SPECDIFF-2: SCALING DIFFUSION DRAFTER ALIGNMENT FOR FASTER SPECULATIVE DECODING

Jameson Sandler [*1] **Jacob K. Christopher** [*1] **Thomas Hartvigsen** [1] **Ferdinando Fioretto** [1]

ABSTRACT

Speculative decoding has become the standard approach for accelerating Large Language Model (LLM) inference. It exploits a lossless draft-then-verify procedure to circumvent the latency of autoregressive decoding, achieving impressive speed-ups. Yet, current speculative decoding approaches remain limited by two fundamental bottlenecks: **(1)** the autoregressive dependency during drafting which limits parallelism, and **(2)** frequent rejections of draft tokens caused by misalignment between the draft and verify models. This paper proposes *SpecDiff-2*, a novel framework to jointly address these two bottlenecks. It leverages discrete diffusion as a non-autoregressive drafter to address bottleneck (1) and develops novel techniques to calibrate discrete diffusion drafters with autoregressive verifiers, addressing bottleneck (2). Experimental results across a comprehensive benchmark suite show that *SpecDiff-2* achieves a new state-of-the-art across reasoning, coding, and mathematical benchmarks, improving tokens-per-second by up to an average of $+55\%$ over previous baselines and obtaining up to $5.5\times$ average speed-up over standard decoding, without any loss of accuracy.

1 INTRODUCTION

The performance of large language models (LLMs) has rapidly improved alongside increases in both model size and computational budget. Inference-time compute scaling, for instance, has been shown to significantly improve performance on complex tasks, with techniques such as long chain-of-thought reasoning and self-consistency, increasing success rates by allocating more computation at inference time (Wei et al., 2022; Wang et al., 2022; Zelikman et al., 2022; Schick et al., 2023). However, these gains are obtained at the cost of higher wall-time latency; because LLMs, being predominantly based on autoregressive (AR) architectures, produce tokens sequentially, deeper reasoning chains will translate into slower response times. As a result, the depth of problem solving that can be realized, in practice, is constrained by wall-time budgets.

Consequently, inference-time acceleration techniques have become an important area of research. The goal is to reduce the latency of LLM generation without sacrificing model quality. In particular, *speculative decoding* has emerged as a leading framework for overcoming this constraint (Leviathan et al., 2023; Chen et al., 2023). This framework is built around a draft-then-verify procedure: a small drafter model proposes multiple tokens, and a heavier verifier model evaluates the proposal in parallel to accept the longest matching sequence. When the draft model can generate "high-quality" tokens quickly, multiple tokens can be accepted per verification cycle, recovering the outputs of vanilla decoding while reducing end-to-end latency. However, as also noted by Yan et al. 2024, the realized speed-up depends on two key factors: **(1)** *the drafter latency*, since

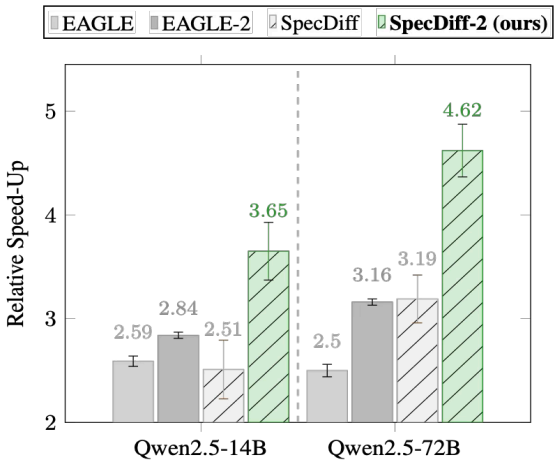


Figure 1. Throughput increase (y-axis) relative to vanilla inference across SoTA acceleration algorithms. Tested on Math500 across 14B and 72B, Qwen2.5-Instruct models. The figure also shows how the *aligned* drafter in *SpecDiff-2* outperforms the base *SpecDiff* drafter by over 40%.

[*]Equal contribution [1]Department of Computer Science, University of Virginia, Charlottesville, USA. Correspondence to: Nando Fioretto <fioretto@virginia.edu>, Jameson Sandler <jmz4ds@virginia.edu>, Jacob K. Christopher <csk4sr@virginia.edu>.

Proceedings of the 8th MLSys Conference, Santa Clara, CA, USA, 2025. Copyright 2025 by the author(s).

generating draft proposals incurs extra time and the drafter must be fast relative to the verifier; and **(2)** *the drafter-verifier alignment*, since misaligned proposals are likely to be rejected, forcing regeneration from the rejection point onward. Thus, the realized throughput hinges on increasing the expected acceptance per cycle while keeping drafter cost low relative to the verifier.

To address the aforementioned challenges, this paper develops *SpecDiff-2*, a speculative decoding system that targets both bottlenecks simultaneously. The approach leverages diffusion language models (DLMs) as non-autoregressive drafters to address bottleneck (1) and develops novel alignment mechanisms to calibrate diffusion drafters with verifiers at train and test-time, addressing bottleneck (2).

Discrete diffusion models generate text by iteratively transitioning the token space toward a fluent sequence in a small, fixed number of steps (Sahoo et al., 2024; Shi et al., 2024). Each denoising step updates all token positions in parallel, so the drafting cost depends primarily on the number of steps rather than the sequence length. This makes discrete diffusion particularly suitable for addressing bottleneck (1): it eliminates the token-by-token dependency of autoregressive drafting, exploits accelerator-friendly batching, and delivers low latency for proposing multi-token drafts (Christopher et al., 2025). However, *diffusion drafters and autoregressive verifiers produce fundamentally different objects*. Diffusion models learn a joint distribution over entire sequences through denoising trajectories, whereas autoregressive models learn local next-token conditionals tied to causal prefixes. As a result, raw diffusion samples can be well-formed globally yet miscalibrated locally with respect to the verifier token-wise decisions. *Aligning these two views is nontrivial and requires mechanisms that bridge joint-generation behavior with prefix-conditional acceptance.* SpecDiff-2 achieves this goal via two complementary mechanisms: a train-time (de-facto, fine-tuning) procedure,Ju called *streak-distillation*, that improves proposer alignment with the verifier by targeting theoretical acceleration, and a test-time acceptance mechanism, called *self-selection acceptance*, that uses the verifier to select drafts that maximize throughput. The consequent framework results in significant throughput improvements ($> 42\%$) over prior diffusion-based drafters, as illustrated in Figure 1, and over 300% improvements over vanilla decoding, without degrading accuracy.

Contributions. This work makes the following key contributions: **(1)** It introduces a parallel drafting mechanism based on discrete diffusion models for speculative decoding, **(2)** to cope with the misalignment between diffusion drafters and autoregressive verifiers, it develops *streak-distillation*, a novel train-time alignment method that encourages the drafter to produce long streaks of accepted tokens, **(3)** this procedure is coupled with *self-selection acceptance*, a test-time mechanism that selects drafts most consistent with the verifier. **(4)** Finally, it demonstrates state-of-the-art throughput across a comprehensive benchmark suite, improving tokens-per-second by up to an average of $+55\%$ over previous baselines and obtaining up to $5.5\times$ average speed-up over standard decoding, without any loss of accuracy.

2 PRELIM: SPECULATIVE DECODING

Speculative decoding (SD) algorithms are built on two language models, a small *drafter* model Q, and a heavier target *verifier* P (Leviathan et al., 2023; Chen et al., 2023). These models generate a sequence of tokens $\boldsymbol{x}_{1:n} = (x_1, \ldots, x_n)$, denoted $\boldsymbol{x}$ when clear from context, within a vocabulary $\mathcal{V}$. For a prefix $\boldsymbol{s} \in \mathcal{V}^\star$, the models expose next-token posteriors $q(\cdot \mid \boldsymbol{s})$ and $p(\cdot \mid \boldsymbol{s})$, associated, respectively, to models Q and P. Generation proceeds in blocks of size γ: the drafter proposes $\boldsymbol{x}_{1:\gamma} \sim Q(\cdot \mid \boldsymbol{s})$ while the verifier scores all positions in parallel. Tokens are committed *left-to-right* using the standard lossless acceptance rule (Leviathan et al., 2023):

For a drafted token x_i at a given position i,

$$\Pr(\text{accept } x_i) = \min\left\{1, \frac{p(x_i \mid \boldsymbol{s})}{q(x_i \mid \boldsymbol{s})}\right\}, \tag{1}$$

where $\boldsymbol{s}$ here, and throughout the paper, denotes the prefix preceding x_i: i.e., $\boldsymbol{s} = \boldsymbol{x}_{1:i-1}$ (for $i > 1$).

This rule results in immediate acceptance whenever $q(x_i \mid \boldsymbol{s}) \leq p(x_i \mid \boldsymbol{s})$. On the first rejection at position k, the remaining draft is discarded and a *replacement* is drawn from the normalized residual:

$$r(x) \;\propto\; \big[p(x \mid \boldsymbol{s}) - q(x \mid \boldsymbol{s})\big]_+.$$

These rules are distributionally faithful: the final transcript matches vanilla decoding from P while enabling parallel proposal and verification (Leviathan et al., 2023), (see Appendix C.1 for proof).

3 ACCEPTANCE & DISTILLATION

The acceptance rate governs the realized speed-up in draft-then-verify decoding.

Definition 3.1 (SD Acceptance Rate)**.** For a prefix $\boldsymbol{s}$ and drafter and verifier pairs Q and P, the speculative-decoding *acceptance rate*, $\alpha(\boldsymbol{s})$, over all prefixes $\boldsymbol{s} = \boldsymbol{x}_{1:i-1}$, is defined as:

$$\alpha(\boldsymbol{s}) \triangleq \Big[1 - \underbrace{\tfrac{1}{2}\sum_{x_i \in \mathcal{V}} \mid P(x_i \mid \boldsymbol{s}) - Q(x_i \mid \boldsymbol{s}) \mid}_{\mathrm{TV}(P,Q)}\Big]. \tag{2}$$

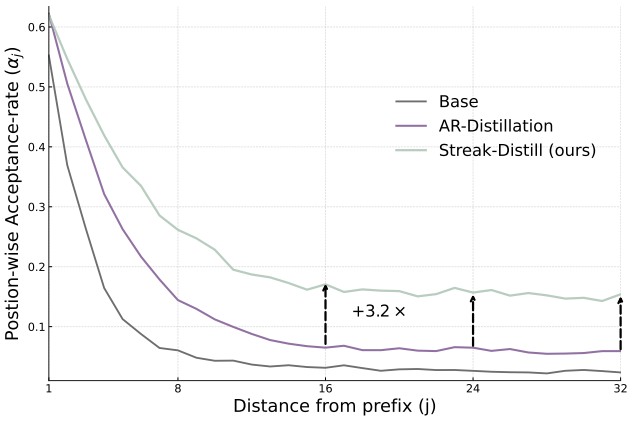


Figure 2. Position-wise acceptance (y-axis, higher is better) against prefix index j (x-axis, distance from prefix $\boldsymbol{s}$). Autoregressive alignment (AR-Distillation above) struggles to align later window indices to verifier (Qwen2.5-14B-Instruct), given lack of window-wise consideration. Both algorithms ran for 20k gradient steps, tested on the Alpaca benchmark.

Further, the acceptance-rate exactly controls the throughput that speculative-decoding benefits from. Let $\alpha_j(\boldsymbol{s})$ denote the position-wise acceptance, that is, the probability that the j-th drafted token is accepted *conditional* on the first $j-1$ drafted tokens having been accepted. Under the standard product-of-accepts identity, the expected number of committed tokens from a draft of length γ is:

$$\tfrac{\text{Tokens}}{\text{Draft}}(\gamma, \boldsymbol{s}) = \mathbb{E}_{\boldsymbol{x}_{1:\gamma}\sim Q(\cdot|\boldsymbol{s})}\big[\sum_{m=1}^{\gamma}\prod_{j=1}^{m}\alpha_j(\boldsymbol{s}\circ\boldsymbol{x}_{1:j-1})\big], \quad (3)$$

where $\circ$ denotes sequence concatenation.

Thus, improvements in the $\alpha_j(\boldsymbol{s})$ translate monotonically into longer accepted streaks and higher throughput at fixed drafter cost.

