mathbb{E}_{P_\lambda}$. Conditional distributions inside divergences are evaluated at H_i. Let $N_F(\lambda) = \sum_{i=1}^T Z_i$ count proxy-finalized tokens, excluding EOS padding. Then

$$\mathbb{E}[N_F(\lambda)] = \mathbb{E}_{P_\lambda}\left[\sum_{i=1}^T \alpha_{\lambda,i}(H_i)\right].$$

Define the accumulated target-route contribution

$$B_\lambda = \mathbb{E}_{P_\lambda}\left[\sum_{i=1}^T (1 - \alpha_{\lambda,i}(H_i))\,\mathrm{KL}\Big(s_{\lambda,i}(\cdot \mid H_i)\Big\|p_i^{(t)}(\cdot \mid H_i)\Big)\right].$$

For $\mathbb{E}[N_F(\lambda)] > 0$, define the exposure-weighted proxy-route divergence

$$\varepsilon_\lambda = \frac{\mathbb{E}_{P_\lambda}\left[\sum_{i=1}^T \alpha_{\lambda,i}(H_i)\,\mathrm{KL}\Big(r_{\lambda,i}(\cdot \mid H_i)\Big\|p_i^{(t)}(\cdot \mid H_i)\Big)\right]}{\mathbb{E}[N_F(\lambda)]}.$$

If $\mathbb{E}[N_F(\lambda)] = 0$, every proxy-route weight is zero almost surely; its accumulated weighted KL contribution is defined as zero. We use $\mathbb{E}[N_F(\lambda)]\varepsilon_\lambda = 0$ as shorthand for that contribution, without dividing by zero.

The autoregressive KL chain rule and convexity in the first argument yield

$$\begin{aligned}\mathrm{KL}\Big(P_\lambda^{(T)}\Big\|P_t^{(T)}\Big) &= \mathbb{E}_{P_\lambda}\left[\sum_{i=1}^T \mathrm{KL}\Big(q_{\lambda,i}\Big\|p_i^{(t)}\Big)\right]\\ &\le \mathbb{E}_{P_\lambda}\left[\sum_{i=1}^T \Big\{(1 - \alpha_{\lambda,i})\mathrm{KL}\Big(s_{\lambda,i}\Big\|p_i^{(t)}\Big) + \alpha_{\lambda,i}\mathrm{KL}\Big(r_{\lambda,i}\Big\|p_i^{(t)}\Big)\Big\}\right]\\ &= B_\lambda + \mathbb{E}[N_F(\lambda)]\varepsilon_\lambda.\end{aligned} \tag{14}$$

Pinsker's inequality, $\mathrm{TV}(P,Q) \le \sqrt{\mathrm{KL}(P\|Q)/2}$, gives the second inequality in Eq. (11). For a finite-valued bound, require $P_\lambda^{(T)} \ll P_t^{(T)}$ and finite weighted KL terms. Absolute continuity here is equivalent to requiring that, at every TriSpec-positive history, the support of $q_{\lambda,i}$ (and hence of each positive-weight route conditional) must be contained in that of $p_i^{(t)}$. This also makes all such histories reachable under the target policy. If this support condition fails, sequence KL is infinite; the chain rule and inequalities are understood in the extended-real sense and supply no finite KL guarantee. Positive temperature does not by itself resolve this issue when top-k/top-p truncation changes the support.

Simplified case and interpretation. The general bound does not assume target-route conditional consistency. If, additionally, $s_{\lambda,i}(\cdot \mid H_i) = p_i^{(t)}(\cdot \mid H_i)$ almost surely on active target routes, then $B_\lambda = 0$ and

$$\mathrm{KL}\Big(P_\lambda^{(T)}\Big\|P_t^{(T)}\Big) \le \mathbb{E}[N_F(\lambda)]\varepsilon_\lambda, \qquad \mathrm{TV}\Big(P_\lambda^{(T)}, P_t^{(T)}\Big) \le \sqrt{\frac{\mathbb{E}[N_F(\lambda)]\varepsilon_\lambda}{2}}.$$

When $\mathbb{E}[N_F(\lambda)] = 0$, the general bound still contains B_λ. Equality with the target distribution follows only with further conditions such as target-route consistency, or when disabling proxy finalization restores standard lossless SD under the same sampling configuration.

B_λ measures target-route conditional divergence, including routing-induced selection effects; it is neither computational overhead nor the actual output error. Convexity can make the bound loose: individual route divergences may be positive even when their mixture equals the target conditional. Since B_λ and ε_λ are not directly estimated here, the result characterizes sources of bias rather than establishing that actual sequence-level bias is small. Increasing λ tightens the margin eligibility test at a fixed state, but changes exposure, conditional distributions, and visited histories; no monotonic decrease in sequence KL follows without additional conditions. Margin confidence therefore supplies no distribution-free guarantee.

