FAIL FAST, WIN BIG: RETHINKING THE DRAFTING STRATEGY IN SPECULATIVE DECODING VIA DIFFUSION LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding efforts have largely optimized draft quality for enhanced performance, treating policy as a secondary focus. While effective, this approach can be costly and brittle, as speedups depend on continually improving drafter quality. We argue that *drafting policy* should be a first-class driver of performance. Our key insight is that intrinsic properties of emerging diffusion LLMs (dLLMs) make this shift both possible and effective. Their accuracy improves sublinearly with compute, yielding diminishing returns to quality, and unlike with autoregressive models, drafting cost is nearly independent of length, enabling long drafts at low marginal cost. Together, these properties favor policies that hedge against draft quality – dynamically allocating compute across the sequence based on heterogeneous decoding difficulties across the sequence – rather than attempting to maximize it. We present FailFast, a speculative decoding framework that operationalizes this perspective. FailFast allocates minimal compute to hard regions ("fail fast") while aggressively extending drafts in easy regions ("win big"), often accepting tens of tokens per step. Without any fine-tuning, FailFast provides lossless acceleration of autoregressive LLMs, achieving up to 4.9× speedup over standard decoding, 1.7× over a naive dLLM drafter baseline, and 1.7× over EAGLE-3 across diverse models and workloads.

1 INTRODUCTION

Speculative decoding (Leviathan et al., 2023) has become a leading approach for accelerating large language model inference, with much of its progress driven by improvements in draft quality via better-trained drafters (Li et al., 2025), stronger target conditioning (Li et al., 2025; Cai et al., 2024), or ensemble drafting (Cai et al., 2024; Miao et al., 2024). Though effective, this quality-centric trajectory increasingly ties performance to draft quality improvements, making speedups dependent on continual investment in drafter capability. As model releases accelerate and deployment settings diversify, this can significantly limit the portability of speculative decoding strategies.

In this work, we revisit this design approach and argue that performance can instead be effectively driven by *drafting policy*, a shift enabled by intrinsic properties of emerging Diffusion Large Language Models (dLLMs) (Khanna et al., 2025; Song et al., 2025; gem, 2025; Bie et al., 2025; Wu et al., 2025b). dLLMs have been primarily studied as standalone generators, where parallel decoding enables low-latency inference but introduces a fundamental tension between speed and quality: simultaneous token sampling assumes conditional independence and ignores mutual dependencies (Wu et al., 2025b; Kang et al., 2025), so generating more tokens per step exacerbates quality degradation, while maximizing quality forces a fallback to left-to-right decoding at autoregressive speeds. Existing work attempts to alleviate this tradeoff for standalone generation (Kang et al., 2025; Bie et al., 2025; Qian et al., 2026). We instead study dLLMs as drafters in speculative decoding with autoregressive verifiers – a setting that sidesteps this tension (verification catches errors, so draft quality need not be perfect) and turns it into an opportunity.

We identify two fundamental properties of dLLMs that enable this shift. First, **dLLM drafting cost is nearly independent of draft length**. Whereas latency for autoregressive LLM inference scales with output length (Yan et al., 2024; Agrawal et al., 2024), dLLMs can unmask multiple tokens per forward pass, making latency primarily a function of forward passes rather than tokens generated.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

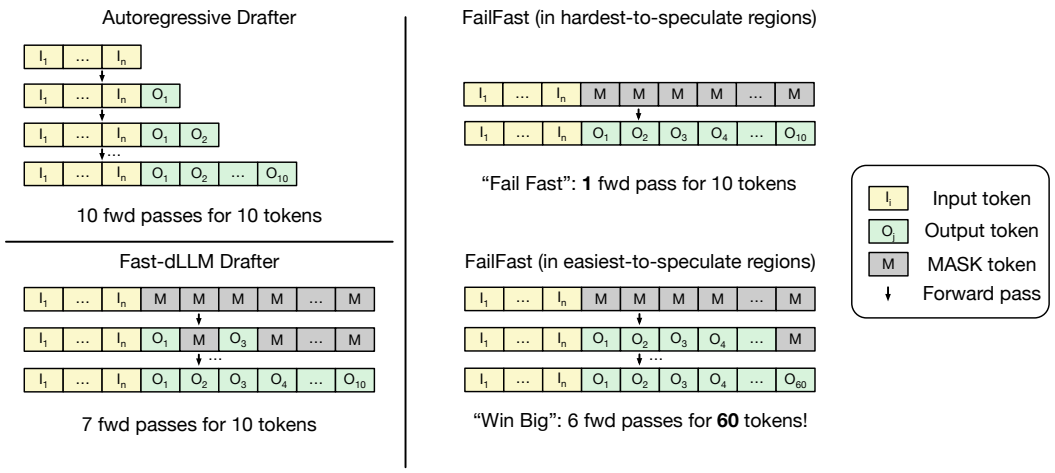


Figure 1: Behavior of FailFast vs. baseline drafters per speculation round. The AR drafter uses 10 forward passes for 10 tokens; the naive dLLM drafter uses confidence-aware parallel decoding to reach similar quality in ¡10 passes. FailFast dynamically adjusts speculation length based on token confidence: in harder regions (top right), it shortens the draft to "fail fast"; in easier regions (bottom right), it extends aggressively to "win big". In practice, FailFast navigates a continuous spectrum between these extremes based on decoding difficulty.

This substantially reshapes the central tradeoff in speculative decoding: balancing longer accepted drafts against exponentially decreasing acceptance probability (Liu et al., 2024b; Huang et al., 2024). With dLLM drafters, long drafts are no longer prohibitively expensive, enabling regimes inaccessible to autoregressive drafting.

Second, **dLLM accuracy improves sublinearly with compute, yielding diminishing returns**. At the sequence level, each additional forward pass produces progressively smaller accuracy gains. This concavity relaxes the latency-accuracy tradeoff: while additional compute is important for standalone generation, where each token affects end-to-end quality, it is often unnecessary for drafting, where all outputs are verified and many regions are inherently easy. In these regions, even minimal compute yields sufficiently accurate drafts, while in harder regions additional compute provides limited benefit due to intrinsic modeling constraints.

Together, these properties shift how useful compute is spent. Quality investments yield bounded returns due to diminishing gains in dLLM accuracy, while compute allocated to *how* drafts are produced – long in easy regions, minimal in hard ones – remains effective because it adapts to variation in difficulty across tokens. This naturally favors policies that hedge against draft quality rather than chase it, dynamically adapting computation based on per-token difficulty (Pan et al., 2025a; Liu et al., 2024b; Huang et al., 2024). Such a stance is impractical for autoregressive drafters, where long low-quality drafts are prohibitively expensive, but becomes both feasible and highly effective with dLLM drafters.

We present FailFast, a training-free speculative decoding framework that operationalizes this stance through two complementary hedges. First, we ***fail fast*** by using a single denoising pass to generate drafts, avoiding additional compute on hard-to-predict tokens while empirically maintaining high accuracy in easy regions. Second, in easy regions where minimal-compute drafts are reliably accepted, we ***win big*** by aggressively extending speculation length – in many cases speculating up to 70 tokens in a single round and having them accepted – reducing verification overhead by amortizing repeated drafter-verifier interactions. We use drafter confidence as a signal of region difficulty, and show that it strongly correlates with token acceptance.

Across diverse models and workloads, FailFast delivers lossless acceleration of autoregressive LLM inference, achieving up to 4.9× speedup over vanilla decoding, 1.7× over a naive dLLM drafter baseline, and 1.7× over EAGLE-3—all without fine-tuning the dLLM drafter. These gains arise purely from drafting policy applied atop off-the-shelf aligned dLLMs (Wu et al., 2025a; Bie et al., 2025; Gong et al., 2024; Zhou et al., 2025), demonstrating that policy design alone can yield substantial speedups without the per-target drafter training required by quality-centric approaches. Because FailFast depends only on intrinsic dLLM properties rather than model-specific retraining, it naturally benefits from continued improvements in dLLM quality and alignment (Sandler et al., 2025).

2 BACKGROUND

Diffusion language models. Recent research has established Diffusion Language Models (dLLMs) as a promising alternative to standard autoregressive generation, with examples spanning both open-source (Nie et al., 2025; Bie et al., 2025; Ye et al., 2025) and closed-source models (Khanna et al., 2025; gem, 2025; Song et al., 2025). Architecturally, dLLMs retain the standard Transformer backbone, preserving access to familiar metrics such as log-probabilities and token confidence scores. However, they diverge fundamentally in their decoding mechanism: instead of generating tokens strictly left-to-right, dLLMs generate text through an iterative *unmasking* process. This decoding is typically semi-autoregressive (Arriola et al., 2025; gem, 2025; Wu et al., 2025a;b): the output sequence is divided into blocks, with inter-block attention remaining causal (later blocks attend to earlier blocks), but intra-block attention is bidirectional. (This semi-autoregressive decoding also enables optimizations like KV caching (Wu et al., 2025b).) Within each block, tokens are unmasked in a non-deterministic order determined by token confidence rather than position, allowing the model to unmask "easy" tokens before "hard" ones. To further improve efficiency, recent works have proposed acceleration techniques such as approximate KV caching (Wu et al., 2025b; Ma et al., 2025; Liu et al., 2025b), confidence-aware parallel decoding (Wu et al., 2025a;b), and self-speculative decoding (Agrawal et al., 2025).