The acceptance-rate α is then a measure of "alignment", scaling inversely to the total-variation distance (TV) between P and Q. *Increasing the alignment* between P and Q, thus translates to *increasing inference throughput*. Therefore, distillation based alignment methods that target TV reduction directly have become a popular practice for accelerating speculative decoding in autoregressive drafters (Zhou et al., 2023). However, these methods commonly treat the per-position acceptances as exchangeable and only optimize the alignment immediately following the prefix (i.e., $\max[\alpha_1(\boldsymbol{s}\circ\boldsymbol{x}_1)]$). Indeed, when the loss function is defined over a windows of size $= 1$, this "simplified" minimization produces an identical minimizer to the true acceptance criteria (derivation reported in Appendix C.2).

Yet, the position-wise alignment levels vary significantly for diffusion drafters, and thus, the full window of draft predictions must be carefully optimized to obtain the expected speed-up. This is illustrated empirically in Figure 2, showing the position-wise acceptance α_j (y-axis) against the draft index j (x-axis) for three settings: the base diffusion drafter ("Base"), the diffusion drafter aligned via an autoregressive objective ("AR-distillation"), and the proposed streak-aware objective ("Streak-distillation", Section 5.1). Notice that using an AR-style alignment concentrates gains at early positions and degrades rapidly with j, leaving later tokens poorly calibrated. This motivates the need for alignment procedures that align over the full window. Our approach contrasts this behavior by correctly aligning the later indices in the draft window at an average of $3.2\times$ greater acceptance than the AR approach at these positions.

4 Speculative Diffusion Decoding

Before introducing our novel alignment tecniques, this section reviews Speculative Diffusion Decoding (or SpecDiff) (Christopher et al., 2025). SpecDiff builds on the speculative decoding paradigm by substituting the autoregressive drafter model, Q, for a low-latency *diffusion drafter*. This exploits a key property of parallel drafting: proposed tokens in a window of size γ are generated *simultaneously*, so drafter cost depends primarily on the number of denoising steps of the diffusion model rather than on γ, addressing the latency bottleneck noted earlier. Below the paper writes Q and Q^{diff} to distinguish between autoregressive and diffusion drafters, respectively. In the following, we briefly review discrete diffusion models for text generation, and, in particular, masked discrete diffusion models on account of their high performance on language-modeling tasks (Sahoo et al., 2024; Shi et al., 2024).

Masked-discrete diffusion language models. Discrete diffusion language models implement non-autoregressive drafting via corruption-denoising steps. A forward process produces a partially corrupted sequence $\boldsymbol{x}^{(t)}$ for $t \in [0, 1]$ from a clean sequence $\boldsymbol{x}$, and a denoiser Q^{diff} predicts token distributions conditioned on $\boldsymbol{x}^{(t)}$. In masked diffusion models (MDMs) (Sahoo et al., 2024), a special [MASK] token is introduced and each position i is independently replaced by [MASK] with probability $1 - \varepsilon(t)$; otherwise it remains x_i. The denoiser outputs $Q^{\text{diff}}(\cdot \mid \boldsymbol{x}^{(t)}; \theta)_i$ for all positions in parallel and is trained with masked cross-entropy over the currently masked set $M(t)$:

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{x}^{(t)}}\left[\frac{1}{|M(t)|}\sum_{i\in M(t)} -\log Q^{\text{diff}}\big(x_i \mid \boldsymbol{x}^{(t)}; \theta\big)_i\right].$$

Drafting at prefix $\boldsymbol{s}$ with window γ then proceeds as follows. It first populates the draft tokens $\boldsymbol{x}_{1:\gamma}$ with [MASK] tokens; a small number of denoising steps (often a single step in practice) then is applied to yield a joint proposal $\boldsymbol{x}_{1:\gamma} \sim Q^{\text{diff}}(\cdot \mid \boldsymbol{s}\circ[\text{MASK}]^{1:\gamma})$, produced in parallel across positions. The paper denotes $Q^{\text{diff}}(\cdot|\boldsymbol{s})$ to ex-

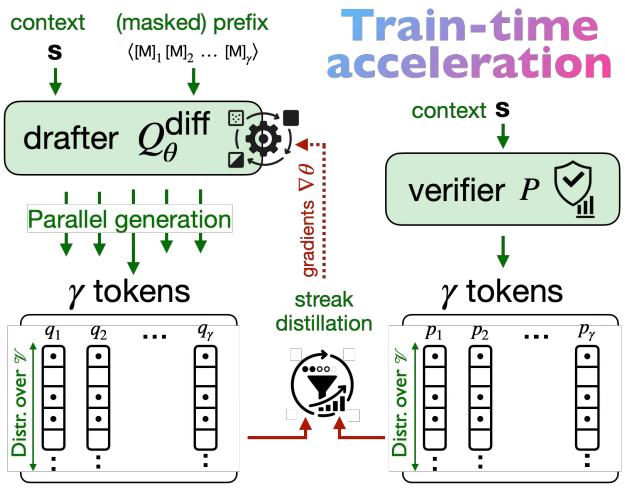


Figure 3. **Train-time acceleration** of *SpecDiff-2*. Q_θ^{diff} (parameters θ) via the streak-distillation equation (see Definitzion 5.1). References are supplied via P, then logits $q_1, \ldots, q_\gamma$ are computed with Q^{diff}. Streak-distill equation is computed, and Q is updated via gradient ascent over expected throughput.

press this joint process, with parameters θ generally omitted where irrelevant, though this should not be confused with an autoregressive conditional.

As motivated in the previous section, while speculative diffusion delivers low and predictable drafter latency, *it does not by itself solve alignment*. Diffusion drafters are trained to model joint denoising distributions over blocks, whereas the verifier evaluates prefix-conditional next-token posteriors; miscalibration at the token level therefore reduces acceptance even when joint samples appear fluent. The next sections introduce alignment objectives and test-time mechanisms tailored to MDM drafters (Section 5) and report the resulting throughput gains (Section 6 and 7).

5 Aligning Diffusion Drafters For Accelerated Inference

We are now ready to introduce the key contribution of this work: two alignment mechanisms tailored to diffusion drafters, a *train-time* (finetuning) objective that scales with distillation compute (Section 5.1) and a *test-time* selection rule that scales with verifier compute (Section 5.2). Importantly, they both act solely on Q^{diff}, while the verifier P remain frozen throughout. Crucially, these mechanisms yield robust drafter–verifier alignment that *generalizes beyond the finetuning data*: indeed, all evaluations are conducted on datasets disjoint from those used in finetuning (see Section 7). Together, and coupled with the use of speculative diffusion, they give rise to SpecDiff-2, a new state of the art speculative decoder.

5.1 Train-time alignment: Streak-distillation

The goal of this section is to align diffusion drafters to the verifier in a manner that *directly* improves the expected accepted streak in $\frac{\text{Tokens}}{\text{Draft}}(\gamma, \boldsymbol{s})$, defined in Equation (3), while keeping training tractable.

The construction begins with a proxy for acceptance, *greedy acceptance*, that yields a smooth training signal. At a prefix $\boldsymbol{s}$, the probability of accepting a drafted token is set to the verifier probability:

$$\Pr(\text{accept } x_i \mid \boldsymbol{s}) = P(x_i \mid \boldsymbol{s}).$$

Note that this scheme does not use the drafter posterior during verification and acts only as an analytical device for deriving a distillation objective. Under this greedy scheme, the position-wise acceptance $\tilde{\alpha}_j$ along a teacher path $\boldsymbol{x}_{1:j-1} \sim P(\cdot \mid \boldsymbol{s})$ reduces to the product:

$$\tilde{\alpha}_j(\boldsymbol{s}) = \mathbb{E}\left[\sum_{x_j \in \mathcal{V}} P(x_j \mid \boldsymbol{s} \circ \boldsymbol{x}_{1:j-1})\, Q^{\text{diff}}(x_j \mid \boldsymbol{s})\right], \quad (4)$$

where the expectation is taken over $\boldsymbol{x}_{1:j-1}$. Note that the diffusion drafter is not conditioned on $\boldsymbol{x}_{1:j-1}$, as all tokens are generated in parallel. Since the inner sum in Equation (4) is a dot product between two categorical distributions over $\mathcal{V}$, it can be rewritten as an expectation w.r.t. either distribution. Thus, the dependency from the verifier P can be made implicit by absorbing it into the sampling distribution for x_j, yielding:

$$\tilde{\alpha}_j(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{x}_{1:j-1}}\left[\mathbb{E}_{x_j \sim P(\cdot \mid \boldsymbol{s} \circ \boldsymbol{x}_{1:j-1})}\left[Q^{\text{Diff}}(x_j \mid \boldsymbol{s})\right]\right] \quad (5a)$$

$$= \mathbb{E}_{\boldsymbol{x}_{1:j-1}}\left[\mathbb{E}_{x_j \sim Q^{\text{diff}}(\cdot \mid \boldsymbol{s})}\left[P(x_j \mid \boldsymbol{s} \circ \boldsymbol{x}_{1:j-1})\right]\right]. \quad (5b)$$

The outer expectation remains over teacher prefixes $\boldsymbol{x}_{1:j-1}$ drawn from the verifier chain, while the inner expectation may be evaluated by sampling x_j either from the verifier P (i.e., during distillation, see Definition 5.1) or from the drafter Q^{diff} (i.e., during verification, see Section 5.2).

This identity is important, as it permits a pathwise re-expression of the streak proxy and leads directly to a tractable training objective. Substituting Equation (5a) into the product-of-accepts construction of Equation (3) yields a pathwise estimator for $\frac{\text{Tokens}}{\text{Draft}}(\gamma, \boldsymbol{s})$ in which the inner product is evaluated along teacher trajectories. Maximizing this quantity across prefixes and verifier continuations yields the *streak-distillation* objective.

Definition 5.1 (Streak-distillation)**.** *Let P be a frozen verifier and Q_θ^{diff} a diffusion drafter with position-wise marginals $q_j(\cdot \mid \boldsymbol{s}; \theta)$. The streak-distillation objective is*

$$\frac{\text{Tokens}}{\text{Draft}}(\gamma, \boldsymbol{s}) \triangleq \mathbb{E}_{\boldsymbol{s}}\, \mathbb{E}_{\boldsymbol{x}_{1:\gamma} \sim P(\cdot \mid \boldsymbol{s})}\left[\sum_{m=1}^{\gamma} \prod_{j=1}^{m} q_j(x_j \mid \boldsymbol{s}; \theta)\right]. \quad (6)$$

This *streak-distillation* objective mirrors the ideal target in Equation (3): the product-of-accepts is preserved, and each factor is replaced by the greedy acceptance term evaluated at verifier tokens. Thus, *the goal is to optimize over all the draft window*, and this is done via the tractable surrogate in Definition 5.1 rather than the intractable path-dependent accept/reject process. Figure 3 illustrates this procedure. Crucially, the streak-distillation objective also forms a tangible bound on speculative-decoding acceptance (Leviathan et al., 2023):

$$\alpha_{\text{SD}} \geq \alpha_{\text{greedy}}, \tag{7}$$

proven in C.3. This notably makes trained DLM drafters versatile during test-time, where differing acceptance strategies can be used with the same alignment scheme.

Alternative streak-distillation forms. A common heuristic for handling the variance associated with aligning parallel prediction systems (Cai et al., 2024) is the exponential-cross entropy function:

$$\mathcal{L}_{\text{expCE}} \triangleq \mathbb{E}_{\boldsymbol{s}}\, \mathbb{E}_{\boldsymbol{x}_{1:\gamma}\sim P(\cdot|\boldsymbol{s})} \left[\sum_{m=1}^{\gamma} -\log q_j\big(x_j \mid \boldsymbol{s};\theta\big) \cdot \lambda^{m-1} \right], \tag{8}$$

where the exponential decay factor $\lambda \in (0:1)$ heuristically applies less weight to later, more chaotic alignment objectives. This behavior is also present implicitly within streak-distillation (equation 6), which compounds acceptance rates over the prediction window. Making this connection more explicit, the heuristic λ can be replace with our dynamic streak-weighting yielding:

$$\mathcal{L}_{\text{Streak-distill(CE)}} \triangleq \mathbb{E}_{\boldsymbol{s}}\, \mathbb{E}_{\boldsymbol{x}_{1:\gamma}\sim P(\cdot|\boldsymbol{s})} \left[\sum_{m=1}^{\gamma} -\log\big(q_m(x_m \mid \boldsymbol{s};\theta)\big) \times \Big(\prod_{j=1}^{m} \text{sg}\,\big(q_j(x_j \mid \boldsymbol{s};\theta)\big)\Big) \right], \tag{9}$$

for stop-gradient operator $\text{sg}(\cdot)$. Equation 9 then pairs the smoothness of cross-entropy optimization, with the dynamic position wise weighting of streak-distillation. Notably, later positions in the window receive more weight once earlier positions become more competent, provided additional conditioning information.

