Draft with Diffusion, Verify with Autoregressive Models

Zicong Cheng [1 2 3] **Guo-Wei Yang** [2] **Jia Li** [1] **Zhijie Deng** [3] **Meng-Hao Guo**✉ [1] **Shi-Min Hu** [1]

Abstract

Efficiency, as a critical practical challenge for LLM-driven agentic and reasoning systems, is increasingly constrained by the inherent latency of autoregressive (AR) decoding. Speculative decoding mitigates this cost through a draft–verify scheme, yet existing approaches rely on AR draft models (*a.k.a.,* drafters), which introduce two fundamental issues: (1) step-wise uncertainty accumulation leads to a progressive collapse of trust between the target model and the drafter, and (2) inherently sequential decoding of AR drafters. Together, these factors cause limited speedups. In this paper, we show that a diffusion large language model (dLLM) drafters can naturally overcome these issues through its fundamentally different probabilistic modeling and efficient parallel decoding strategy. Building on this insight, we introduce **DEER**, an efficient speculative decoding framework that drafts with diffusion and verifies with AR models. To enable high-quality drafting, DEER employs a two-stage training pipeline to align the dLLM-based drafters with the target AR model, and further adopts single-step decoding to generate long draft segments. Experiments show DEER reaches draft acceptance lengths of up to 32 tokens, far surpassing the 10 tokens achieved by EAGLE-3. Moreover, on HumanEval with Qwen3-30B-A3B, DEER attains a 5.54× speedup, while EAGLE-3 achieves only 2.41×. Code, model, demo, etc, will be available at https://czc726.github.io/DEER/

1. Introduction

Large language models (LLMs) have fundamentally reshaped the modern AI ecosystem, owing to their remarkable generalization (Chen et al., 2025; Le et al., 2024; Zhang et al., 2025). Meanwhile, as the demand for extended context continues to rise, particularly in complex reasoning and agentic tasks, efficiency becomes increasingly critical. To mitigate this growing bottleneck while preserving model fidelity, speculative decoding (Leviathan et al., 2023; Chen et al., 2023) has emerged as an effective approach for efficient decoding, providing lossless acceleration by enabling lightweight drafters to propose candidate continuations that are verified by the target model.

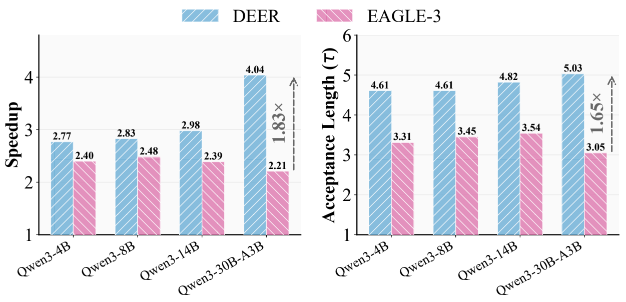


Figure 1. Performance Comparison of DEER and EAGLE-3: Speedup and τ Across Models (tokens/s) at Temperature=0

However, existing speculative decoding methods overwhelmingly rely on AR drafters, which impose two structural limitations. On the one hand, left-to-right decoding induces step-wise uncertainty accumulation, where uncertainties in early draft tokens propagate through the sequence, progressively degrading alignment with the target model and sharply limiting acceptance length. We term this phenomenon as gradual **collapse of trust** between the drafter and the target model. As illustrated in Figure 2, when the drafter conditions on its own unverified outputs, even a small discrepancy from the target model at early positions is recursively amplified through left-to-right decoding. The draft trajectory gradually drifts outside the acceptance region, causing the verifier to reject increasing portions of the draft. On the other hand, AR drafters themselves must decode sequentially, preventing them from exploiting parallel generation. Together, above structural limitations become the bottleneck that restricts attainable speedups.

We address above two challenges by introducing DEER, a novel framework that utilizes discrete-space dLLMs as

[1]Tsinghua University [2]Proxseer Inc [3]Shanghai Jiao Tong University. Correspondence to: Meng-Hao Guo <gmh@tsinghua.edu.cn>.

Preprint. December 10, 2025.