Speculative decoding. Borrowing from classical computer architecture principles (Burton, 1985), speculative decoding has emerged as a standard technique to alleviate the memory-bound nature (Agrawal et al., 2024) of LLM inference (Stern et al., 2018; Leviathan et al., 2023; Yan et al., 2024; Liu et al., 2025a). The process operates on a propose-and-verify cycle: in each round, a lightweight drafter first generates a draft, which is subsequently validated by the target model. The speculation phase prioritizes efficiency, relying on methods such as a standalone draft model (Leviathan et al., 2023; Miao et al., 2024), a trainable module on top of the target model (Cai et al., 2024; Li et al., 2025), a tree-based token cache (Oliaro et al., 2024; Luo et al., 2024; Zhao et al., 2024), an n-gram lookup table (Fu et al., 2024), or a retrieval-based datastore (He et al., 2023). In the verification phase, the target model performs a parallel chunked-prefill over these candidates, which usually consists of either a single sequence of tokens as in (Leviathan et al., 2023) or tree-like structures to further boost the quality of speculation (Miao et al., 2024; Cai et al., 2024; Li et al., 2025; Chen et al., 2024), and accepts the longest valid prefix. Consequently, the speculation length n is typically conservative (e.g., $n = 3$-10) (Liu et al., 2024b; Fu et al., 2024) to maintain an optimal balance between the speculation overhead and the rate of token acceptance.

3 MOTIVATION: DLLM PROPERTIES THAT ENABLE POLICY-DRIVEN SPECULATION

We now present an empirical foundation for our argument to shift to policy-centric speculation.

3.1 DLLMS' CONCAVITY OF ACCURACY IMPROVEMENTS

The quality of a dLLM's output is a function of compute. As non-autoregressive models, dLLMs generate multiple tokens in parallel during an iterative denoising process. In each forward pass, a select few masked tokens with higher confidence within the full sequence of regular and mask tokens are unmasked into regular tokens. As such, the computational cost of a dLLM is predominantly determined by the number of denoising steps (forward passes), not the number of tokens generated. By increasing the number of denoising steps, the model iteratively refines the output, trading latency for higher quality/fidelity.

dLLMs can outperform AR LLMs in generation speed while maintaining accuracy by spending fewer forward passes to generate the same number of output tokens (Wu et al., 2025b). However, naively deploying them as if they are equally-accurate but faster AR drafters in speculative decoding does not capitalize on their full latency benefit. Crucially, we find that dLLMs show concavity in accuracy wins as more compute is spent – improvements with each new forward pass have diminishing returns. In Fig. 2, we naively adopt a dLLM as the drafter and show that a higher compute budget leads to diminishing returns in overall acceptance rate. A dLLM can, in theory, unmask a sequence of unlimited length using a single forward pass. Although the quality of such one-step generation is relatively low and more denoising steps improve the quality – and those denoising steps are crucial

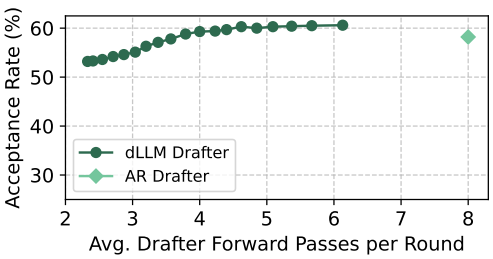


Figure 2: [Qwen2.5-32B, MATH] dLLMs' concavity of accuracy improvements when speculating 8 tokens/round. Doing 2.6× more drafter forward passes only increased the acceptance rate by 7.3%.

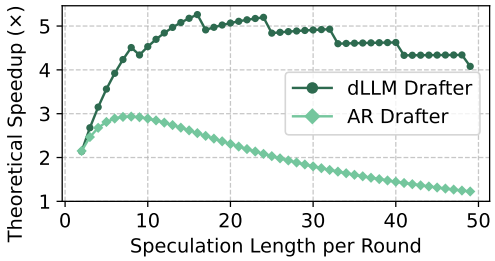


Figure 3: [Qwen2.5-32B, MATH] A semi-autoregressive dLLM that performs one-step generation achieves a higher theoretical speedup than that of an autoregressive drafter.[1]

for quality if the dLLM is doing standalone generation – the role of draft models in speculative decoding is, by definition, to correctly decode the easier tokens, which is a nice match for the fast yet inaccurate one-step generation.

In Fig. 3, we analyze the theoretical speedup (Theorem 3.8 in (Leviathan et al., 2023)) of speculative decoding over vanilla decoding using different drafters of the same size. The autoregressive drafter generates one token in each forward pass, incurring drafting latency linear in the draft length, whereas the semi-autoregressive dLLM drafter performs one-step generation and generates 8 tokens in each forward pass (Wu et al., 2025a), resulting in sublinear drafting latency. Even though longer speculation lengths exponentially reduce the acceptance probabilities of the tokens later in the draft, the dLLM's one-step generation compensates by dramatically lowering the average latency to generate each draft token, increasing the maximum speedup from 3.0× (at speculation length 8) to 5.2× (at length 16).

3.2 The Varying Decoding Difficulty within Sequences

Importantly, even though the concavity property manifests across entire sequences, within each sequence, there are easier regions and tokens where minimal compute suffices for high-quality generation. Similar to prior work (Liu et al., 2024a; Huang et al., 2024; Pan et al., 2025a), we observe that in natural language generation, the difficulty of generation and speculation differs between regions of the output sequence. Easier regions often consist of syntactic copying, summarization of prior context and input prompt, formulaic enumerations, simple arithmetic, etc. In these regions of lower difficulty, a capable draft model can achieve a near-perfect acceptance rate. Conversely, harder regions involve complex planning, multi-step reasoning, or knowledge retrieval where the draft model's capacity is insufficient, leading to frequent rejections. In Fig. 4, we pick five random queries from one of our evaluation datasets, run an AR draft model, and classify tokens in the output sequence into "easier" tokens (including both correct speculations and the "bonus tokens" when all drafted tokens are accepted) and "harder" tokens (rejected by the target model). Further, in Fig. 5, we report the fraction of tokens that belong to a consecutive block of more than X easy tokens across different tasks. If an easy region is defined to be a consecutive block of $>$10 "easy" tokens, 50-70% of tokens belong to an easy region. Moreover, 15-26% of tokens belong to an easy region of $>$50 tokens.

We emphasize that this easy/hard heterogeneity is a property of the generation workload rather than of any particular drafter architecture. The role of the drafter is instead to determine how efficiently this heterogeneity can be exploited: as we discuss next, dLLMs expose a more flexible compute-length tradeoff than AR drafters, which typically utilize a static speculation length (e.g., $n = 10$ tokens). While this aims to balance speculation overhead and potential speedup, it is suboptimal for the dynamic nature of generation:

- **In easier regions** (e.g., tokens 150-380 for query 4 in Fig. 4): A short, fixed speculation length results in undue verification overhead. Even though the drafter could correctly speculate the next 10 tokens, its token generation is forced to pause every 10 tokens to invoke the target model for verification. Since target model inference is memory-bound, the frequent loading of model weights incurs significant latency overhead, preventing the system from reaching peak overall efficiency.
- **In harder regions** (e.g., tokens 200-250 for query 18 in Fig. 4): A fixed speculation length results in undue speculation overhead. The drafter wastes compute generating 10 tokens, most of which are likely to be rejected (after the first or second position), resulting in wasted computation and hurting the end-to-end speedup.

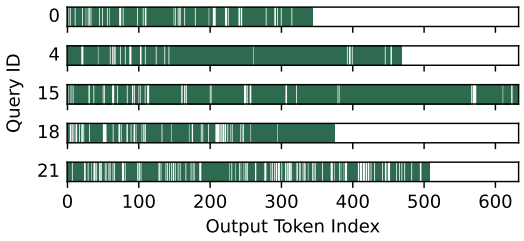


Figure 4: [Qwen2.5-32B, MATH] Varying decoding difficulty within sequences. Each cell shows whether a token is accepted (green, "easier") or rejected (white, "harder"); all-white trailing cells indicate a shorter sequence.

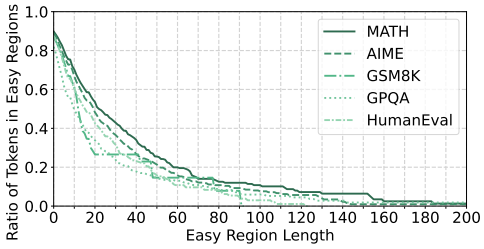


Figure 5: [Qwen2.5-32B] A large fraction of tokens belong to a consecutive block of more than X "easy" tokens.

Altogether, these properties provide a unique opportunity to change the way drafters are used in speculative decoding. To exploit varying decoding difficulty, an ideal speculative decoding paradigm would dynamically increase the speculation length in easy regions to reduce the frequency of verification and amortize its cost, while minimizing the drafter's compute in hard regions to minimize wasted effort. Standard autoregressive (AR) draft models are ill-suited for this dynamic strategy – each AR drafter's forward pass only generates a single token, which makes large speculation lengths (e.g., speculating $>$30 tokens at a time) prohibitively expensive and risky (Fig. 3), since the probability of each token's acceptance drops exponentially, and the drafting latency eventually overshadows the savings from parallel verification. In contrast, dLLM drafters expose a more favorable control surface. By generating multiple tokens in parallel and allowing the compute budget per draft to be adjusted, they can cheaply produce lengthy drafts in easy regions while spending less compute when the workload becomes hard.