5.2 Test-time alignment: Self-selection acceptance

While streak-distillation improves drafter-verifier alignment at train-time, additional gains can be obtained by leveraging verifier compute at test-time. Prior speculative methods (e.g., EAGLE-x) realize this with autoregressive multi-path expansion, generating several proposal paths with Q^{ar} to improve the likelihood the verifier will accept a continuation (Li et al., 2024b). However, this test-time approach has some drawbacks when using autoregressive drafters. Firstly, the drafting complexity increases: if N tokens are drafted across K distinct paths, then under ideal branching $N \propto \log(K \cdot \gamma)$ and drafting compute scales linearly with the number of tokens $\mathcal{O}(N)$ (for fixed γ). Further, generation of multiple paths must happen in a parallel tree-like manner, increasing complexity.

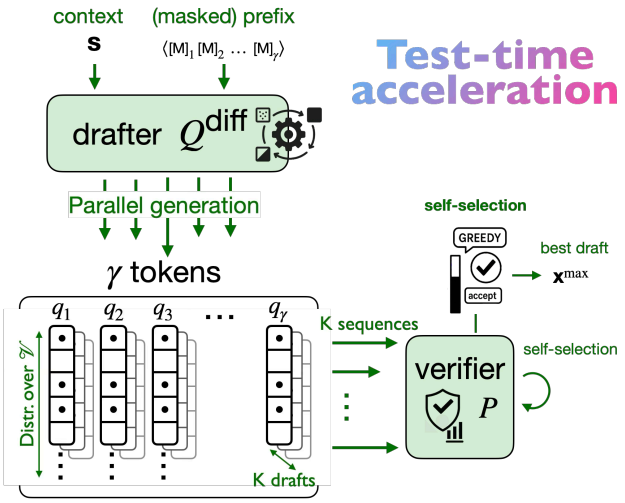


Figure 4. **Test-time acceleration** of *SpecDiff-2*. For prefix $\boldsymbol{s}$ and draft of size γ, the drafter generates marginals $q_1, \ldots, q_\gamma$, and expands them into K discrete drafts of length γ via sampling. Then P is used to select $\boldsymbol{x}^{\max}$, the best draft (*self-selection*) to produce the final sequence.

In contrast, this paper argues that diffusion drafters admit a considerably more simple and efficient multi-draft regime. A single denoising pass exposes all position-wise marginals, from which multiple joint drafts can be sampled with negligible additional cost. This observation motivates a *test-time mechanism* that uses the verifier to select among these drafts before lossless verification.

Self-selection acceptance. The proposed test-time procedure, called *self-selection acceptance*, benefits from the following properties: **(1)** it requires minimal adaptation from vanilla Speculative Diffusion; **(2)** the diffusion drafters provide both time and compute efficient drafting, where for K paths, drafting compute scales with $\mathcal{O}(1)$ (for fixed γ); and **(3)** the proposed approach scales with K (up to the model drafting size ability), allowing additional throughput with negligible sequential overhead in K, as illustrated in Figure 5 (discussed later in Section 7.3).

First, note that diffusion models bring a unique *multi-draft capability* for efficient drafting of several paths. In the speculative diffusion setting (see Section 4), a single denoising step of Q^{diff} over $[\text{MASK}]^\gamma$ produces position-

Algorithm 1: Self-selection acceptance

Input : prefix $\boldsymbol{s}$; verifier $P(\cdot \mid \boldsymbol{s}, \cdot)$; drafter $Q^{\text{diff}}(\cdot \mid \boldsymbol{s})$; integers K, γ
Output: Final generation $\boldsymbol{y} \in \mathcal{V}^{\leq \gamma}$ from $P(\cdot \mid \boldsymbol{s})$
$(q_1, \ldots, q_\gamma) \sim Q^{\text{diff}}(\cdot \mid \boldsymbol{s})$ *(draft marginals)*
for $k \leftarrow 1$ **to** K (in parallel) **do**
  $\boldsymbol{x}^k_{1:\gamma} \sim (q_1, \ldots, q_\gamma)$ *(select best draft)*
  $\tau_k \leftarrow \frac{\text{Tokens}}{\text{Draft}}(\boldsymbol{x}^k_{1:\gamma}, \boldsymbol{s})$ *(compute throughput)*
$(\boldsymbol{x}^{\max}, \boldsymbol{y}) \sim (\arg\max_k \tau_k, [\,])$ *(select best draft)*
$\boldsymbol{y} \sim [\,]$;
for $i \sim 1$ **to** γ **do**
  $p_i \leftarrow P(\boldsymbol{x}^{\max}_i \mid \boldsymbol{s} \circ \boldsymbol{y})$;
  $b \sim \text{Bernoulli}(p_i)$;
  if $b = 1$ **then**
    $\boldsymbol{y} \leftarrow \boldsymbol{y} \circ \boldsymbol{x}^{\max}_i$ *(Accept drafted token)*
  else
    $x_i \sim \dfrac{P(\cdot \mid \boldsymbol{s} \circ \boldsymbol{y})}{1 - P(\boldsymbol{x}^{\max}_i \mid \boldsymbol{s} \circ \boldsymbol{y})}$ *(Replace token)*
    $\boldsymbol{y} \leftarrow \boldsymbol{y} \circ x_i$;
return $\boldsymbol{y}$

wise marginals $\{q_j(\cdot \mid \boldsymbol{s})\}_{j=1}^{\gamma}$. A joint draft is then obtained by independent sampling, $\boldsymbol{x}_{1:\gamma} \sim \prod_{j=1}^{\gamma} q_j(\cdot \mid \boldsymbol{s})$, at negligible cost relative to the neural forward pass.[1] Repeating this sampling K times yields $\{\boldsymbol{x}^{(k)}_{1:\gamma}\}_{k=1}^{K}$ once again with negligible sequential overhead in K. The proposed self-selection method ranks these candidates using a streak-oriented verifier score and selects the best draft for lossless verification. The approach is illustrated in Figure 4.

To evaluate which of the K generated drafts exhibits the highest alignment with the distribution $p_j(\cdot \mid \boldsymbol{s})$ induced by P, the paper uses a theoretically equivalent adaptation of the *streak-distillation* objective defined in Definition 5.1.

Building from this streak-objective, substituting with the equivalency in Equation (5b) yields,

$$\frac{\text{Tokens}}{\text{Draft}}(\boldsymbol{x}, \boldsymbol{s}) = \sum_{m=1}^{\gamma} \prod_{j=1}^{m} p_j\left(x_j \mid \boldsymbol{x}_{t<j}\right), \qquad (10)$$

where $\boldsymbol{x} \sim Q^{\text{diff}}(\cdot|\boldsymbol{s})$ is now a draft generated by the diffusion model Q^{diff}. This alternate form allows us to determine the expected accepted tokens yielded from a draft $\boldsymbol{x}$ using directly the verifier P.

Importantly, Equation (10) formalizes the degree to which draft $\boldsymbol{x}$ is expected to contribute to inference throughput. Thus, for K drafts sampled from $Q^{\text{diff}}(\cdot|\boldsymbol{s})$, the throughput-maximizing draft, denoted $\boldsymbol{x}^{\max}$, can be selected by,

$$\boldsymbol{x}^{\max} \triangleq \arg\max_k \left(\frac{\text{Tokens}}{\text{Draft}}(\boldsymbol{x}^1, \boldsymbol{s}), \ldots, \frac{\text{Tokens}}{\text{Draft}}(\boldsymbol{x}^K, \boldsymbol{s})\right),$$

for a shared prefix $\boldsymbol{s}$. This approach maximizes expected speed-up by construction, selecting based on the expected throughput of each generated draft.

Operationally, the verifier scores each candidate using token-wise posteriors $\left\{p_j\left(x^k_j \mid \boldsymbol{s} \circ \boldsymbol{x}^k_{<j}\right)\right\}_{j=1}^{\gamma}$, then selects $\boldsymbol{x}^{\max}$ before applying the acceptance rule, yielding the *self-selection* mechanism. As detailed in App. B.8, these scores across all K drafts can be computed efficiently via tree-style attention (Xiong et al., 2024). A pseudo-code of the *self-selection acceptance* mechanism is provided in Algorithm 1.

Lossless verification with greedy-acceptance. Once a draft $\boldsymbol{x}^{\max}$ has been selected by P, it then must be verified to determine which tokens to accept and reject. Deploying the greedy-acceptance rule then becomes trivial, given that verifier probabilities over $\boldsymbol{x}^{\max}$ are cached from the previous ranking phase. Additionally, note that the drafting probabilities for $\boldsymbol{x}^{\max}$ are never considered, enabling cross-tokenizer support and extensions to diffusion drafters that do not output calibrated probability distributions (Sahoo et al., 2024; Shi et al., 2024).

6 EXPERIMENTAL SETTINGS

This section presents a comprehensive evaluation of SPECDIFF-2 against state-of-the-art speculative decoding methods, spanning both autoregressive (AR) and diffusion-based drafters. We study four drafter–verifier pairs across three datasets and report *end-to-end wall-clock speedups*, *token-level acceptance*, and *average accepted streak length* under matched decoding budgets. Our goals are twofold: **(i)** quantify the gains from our test- and train-time alignment in speculative decoding, and **(ii)** characterize when diffusion-based drafting can serve to scale efficient reasoning on low budgets. All evaluation keep the output lossless (identical to the verifier model) across all methods.

Models and datasets. The evaluation compares SPECDIFF-2 to strong, publicly released baselines: **(i)** *Speculative Sampling (SpS)* (Leviathan et al., 2023) (classical AR draft-then-verify with a smaller AR drafter), **(ii)** *EAGLE*, and **(iii)** *EAGLE-2* (Li et al., 2024a;b) (AR drafters with verifier-aligned early-accept mechanisms). These algorithms represent the current state-of-the-art in AR-based speculative decoding. Finally, **(iv)** the evaluation includes the original *SpecDiff* (Christopher et al., 2025) (which uses a unaligned diffusion drafter). SpecDiff is the previous state-of-the-art speculative diffusion method, and it is also used to isolate the impact of the proposed train- and test-time alignment methods.

[1] Sampling is inexpensive relative to the denoising computation and parallel over j.

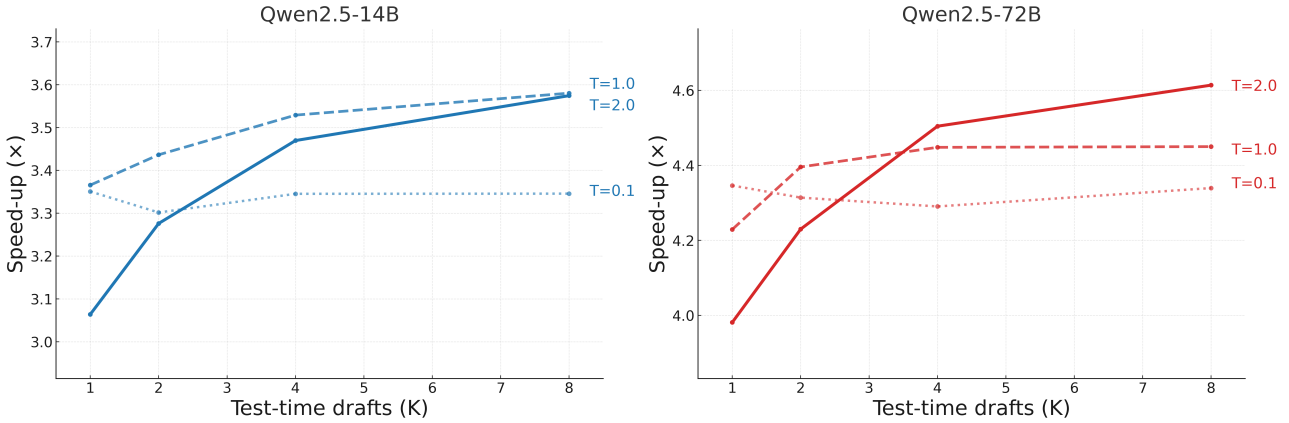


Figure 5. Test-time speed-up scaling w.r.t. parallel drafts K when deploying self-selection. Tested across Qwen2.5-14B-Instruct, and Qwen2.5-72B-Instruct on Math500 prompts (out of distribution). Tested across different drafter temperatures T above. Showing the positive effect of additional draft variance at large K.