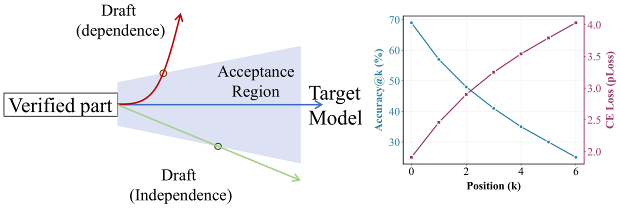


Figure 2. **Left**: Comparison between the dependence-based and independence-based drafting strategies. T **Right**: Accuracy@k and cross-entropy loss across intermediate checkpoints of the EAGLE-3 training pipeline. Together, the plots illustrate draft–backbone alignment and the progression of training performance.

efficient draft generators. Ideally, unlike AR drafters, a dLLM-based drafter can naturally generate an entire sequence of tokens in a single denoising process, thereby not only overcoming the efficiency limitations of serial AR decoding, but also theoretically eliminating the multi-step uncertainty accumulation inherent in AR sequential generation. However, naïvely applying dLLMs as drafters for speculative decoding bring serious distribution mismatch, as standard dLLMs are trained for global sequence generation rather than prefix-conditioned local continuation, leading draft proposals are misaligned with the target AR distribution.

To address this challenge, we develop a two-stage training strategy that adapts pretrained dLLMs into efficient and high-fidelity blockwise drafters. Stage-1, *AR-Style Continuation Distillation*, aligns the diffusion model with the target AR distribution by training on truncated teacher answers appended with a special SEP marker, enabling stable prefix-conditioned continuation. Stage-2, *Prefix-Conditioned Accuracy Refinement*, enhances continuation fidelity through weighted suffix masking with an exponentially decaying loss, improving token-level stability near the AR verification boundary. Combined, these stages produce a dLLM-based drafter that supports reliable blockwise generation. Furthermore, we observe above training pipeline unlocks an emergent capability we term reliable block regeneration: the ability of the dLLM to repeatedly accept partially masked suffixes and regenerate them in a coherent manner.

During inference, DEER supports one-step block drafting, eliminating left-to-right dependency and enabling significantly longer accepted drafts. Across various code-generation benchmarks, DEER achieves acceptance lengths of up to 32 tokens, far exceeding the 10 tokens typical of advanced methods such as EAGLE-3. On HumanEval with Qwen3-30B-A3B, DEER delivers a 5.54× speedup, surpassing the 2.41× speedup of EAGLE-3 and establishing dLLM-based drafting as a promising path for acceleration.

In summary, our key contributions are as follows:

- We introduce DEER, the first speculative decoding framework that relies *exclusively* on a discrete-space dLLM as the drafter, removing the need for auxiliary AR models or hybrid drafting architectures. Further, we reveal a novel generative capability in DEER, termed reliable block regeneration, wherein the dLLM can perform genuinely blockwise generation from incrementally masked suffixes.

- We propose a two-stage alignment method that adapts dLLMs to the structural requirements of speculative decoding: Stage-1 resolves the distribution mismatch for prefix-conditioned continuation, while Stage-2 provides fine-grained local accuracy through exponentially weighted suffix masking.

- Experiments on diverse benchmarks and model scales (Qwen3–4B to 30B), DEER consistently outperforms existing approaches. For instance, on HumanEval with Qwen3-30B-A3B, DEER attains a 5.54× speedup, while EAGLE-3 achieves only 2.41×.

2. Related Work

2.1. Speculative Decoding

Autoregressive and Tree-based Drafting. Standard methods employ small auxiliary AR models (Leviathan et al., 2023) or manipulate hidden states to predict tree-structured continuations (e.g., Medusa (Cai et al., 2024), Hydra (Ankner et al., 2024), EAGLE series (Li et al., 2024a;b; 2025b)). While effective, these methods remain inherently sequential: draft tokens are generated left-to-right, meaning early uncertainties propagate and corrupt the draft chain. **In contrast**, DEER eliminates this serial dependency by generating a full block of tokens in a single diffusion step, preventing uncertainty accumulation even at long draft lengths.

***N*-gram and Heuristic Drafting.** Lightweight approaches like Lookahead (Fu et al., 2024) and DiffuSpec (Li et al., 2025a) construct drafts via *n*-gram matching or retrieval. While efficient, they lack global context, causing acceptance rates to drop on complex sequences. DEER leverages dLLMs' global modeling to keep drafts coherent and contextually consistent over long spans.

Diffusion-based Drafting. Speculative Diffusion Decoding (SDD) (Christopher et al., 2025) pioneered using dLLMs for drafting but relies on continuous-space, multi-step denoising. This prohibits precise temperature control and introduces step-wise drift from the AR verifier. DEER diverges by operating in discrete space with a strictly aligned one-step generation process, ensuring high compatibility with the target AR distribution.

2.2. Self-Drafting and Hybrid Architectures

Latent and Intermediate Self-Drafting. Methods such as SSDD (Gao et al., 2025) and SSMD (Campbell et al., 2025) exploit intermediate noisy states or self-generated diffusion logits to form drafts. However, these intermediate representations are often noisy and unaligned with the final autoregressive objective. DEER avoids this by utilizing a fully denoised, alignment-tuned output distribution, yielding significantly cleaner and more stable proposals.