4 Method

We now present FailFast, a training-free framework that operationalizes the policy-driven stance argued for in Sec. 1&3. FailFast realizes the stance through two complementary hedges – fail fast and win big – that together exploit the dLLM properties identified above to dynamically allocate compute based on per-token difficulty.

4.1 "Fail Fast": Embracing Error-Proneness for Faster Speculation

There has been a myriad of related work on improving the efficacy of speculative decoding by improving the draft acceptance rate. Typically, this is achieved by enhancing the drafted tokens' quality through using an ensemble of small drafters to construct token trees (Miao et al., 2024; Cai et al., 2024; Li et al., 2025; Chen et al., 2024), extensive fine-tuning of the draft model (Liu et al., 2023; atl, 2025; Sandler et al., 2025), performing online drafter selection (Liu et al., 2025a), etc.

In our work, we propose a counter-intuitive approach: **we explicitly embrace the error-proneness of drafters in speculative decoding**. Instead of refining the draft output to match the target distribution better, we restrict the dLLM to the absolute minimum computational budget – typically only using one denoising step (model forward pass). Our rationale is twofold:

- **Easy regions:** In regions of low difficulty (e.g., summarization), even a coarse, one-step generation from a small dLLM is often sufficient to speculate the correct tokens (Fig. 2). Additional refinement steps in these regions are wasted compute.
- **Hard regions:** In regions of high difficulty (e.g., complex reasoning), a small draft model is statistically likely to diverge from the target model regardless of how many refinement steps are applied. Although spending additional compute to refine the draft tokens does indeed improve the acceptance rate, it yields diminishing returns (Fig. 2): the quality of speculated tokens is still inherently bounded by the capability of the small drafter, and a significant portion of tokens is eventually rejected anyway.

By adopting dLLMs as draft models, we relax the requirement for high-quality language modeling compared to using them as standalone models, allowing us to operate in a low-quality but blazing-fast mode that minimizes speculation latency per token. The verification stage then catches errors, enabling a "fail fast" approach that quickly returns the task of decoding to the target model with minimal speculation overhead when the task becomes difficult.

4.2 "WIN BIG": DYNAMIC EXPANSION OF SPECULATION LENGTH FOR FASTER VERIFICATION

While reducing speculation latency addresses the cost of generating drafts, it does not lower the verification latency – the overhead incurred by executing the large target model, which is a factor of both the number of verifications (rounds of speculation) and the verification latency in each round. If we strictly use short speculation lengths, the target model is invoked frequently, creating a latency bottleneck.

To mitigate this, FailFast capitalizes on "easy" segments by opportunistically increasing the speculation length. When the draft model encounters a subsequence that is easier to speculate, we defer verification and aggressively extend the length of the speculated draft. Because speculation has, by design, such a low cost in FailFast, the risk of "failing" is low, and we can afford these aggressive speculations.

Confidence as a difficulty signal. To detect these easy regions without ground-truth knowledge of difficulty, we utilize the dLLM's internal confidence (Fu et al., 2025; Dai et al., 2024) as a proxy for speculation difficulty. Although dLLMs are non-autoregressive, they use the same Transformers backbone as AR LLMs, so we still have access to the probability distribution over the vocabulary for each token. For a dLLM predicting a token x_i at position i, we define its confidence C_i as the maximum probability in the output distribution P_i: $C_i = \max_{v \in V} P_i(x_i = v)$, where V is the vocabulary. Even though LLM confidence suffers from miscalibration issues when used to signal final answer correctness (Chhikara, 2025; Geng et al., 2024), we instead use confidence as a signal for whether the draft model matches the target model's token distribution (similar to prior work in speculative decoding, i.e., EAGLE-2 (Li et al., 2024b)). Empirically, Fig. 6 shows a strong correlation between low confidence and the likelihood of rejection. We further show that this behavior generalizes in Appendix D.3.

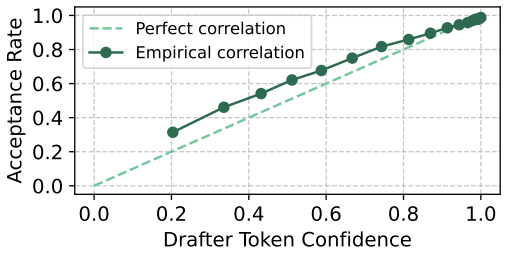


Figure 6: Draft model confidence strongly correlates with token acceptance (i.e., "easiness").

Dynamic speculation length expansion logic. Our adaptive strategy is detailed in Algorithm 1. We begin with a default speculation length N. After generating these initial tokens (using minimal compute), we inspect their confidence scores. If all tokens' confidence in the current speculated sequence exceeds a confidence threshold τ (indicating an "easy" region), we assume the draft will likely be accepted. Instead of stopping to verify, we immediately extend the speculation length by another N tokens and invoke the drafter again. We repeat this extension process until a low-confidence token is detected or a maximum sequence length $N_{\max}$ is reached. This mechanism allows FailFast to generate massive chunks of tokens (e.g., 70 tokens) in easy regions in a single round of speculation, and to get a majority of them accepted in many cases, which drastically reduces the number of times the target model needs to be invoked (verification rounds) – the verification overhead in each round stays the same because short prefills are memory-bound (Agrawal et al., 2024) – while reverting to short bursts in harder regions.

Algorithm 1: FailFast Main Logic in Each Round

Hyperparameters: step size N, confidence threshold τ, max length $N_{\max}$
$L \leftarrow 0$ {Num. speculated tokens in this round}
while true **do**
  $L \leftarrow L + N$
  Speculate the next N tokens.
  if any speculated token has confidence $C_i < \tau$ **or** $L \geq N_{\max}$ **then**
    break
  end if
end while
Submit all L tokens for verification.

While the notion of dynamically adjusting the draft length is an established concept in speculative decoding (Liu et al., 2024b), the resulting speedups are often limited, hindered by the risk of invoking undue speculation latency. For example, SpecDec++ (Huang et al., 2024) achieves an average speedup of $\sim$10% over its baselines. In comparison, FailFast overcomes this by using minimal compute to

minimize the risk of incorrect speculation, allowing it to drastically adapt the speculation length on-the-fly. We include a discussion of alternative design choices and optimizations in Appendix F.

4.3 Policy Design vs. Training-Based Approaches

Our work focuses specifically on **drafting policy as a first-class lever**, rather than improving performance through additional training of the drafter. Indeed, most state-of-the-art dLLMs are adapted from pretrained autoregressive (AR) weights (Wu et al., 2025a; Bie et al., 2025; Gong et al., 2024; Zhou et al., 2025) and are therefore already implicitly aligned with the target AR models when used as drafters. Leveraging this property, we study how to better exploit dLLMs at inference time via policy design. That said, our approach is complementary to prior optimizations that aim to improve the overall acceptance rate of drafted tokens (Miao et al., 2024; Chen et al., 2024; Liu et al., 2023; atl, 2025; Liu et al., 2025a), including recent work that fine-tunes the dLLM drafter for improved alignment and higher acceptance rates (Christopher et al., 2025; Sandler et al., 2025). These techniques are *complementary* to FailFast, as the dLLMs we employ can be further fine-tuned to fit the base model's distribution better. Our claim is not that policy-driven design uniformly outperforms quality-centric design on raw speedup, but that it delivers substantial gains without per-target training. We discuss additional related work in Section H.

5 Evaluation

5.1 Setup

Models and Baselines. We evaluate FailFast across diverse models and workloads to demonstrate that policy-driven speculation can deliver substantial speedups without per-target drafter training. In our main experiments, we evaluate three target models of different sizes: `Qwen2.5-{32B,14B,7B}-Instruct` (Yang et al., 2024). To demonstrate cross-model generalization, we also evaluate on LLaMA-based draft/target models (Gong et al., 2024). The different speculative decoding schemes we evaluate are listed below. For all schemes, we set the generation temperature of all models to 0 and set the maximum number of output tokens to 1024. We evaluate stochastic decoding in Appendix D.2 and discuss more details of our best-effort reproduction of all baselines and additional baselines in Appendix E.

- **AR drafter (Leviathan et al., 2023):** We adopt `Qwen2.5-1.5B-Instruct` as our autoregressive speculative sampling baseline, which uses the same training data as Qwen2.5-32B. For each dataset, we conduct an extensive parameter sweep (n = 3-20) and pick the speculation length that achieves the best speedup.
- **Fast-dLLM (Wu et al., 2025a):** For the dLLM drafter baseline, we employ `Fast_dLLM_v2_1.5B` (Wu et al., 2025a), the state-of-the-art semi-autoregressive diffusion LLM that embeds efficiency optimizations such as approximate caching and confidence-aware parallel decoding. We adopt its default hyperparameters that achieve the best balance between accuracy and generation speed. This baseline represents naively plugging in a dLLM as the draft model without the additional optimizations of FailFast.
- **EAGLE-3 (Li et al., 2025):** EAGLE-3 is one of the most widely deployed state-of-the-art speculative decoding drafters. As a lightweight single-layer autoregressive Transformer, EAGLE-3 introduces a multi-layer feature fusion and a training-time test mechanism for improving drafting quality. Different from other strategies where a single chain of draft tokens is proposed for verification, EAGLE-3 leverages a draft-tree-based mechanism that improves the overall token acceptance rate. We enable the draft tree in all experiments as its speedup is higher than without it.
- **FailFast:** FailFast employs `Fast_dLLM_v2_1.5B` as the underlying model and performs one-step generation and dynamic speculation length expansion. For the hyperparameters in Alg. 1, we set $\tau = \{0.4, 0.45, 0.5\}$ for the {32,14,7}B target model and $N = 10$. Although picking specific hyperparameters for each dataset/model combination yields higher speedups, we use the same set of hyperparameters across datasets to demonstrate generalizability. FailFast's performance degrades gracefully as hyperparameters are tweaked (Appendix C).