Verifier	Drafter	Tokenizer
LLaMA-2-70B-chat LLaMA-2-13B-chat	DiffuLLaMA-7B	LLaMA-2
Qwen2.5-72B-Instruct Qwen2.5-14B-Instruct	DiffuCoder-7B	Qwen2

Table 1. Verifier–drafter pairings and shared tokenizers.

The evaluation covers three settings that stress different acceptance regimes: **(i)** *question answering* (long-form, open-ended reasoning using the *GPQA* dataset (Rein et al., 2024)), **(ii)** *mathematical reasoning* (which has sparse-support, and requires high-precision acceptance, using dataset *Math-500* (Hendrycks et al., 2021)), and **(iii)** *code generation* (structured outputs with exactness constraints, using *HumanEval* (Chen et al., 2021)). These domains jointly probe **(a)** calibration of drafter marginals, **(b)** robustness of acceptance under distribution shift, and **(c)** downstream faithfulness when drafts are partially rejected. All methods use identical prompts, stopping criteria, and verifier decoding parameters per setting. Further, wall-clock is measured using A100 80GB GPUs. *Importantly, note that all tested benchmarks lie outside the training/fine-tuning distribution* for all reported results, thus showcasing strong generalization capabilities of SpecDiff-2.

Verifiers. The evaluation selects verifiers to satisfy **(i)** compatibility with high-quality diffusion-LM tokenization schemes and **(ii)** availability of stable, open implementations of state-of-the-art speculative decoders: it uses QWEN2.5-72B-INSTRUCT (Yang et al., 2024) and LLAMA-2-70B-CHAT (Touvron et al., 2023), which align with released EAGLE/EAGLE-2 toolchains and tokenizers. Appendix A also reports additional results on smaller verifiers (QWEN2.5-14B-INSTRUCT and LLAMA-2-13B-CHAT).

Drafters. For diffusion drafting, the settings adopts adapted Diffusion Language Models (DLMs) with tokenizer alignment to the chosen verifiers: DIFFUCODER-7B (Qwen2.5 tokenizer) (Gong et al., 2025) and DIFFULLAMA-7B (LLaMA-2 tokenizer) (Gong et al., 2024). Notably these diffusion drafters are pretrained, thus they enable efficient application of Streak-distillation as *purely a finetuning paradigm.* For AR drafting baselines, *SpS* uses QWEN2.5-7B-INSTRUCT and LLAMA-2-7B-CHAT as the small draft models. The verifier–drafter pairings and the associated shared tokenizers are summarized in Table 1. Note that we rely on the EAGLE and EAGLE-2 implementations for comparability.[2]

Implementation notes. Details of the drafter hyperparameters and ablations are reported in Appendix B, with key details highlighted here to contextualize the evaluation:

- **Diffusion Steps:** Given the unconventionally large size of our drafter models ($\sim$7$\times$ larger than EAGLE's drafters), it is not neither empirically optimal nor necessary to iterate on drafts for multiple diffusion steps. The diminishing returns of additional diffusion steps is illustrated in Appendix A.2, which provided only slightly better predictions while scaling draft time linearly.
- **Drafter Temperature:** It is also observed that the drafting temperature can be optimized to balance draft quality (suffering when temperature is high) and sufficient variance among the K drafts to take full advantage of self-selection (suffering when temperature is low); as shown in Figure 5, $\text{temperature} = 1.5$ serves as a consistent middle ground between the two extremes.

[2]Despite our best attempts, the results from the original EAGLE-3 work were not reproducible in our environment.

Beyond these hyperparameters, the study defers over-optimization of the the drafter mechanics, to avoid over-fitting to specific settings, and, consequentially, presenting results that are not robust.

7 EXPERIMENTAL EVALUATION

7.1 Wall-Clock Speedups and Accepted Streaks

To assess the performance of different speculative decoders, the speed-up is reported as compared to vanilla generation with the verifier, alongside the average accepted streak $\frac{\text{Tokens}}{\text{Draft}}(\boldsymbol{x}, \boldsymbol{s})$. Other text quality metrics (e.g., PPL, ROUGE, BLEU) are omitted, as the lossless decoding scheme results in outputs that match the performance of the verifier model exactly. The results are reported in Table 2.

Note that *SpecDiff-2* consistently achieves the highest speed-ups and longest accept streaks, outperforming state-of-the-art models, and reporting an average **4.22× speed-up** across all settings, increasing from EAGLE-2 by over 30%. Furthermore, when applied to settings where the drafter specialize, such as when DIFFUCODER is applied for drafting on coding questions, speed-ups soar to over **5× faster** than autoregressive generation.

The table highlights two key trends. First, larger-capacity drafters (7B vs. ∼1B for EAGLE-2) yield substantially higher $\frac{\text{Tokens}}{\text{Draft}}$, and, as expected, these longer accepted streaks translate directly into larger speed-ups. Second, because diffusion drafting is natively parallel, the draft latency remains comparable to EAGLE/EAGLE-2 despite their smaller drafters, which require multiple forward passes to produce the same sequence.

SpecDiff-2 realizes the highest speed-ups when operating on math and coding questions, reporting an average **4.71× speed-up** across verifiers and temperatures compared to EAGLE-2's 3.43× speed-up. In more open-ended QA, the margin tightens to 3.24× for *SpecDiff-2* and 2.80× for EAGLE-2. While both methods slow on open-ended generation where semantic diversity is higher, *SpecDiff-2* still improves over the state of the art, with relative gains tapering from (37%) (reasoning) to (16%) (Q&A). These results suggest that diffusion drafters excel when the target distribution is more structurally constrained (e.g., code, stepwise reasoning), *motivating SpecDiff-2 specifically for accelerating structured LLM reasoning*.

7.2 Scaling Efficient Reasoning With SpecDiff-2

This paper argues that a significant motivation for the development of inference acceleration algorithms is addressing the run-time latency introduced with the emergence of test-time compute scaling.

As shown in several recent works (Tian et al., 2025; Eisenstadt et al., 2025; Zhang et al., 2025) additional wall-time spent while "thinking" at test-time produces increasing accuracy on downstream tasks. In time-constrained settings, however, accuracy is bounded by throughput: faster decoding converts the same wall-time into more usable reasoning tokens. Here, we quantify that link and shows how acceleration from *SpecDiff-2* compounds with test-time compute to raise task accuracy under fixed budgets.

To show this we use Qwen2.5-72B-Instruct (Yang et al., 2024) (a post-trained model with strong instruction following capabilities). Provided a prompt from the Math500 benchmark, the model is instructed to think through its answers before responding in a chain of thought (CoT) style. After a budget of b seconds has been reached, the prompt:

"thinking time is up, wrap up your answer"

is appended, and the correctness of the final answer is evaluated (see Appendix D for example completions).

Figure 6 (left) shows that increasing the reasoning budget (x-axis) directly influences the accuracy on Math500 prompts, where as expected, more reasoning time allows for greater accuracy. Consequently, because *SpecDiff-2* increases throughput, the accelerated system attains higher accuracy at the same wall-time. In Figure 6 (right), the accelerated version of Qwen2.5-72B experiences a $+\mathbf{63}\%$ boost in accuracy over the base model, when restricted to think for $b = 15$ seconds, and a further ($+\mathbf{11}\%$) over unaligned SpecDiff at the same budget.

A clear relationship emerges between the additional compute spent on acceleration as well as alignment, and the accuracy of the system under fixed wall-time constraints. As shown in Figure 6 (right), increasing alignment/training compute (e.g., streak-distillation steps from `30k` to `60k`), adding test-time self-selection, and using speculative diffusion each raise acceptance and effective parallelism, which translates into higher accuracy within the same budget. This perspective positions *acceleration compute* as a practical scaling knob: investing more in alignment and fast drafting yields monotonic improvements in time-limited reasoning, making *SpecDiff-2* particularly attractive for structured, CoT-heavy workloads. The paper therefore presents 'acceleration-compute' scaling as a *unique and novel scaling paradigm*, enabling a new axis along which model performance can be scaled.

7.3 Ablations: Train-time and Test-time Scaling

In Table 3, the baseline SpecDiff provides a qualitative comparison of diffusion drafters in the absence of the train-time and test-time techniques introduced by this paper. Next, these contributions are isolated to assess their individual attribution to the reported speed-ups.

		Math-500		HumanEval		GPQA		Mean	
Model	**Accelerator**	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$
Temperature = 0									
Qwen-2.5-72B	SpS	1.87×	1.77 toks	1.75×	1.77 toks	1.47×	1.78 toks	1.70×	1.77 toks
	EAGLE	2.50×	3.78 toks	2.28×	3.34 toks	1.93×	2.79 toks	2.24×	3.30 toks
	EAGLE-2	3.16×	4.69 toks	3.16×	4.87 toks	2.50×	3.67 toks	2.94×	4.41 toks
	SpecDiff-2	**4.62×**	**6.47 toks**	**4.98×**	**6.98 toks**	**3.28×**	**4.59 toks**	**4.29×**	**5.98 toks**
LLaMA-2-70B	SpS	1.32×	1.72 toks	1.39×	1.69 toks	1.15×	1.70 toks	1.29×	1.70 toks
	EAGLE	3.00×	4.03 toks	3.11×	4.28 toks	2.47×	3.34 toks	2.86×	3.88 toks
	EAGLE-2	3.48×	4.69 toks	3.87×	5.29 toks	3.08×	4.18 toks	3.48×	4.72 toks
	SpecDiff-2	**3.61×**	**5.04 toks**	**4.69×**	**6.57 toks**	**3.48×**	**4.87 toks**	**3.93×**	**5.49 toks**
Temperature = 1									
Qwen-2.5-72B	SpS	1.70×	1.87 toks	1.78×	1.96 toks	1.49×	1.98 toks	1.66×	1.94 toks
	EAGLE	2.19×	3.38 toks	2.16×	3.20 toks	1.69×	2.50 toks	2.01×	3.03 toks
	EAGLE-2	3.12×	4.63 toks	3.15×	4.77 toks	2.52×	3.67 toks	2.93×	4.36 toks
	SpecDiff-2	**5.01×**	**7.00 toks**	**5.51×**	**7.71 toks**	**2.65×**	**3.71 toks**	**4.39×**	**6.14 toks**
LLaMA-2-70B	SpS	1.39×	1.79 toks	1.39×	1.76 toks	1.24×	1.76 toks	1.34×	1.77 toks
	EAGLE	2.81×	3.80 toks	3.22×	4.38 toks	2.55×	3.38 toks	2.86×	3.85 toks
	EAGLE-2	3.56×	4.75 toks	3.91×	5.33 toks	3.08×	4.20 toks	3.52×	4.76 toks
	SpecDiff-2	**3.99×**	**5.58 toks**	**5.28×**	**7.40 toks**	**3.54×**	**4.95 toks**	**4.27×**	**5.98 toks**

Table 2. Comparison on *Math-500*, *livecodebench*, and *MT-Bench* for four base models at temperatures 0 and 1. Each cell reports relative *Speed-up* and average acceptance length $\frac{\text{Tokens}}{\text{Draft}}$ (where +1 is added to represent token from verifier, consistent with previous literature). Experiments utilize two A100 GPUs (80Gb).