Under deterministic greedy decoding with a fixed prompt, different T-token outputs give two distinct point masses: sequence-level TV is one and KL is infinite. Token-level disagreement can provide empirical reliability evidence, but low local disagreement does not by itself establish small sequence TV. Likewise, margin-bin disagreement and target-support statistics are not direct estimates of the conditional KL terms above. These limitations do not alter the bound, but restrict its informativeness.

E Case Study of Complete Generation

We present a TriSpec case study of complete generation and compare it with target-only and proxy-only decoding results(Figure 10). Tokens in blue denote outputs verified or generated by the proxy model (DSQ-1.5B), while tokens in black indicate those verified or generated by the target model (DSQ-32B). In this example, over half of the tokens are validated by the proxy model, highlighting TriSpec's efficiency. When the proxy alone would err, TriSpec attains correctness by involving the target, preserving accuracy.

Specifically, the yellow boxes highlight disagreements between the two models. The proxy incorrectly predicts "He runs 3 sprints each week" instead of the correct "He runs 3 sprints each session." In TriSpec inference, however, the proxy expresses uncertainty at this point and defers to the target model, thereby avoiding the error and showcasing strong robustness.

F Limitations

TriSpec relaxes the exact losslessness guarantee of standard speculative decoding by allowing the proxy model to finalize part of the verification process. Although our theoretical analysis and extensive experiments suggest that the resulting accuracy degradation is limited, such approximate verification cannot guarantee identical behavior on novel tasks and model families. In addition, TriSpec requires the proxy and target models to share a tokenizer for direct token-level verification, which limits proxy selection, particularly across different model families. Finally, introducing an additional proxy model brings new system-level challenges, making optimized deployment in batched serving and production inference systems an important direction for future work.

Case: GSM8K

Question: James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. How many total meters does he run a week?

GT: 540

DSQ-32B

First, I need to determine the total number of sprints James runs each week. He runs 3 sprints each session and does this 3 times a week. So, the total number of sprints is 3 multiplied by 3, which equals 9 sprints.
Next, I need to calculate the total distance James runs. Each sprint is 60 meters, and he completes 9 sprints in total. Therefore, the total distance is 60 meters multiplied by 9, which equals 540 meters.
Finally, the total number of meters James runs each week is 540 meters.
Solution:
To determine the total number of meters James runs in a week, follow these steps:
1. Calculate the total number of sprints per week:
James runs 3 sprints each session and does this 3 times a week.
Total sprints = 3 sprints/session × 3 sessions/week = 9 sprints/week
2. Determine the total distance run per week:
Each sprint is 60 meters. Therefore, the total distance is:
Total distance = 9 sprints × 60 meters/sprint = 540 meters
Final Answer:
540

DSQ-1.5B

I need to determine how many total meters James runs in a week. He runs 3 sprints each week, and each sprint is 60 meters long. To find the total distance, I'll multiply the number of sprints by the distance per sprint.
So, 3 sprints multiplied by 60 meters per sprint equals 180 meters. Therefore, James runs a total of 180 meters each week.
James runs 3 sprints each week, with each sprint being 60 meters long. To find the total distance he runs in a week, multiply the number of sprints by the distance per sprint.
Total meters = Number of sprints × Distance per sprint = 3 × 60 = 180
180

TriSpec

First, I need to determine the total number of sprints James runs in a week. He runs 3 sprints each session and does this 3 times a week. So, the total number of sprints is 3 multiplied by 3, which equals 9 sprints.
Next, each sprint is 60 meters long. To find the total distance James runs in a week, I multiply the number of sprints by the distance of each sprint. That is 9 sprints multiplied by 60 meters per sprint, resulting in 540 meters.
Therefore, James runs a total of 540 meters each week.
Solution:
To determine the total number of meters James runs in a week, follow these steps:
1. Calculate the total number of sprints per week:
James runs 3 sprints each session and does this 3 times a week.
Total sprints per week = 3 sprints/session × 3 sessions/week = 9 sprints/week
2. Determine the total distance run per sprint:
James runs 60 meters per sprint
3. Calculate the total distance run in a week:
Multiply the total number of sprints by the distance per sprint.
Total distance = 9 sprints × 60 meters/sprint = 540 meters
Final Answer:
540

Figure 10: Case study of TriSpec in GSM8K.