Datasets. Similar to prior work (Wu et al., 2025a; Sandler et al., 2025; Li et al., 2025), we evaluate FailFast on a wide range of diverse benchmarks: MATH, AIME, and GSM8K (Hendrycks et al., 2021; aim, 2025; Cobbe et al., 2021) for mathematical reasoning, GPQA (Rein et al., 2024) for knowledge-intensive question answering, and HumanEval (Chen, 2021) for code generation.

Table 1: Comparison of average (±standard deviation) speedups of different drafting methods over vanilla decoding. FailFast achieves the **highest** average speedup.

Drafting Method	Dataset					Average
	MATH	AIME	GSM8K	GPQA	HumanEval	
Target Model: Qwen2.5-32B-Instruct						
AR Draft Model	2.93±0.48×	2.84±0.48×	2.82±0.42×	2.26±0.61×	2.72±0.34×	2.71±0.47×
Fast-dLLM	3.57±0.71×	3.29±0.61×	3.10±0.39×	2.42±0.50×	3.16±0.30×	3.11±0.50×
EAGLE-3	2.87±0.51×	2.85±0.41×	2.59±0.28×	2.42±0.25×	3.31±0.27×	2.81±0.34×
FailFast (ours)	**4.90**±0.92×	**4.40**±0.98×	**3.71**±0.48×	**3.11**±0.94×	**4.06**±0.58×	**4.04**±0.78×
Target Model: Qwen2.5-14B-Instruct						
AR Draft Model	2.05±0.35×	1.94±0.34×	1.95±0.27×	1.71±0.35×	1.91±0.20×	1.91±0.30×
Fast-dLLM	2.57±0.54×	2.22±0.39×	2.22±0.28×	1.86±0.29×	2.23±0.15×	2.22±0.33×
EAGLE-3	2.48±0.66×	2.38±0.38×	2.23±0.21×	2.04±0.31×	2.68±0.40×	2.36±0.40×
FailFast (ours)	**3.92**±1.17×	**3.37**±0.91×	**3.04**±0.53×	**2.54**±0.58×	**3.41**±0.51×	**3.26**±0.74×
Target Model: Qwen2.5-7B-Instruct						
AR Draft Model	1.43±0.18×	1.42±0.25×	1.41±0.19×	1.25±0.21×	1.40±0.17×	1.38±0.20×
Fast-dLLM	1.95±0.38×	1.57±0.19×	1.54±0.18×	1.34±0.22×	1.61±0.15×	1.60±0.22×
EAGLE-3	2.31±0.54×	2.36±0.51×	2.02±0.45×	**1.98**±0.27×	2.39±0.56×	2.21±0.47×
FailFast (ours)	**3.06**±0.70×	**2.63**±0.65×	**2.34**±0.43×	1.89±0.47×	**2.71**±0.71×	**2.52**±0.59×

Hardware. We run our evaluations on two NVIDIA A6000-48GB GPUs connected via PCIe 4.0 x16. We profile latency using SGLang (Zheng et al., 2023), enable prefix caching (Pan et al., 2025b) for both the draft model and target model, and use TP=2 for all target models.

5.2 End-to-End Results

Table 1 presents the end-to-end speedups over vanilla decoding across different schemes and datasets. Due to space constraints, we refer readers to the Appendix for additional experiments (hyperparameter sensitivity studies and generalization studies) and detailed analyses and discussions.

Workload sensitivity. Across models and datasets, FailFast achieves an overall speedup of 1.9-4.9×. Compared to baselines, FailFast achieves a speedup of 1.3-2.1× over the AR drafter baseline and a speedup of 1.2-1.7× over the Fast-dLLM baseline. FailFast has a higher win over baselines on datasets that are easier to speculate: math and coding workloads exhibit centralized regions of predictable tokens (e.g., intermediate calculations and code generation), allowing FailFast to capitalize on long speculation windows. Conversely, datasets where "easy" tokens are scattered rather than clustered offer fewer opportunities for aggressive speculation length expansion, though FailFast still outperforms fixed-length baselines with moderate length expansion. These speedups arise from drafting policy alone, applied to off-the-shelf dLLMs without any fine-tuning, validating the claim that policy can drive substantial gains independent of drafter quality investment.

Efficiency of FailFast. Naively plugging in a dLLM as the drafter (Fast-dLLM) retains the acceptance rate (Tab. 4) and achieves a speedup over the AR drafter. However, as shown in Fig. 7, this gain is primarily derived from a reduction in speculation latency (22.9% on average)[2] while the verification latency remains near-constant. In contrast, FailFast further reduces speculation latency significantly (a further 41.1% over Fast-dLLM) by employing an ultra-small compute budget for the drafter. Although our approach inevitably degrades the acceptance rate (Tab. 4) – a combined result of a low drafter compute budget and proposing more tokens on average in each round – we achieve superior end-to-end speedups (e.g., 3.1-4.9× for Qwen2.5-32B). In addition, FailFast reduces verification latency (Fig. 7) by an average of 17.1% via adopting adaptive speculation lengths: through dynamically expanding the speculation window in "easy" regions, we can speculate and verify up to 70 tokens (Fig. 8) in a single round. This drastic reduction in the total number of speculation-verification rounds

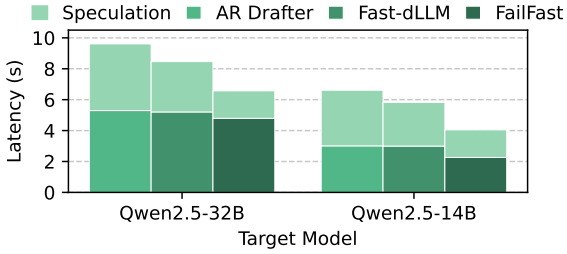


Figure 7: End-to-end latency broken down into speculation (lighter, upper) and verification (darker, lower). Fast-dLLM reduces speculation latency; FailFast further reduces both speculation latency per round and verification latency (fewer rounds).

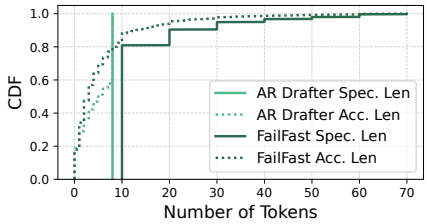


Figure 8: [Qwen2.5-32B, MATH] CDF of the number of accepted/speculated tokens in each round. Full results are in Appendix B.

and the speculation overhead in each round outweighs the cost of a lower per-token acceptance rate. We include an example trajectory in Appendix G to visualize FailFast in action.

FailFast is drafter-agnostic. We further test whether FailFast's gains are tied to a specific model architecture by using DiffuLLaMA-7B, a dLLM adapted from LLaMA2-7B, as the drafter for LLaMA2-13B. Averaged over MATH, GPQA, and HumanEval, FailFast outperforms the AR drafter and vanilla dLLM drafter by 2.7× and 3.7×, respectively. These results suggest that FailFast's benefit is not architecture-specific: even with a relatively large and lower-quality dLLM drafter, it wins by sharply reducing drafting latency and exploiting the general concavity of dLLM accuracy improvements. Full results are in Appendix D.1.

Table 2: FailFast's win generalizes across different drafter architectures.

Drafting Method	Speedup
LLaMA2-7B	1.61×
DiffuLLaMA-7B	1.21×
DiffuLLaMA-7B + FailFast	**4.24**×

Comparisons with EAGLE-3. Across models, FailFast achieves an average speedup of 1.0–1.4× over EAGLE-3, with gains of up to 1.3–1.7× on specific datasets. Even though EAGLE-3 incurs a smaller drafting latency in each of its forward passes – it only has one layer compared to the 28 layers in other drafters, reducing overheads associated with loading weights and launching kernels – FailFast still outperforms EAGLE-3. This is because EAGLE-3 remains an autoregressive drafter that generates only one token per forward pass, whereas FailFast generates multiple tokens in each pass. Moreover, while EAGLE-3 speculates a tree of 50–60 tokens per round, its draft tree is relatively shallow (default depth 8 (Li et al., 2025)) compared to FailFast, which drafts a chain of up to 70 tokens in a single round. As a result, EAGLE-3's upper bound on the number of accepted tokens is constrained by tree depth, limiting its ability to exploit extended regions of easier tokens.

6 Conclusion

We present FailFast, a training-free speculative decoding framework that operationalizes a shift in design philosophy: from quality-centric drafting to policy-driven design. By identifying two intrinsic dLLM properties – sublinear accuracy gains and near-length-independent drafting cost – and combining them with workload-level heterogeneity in decoding difficulty, we showed that drafting policy alone can drive substantial speedups (up to 4.9× over vanilla decoding, 1.7× over a naive dLLM drafter baseline, 1.7× over EAGLE-3) using off-the-shelf aligned dLLMs without any fine-tuning. Beyond FailFast specifically, our findings suggest that the policy axis of speculative decoding deserves attention proportional to its impact, and that dLLM-specific properties enable design choices that AR drafters structurally cannot. As dLLM alignment continues to improve, policy-driven approaches like FailFast will benefit naturally, without requiring per-target retraining.