Model	**Accelerator**	**Speed-Up**
Qwen2.5-72B	SpecDiff	3.19×
	SpecDiff-2	**4.62**× (+44.8%)
Qwen2.5-14B	SpecDiff	2.51×
	SpecDiff-2	**3.65**× (+45.5%)

Table 3. Math500 comparison for SpecDiff (unaligned drafter) and SpecDiff-2 (this work), with greedy decoding.

The **train-time scaling** curve in Figure 7, illustrates the increased throughput scaling with respect to *streak-distillation* steps. Validating over the first 60,000 train-steps, the figure showcases the efficacy of streak-distillation in improving throughput with minimal train-time compute, (≤ 75) GPU-hours (see Appendix B for hardware details). Both Qwen models see a $\sim 30\%$ increase in speed-up, comparable to improvements seen in autoregressive drafters from DistillSpec (Zhou et al., 2023). Ultimately, the distilled drafters exceed both EAGLE-2 baseline throughput's, by up-to $+32\%$.

Next, the ablation investigates empirical **test-time scaling** trends associated with the novel acceptance scheme *self-selection*. *Self-selection* is tested on Math500 (Hendrycks et al., 2021) prompts across both Qwen2.5 models (see Appendix A.1 for LLaMA scaling). The drafting temperature is varied, ('T' in Figure 5), and the number of drafts is scaled $K = 1, \ldots, 8$, measuring the speed-up at each stage. The results illustrate smooth test-time scaling in speed-up for all model pairs, where a larger number of drafts K increases the probability that the verifier can select an aligned sample, as intended. The largest gains appear at $T = 2.0$, reaching up to $+20\%$ additional throughput at $K = 8$. By contrast, low temperatures $T = 0.1$ show little scaling, reflecting low variance across drafts.

These results show that the proposed train-time distillation provides on average a +30% increase in the overall speed-up, while the test-time algorithm pushes the speed-up by an additional +15%. This effectively results in 40-50% performance improvement over the unaligned SpecDiff.

7.4 Inference Compute Efficiency

Next, an illustration of the inference compute usage of the SpecDiff-2 models is provided relative to auto-regressive baselines. First, note that while inference compute usage can be optimized via diffusion drafter scale and prediction window ablation, this analysis falls outside the scope of this work. Rather, the results below serve as an upper bound on the inference time compute requirements for SpecDiff-2. The compute distinction between Eagle and SpecDiff-2 methods emerges mainly from the verification cost of the large draft-tree generated via Eagle decoding (60 tokens). This verification cost dominates inference usage, despite

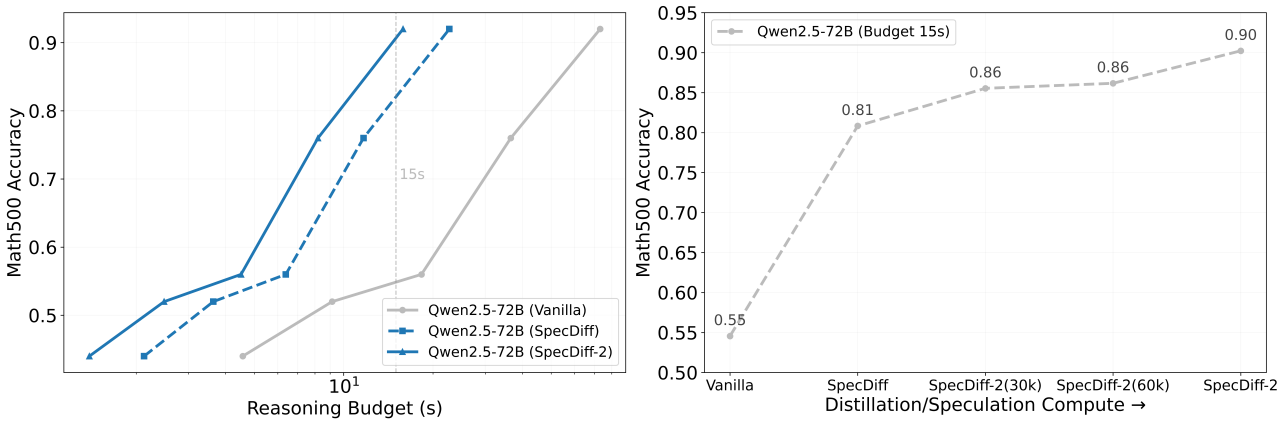


Figure 6. Test-time scaling with respect to wall-time reasoning budget with CoT style prompting. Additional budget results in greater accuracy, accelerated model [Qwen2.5-72B-(SpecDiff-2) above] exhibits $+63\%$ accuracy increase over vanilla model [Qwen2.5-72B(Vanilla) above], and $+11\%$ accuracy over Qwen2.5-72B-(SpecDiff) under identical wall-time constraints (15s). Checkpoints at (30k) and (60k) distillation steps are also shown.

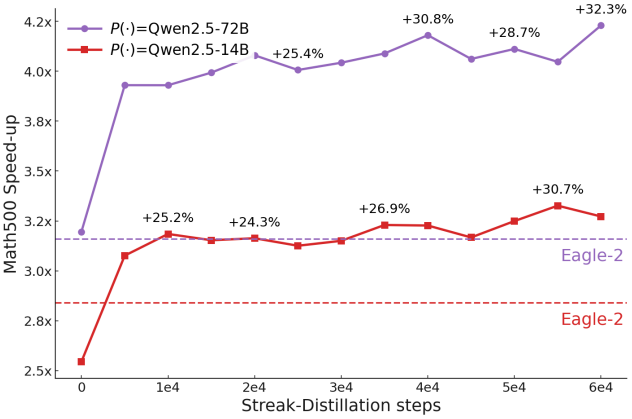


Figure 7. Scaling Streak-distillation for larger scale drafter alignment on reasoning corpus (see Appendix B for details). We see $\sim 30\%$ increases in acceleration across out-of-distribution Math500 prompts, surpassing SOTA baselines (Eagle-2) in the process. $P(\cdot)$ shows verifier model, red is Qwen2.5-14B-Instruct (Qwen2.5-14B above), purple is Qwen2.5-72B-Instruct (Qwen2.5-72B above), with corresponding Eagle-2 baselines.

the higher drafting cost of the diffusion model. The speed-up distinction emerges because the larger diffusion drafter is capable of generating better predictions that the Eagle, which must rely on costly draft trees in order to obtain competitive acceptance.

7.5 Batched Inference and Memory Usage

SpecDiff-2 generalizes the batched inference case, conceptually, in a similar manner to AR speculative-decoding methods like Eagle. Memory and compute usage scale linearly with batch size B.

Given that the FLOPs remains competitive with Eagle, the distinction lies primarily in memory usage. Table 4 illustrates that SpecDiff-2 uses similar levels of memory to Eagle, with the gap narrowing as model size increases. It is anticipated that additional advantage could be realized by optimizing the drafter size, thus the memory metrics reported here remain an upper bound.

Model	Accelerator	Speed-Up ↑	Vram ↓
Qwen2.5-72B	Eagle (60)	2.19×	**135.57 Gibs**
	SpecDiff-2 (32)	**3.99×**	149.77 Gibs
Qwen2.5-14B	Eagle (60)	2.43 ×	**27.47 Gibs**
	SpecDiff-2 (32)	**3.08×**	41.67 Gibs

Table 4. Vram usage for SDD-2 ($k = 1$, 32 tokens per step) vs Eagle (60 tokens draft tree), on an Ax100 Gpu (512 tokens).

Model	Accelerator	Speed-Up ↑	Flops ↓
Qwen2.5-72B	Eagle (60)	2.19×	8.78×10^{14}
	SpecDiff-2 (32)	**3.99×**	$\mathbf{5.06}\times10^{14}$
Qwen2.5-14B	Eagle (60)	2.43 ×	1.67×10^{14}
	SpecDiff-2 (32)	**3.08×**	$\mathbf{1.31}\times10^{14}$

Table 5. Inference compute usage for SDD-2 ($k = 1$, 32 tokens per step) vs Eagle (60 tokens draft tree), on an Ax100 Gpu (512 tokens).

7.6 Latency Analysis

The proposed diffusion drafters perform efficient parallel computation over the prefix, with an effect on latency only emerging as utilization reaches hardware capacity. Up to $\gamma = 16$ and a context size of 256 tokens, a single NVIDIA A6000 GPU does not reach full utilization. Consequently, minimal effect is observed on diffusion latency prior to this saturation. This latency starts to increase as the hardware

Table 6. Diffusion (Diffucoder-7B) forward latency vs. γ (ctxt=256).

γ	latency (ms)
8	36.13 ± 0.51
16	36.13 ± 0.25
32	37.83 ± 2.61
64	38.77 ± 0.86

throughput becomes exceeded, and similar trends can be expected in high compute regimes. Additionally, our multi-draft framework enables extra drafting with respect to K with negligible latency overhead.

8 Related Work

Early work in LLM inference acceleration focused primarily on architectural changes, such as model pruning (Sun et al., 2024) and quantization (Frantar et al., 2023), enabling more efficient generation at the cost of overall generation quality. Addressing this, speculative decoding emerged as a promising alternative, accelerating generation without degrading the generations (Leviathan et al., 2023; Chen et al., 2023). While broadly regarded as a trans formative contribution, however, early works presented moderate speed-ups, motivating further exploration of speculative drafting techniques. Existing literature on the topic can be partitioned into two categories, with prior works addressing either the autoregressive dependency during drafting or the misalignment between the draft and verifier models (e.g., one of the two *speculative-bottlenecks* reviewed in the Introduction section.

Draft model alignment has been improved through both specialized training procedures and test-time drafting enhancements. In particular, several approaches build on principles from knowledge distillation (Zhou et al., 2023) and optimal transport (Sun et al., 2023) to design training-time objectives that align the draft model distributions more directly with the verifier posterior. These approaches, however, are specifically catered to AR drafting architectures. Another line of work has realized significant success by implementing clever drafting algorithms, such as employing draft-trees, which generally result in much higher throughput per draft-then-verify step (Li et al., 2024a;b; 2025). This work has compared against such methods.

Draft model latency has been studied separately, leveraging draft models with parallel draft heads (Ankner et al., 2024). More recent work demonstrated that non-autoregressive diffusion language models can operate as highly efficient drafters, similarly removing sequential dependencies from the drafting step (Christopher et al., 2025).

Our analysis reports substantial improvements against these proposals.

Dynamic steering methods Recent work has explored dynamic steering and adaptive drafting strategies as alternatives to static speculative pipelines. Steering methods, such as those proposed by (Xiong et al., 2024), modify the drafter distribution at inference time to better align with the verifier, enabling improved acceptance without retraining. In contrast, tree-based approaches like DySpec (Xiong et al., 2024) restructure the drafting process itself, constructing dynamic token trees to increase parallelism and improve verification efficiency. The key advantage of the proposed approach is that in SpecDiff-2 draft tree generation cost is negligible as a function of γ, due to the joint distributions generated by diffusion drafters. We leave the synthesis between dynamic drafting methods and diffusion drafters open for future work.

9 Future Work and Limitations

We conclude by outlining the main limitations of *SpecDiff-2* and the most promising directions they expose. At a high level, the presented results broaden the design space of acceptance rules and drafting strategies for speculative decoding, but several systems and theory questions arise.

First, note that acceptance in speculative decoding is typically fixed. The *greedy-acceptance* rule proposed in this paper shows that alternative criteria can be both practical and advantageous. Formal analysis of these rules (e.g., their bias/variance trade-offs, worst-case acceptance under mis-calibration, and interactions with verifier temperature) remain an open direction. Because greedy-acceptance naturally supports heterogeneous drafters and mismatched tokenizers, a systematic study of *cross-family* and *cross-tokenizer* drafting (rather than same-family pairs) is a key next step.

Beyond tokenization schemes, the proposed self-selection drafting mechanism sets the stage for different drafting procedure producing multiple drafts in parallel. A natural extension is to deploy K *distinct* drafters specialized to sub-distributions (e.g., algebraic manipulation vs. numeric computation vs. code) and to select among them at test time. Doing so raises open questions about diversity–alignment trade-offs, cost-aware selection policies, and how best to allocate a fixed compute budget across K, temperature, and proposal depth.