Hybrid AR-Diffusion Training. Approaches like TiDAR (Liu et al., 2025) retrain AR models to jointly perform diffusion-style generation. While this unifies drafting and verification, the dual-objective training is computationally expensive and often induces conflict that degrades base model performance. DEER employs a modular design with a dedicated, lightweight dLLM. This avoids expensive retraining of the target LLM and preserves its original capabilities without objective conflicts.

2.3. Diffusion language models

Early work such as D3PM (Austin et al., 2021a) and Diffusion-LM (Li et al., 2022) introduced diffusion language models in continuous spaces, while later approaches like LLADA (Nie et al., 2025) and Dream (Ye et al., 2025) scaled them to discrete tokens and larger model sizes. A key benefit of diffusion models is their ability to generate multiple tokens in parallel, reducing autoregressive dependency. Leveraging this, dLLMs support coherent block-wise token generation for efficient speculative decoding.

3. DEER

Speculative decoding accelerates autoregressive (AR) inference by enabling a lightweight model to propose multiple tokens in parallel, which are then verified by the target AR model. However, existing AR-based drafters inevitably suffer from **left-to-right uncertainty accumulation** (*a.k.a.,* gradual collapse of trust between drafters and target model): each drafted token conditions on previously unverified ones, amplifying early deviations. As draft depth increases, acceptance sharply declines, limiting speedup.

DEER resolves this bottleneck by leveraging a dLLM as the drafter. Unlike AR generation, diffusion models jointly reconstruct the entire suffix, making proposal quality largely invariant to token depth. To adapt pretrained dLLMs to prefix-conditioned continuation, we propose a two-stage **Diffusion-to-Autoregressive (D2A) Alignment** pipeline, followed by a lightweight block-wise verification procedure.

3.1. Diffusion-to-AR Alignment

Diffusion models operate through global denoising and are not inherently consistent with AR-style prefix continuation. Directly employing a pretrained dLLM as a drafter causes severe distribution mismatch, producing unstable suffix predictions. To make dLLMs suitable for as drafters, D2A adapts them to prefix-conditioned continuation behavior.

Notation We denote: p_{AR}: AR teacher model; x_0: original tokens with masked future spans; x_t: noised tokens at timestep t; L: full length of a teacher-generated answer; l_q: prefix/question length; $\mathbf{M}$: mask token; `SEP`: separator token indicating truncation; p_θ: aligned dLLM.

3.1.1. Stage *I*: AR-style Distillation

A dLLM pre-trained with full-sentence denoising does not inherently model causal continuation: if the prefix is truncated, its denoising process still implicitly relies on future tokens that are no longer available. To enable prefix-conditioned generation, we finetune the model to imitate an AR teacher on continuation-style data.

Given a teacher-generated answer $\mathcal{A} = \{a_n^{1:l_n}\}$, we randomly truncate each answer, mask the suffix, and append a `SEP` token to mark the continuation boundary. The dLLM observes a noisy version x_t and is trained to denoise only the masked continuation:

$$\mathcal{L}_{\text{Distill}} = -\mathbb{E}_{t,x_0,x_t}\left[\frac{1}{t}\sum_{i=l_q}^{L-1}\mathbf{1}[x_t^i = \mathbf{M}]\, r_i\right]. \tag{1}$$

$$r_i = \log p_\theta(x_0^i \mid x_t) \tag{2}$$

This first stage adapts the dLLM to a setting where the past is observed but the future must be predicted, aligning its continuation behavior with the AR teacher and making it compatible with speculative decoding.

3.1.2. Stage *II*: Scribe Refinement

While the above training enables causal continuation, speculative acceptance is particularly sensitive to the tokens that appear immediately after the prefix. To refine accuracy precisely in this region, we mask only the last $R \sim \text{Uniform}(1, 96)$ tokens of the answer, instead of the entire suffix. Tokens closer to the prefix are emphasized with exponentially increasing weights:

$$w_i = \alpha^{R-i}, \qquad i = 1, \dots, R, \tag{3}$$

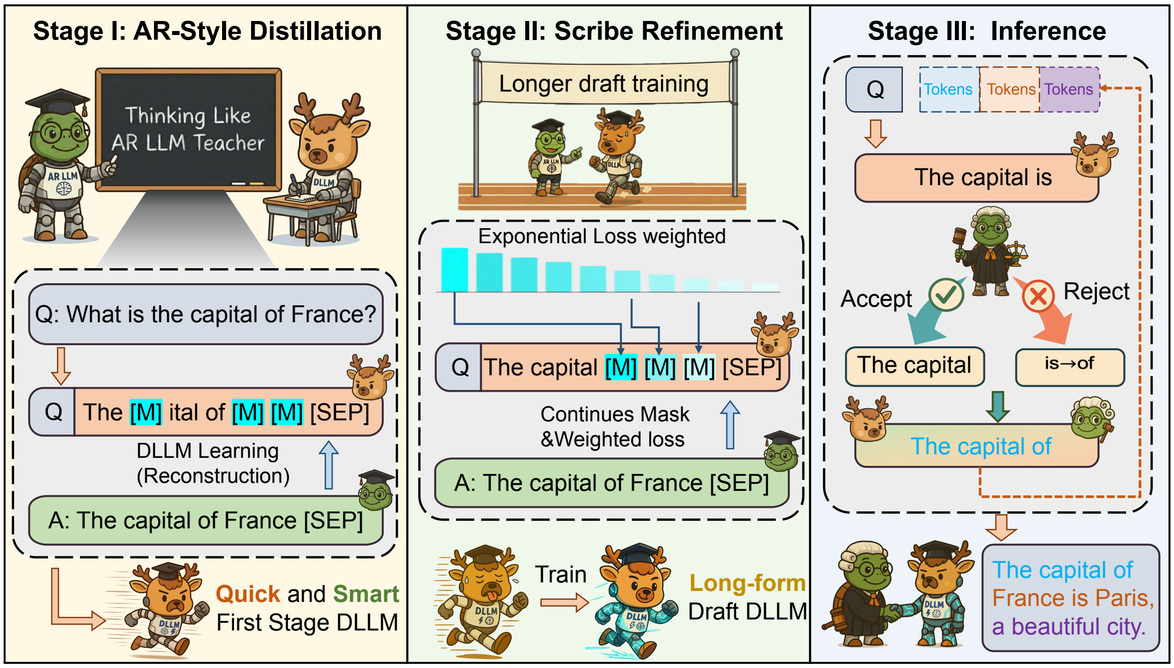

Stage I: AR-Style Distillation
Thinking Like AR LLM Teacher
AR LLM
DLLM
Q: What is the capital of France?
Q
The [M] ital of [M] [M] [SEP]
DLLM Learning (Reconstruction)
A: The capital of France [SEP]
Quick and Smart First Stage DLLM
Stage II: Scribe Refinement
Longer draft training
Exponential Loss weighted
Q
The capital [M] [M] [M] [SEP]
Continues Mask &Weighted loss
A: The capital of France [SEP]
Train
Long-form Draft DLLM
Stage III: Inference
Q
Tokens
Tokens
Tokens
The capital is
Accept
Reject
The capital
is→of
The capital of
The capital of France is Paris, a beautiful city.

Figure 3. Overview of the DEER pipeline. **Stage *I* (AR-Style Continuation Distrilling)** structurally adapts the dLLM to generate full suffix blocks from prefix + [SEP] using truncated teacher answers. **Stage *II* (Scribe Refinement)** stabilizes local coherence through weighted suffix masking with an exponential decaying loss. **Stage *III* Inference** performs draft-then-verify decoding, where large draft blocks proposed by the dLLM are accepted or corrected by the target AR model, accelerating inference while preserving quality.

leading to the refinement objective:

$$\mathcal{L}_{\text{Refine}} = -\mathbb{E}_{t,x_0,x_t}\left[\frac{1}{t}\sum_{i=l_q}^{L-1} w_i\,\mathbf{1}[x_t^i=\mathbf{M}]\,r_i\right]. \tag{4}$$

Through this masked-span curriculum, the dLLM increasingly concentrates capacity on the region where speculative verification first interacts with the draft, leading to more reliable block acceptance.

3.2. Inference

At inference time, we use the aligned dLLM as a parallel drafter and an AR model as an exact verifier within a blockwise speculative decoding scheme. Given a current prefix $\mathbf{x}_{1:j}$, the dLLM proposes a block of k tokens in parallel: $\hat{\mathbf{y}}_{j+1:j+k} \sim q_\theta(\cdot \mid \mathbf{x}_{1:j})$, and the AR model then decides, token by token, whether to accept each proposal.