AI use statement

We have not used generative AI tools for any research task requiring disclosure under the ICLR policy. Additionally, we used GitHub Copilot to assist with creating and editing software code, and used ChatGPT to identify and correct grammatical issues in the manuscript. All AI-assisted code

and text edits were reviewed by the authors; the research code was tested for correctness. We take responsibility for the final content of this work, including all code, experiments, results, and claims.

Reproducibility statement

Sections 4 and 5 describe the proposed policy, experimental setup, evaluated models and baselines, datasets, decoding settings, and hardware configuration. The appendix provides further details on hyperparameter sensitivity, stochastic decoding, generalization across model families, and baseline reproduction. All base models and benchmark datasets used in our experiments are publicly available, and the relevant versions and settings are specified in the paper.

References

Aime 2024 dataset card. https://huggingface.co/datasets/HuggingFaceH4/aime_2024, 2025.

Accelerating llm inference: Up to 3x speedup on mi300x with speculative decoding. https://rocm.blogs.amd.com/artificial-intelligence/spec_decode_mi300x/README.html, 2025.

Adaptive-learning speculator system (atlas): A new paradigm in llm inference via runtime-learning accelerators — together ai. https://www.together.ai/blog/adaptive-learning-speculator-system-atlas, 2025.

Fly eagle(3) fly: Faster inference with vllm & speculative decoding — red hat developer. https://developers.redhat.com/articles/2025/07/01/fly-eagle3-fly-faster-inference-vllm-speculative-decoding, 2025.

Gemini diffusion - google deepmind. https://deepmind.google/models/gemini-diffusion/, 2025.

Specforge. https://github.com/sgl-project/SpecForge, 2025.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Sudhanshu Agrawal, Risheek Garrepalli, Raghavv Goel, Mingu Lee, Christopher Lott, and Fatih Porikli. Spiffy: Multiplying diffusion llm acceleration via lossless speculative decoding. *arXiv preprint arXiv:2509.18085*, 2025.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, et al. Llama-nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2.0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

F Warren Burton. Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, 100(12):1190–1193, 1985.

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.

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

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yu-Hsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. *Advances in Neural Information Processing Systems*, 37:129531–129563, 2024.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

Prateek Chhikara. Mind the confidence gap: Overconfidence, calibration, and distractor effects in large language models. *arXiv preprint arXiv:2502.11028*, 2025.

Jacob K Christopher, Brian R Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In *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, 2025.

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.

Yinwei Dai, Rui Pan, Anand Iyer, Kai Li, and Ravi Netravali. Apparate: Rethinking early exits to tame latency-throughput tensions in ml serving. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 607–623, 2024.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, 2023.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Jiahui Geng, Fengyu Cai, Yuxia Wang, Heinz Koeppl, Preslav Nakov, and Iryna Gurevych. A survey of confidence estimation and calibration in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6577–6595, 2024.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

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

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

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*, 2024b.

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.

Hongyi Liu, Jiaji Huang, Zhen Jia, Youngsuk Park, and Yu-Xiang Wang. Not-a-bandit: Provably no-regret drafter selection in speculative decoding for llms. *arXiv preprint arXiv:2510.20064*, 2025a.

Jiahao Liu, Qifan Wang, Jingang Wang, and Xunliang Cai. Speculative decoding via early-exiting for faster llm inference with thompson sampling control mechanism. *arXiv preprint arXiv:2406.03853*, 2024a.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023.

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

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025b.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. Turning trash into treasure: Accelerating inference of large language models with token recycling, 2024. URL https://arxiv.org/abs/2408.08696.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. Suffixdecoding: A model-free approach to speeding up large language model inference, 2024. URL https://arxiv.org/abs/2411.04975.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*, 2025a.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid llms. In *Eighth Conference on Machine Learning and Systems*, 2025b.

Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. d3llm: Ultra-fast diffusion llm using pseudo-trajectory distillation. *arXiv preprint arXiv:2601.07568*, 2026.

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.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. *arXiv preprint arXiv:2408.11049*, 2024.

Jameson Sandler, Jacob K Christopher, Thomas Hartvigsen, and Nando Fioretto. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding. *arXiv preprint arXiv:2511.00606*, 2025.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025a.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025b.

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.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*, 2024.

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

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 6344–6355. Association for Computing Machinery, 2024. ISBN 9798400704901. doi: 10.1145/3637528.3671614.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody_Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Efficiently programming large language models using sglang. 2023.

Zhanhui Zhou, Lingjie Chen, Hanghang Tong, and Dawn Song. dllm: Simple diffusion language modeling. `https://github.com/ZHZisZZ/dllm`, 2025.

A List of abbreviations and notations

Table 3: Glossary of abbreviations, notations, and terminology.

Notation	Definition
AR	Autoregressive
dLLM	Diffusion language models
Round	A speculation-verification round
n / N	Speculation length (number of tokens) in each round

B Detailed Performance Comparison

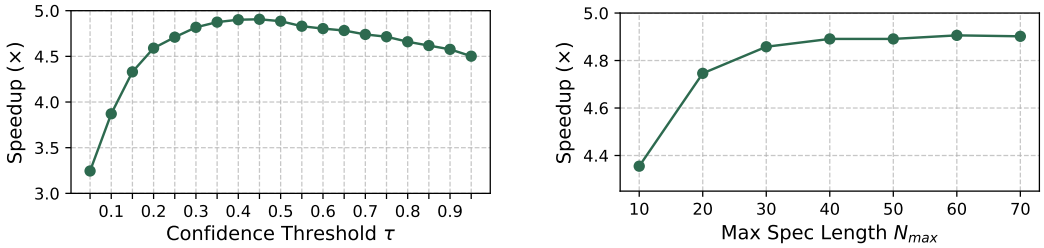
Table 4: Detailed performance comparison. We report the average acceptance rate, average accepted/speculated lengths in each round[3], maximum accepted/speculated lengths across rounds, average number of speculation-verification rounds, and the average number of drafter forward passes in each round. **Note that, for fixed-draft-length baselines, we select lengths with the highest end-to-end speedup to best represent the baselines' performance.**

Drafter	Metric	Dataset					
		MATH	AIME	GSM8K	GPQA	HumanEval	Average
Target: Qwen2.5-32B-Instruct							
AR drafter	Acceptance Rate	58.2%	56.0%	55.3%	51.2%	56.1%	55.4%
	Avg (Acc./Spec.) Len	4.7 / 8	4.5 / 8	4.4 / 8	2.6 / 5	3.9 / 7	4.0 / 7.2
	Max (Acc./Spec.) Len	8 / 8	8 / 8	8 / 8	5 / 5	7 / 7	-
	Num Speculation Rounds	93.6	138.8	56.1	138.3	76.6	100.7
	Avg Drafter Passes / Round	8	8	8	5	7	7.2
Fast-dLLM	Acceptance Rate	60.5%	56.9%	53.2%	52.7%	57.7%	56.2%
	Avg (Acc./Spec.) Len	4.8 / 8	4.5 / 8	4.3 / 8	2.6 / 5	4.0 / 7	4.0 / 7.2
	Max (Acc./Spec.) Len	8 / 8	8 / 8	8 / 8	5 / 5	7 / 7	-
	Num Speculation Rounds	90.4	137.4	57.3	135.3	74.3	98.9
	Avg Drafter Passes / Round	5.7	6.1	6.1	4.4	5.2	5.5
FailFast	Acceptance Rate	40.6%	38.0%	33.6%	26.4%	31.7%	34.1%
	Avg (Acc./Spec.) Len	6.0 / 14.6	5.1 / 13.3	4.1 / 12.1	3.3 / 12.0	4.5 / 11.3	4.6 / 12.7
	Max (Acc./Spec.) Len	70 / 70	70 / 70	50 / 70	70 / 70	60 / 70	-
	Num Speculation Rounds	78.6	129.9	59.1	119.0	66.9	90.7
	Avg Drafter Passes / Round	3.6	3.4	3.2	3.1	3.1	3.3
Target: Qwen2.5-14B-Instruct							
AR drafter	Acceptance Rate	68.3%	66.8%	64.1%	60.2%	69.9%	65.9%
	Avg (Acc./Spec.) Len	4.1 / 6	3.3 / 5	3.8 / 6	2.4 / 4	2.8 / 4	3.3 / 5.0
	Max (Acc./Spec.) Len	6 / 6	5 / 5	6 / 6	4 / 4	4 / 4	-
	Num Speculation Rounds	93.0	175.5	75.5	176.3	89.3	121.9
	Avg Drafter Passes / Round	6	5	6	4	4	5
Fast-dLLM	Acceptance Rate	68.9%	66.8%	61.9%	61.3%	73.9%	66.6%
	Avg (Acc./Spec.) Len	4.1 / 6	3.3 / 5	3.7 / 6	2.5 / 4	3.0 / 4	3.3 / 5.0
	Max (Acc./Spec.) Len	6 / 6	5 / 5	6 / 6	4 / 4	4 / 4	-
	Num Speculation Rounds	93.0	175.3	78.1	173.8	86.6	121.4
	Avg Drafter Passes / Round	4.2	4.0	4.6	3.6	3.2	3.9
FailFast	Acceptance Rate	43.5%	39.2%	36.4%	29.2%	39.3%	37.5%
	Avg (Acc./Spec.) Len	6.5 / 14.5	5.2 / 12.7	4.4 / 12.2	3.5 / 11.7	5.2 / 13.2	5.0 / 12.9
	Max (Acc./Spec.) Len	60 / 70	70 / 70	60 / 60	70 / 70	60 / 70	-
	Num Speculation Rounds	71.4	127.8	67.1	136.1	55.4	91.6
	Avg Drafter Passes / Round	3.6	3.2	3.2	3.0	3.3	3.3

We further dissect FailFast's performance improvements in Tab. 4. Zooming in on Qwen2.5-32B on the MATH dataset, even though the AR drafter has a higher overall acceptance rate, for each

of its forward passes, it produces only one token and accepts 0.6 tokens on average. In contrast, FailFast can produce an average of 4.1 tokens per forward pass and accept 1.7 of them. Another source of FailFast's win comes from its aggressive speculation length expansion. Fig. 8 shows a CDF of the accepted/speculated length across rounds. In ∼20% of the rounds, we extended the speculation length beyond the default, reducing the average number of speculation-verification rounds by 16.0%. Interestingly, FailFast's dynamic policy converges to an average speculation length of 14.6 tokens, closely matching the theoretically optimal of 16 tokens from our prior analysis (Fig. 3).