This work also introduces an important question of the optimal size for diffusion draft models. While this has been well established for autoregressive architectures, the empirical results show that the parallel drafting properties of

diffusion enable scaling to much larger drafter architectures, while maintaining similar drafter latency. In this work, drafter selection was largely determined by diffusion models quality, with DIFFULLAMA and DIFFUCODER being some of the strongest open-source DLMs, but the lower speed-ups for smaller verifiers suggest that greater speed-ups could be realized by scaling the DLM drafters proportionally to the verifiers. Deriving scaling laws and compute–latency frontiers for diffusion drafters, ideally as functions of γ, acceptance, and verifier size, remains an open direction.

Finally, there is significant room for hardware based optimization. Efficient kernels specialized for semi-autoregressive inference are lacking, and support for KV caching in the context of γ length diffusion drafts is important for efficient inference.

10 CONCLUSION

This study has presented *SpecDiff-2*, providing the first work on model alignment for speculative decoding with diffusion draft models. Motivated by the absence of approaches which simultaneously address drafter latency and drafter-verifier alignment, this paper introduces train-time distillation techniques and test-time parallel draft generation catered for diffusion drafters. Leveraging these novel diffusion alignment methods, the proposed framework observes significant increase in token throughput and achieves up to an average of $+55\%$ faster generation over previous baselines and up to $5.5\times$ average speed-up over standard decoding, without any loss of accuracy. *SpecDiff-2* achieves state-of-the-art performance across a breadth of reasoning, coding, and mathematical benchmarks, establishing a new standard for lossless acceleration.

We believe that the use of diffusion drafter represents a significant step towards inference-time acceleration, opening several promising directions for accelerating LLMs.

ACKNOWLEDGMENTS

This work was partially supported by an NVIDIA Academic Grant Program Award. The work was also supported by NSF grants 2334448, 2401285, and 2533631, DARPA Contracting activity #HR0011252E005, and UVA's National Security Data & Policy Institute, ODNI Contracting Activity #2024-24070100001. Its view and conclusions reflect those of the authors only.

REFERENCES

Ankner, Z., Parthasarathy, R., Nrusimha, A., Rinard, C., Ragan-Kelley, J., and Brandon, W. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

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

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

Christopher, J. K., Bartoldson, B. R., Ben-Nun, T., Cardei, M., Kailkhura, B., and Fioretto, F. Speculative diffusion decoding: Accelerating language generation through diffusion. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 12042–12059, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.601. URL https://aclanthology.org/2025.naacl-long.601/.

Eisenstadt, R., Zimerman, I., and Wolf, L. Overclocking llm reasoning: Monitoring and controlling thinking path lengths in llms. *arXiv preprint arXiv:2506.07240*, 2025.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Gong, S., Agarwal, S., Zhang, Y., Ye, J., Zheng, L., Li, M., An, C., Zhao, P., Bi, W., Han, J., et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

Gong, S., Zhang, R., Zheng, H., Gu, J., Jaitly, N., Kong, L., and Zhang, Y. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Hendrycks, D., Burns, C., Basart, S., Critch, A., Li, J., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

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

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024b.

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

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Sahoo, S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Schick, T., Dwivedi-Yu, K., Dessi, R., Raileanu, R., Lomeli, M., and Severyn, A. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Shi, J., Han, K., Wang, Z., Doucet, A., and Titsias, M. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models, 2024. URL https://arxiv.org/abs/2306.11695.

Sun, Z., Suresh, A. T., Ro, J. H., Beirami, A., Jain, H., and Yu, F. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36:30222–30242, 2023.

Tian, X., Zhao, S., Wang, H., Chen, S., Ji, Y., Peng, Y., Zhao, H., and Li, X. Think twice: Enhancing llm reasoning by scaling multi-round test-time thinking. *arXiv preprint arXiv:2503.19855*, 2025.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837, 2022.

Xiong, Y., Zhang, R., Li, Y., Wu, T., and Zou, L. Dyspec: Faster speculative decoding with dynamic token tree structure, 2024. URL https://arxiv.org/abs/2410.11744.

Yan, M., Agarwal, S., and Venkataraman, S. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*, 2024.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Zelikman, M., Wu, Y., Mu, J., and Goodman, N. D. STaR: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

Zhang, Q., Lyu, F., Sun, Z., Wang, L., Zhang, W., Hua, W., Wu, H., Guo, Z., Wang, Y., Muennighoff, N., et al. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025.

Zhou, Y., Lyu, K., Rawat, A. S., Menon, A. K., Rostamizadeh, A., Kumar, S., Kagy, J.-F., and Agarwal, R. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2023.

A ADDITIONAL RESULTS

Additional evaluation is conducted on smaller verifier models QWEN-2.5-14B-INSTRUCT and LLAMA-2-13B-CHAT. As previously noted, the diffusion drafters used for the 70B+ parameter models are already oversize when comparing to conventional AR draft models. For these smaller verifiers, the 7B parameter diffusion drafters are certainly too large to behave as optimal drafters in this ablation. However, as the optimal diffusion drafter size remains in question, it is valuable insight to see how decoding behavior changes with scale of the verifier.

Table 7 reports the speed-up and throughput for the smaller verifiers. While DIFFUCODER remains a viable drafter for the QWEN-2.5-14B-INSTRUCT, outperforming EAGLE-2 in nearly all settings, the weaker diffusion drafter DIFFULLAMA is unable to consistently outperform EAGLE-2. With the diffusion draft model being over $20\times$ larger than EAGLE/EAGLE-2 drafters, the drafting latency is no longer competitive, with the higher tokens per draft being the primary factor leading to *SpecDiff-2's* superior performance on select tasks.

A.1 Llama Model Temperature Ablation

Supplementing the drafter temperature analysis provided in the main text (Figure 5), similar analysis is provided for the DIFFULLAMA draft model.

As shown in Figure 8, maintaining $\text{temperature} = 1.0$ results in the strongest performance when scaling the number of drafts.

A.2 Diffusion Steps Ablation

As mentioned in the main text, the number of diffusion steps is set to $T = 1$. As illustrated by Figure 9, speed-up quickly degrades as the number of diffusion steps increases. This is a byproduct of the draft latency increasing linearly with T, while the increase in accepted tokens per draft remains relatively stagnant.

A.3 Gamma Ablation

The experiments use $\gamma = 32$ for DIFFUCODER and $\gamma = 16$ for DIFFULLAMA. Figure 10 illustrates how different window sizes influence the overall speed-up for DIFFUCODER. Preliminary experiments revealed that the draft window size γ depends largely on the diffusion drafter. The window size is set as $\gamma = 32$ for DIFFUCODER and $\gamma = 16$ for DIFFULLAMA. These sizes effectively balance between small γ values which cannot accept long drafts and large γ values which lower draft quality, as ablated in Appendix B and established in previous work (Christopher et al., 2025).

Empirical observations suggest DIFFUCODER more effec-

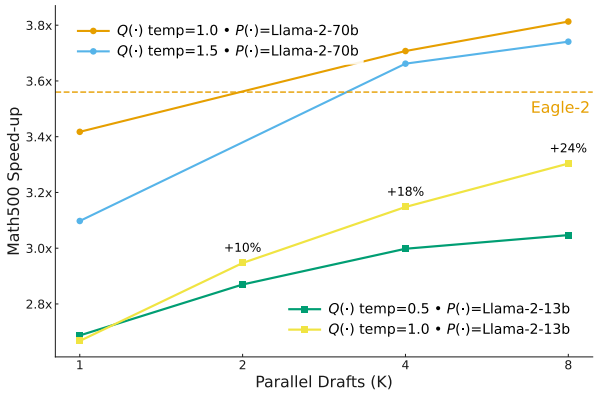


Figure 8. Train-time and test-time alignment methods for all model pairs. We show scaling in speed-up on Math500 prompts as test-time speed-up increases during self-selection acceptance.

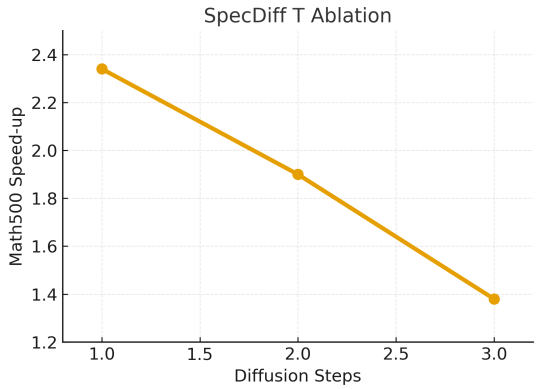


Figure 9. SpecDiff (with an unaligned drafter DIFFUCODER) ablated over increasing diffusion steps. Clear downward trend where acceptance vs latency trade-off is undesirable.

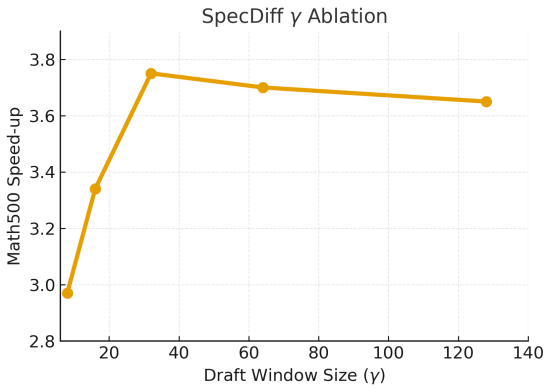


Figure 10. SpecDiff (unaligned drafter - DIFFUCODER) ablated over increasing diffusion window size (gamma). Degradation for large gamma, insufficient streak-acceptance for lower gamma.

		Math-500		HumanEval		GPQA		Mean	
Model	**Accelerator**	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$	**Speed-up**	$\frac{\text{Tokens}}{\text{Draft}}$
Temperature = 0									
Qwen-2.5-14B	EAGLE	2.59×	3.83 toks	2.57×	4.02 toks	1.94×	2.79 toks	2.37×	3.55 toks
	EAGLE-2	2.84×	4.25 toks	3.05×	4.89 toks	**2.29**×	3.46 toks	2.73×	4.20 toks
	SpecDiff-2.0	**3.65**×	**6.20 toks**	**6.17**×	**10.8 toks**	2.14×	**3.76 toks**	**3.98**×	**6.92 toks**
LLaMA-2-13B	EAGLE	3.43×	4.65 toks	3.47×	4.82 toks	2.83×	3.92 toks	3.24×	4.46 toks
	EAGLE-2	3.48×	4.83 toks	**3.92**×	5.63 toks	**3.04**×	**4.46 toks**	**3.48**×	4.97 toks
	SpecDiff-2.0	**3.61**×	**6.31 toks**	3.55×	**6.21 toks**	2.49×	4.37 toks	3.22×	**5.63 toks**
Temperature = 1									
Qwen-2.5-14B	EAGLE	2.43×	3.52 toks	2.72×	4.14 toks	1.94×	2.81 toks	2.36×	3.49 toks
	EAGLE-2	3.00×	4.32 toks	3.21×	5.03 toks	2.38×	3.53 toks	2.86×	4.29 toks
	SpecDiff-2.0	**3.88**×	**5.79 toks**	**3.83**×	**5.69 toks**	**2.53**×	**4.43 toks**	**3.41**×	**5.30 toks**
LLaMA-2-13B	EAGLE	3.39×	4.63 toks	3.30×	4.73 toks	2.77×	3.90 toks	3.15×	4.42 toks
	EAGLE-2	**3.43**×	4.77 toks	**3.85**×	**5.60 toks**	**3.14**×	**4.49 toks**	**3.47**×	4.95 toks
	SpecDiff-2.0	3.36×	**5.88 toks**	3.15×	5.52 toks	2.27×	3.98 toks	2.92×	**5.12 toks**

Table 7. Comparison on *Math-500*, *livecodebench*, and *MT-Bench* for four base models at temperatures 0 and 1. Each cell reports relative *Speed-up* and average acceptance length τ. Experiments utilize a single Ax100 GPU (80Gb).

tively handles large drafts than DIFFULLAMA, likely due to more extensive pretraining. DIFFUCODER *was trained on 130B tokens of specialized data, while DiffuLLaMA was trained on 60B tokens mostly from the web-text corpus SlimPajama.*

A.4 Joint Alignment Scaling

Supplementing the visualizations of train-time and test-time scaling, Figure 11 illustrates the joint impact of applying both alignment techniques. By combining streak-distillation and parallel drafting for test-time alignment, we see a $+39\%$ improvement over base diffusion drafter, and EAGLE-2 baseline, achieving $4.3\times$ throughput increase over QWEN2.5-72B model on Math500 data.