G Licenses for existing assets

Our paper builds on several existing models, datasets, code repositories, and external services. We acknowledge the original creators of these assets and summarize their licenses below:

Qwen3-32B is available at `https://huggingface.co/Qwen/Qwen3-32B` under Apache License 2.0.

Qwen3-1.7B is available at `https://huggingface.co/Qwen/Qwen3-1.7B` under Apache License 2.0.

Qwen3-235B is available at `https://huggingface.co/Qwen/Qwen3-235B-A22B` under Apache License 2.0.

DeepSeek-R1-Distill-Qwen-32B is available at https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-32B under MIT License.

DeepSeek-R1-Distill-Qwen-1.5B is available at https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B under MIT License.

Llama-3.1-8B-Instruct is available at https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct under Llama 3.1 License.

Llama-3.2-1B-Instruct is available at https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct under Llama 3.2 License.

DeepSeek-R1-Distill-Llama-70B is available at https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B under MIT License.

DeepSeek-R1-Distill-Llama-8B is available at https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B under MIT License.

ShareGPT is available at https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered under Apache License 2.0.

UltraChat-200K is available at https://huggingface.co/datasets/HuggingFaceH4/ultrachat_200k under MIT License.

GSM8K is available at https://huggingface.co/datasets/openai/gsm8k under MIT License.

MATH500 is available at https://huggingface.co/datasets/HuggingFaceH4/MATH-500 under MIT License.

Gaokao-2023-EN is available at https://github.com/OpenLMLab/GAOKAO-Bench under Apache License 2.0.

HumanEval is available at https://huggingface.co/datasets/openai/openai_humaneval under MIT License.

MBPP is available at https://huggingface.co/datasets/google-research-datasets/mbpp under CC-BY-4.0 License.

SpecBench is available at https://github.com/hemingkx/Spec-Bench under Apache License 2.0.

HotpotQA is available at https://huggingface.co/datasets/hotpotqa/hotpot_qa under CC-BY-SA-4.0 License.

PolyMath is available at https://huggingface.co/datasets/Qwen/PolyMath under Apache License 2.0.

AIME 2024 is available at https://huggingface.co/datasets/math-ai/aime24 under Apache License 2.0.

AIME 2025 is available at https://huggingface.co/datasets/math-ai/aime25 under Apache License 2.0.

GPQA-Diamond is available at https://github.com/idavidrein/gpqa under MIT License.

EAGLE is available at https://github.com/SafeAILab/EAGLE under Apache License 2.0.

HASS is available at https://github.com/HArmonizedSS/HASS under Apache License 2.0.

FastChat is available at https://github.com/lm-sys/FastChat under Apache License 2.0.

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: We state our main claims in the abstract and introduction, and we support them in Sections 3– 4 and Tables 1– 5.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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 our method in Appendix F

   Guidelines:

   - The answer [N/A] 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: Section 3.3 distinguishes a first-order task-level approximation from a sequence-level distributional bound. Appendix D.1 states the reference construction and modeling assumptions; Appendix D.2 provides the bound's definitions, proof, support conditions, and boundary cases.

Guidelines:

- The answer [N/A] 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: We describe our method in Sections 3– 4 and Algorithm 1, and we provide experimental details in Appendix B.

Guidelines:

- The answer [N/A] 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: [Yes]

   Justification: We provide an anonymized code package in the supplemental material for reproducing the main experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify the main training and evaluation settings in Section 4.1 and Appendix B.

   Guidelines:

   - The answer [N/A] 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: [Yes]

   Justification: We report statistical robustness results in Appendix B.4 and Figure 6.

   Guidelines:

   - The answer [N/A] 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: We provide the main compute resource information in Appendices B.1 and B.3.

   Guidelines:

   - The answer [N/A] 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 have reviewed the NeurIPS Code of Ethics and believe that our paper conforms to it.

   Guidelines:

   - The answer [N/A] 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: [N/A]

    Justification: Our paper focuses on inference acceleration, and we do not identify clear negative societal impacts that are specific to our method.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our paper does not release a new high-risk model or dataset.

    Guidelines:

    - The answer [N/A] 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 cite the sources of the models, datasets, codebases, and external services we use, and we summarize their licenses in Appendix G.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: Our paper does not release new assets in the current submission.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: Our paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: Our paper does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] 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 rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core and non-standard component in our method, since we studies collaborative speculative decoding with a target model, a proxy verifier, and a drafter; we also describes LLM-based judging used in parts of the evaluation.

    Guidelines:

    - The answer [N/A] 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 in the NeurIPS handbook for what should or should not be described.