C Hyperparameter Sensitivity Analysis of FailFast



(a) Impact of the confidence threshold τ.

(b) Impact of the maximum speculation length N_{max}.

Figure 9: Impact of FailFast's hyperparameters in Alg. 1 on its performance. FailFast's performance degrades gracefully as the hyperparameters are tweaked.

We analyze the sensitivity of FailFast to its key hyperparameters to understand the trade-offs between speculation overhead and verification efficiency. We focus our analysis on Qwen2.5-32B on MATH, and vary its default hyperparameters $\tau = 0.4$ and $N = 10$. One advantage of FailFast is that, because drafting is low-latency by design, it does not require extensive tuning of hyperparameters to reach optimal performance, and the speculation length is self-adaptive based on token difficulty, whereas traditional speculative decoding schemes typically require workload-aware tuning of the speculation length N to reach optimal performance.

Confidence Threshold (τ). The confidence threshold controls the aggressiveness of the speculation length expansion. We find that $\tau \in [0.3, 0.55]$ yields optimal performance. A threshold that is too high (e.g., 0.7) makes FailFast overly conservative, preventing the proposal of long sequences in easy regions. Conversely, a threshold that is too low (e.g., 0.1) increases speculation latency overhead by generating tokens that are statistically likely to be rejected by the verifier.

Maximum Speculation Length ($N_{\max}$). We observe that $N_{\max}$ values between 30 and 70 perform reliably well. We adopt a default value of $N_{max} = 60$ because LLM prefill for short sequences is typically memory-bandwidth bound, and the typical tipping point between memory-bound and compute-bound for medium-sized LLMs typically ranges between 64 and 128 tokens on modern GPUs (Agrawal et al., 2024). In high-throughput inference settings where batch sizes are large, bigger values of $N_{\max}$ still work, although they might inflate the verification latency slightly – the prefill pass would eventually become compute-bound for large sequence lengths, even with prefix caching (Pan et al., 2025b). In that case, synchronous verification might force a reduction in $N_{\max}$ to avoid inflating the per-round verification latency, and FailFast's performance improvement from "winning big" might drop slightly. However, FailFast's performance improvement from "failing fast" persists. We note that speculative decoding is inherently more suitable for inference acceleration at smaller batch sizes to begin with and provides diminishing gains as batch size increases, where target model inference becomes less memory-bound and more compute-bound (Sadhukhan et al., 2024; amd, 2025).

D GENERALIZATION

D.1 MODEL ARCHITECTURES

To demonstrate generalization across model architectures, we adopted DiffuLLaMA-7B (Gong et al., 2024), a dLLM adapted from LLaMA2-7B, as a draft model for LLaMA2-13B (in line with the evaluation setup in prior work (Sandler et al., 2025)) and evaluated its speedup over vanilla decoding in Tab. 5. Similar to our main experiments, we pick the draft length from 3 to 15 that achieves the best end-to-end speedup. On LLaMA-based models, FailFast's win generalizes across different model architectures, outperforming the AR drafter by 2.7× and the dLLM drafter by 3.7×. Despite DiffuLLaMA-7B being a less-than-ideal draft model (it's too big relative to the 13B target model compared to typical spec decoding, where drafter size is 0.3-1.5B; its quality is also much worse than the corresponding AR drafter (Gong et al., 2024)), FailFast can still outperform the AR drafter through drastically reducing the drafting latency by 5.5×. Zooming out, the reason for these wins is that FailFast's performance is not tied to architecture-specific traits of any model family, but instead revolves around exploiting an intrinsic difference between AR and diffusion-based drafting techniques, i.e., dLLM's concavity of accuracy improvements is fundamental to dLLMs of different architectures.

Table 5: FailFast's win generalizes across different model architectures, outperforming the AR drafter by 2.7× and the dLLM drafter by 3.7×.

Drafting Method	**MATH**	**GPQA**	**HumanEval**	**Average**
LLaMA2-7B	1.59×	1.66×	1.59×	1.61×
DiffuLLaMA-7B	1.17×	1.27×	1.19×	1.21×
DiffuLLaMA-7B + FailFast	**4.01×**	**4.36×**	**4.35×**	**4.24×**

D.2 STOCHASTIC DECODING (TEMPERATURE = 1)

Tab. 6 reports end-to-end speedups under stochastic decoding (temperature = 1). Consistent with prior work on speculative decoding (Yan et al., 2024), all methods experience a reduction in speedup compared to the greedy (temperature = 0) setting. This is expected: at higher temperatures, the target model's output distribution becomes more flattened out, reducing the likelihood that the draft model's sampled tokens align with the target model's samples. As a result, the per-round acceptance rate drops, fewer speculative tokens are accepted on average, and more decoding work falls back to the (slow) target model – since end-to-end speedup scales with the average number of accepted draft tokens per round, this translates directly into lower overall acceleration. Despite this, FailFast retains its advantage over all baselines, achieving speedups of up to 3.9× across workloads, and a performance gain of 1.3-1.6× over baselines. These results confirm that FailFast's gains are robust to the choice of decoding temperature.

Table 6: Comparison of speedups of different drafting methods over vanilla decoding at temperature = 1. FailFast achieves the **highest** average speedup.

Drafting Method	**Dataset**					**Average**
	MATH	**AIME**	**GSM8K**	**GPQA**	**HumanEval**	
Target Model: Qwen2.5-32B-Instruct						
AR Draft Model	2.47×	2.20×	2.56×	1.78×	2.47×	2.30×
Fast-dLLM	3.45×	3.10×	3.05×	2.28×	3.16×	3.01×
EAGLE-3 (w/ draft tree)	2.53×	2.36×	2.39×	2.01×	2.89×	2.44×
FailFast (ours)	**3.90×**	**3.40×**	**3.23×**	**2.42×**	**3.40×**	**3.27×**

D.3 Generalization of Confidence as a Difficulty Signal

We compare confidence scores for "easier" and "harder" tokens (as defined in Sec. 4.2) across datasets. As shown in Fig. 10, confidence as a difficulty signal for acceptance exhibits a strong correlation across datasets, is intuitive, and incurs near-zero overhead.

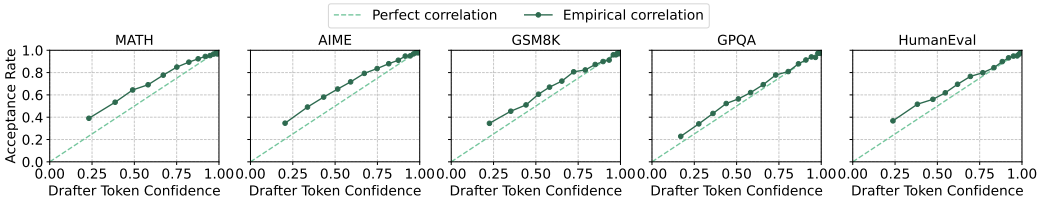


Figure 10: Draft model confidence strongly correlates with token acceptance (i.e., "easiness") across datasets.

E Best-Effort Baselines

We evaluate FailFast against a diverse set of strong baselines, covering both single-layer and multi-layer drafters and single-token predictors and multi-token predictors (in each drafter forward pass). We acknowledge that there exist many other speculative decoding methods, e.g., Medusa (Cai et al., 2024), Lookahead Decoding (Fu et al., 2024), etc. Since none of them have been incorporated into high-throughput inference engines (Kwon et al., 2023; Zheng et al., 2023) and EAGLE-3 outperforms them (Li et al., 2025; Xia et al., 2024), we omit those schemes from our evaluations.

E.1 Fast-dLLM Drafter

Our Fast-dLLM baseline adopts the same speculation length as the AR drafter. To show the full potential of a naive baseline, we also conduct an extensive parameter sweep (n = 3-20) for the optimal speculation length of the dLLM drafter and pick a (different) length that achieves the best speedup for each dataset. The end-to-end speedup is reported in Tab. 7. This baseline is idealistic because it requires extensive offline profiling to find the optimal tradeoff between the acceptance rate and speculation latency, which is indeterministic because of Fast-dLLM's confidence-aware parallel decoding. More importantly, we find that on average, picking the best speculation length only brings a 2.3-3.3% performance improvement over using the same speculation length as the AR drafter, so we omit Fast-dLLM+ from the main paper for conciseness.