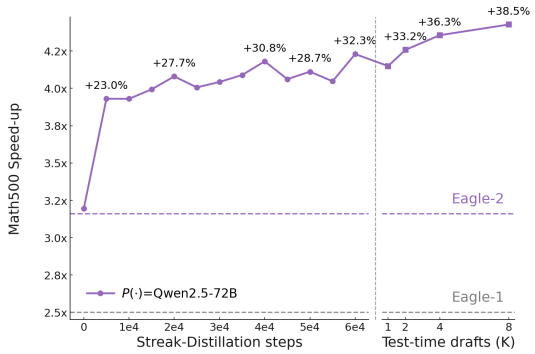


Figure 11. The test-time and train-time phases of *SpecDiff-2.0*.

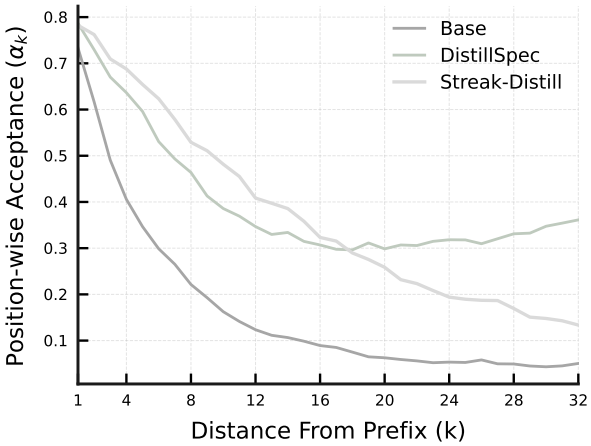
A.5 Streak-distillation versus DistillSpec

To further justify the neceesity of streak-distillation, the results in Figure 2 are supplemented with another variation of DistillSpec (Zhou et al., 2023). We refer to the baseline *DistillSpec* in Figure 12 as a naive adjustment of the *AR-Distillation* approach in Figure 2. Specifically, it extends the window to γ, matching a semi-autoregressive training approach often used for DLMs, where an average over all position-wise acceptances is used for alignment. However, it continues to operate invariant to positional dependencies, given that later tokens are over emphasized relative to their theoretic importance.

The results in Figure 12 (left) illustrate that this naive implementation results in significantly lower expected throughput, given mismatch from the theoretic definition of throughput. This results in the tangible gap between DistillSpec and streak-distillation reported throughputs, shown in Figure 12 (right).

B EXPERIMENTAL DETAILS

This appendix documents the hardware, software, model-loading pipeline, and evaluation protocol used in all speculative diffusion experiments. Unless stated otherwise, all numbers reported in the main text were obtained under the configuration described below.



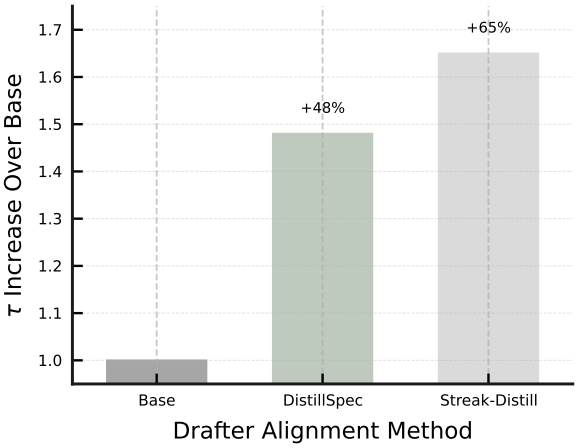


Figure 12. **Streak-distillation is compute-effective.** Streak-distillation (Streak-Distill above) compared with DistillSpec at optimizing alignment over the base model. Both algorithms ran for 40k training steps ($\sim$50 Gpu-Hours) on Math500 prompts, for Qwen2.5-14B-Instruct verifier. We see Streak-Distill out-preforms DistillSpec by $+35\%$ alignment/throughput increase (τ increase above) over the base model, illustrating the weighting issues of naive DistillSpec, and the advantages of the theoretically motivated Streak-Distill.

B.1 Hardware Setup

All experiments were executed on a single NVIDIA A100 GPU with 80 GB (two GPUs for 70B+ experiemnts). The GPU was used for inference only; model assembly (including LoRA merging) was performed on CPU first, then the fully assembled drafter model was moved to CUDA *once* to avoid repeated device transfers and to limit allocator fragmentation. CPU nodes were standard x86_64 Linux servers with at least 256 GB system memory to accommodate temporary model objects during LoRA merge.

B.2 Software Environment

Experiments were run with:

- **Python**: 3.10
- **PyTorch**: 2.3+ (with CUDA enabled)
- **Transformers**: 4.46.x (for both drafter and verifier loading)
- **PEFT**: for loading and merging LoRA adapters
- **Datasets**: Hugging Face `datasets` for benchmark streaming (MATH-500, LiveCodeBench, GSM8K, MT-Bench)
- **Matplotlib**, **NumPy**: for lightweight timing/diagnostic plots emitted by the script

Mixed-precision and kernel configuration followed:

```
torch.backends.cuda.matmul.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
torch.set_float32_matmul_precision("high")
```

to enable tensor-core friendly execution on A100 while retaining numerical stability.

B.3 Models

Verifier. The verifier was loaded with `AutoModelForCausalLM.from_pretrained(...)` on GPU, using the same tokenizer and with sampling flags normalized to avoid configuration warnings (e.g. forcing `do_sample=True` and setting a small but positive temperature). The verifier was evaluated in `bfloat16` on GPU, with `max_memory={0: "75GiB", 1: "75GiB"}` passed to `from_pretrained` to make large-model loading explicit on multi-GPU machines.

Drafter. The drafter model was loaded through a CPU-first path:

1. detect whether `--model-id` is a *LoRA adapter dir*, a *merged full model*, or a *base model*, by checking for `adapter_config.json` and `config.json`;
2. if an adapter was provided (either as `--model-id` or as `--drafter-lora`), attach it to the base model on CPU;
3. if vocab sizes mismatch, resize the base embeddings to the adapter's expected vocab and re-attach;
4. call `merge_and_unload()` so that the final drafter is a single model object;
5. move the merged model to CUDA once, and cast to the target dtype (typically `bfloat16`).

This procedure ensures that all speculative runs (and their timing) are not polluted by runtime adapter merges.

B.4 Acceptance and Distribution Collection

A key component of the script is the `_DraftProbCollector`, which is attached as a hook to the drafter's generation loop. For each speculative pass, after the drafter emits a block of γ tokens:

1. we record the *positions* of newly generated tokens (the "tail" of the sequence);
2. we record the drafter's per-token probabilities (or log-probabilities) for those positions;
3. optionally, we record the *full* token distribution for each of the γ positions (not just the chosen token), in model dtype, to enable verifier-side reweighting.

On the verifier side, for each drafted token we compute the verifier distribution, compare it with the drafter distribution, and apply a greedy-style acceptance rule:

- if a uniform $r \sim \mathcal{U}(0, 1)$ is less than the verifier probability of the drafted token, the token is *accepted*;
- otherwise, the token is *replaced* by sampling from a re-normalized verifier distribution in which the rejected draft token is zeroed out.

This produces a per-pass *accepted streak* length; over the course of generation we log all streaks and report their mean tokens per draft.

B.5 Evaluation Protocol

For each benchmark example x, the script executes:

1. **Speculative run:** call `speculative_loop(...)` to generate up to `--max-new` tokens, logging:
   - wall-clock time for drafter forward(s);
   - wall-clock time for verifier scoring;
   - KV-cache truncation / advance time;
   - number of passes and per-pass accepted streaks;
   - final sequence (for manual auditing).
2. **Baseline run (optional):** call `verifier_only_generate(...)` on the *same* prompt with the same sampling parameters; this supplies an actual tokens-per-second baseline for the big model on that hardware.

Throughput is then reported as

$$\text{throughput} = \frac{\text{new tokens}}{\text{wall time (s)}},$$

and speculative speed-up is computed as the ratio of baseline throughput to speculative throughput on the same hardware and prompt.

B.6 Dataset Streaming

When `--benchmark` is set, problems are streamed directly from Hugging Face:

- **MATH-500:** `HuggingFaceH4/MATH-500`, split=`test`; each problem is wrapped as `"Problem: <text> \n Answer:"`.
- **LiveCodeBench:** `livecodebench`, `version_tag=release_latest`, including platform and starter code in the prompt.
- **MT-Bench:** `HuggingFaceH4/mt_bench_prompts`; only the first turn is used as a single-turn prompt.
- **GSM8K:** `openai/gsm8k`, split=`test`; questions are wrapped as `"Question: ... \n Answer:"`.

A global example limit is enforced via `--limit` to make runs tractable on single GPUs.

B.7 Datasets

Distillation datasets were split into three classes.

(1) Qwen 14B, and 72B corpus. Corpus of temperature 1.0 completions were generated on prompts from gsm8k, alpaca, and livecodebench. Samples were generated with temp=1.0 by verifiers for smoother distributions, and further generated up-to 32 completions per prompt for distributional coverage, across 128 prompts. Completions were mixed evenly at 33% per benchmark. The full corpus of completions were not observed during training, given i.e., distillation compute used was insufficient for a full epoch. Observed high generalization across different tasks. Precise mix appeared un-important.

(2) Llama 13b corpus. Filtered Llamatoloka/beemo and openbmb/UltraFeedback SFT datasets for filtered 13b completions. Maintained even 50, 50 mix. Specifically, *toloka/beemo* — A curated set of machine-generated responses with per-row model labels; ($\sim$2.19k rows in the default split. *openbmb/UltraFeedback* — Preference dataset with 64k prompts / 256k responses; We filtered to entries generated by LLaMA-2-13B-chat.

(3) Llama 70b corpus. Completions filtered from togethercomputer/llama-instruct for Llama-2-70b. *togethercomputer/llama-instruct* — 19,004 English instruction–response conversations in [INST]... [/INST] format.

Generally, we found distillation to be very robust to differing datasets, exhbiting strong transfer.

B.8 Test-time verification.

K distinct drafts may entail redundancy at different token positions. We construct a tree from drafted tokens, pass flattened block to verifier, with custom attention mask. Verifier outputs probability distributions over tokens, conditioned on respective paths. Outputs are cached, and evaluated in parallel. This multi-branch, tree-based test-time verification was implemented using standard PyTorch and Hugging Face tooling, together with a few lightweight utility layers, and was integrated into the same speculative-diffusion pipeline as described. We describe the relevant components below.

Core frameworks. All tensor-level operations (tree flattening, parent-index bookkeeping, mask construction, and batched logit gathering) were implemented in `PyTorch` (2.3+). We relied on PyTorch's native advanced indexing to map from *tree order* to *flattened order*, and we used batched `torch.gather` to pull out verifier probabilities corresponding to the drafted tokens at each depth. The verifier itself was loaded via `transformers.AutoModelForCausalLM` and executed in `bfloat16` on GPU, so the custom mask had to be materialized on the same device and dtype-compatible with the model's attention stack (i.e. `bool` or `int` mask, depending on the architecture).

Tokenizer and model plumbing. We kept using `transformers.AutoTokenizer` to ensure that the flattened sequence $\tilde{x}$ remained consistent with the underlying causal LM. Since the tree construction introduced non-contiguous branches, we stored an auxiliary Python-side structure:

```
List[Dict[str, Tensor]]
```

for each level of the tree, holding (i) token ids, (ii) branch ids, and (iii) parent indices. This allowed us to re-associate the model's output logits with the correct logical branch after the forward pass. Because we had already aligned tokenizer and embeddings to avoid vocab mismatches when loading LoRA or merged checkpoints, no extra resizing was needed at this stage.