Table 7: Comparison of Fast-dLLM (same speculation length as the AR drafter) with Fast-dLLM+ (best speculation length on each dataset from a parameter sweep).

Drafting Method	Dataset					
	MATH	AIME	GSM8K	GPQA	HumanEval	Average
Target Model: Qwen2.5-32B-Instruct						
Fast-dLLM	3.57×	3.29×	3.10×	2.42×	3.16×	3.11×
Fast-dLLM+	**3.75×**	**3.34×**	**3.10×**	**2.48×**	**3.23×**	**3.18×**
Target Model: Qwen2.5-14B-Instruct						
Fast-dLLM	2.57×	2.22×	2.22×	1.86×	2.23×	2.22×
Fast-dLLM+	**2.69×**	**2.28×**	**2.23×**	**1.87×**	**2.40×**	**2.30×**

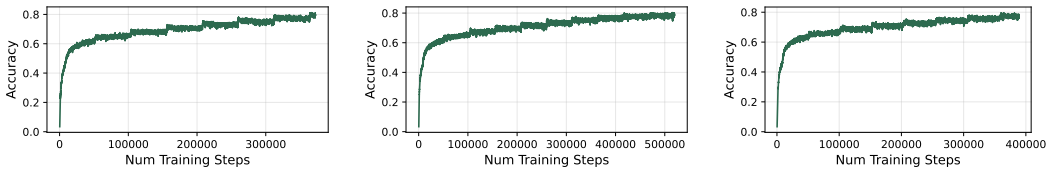
E.2 SuffixDecoding

SuffixDecoding (Oliaro et al., 2024) represents the state of the art non-parametric drafting method in speculative decoding. It maintains a suffix tree of tokens from previously generated text and the current prompt. If it finds an exact match of the most recent tokens generated, it "speculates" that the LLM will continue the sequence the same way it did in the past. SuffixDecoding only captures wins on the easy regions that exactly repeat the previous context. In contrast, FailFast adaptively

determines which subsequences are easy and captures wins on easier segments that aren't just simply repeating prior context, gaining more wins.

In our evaluations, we find that SuffixDecoding provides negligible speedups over vanilla decoding on the workloads that we evaluate. Fundamentally, SuffixDecoding operates as a pattern-matcher rather than a predictive model; it yields the greatest efficiency gains when the LLM generates sequences identical to those in prior context, such as SQL schemas, boilerplate code, or repetitive reasoning loops. In agentic coding tasks (e.g., SWE-Bench), where an agent iteratively interacts with file systems and error logs, the method achieves high speedups by matching recurring token sequences. Conversely, in "one-shot" benchmarks such as HumanEval, and in mathematical reasoning chains where sequences are unique and lack frequent reflection, the suffix tree finds few matches.

E.3 EAGLE-3



(a) 32B training. (b) 14B training. (c) 7B training.

Figure 11: EAGLE-3 training accuracy curves.

EAGLE-3 training. There are no public EAGLE-3 weights available for Qwen2.5. Thus, we conducted pretraining of EAGLE-3 weights using the ultrachat-200K (Ding et al., 2023) text corpus in SpecForge (spe, 2025). We adopted the default training hyperparameters in SpecForge and trained EAGLE-3 to match the target model's output until convergence. We plot the moving average of the training accuracy in Fig. 11. During evaluation, we use the model checkpoints after five epochs of training ($\sim$260k training steps). We find that even though further training improves training-time accuracy, it has a negligible impact on the end-to-end speedup of EAGLE-3. We open-source our EAGLE-3 model weights and training scripts.

Draft token trees. To ensure a fair and accurate representation of EAGLE-3's peak performance (Li et al., 2025; eag, 2025), we evaluate EAGLE-3 in SGLang v0.5.6.post2 (Zheng et al., 2023) with a draft token tree, with 50-60 tokens in each tree (following the best default hyperparameter configurations reported in the EAGLE-3 paper (Li et al., 2025), roughly matching N_{max} in FailFast). For completeness, we additionally evaluate EAGLE-3 configured to propose a single chain of 3-20 draft tokens (rather than a tree). This variant consistently underperforms the full draft-tree version of EAGLE-3, and we therefore omit its results from the evaluation. Similar to the AR drafter (Fig. 3), adopting a fixed speculation length of 50-60 tokens in a single chain of draft tokens resulted in a significant slowdown compared to vanilla decoding due to the speculation overhead in autoregressive drafting. We note that FailFast's one-step generation is compatible with token trees, which can be realized via a custom attention mask (Agrawal et al., 2025). We leave this extension as future work.

Data-matched comparisons of parametric drafters. While a data-matched comparison with EAGLE-3 would strengthen our claim, Fast-dLLM-v2 is trained on a proprietary dataset (Wu et al., 2025a), so a fully apples-to-apples comparison is impossible. Nevertheless, in this subsection, we trained an EAGLE-3 drafter on a dataset as close as possible to Fast-dLLM's. Fast-dLLM-v2 notes that its models are fine-tuned on a subset of the LLaMA-Nemotron post-training dataset (Bercovich et al., 2025). Although the authors didn't disclose how exactly the subset was constructed, we sampled 33k random requests from each of the three main splits (math, code, chat). (In our main experiments in Tab. 1, EAGLE-3's training set, UltraChat, contained only chat data.) We trained the EAGLE-3 drafter to convergence and report end-to-end speedups of Qwen2.5-14B-Instruct in Tab. 8. Adding math requests significantly improved EAGLE-3 speedups on MATH and AIME. However, FailFast still outperforms EAGLE-3 by $\sim$20% on math datasets and up to $\sim$46% across all datasets. HumanEval results changed little, likely because coding requests in LLaMA-Nemotron include lengthy chain-of-thoughts in natural language, whereas HumanEval outputs are mostly code.

Table 8: Best-effort comparison of data-matched EAGLE-3 and FailFast.

Drafting Method	Dataset					
	MATH	**AIME**	**GSM8K**	**GPQA**	**HumanEval**	**Average**
EAGLE-3	2.48×	2.38×	2.23×	2.04×	2.68×	2.36×
+ data-matched	3.24× *(+30%)*	2.79× *(+17%)*	2.08× *(-7%)*	2.02× *(-1%)*	2.66× *(-0%)*	2.56× *(+8%)*
FailFast	**3.92×**	**3.37×**	**3.04×**	**2.54×**	**3.41×**	**3.26×**

In summary, even though adding more math data improves the speedup of EAGLE-3 on math-heavy datasets, FailFast retains most of its performance win over EAGLE-3. These data-matched comparisons support EAGLE's claim that "EAGLE exhibits low sensitivity to training data" (Li et al., 2024a), confirming that FailFast's gains are algorithmic rather than dataset-driven.

F (Failed) Explorations of Alternative Design Choices

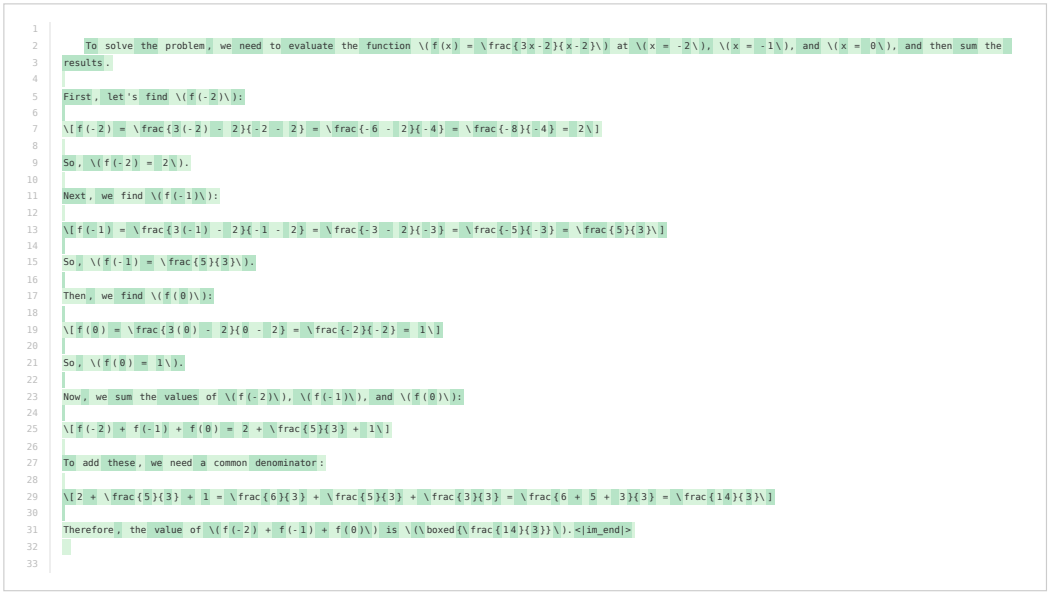
Dynamic speculation length schemes. Initially, borrowing from ideas in network congestion control (Jacobson, 1988), we tried to use the acceptance rate of previous rounds as an indication of token "easiness" in the current round. However, we found that the notion of easiness appears to be highly local and not well correlated across rounds. This finding prompted us to look forward (ahead into the future, i.e., the current tokens being speculated), not backward.