Mask construction. The custom attention mask was built with pure tensor operations. We first created a lower-triangular causal mask of shape $T \times T$ (where $T = |\tilde{x}|$) using `torch.tril`, then selectively *zeroed out* the entries corresponding to cross-branch attention at the same depth. To do this efficiently, we precomputed, per depth, the flat indices of tokens belonging to each branch and used `torch.index_fill_` (or direct boolean assignment) to disable attention among sibling tokens. On models that expected an additive mask (e.g. causal LMs returning `attn_weights + mask`), we converted the boolean mask to the large negative form (e.g. -10^4 or -10^9) and broadcast it to the expected number of heads. This was passed to the model via the same keyword arguments the `transformers` model already supported (typically `attention_mask` or `attention_bias`, depending on the architecture).

KV-cache reuse. We reused the verifier's KV-cache between speculative passes in exactly the same manner as the single-branch setup: the prefix $x_{1:L}$ was scored once, and the resulting `past_key_values` object was stored on GPU. For the multi-branch case we only extended the cache along valid branch paths. This was straightforward because the flattened tree preserved topological order; we simply advanced the KV-cache in the same order in which we had flattened the tokens. Where the underlying model exposed a `.crop()` or similar method (as in our speculative loop), we invoked it to truncate the cache to the accepted depth.

Parallel acceptance and logging. After the batched forward, we obtained logits of shape (T, V), where V is the vocabulary size. We then gathered, for every drafted node, the probability of *its* proposed token and, in parallel, the full verifier distribution for Dirac/SDA variants. This was done with a single `torch.softmax` call (in `float32` for numerical safety) followed by masked sampling. Random draws for all nodes at a given depth were produced in batch using `torch.rand` on GPU, so accept/reject decisions could be applied with simple elementwise comparisons. All acceptance decisions, together with branch ids and depths, were written to the same JSONL logging utility we already used for the speculative loop, so later analysis could reconstruct which branches were accepted and at which depth.

Profiling utilities. To ensure that the tree flattening and mask construction did not dominate runtime, we added lightweight timing scopes (via `time.perf_counter()`) around:

1. tree build,
2. mask materialization,
3. verifier forward.

These were recorded in the per-run `timers` dictionary alongside the existing entries for "drafter," "verifier_score," and "kv_truncate," so that the overhead of multi-branch verification could be compared 1:1 with the single-branch speculative loop reported in the main experiments.

Optional optimizations. On runs where the flattened sequence became long (large K, large γ), we exploited:

- **in-place mask reuse**: for fixed K and γ, the mask pattern per depth was constant, so we cached it and only re-filled the branch-specific indices;
- **AMP / autocast**: where the model permitted it, we wrapped the verifier forward in `torch.autocast("cuda", dtype=torch.bfloat16)` to reduce memory pressure;
- **SDPA backends**: we kept the SDPA context from the original script to allow Flash / memory-efficient attention, controlled by the `--no-flash` flag.

Overall, this combination of (i) PyTorch tensor operations, (ii) Hugging Face causal LMs, (iii) explicit attention-mask construction, and (iv) the existing JSONL logging infrastructure made the multi-draft verification slot cleanly into the rest of the speculative decoding framework, while still letting us profile and report speed-ups in a reproducible manner.

C COMPLETE PROOFS

Theorem C.1 (SD-Invariance)**.** *Tokens sampled via* speculative sampling *from $p(x)$ and $q(x)$ are distributed identically to those sampled from $p(x)$.*

Proof. (As reported in (Leviathan et al., 2023)) Let α be the acceptance probability. Note that as $p'(x) = \text{norm}(\max(0, p(x) - q(x))) = \frac{p(x)-\min(q(x),p(x))}{\sum_{x'}(p(x')-\min(q(x'),p(x')))} = \frac{p(x)-\min(q(x),p(x))}{1-\alpha}$, the normalizing constant for the adjusted distribution $p'(x)$ is $1-\alpha$.

Now:

$$P(x = x') = P(\text{guess accepted}, x = x') + P(\text{guess rejected}, x = x')$$

Where:

$$P(\text{guess accepted}, x = x') = q(x') \min(1, \frac{p(x')}{q(x')}) \quad (11)$$
$$= \min(q(x'), p(x')) \quad (12)$$
$$P(\text{guess rejected}, x = x') = (1 - \alpha)p'(x') \quad (13)$$
$$= p(x') - \min(q(x'), p(x')) \quad (14)$$

And thus, overall we obtained the sought result:

$$P(x = x') = \min(p(x'), q(x')) + p(x') - \min(p(x'), q(x')) = p(x').$$

□

Theorem C.2 (Position-Wise Alpha Identity)**.** *Let position wise acceptance rates $\alpha_1, \ldots, \alpha_\gamma$ be exchangable (i.e., as is asumed for AR drafters), then maximizing $\frac{\textit{Tokens}}{\textit{Draft}}$ reduces to $\max[\alpha_1(\boldsymbol{s} \circ \boldsymbol{x}_1)]$ (acceptance directly proceeding prefix $\boldsymbol{s}$.*

Proof.

$$\begin{aligned}
\max[\sum_{m=1}^{\gamma} \prod_{j=1}^{m} \alpha_j(\boldsymbol{s} \circ \boldsymbol{x}_{1:j})] &= \max[\sum_{m=1}^{\gamma} \prod_{j=1}^{m} \alpha_j(\boldsymbol{s} \circ \boldsymbol{x}_{1:j})], \\
&= \max[\sum_{m=1}^{\gamma} \alpha_j(\boldsymbol{s} \circ \boldsymbol{x}_{1:j})^m], \\
&= \sum_{m=1}^{\gamma} \max[\alpha_j(\boldsymbol{s} \circ \boldsymbol{x}_{1:j})]^m, \\
&\Rightarrow \sum_{m=1}^{\gamma} \max[\alpha_j(\boldsymbol{s} \circ \boldsymbol{x}_{1:j})], \\
&\Rightarrow \max[\alpha_1(\boldsymbol{s} \circ \boldsymbol{x}_1)].
\end{aligned}$$

□

Theorem C.3 (Greedy-acceptance vs SD-acceptance)**.** *Let speculative-decoding acceptance rate, α_{SD}, and greedy-acceptance rate α_{greedy}, be defined as:*

$$\alpha_{SD} = 1 - TV(p(x), q(x)), \ \alpha_{greedy} = p(x) \cdot q(x) \quad (15)$$

for model posteriors p, q over next-token x. Then,

Proof.

$\rightarrow p(x)q(x) + (p(x) - q(x)) - 1 \leq 0,$

$\rightarrow p(x)q(x) \leq 1 - p(x) - q(x),$

$\rightarrow p(x)q(x) \leq 1 - \frac{1}{2}p(x) - \frac{1}{2}q(x),$

$\rightarrow \alpha_{\text{SD}} \leq \alpha_{\text{greedy}}.$

□

D MODEL COMPLETIONS

D.1 Qwen2.5-72B-Instruct

Sampled at temperature 0.0, at a maximum of 3096 tokens. ¡thinking¿ tokens omitted, reasoning formatted below, answers instructed to be in 'boxed{}' format. Prompts sampled from math500 corpus.

Problem 1: Algebra

Problem. Hillary has eleven coins, all dimes and nickels. In total, the coins are worth 75 cents. How many nickels does she have?

Solution. Let d represent the number of dimes and n represent the number of nickels.

$$d + n = 11,$$
$$10d + 5n = 75.$$

Dividing the second equation by 5 gives $2d+n = 15$. Subtracting the first equation from this, we obtain $d = 4$. Substitute back: $4 + n = 11 \implies n = 7$.

Answer. $\boxed{7}$

Problem 2: Intermediate Algebra

Problem. Find the maximum value of

$$f(x, y) = x\sqrt{1 - y^2} + y\sqrt{1 - x^2}, \quad -1 \leq x, y \leq 1.$$

Solution. Let $x = \cos\theta$, $y = \cos\phi$. Then $\sqrt{1 - y^2} = \sin\phi$ and $\sqrt{1 - x^2} = \sin\theta$:

$$f = \cos\theta \sin\phi + \cos\phi \sin\theta = \sin(\theta + \phi).$$

The maximum of $\sin(\theta + \phi)$ is 1, which is attainable (e.g., $x = 1$, $y = 0$).

Answer. $\boxed{1}$

Problem 3: Algebra

Problem. For how many values of x is the expression

$$\frac{x - 5}{x^2 - 9}$$

not defined?

Solution. The denominator is zero when $x^2 - 9 = 0 \Rightarrow (x - 3)(x + 3) = 0$. Thus $x = 3$ or $x = -3$.

Answer. 2

Problem 4: Geometry

Problem. Altitudes $\overline{AD}$ and $\overline{BE}$ of $\triangle ABC$ intersect at H. If $\angle BAC = 54^\circ$ and $\angle ABC = 52^\circ$, find $\angle AHB$.

Solution.

$$\angle ACB = 180^\circ - 54^\circ - 52^\circ = 74^\circ.$$

In $\triangle ABC$, $\angle AHB = 180^\circ - \angle ACB = 106^\circ$.

Answer. $\boxed{106^\circ}$

Problem 5: Precalculus

Problem. Two parallel lines are defined by

$$\begin{pmatrix} 3 \\ -10 \\ 1 \end{pmatrix} + t \begin{pmatrix} 2 \\ -9 \\ -2 \end{pmatrix}, \quad \begin{pmatrix} -5 \\ -3 \\ 6 \end{pmatrix} + u \begin{pmatrix} 4 \\ -18 \\ -4 \end{pmatrix}.$$

Find the distance between the lines.

Solution. Direction vectors are proportional: $\mathbf{b}_2 = 2\mathbf{b}_1$, confirming parallelism. For $\mathbf{a}_1 = (3, -10, 1)$, $\mathbf{a}_2 = (-5, -3, 6)$, and $\mathbf{b}_1 = (2, -9, -2)$,

$$d = \frac{\|(\mathbf{a}_2 - \mathbf{a}_1) \times \mathbf{b}_1\|}{\|\mathbf{b}_1\|}.$$

Compute:

$$(\mathbf{a}_2 - \mathbf{a}_1) \times \mathbf{b}_1 = (31, -6, 58), \quad \|\mathbf{b}_1\| = \sqrt{89}.$$

Hence

$$d = \frac{\sqrt{4361}}{\sqrt{89}} = \sqrt{49} = 7.$$

Answer. $\boxed{7}$

Problem 6: Precalculus

Problem. Two runners A and B start at point O and run in the same direction. Runner B runs three times as fast as A. An observer at point P (with $\overline{OP}$ perpendicular to the track) observes angle $\angle APB$. Find the maximum of $\angle APB$ (in degrees).

Solution. At time t, $A(vt, 0)$, $B(3vt, 0)$, $P(0, d)$. The angle satisfies

$$\theta = \arctan\left(\frac{d}{vt}\right) - \arctan\left(\frac{d}{3vt}\right).$$

Maximizing over $t > 0$ yields $\theta_{\max} = 30^\circ$.

Answer. $\boxed{30^\circ}$

Problem 7: Counting and Probability

Problem. In how many ways can 5 students be selected from a group of 6?

Solution.

$$\binom{6}{5} = \frac{6!}{5!(1!)} = 6.$$

Answer. $\boxed{6}$

Problem 8: Counting and Probability

Problem. A prodigy wishes to arrange 2 indistinguishable IMO and 2 indistinguishable IPhO gold medals in a row. How many distinct arrangements are possible?

Solution.

$$\text{Distinct permutations} = \frac{4!}{2!2!} = 6.$$

Answer. $\boxed{6}$

Problem 9: Number Theory

Problem. If Rosa's age leaves remainder 1 when divided by 2, 3, 4, or 6, and remainder 0 when divided by 7, and she is younger than 75, how old is she?

Solution. From the first set: $R \equiv 1 \pmod{12}$. Also $R \equiv 0 \pmod 7$. Solving $12k + 1 \equiv 0 \pmod 7 \Rightarrow k \equiv 4 \pmod 7$. Thus $R = 12(7m+4)+1 = 84m+49 < 75 \Rightarrow m = 0$. Hence $R = 49$.

Answer. $\boxed{49}$