Reusing previous drafts. In speculative decoding, all tokens following the first rejection are discarded. However, we observe that in many workloads – particularly reasoning chains of thoughts – rejections are often minor corrections (e.g., changing "thus" to "therefore", as can be observed in the example trajectory in Fig. 14), while the subsequent tokens remain of high utility. In traditional speculative decoding, where speculation lengths are relatively short (3–16 tokens), reusing previous drafts offers marginal utility. In contrast, the aggressive speculation length expansion in FailFast makes reuse more beneficial; for example, if out of 60 proposed tokens, we accepted the first 19 tokens and corrected the 20th token, the remaining 40 tokens could be directly plugged back in if the 20th token was the only token that had to be corrected. Nevertheless, because the drafting process in FailFast is inherently low-latency, the marginal speedups gained from reuse are often secondary. When these cases do occur, they represent an opportunistic performance gain. For a specific example in the MATH dataset (Fig. 14), we implemented a preliminary mechanism for reusing drafts: after each speculation round, we determine if the suffix of the current proposal exists within the tokens rejected in the previous round. If a match is found, the new proposal is appended with the tokens following that suffix. For instance, if the rejected tokens from the last round are 'xefghij' and the current draft is 'abcdefg', our new proposal will be 'abcdefghij'. For this specific query, reuse provided a $\sim$15% speedup over FailFast, reaching an impressive end-to-end speedup of 8.8× over vanilla decoding. However, since most rejections in other queries involve more than simple one-token corrections, the average speedup across the dataset is only $\sim$2%, as the gains are diluted by rounds with fewer reuse opportunities. Consequently, we omit this technique from our primary results.

G Visualization of Example Trajectory

In this section, we visualize the speculative decoding trajectory of an example query (question 2 from the MATH dataset). Individual tokens are distinguished by alternating background colors. Accepted tokens are shown as standard text. Rejected draft tokens are marked with a strikethrough and reduced opacity. Target-generated tokens (either corrections applied after a rejection or bonus tokens appended after a fully accepted draft) are highlighted with a solid border. This visualization illustrates the precise behavior of different speculative decoding baselines. The question is as follows:

> If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2)+f(-1)+f(0)$? Express your answer as a common fraction.



```
To solve the problem, we need to evaluate the function \(f(x) = \frac{3x-2}{x-2}\) at \(x = -2\), \(x = -1\), and \(x = 0\), and then sum the results.

First, let's find \(f(-2)\):

\[f(-2) = \frac{3(-2) - 2}{-2 - 2} = \frac{-6 - 2}{-4} = \frac{-8}{-4} = 2\]

So, \(f(-2) = 2\).

Next, we find \(f(-1)\):

\[f(-1) = \frac{3(-1) - 2}{-1 - 2} = \frac{-3 - 2}{-3} = \frac{-5}{-3} = \frac{5}{3}\]

So, \(f(-1) = \frac{5}{3}\).

Then, we find \(f(0)\):

\[f(0) = \frac{3(0) - 2}{0 - 2} = \frac{-2}{-2} = 1\]

So, \(f(0) = 1\).

Now, we sum the values of \(f(-2)\), \(f(-1)\), and \(f(0)\):

\[f(-2) + f(-1) + f(0) = 2 + \frac{5}{3} + 1\]

To add these, we need a common denominator:

\[2 + \frac{5}{3} + 1 = \frac{6}{3} + \frac{5}{3} + \frac{3}{3} = \frac{6 + 5 + 3}{3} = \frac{14}{3}\]

Therefore, the value of \(f(-2) + f(-1) + f(0)\) is \(\boxed{\frac{14}{3}}\).<|im_end|>
```

Figure 12: Ground truth trajectory.



```
To solve for \(f(-2)\) the given problem, we need to evaluate \(f\) problem \(f(-2) + f(-1)\), we need to evaluate \(f(-2)\) the function \(f(x) = \frac{f(x)}{x}\) at three specific points \(f\) = \frac{3x-2}{x-2}\) at three specific \(x = -2\), \(x = -1\), and \(x = 0\) separately, and then sum these\), and then sum these values.

First the results.

First, let's find \(f(-2)\):
\[f(-2)\):

\[f(-2) = \frac{3(-2) - 2}{-2 - 2} = \frac{-6 - 2}{-4} = \frac{-8}{-4} = 2\]

Next, let's find \(f\)So, \(f(-2) = 2\).

Next, let's find \(f\) we find \(f(-1)\):

\[f(-1) = \frac{3(-1) - 2}{-1 - 2} = \frac{-3 - 2}{-3} = \frac{-5}{-3} = \frac{5}{3}\]

So, \(f(-1) = \frac{5}{3}\).

Finally, we findThen, we find \(f(0)\):

\[f(0) = \frac{3(0) - 2}{0 - 2} = \frac{-2}{-2} = 1\]

So, \(f(0) = 1\).

Now, we add these sum these values.

\[f(-2)\] the values

\[f(-2) + of \(f(-2)\), \(f(-1)\), and \(f(0)\):

\[f(-2) + f(-1) + f(0) = 2 + \frac{5}{3} + 1\]

To add these, we need a common denominator:

\[2 + \frac{5}{3} + 1 = \frac{6}{3} + \frac{5}{3} + \frac{3}{3} = \frac{6+5+3}{3} + 5 + 3}{3} = \frac{14}{3}\]

Therefore, the value of \(f(-2) + f(-1) + f(0)\) is \(\boxed{\frac{14}{3}}\).<|im_end|>
```

Figure 13: Autoregressive drafter's trajectory.



Figure 14: FailFast's trajectory.

In Fig. 12, we first show the ground-truth trajectory as a reference. Fig. 13 shows the trajectory of an autoregressive drafter with a static speculation length of 8 tokens. In many segments of the trajectory (e.g., lines 11, 17, and 31-39), there are consecutive rounds where the acceptance rate is near-perfect; however, the target model is frequently invoked for verification in these segments, resulting in excessive latency overhead.

Finally, in Fig. 14, we demonstrate the dynamic speculation length of FailFast in action. The main win from FailFast's aggressive speculation length expansion comes from these rounds:

- Round 4 (line 3), 18/30 accepted (7 drafter forward passes in this round)
- Round 5 (line 3), 20/20 accepted (6 drafter forward passes in this round)
- Round 10 (line 14), 18/20 accepted (6 drafter forward passes in this round)
- Round 11 (line 14), 28/60 accepted (11 drafter forward passes in this round)
- Round 14 (line 38), 48/60 accepted (12 drafter forward passes in this round)
- Round 15 (lines 38-40), 23/60 accepted (11 drafter forward passes in this round)
- Round 16 (lines 42-48), 60/60 accepted (11 drafter forward passes in this round)
- Round 18 (lines 48-52), 50/50 accepted (9 drafter forward passes in this round)
- Round 22 (line 55), 60/60 accepted (12 drafter forward passes in this round)
- Round 23 (line 60), 30/30 accepted (7 drafter forward passes in this round)

We note that the number of actual forward passes deviates from the intuition in Fig. 1 due to a number of reasons: We do one-pass generation following Fast-dLLM's default small block size of 8, so generating 10 tokens might span across three small blocks, which requires three forward passes even

though we are doing one-pass generation for each small block; occasionally, we need extra forward passes on a full block (e.g., 32 tokens) to populate the KV cache of prior drafted tokens (Wu et al., 2025a).

H Additional Related Work

FailFast is concurrent with DFlash (Chen et al., 2026) and DSpark (Cheng et al., 2026), which independently explore closely related design choices for speculative decoding. In particular, DFlash and FailFast share a central operating point: rather than performing additional denoising iterations at inference to improve each draft, both use a single block-diffusion forward pass to keep drafting inexpensive. The methods differ in where they invest effort thereafter. DFlash trains a bespoke, target-conditioned block-diffusion drafter to improve the quality and acceptance of one-pass proposals. In contrast, FailFast uses an off-the-shelf dLLM and invests in drafting policy: it deliberately tolerates lower-quality one-pass proposals, hedges against their errors through verification, and adapts speculation length to exploit regions where the drafter is sufficiently reliable. Thus, DFlash and FailFast represent alternative strategies for realizing efficient one-pass speculative decoding: improving the drafter itself versus adapting the policy to the drafter's quality.

DSpark is closer to FailFast in using confidence to control speculative work, but applies it differently. DSpark learns a calibrated acceptance predictor and prunes low-confidence suffixes before verification, with a load-aware scheduler aimed at high-concurrency serving (Cheng et al., 2026). FailFast instead uses the dLLM's native token confidence as a continue-or-verify signal: it extends a draft across further high-confidence blocks to reduce verification frequency in the batch-1, bounded-length regime studied here. Both methods adapt the effective verification length, but differ in the confidence signal, policy action, and target serving regime.

I Additional Notes

[1]The sawtooth pattern arises because drafting 9–16 tokens costs two forward passes, while 17 tokens requires three (a 50% latency increase due to semi-autoregressive generation of 8-token blocks).

[2]This improvement falls short of the theoretical end-to-end speedups of dLLMs (Wu et al., 2025a) because our termination criterion requires the next continuous n tokens to be unmasked; due to the random decoding order within token blocks, the leftmost tokens are not necessarily decoded first.

[3]The acceptance rate does not exactly equal the average accepted length divided by the average speculation length, because the average of averages does not necessarily equal the overall average. Note that, unlike prior work (Sandler et al., 2025), we **do not** add one to the average accepted length to represent the token (either correcting the rejected token, or the bonus token if none is rejected) from the verifier.