For the i-th token in the block, we compute an acceptance probability

$$\alpha_i = \min\left(1,\ \frac{p_{\text{AR}}(\hat{y}_{j+i} \mid \mathbf{x}_{1:j+i-1})}{q_\theta(\hat{y}_{j+i} \mid \mathbf{x}_{1:j})}\right), \quad i = 1, \dots, k. \tag{5}$$

With probability α_i the draft token is accepted; otherwise it is replaced by an AR sample:

$$\hat{y}_{j+i} \propto \max\Big(0,\ p_{\text{AR}}(\cdot \mid \mathbf{x}_{1:j+i-1}) - q_\theta(\cdot \mid \mathbf{x}_{1:j})\Big), \tag{6}$$

In both cases, the chosen token (either the accepted draft or the AR resample) is appended to the prefix and becomes part of the context for subsequent positions.

Uncertainty accumulation vs. stable block proposals. In classical speculative decoding with an autoregressive drafter, the draft distribution at position i depends on previously sampled draft tokens: $q^{\text{AR}}(\hat{y}_i \mid \mathbf{x}_{1:j}, \hat{\mathbf{y}}_{1:i-1}) \neq q^{\text{AR}}(\hat{y}_i \mid \mathbf{x}_{1:j})$, so any divergence between the drafter and p_{AR} at early positions propagates to later ones. As a result, the distribution mismatch $\text{KL}\big(p_{\text{AR}}(\hat{y}_i \mid \mathbf{x}_{1:j+i-1}) \,\big\|\, q^{\text{AR}}(\hat{y}_i \mid \mathbf{x}_{1:j}, \hat{\mathbf{y}}_{1:i-1})\big)$ tends to grow with i, leading to left-to-right uncertainty accumulation and rapidly decreasing acceptance rates.

In contrast, the dLLM uses block-wise masked conditioning, which yields $q_\theta(\hat{y}_i \mid \mathbf{x}_{1:j}, \hat{\mathbf{y}}_{1:i-1}) = q_\theta(\hat{y}_i \mid \mathbf{x}_{1:j})$, so the proposal at position i is independent of previously drafted tokens. The mismatch between p_{AR} and q_θ at depth i is therefore determined solely by how well $q_\theta(\cdot \mid \mathbf{x}_{1:j})$ matches $p_{\text{AR}}(\cdot \mid \mathbf{x}_{1:j+i-1})$, rather than by accumulated errors in earlier drafts. In other words, the drafter does not *amplify* its own past mistakes, which is the primary source of degradation in AR-based speculative decoding.

Overall decoding procedure. Putting these pieces together, decoding proceeds by alternating between (i) parallel block proposals from the dLLM and (ii) token-wise validation and correction by the AR model. Most computation

Table 1. Performance Comparison of Acceleration Methods Across Models (temperature=0.6), with KV cache

Method	MBPP		CodeAlpacaPy		HumanEval		LiveCodeBench		Mean	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B										
EAGLE3	×1.98	2.60	×1.86	2.43	×2.01	2.63	×1.79	2.51	×1.91	2.54
DEER	×2.82	4.79	×2.51	4.10	×2.59	4.51	×2.14	4.25	×2.52	4.41
Qwen3-8B										
EAGLE3	×2.12	3.46	×1.55	2.58	×2.18	3.23	×1.33	2.42	×1.80	2.92
DEER	×3.35	4.84	×2.40	3.81	×3.18	4.66	×1.59	2.93	×2.63	4.06
Qwen2-7B										
EAGLE3	×2.40	3.28	×2.18	3.23	×2.41	3.25	×2.39	2.91	×2.35	3.18
DEER	×2.45	3.81	×2.27	3.50	×2.52	3.90	×2.84	4.39	×2.52	3.90
Qwen3-14B										
EAGLE3	×1.89	2.22	×2.01	2.44	×1.91	2.61	×1.75	2.30	×1.89	2.39
DEER	×3.67	4.93	×2.80	3.69	×3.50	5.00	×2.53	3.93	×3.13	4.39
Qwen3-30B-A3B										
EAGLE3	×2.07	2.35	×1.91	2.31	×2.14	2.57	×1.90	2.38	×2.01	2.40
DEER	×3.69	4.79	×3.23	3.82	×4.32	5.48	×3.24	4.09	×3.62	4.45

Algorithm 1 DEER Inference

```
Input: prefix $\mathbf{x}_{1:j}$, block size $k$
while not EOS and length limit not reached do
    Sample draft block $\hat{\mathbf{y}}_{j+1:j+k} \sim q_\theta(\cdot \mid \mathbf{x}_{1:j})$
    for $i = 1 \dots k$ do
        Compute $\alpha_i$ via Eq. 5
        Sample $u \sim \mathrm{Uniform}(0, 1)$
        if $u \leq \alpha_i$ then
            accept $\hat{y}_{j+i}$ and set $\mathbf{x}_{1:j+i} \leftarrow \mathbf{x}_{1:j+i-1} \circ \hat{y}_{j+i}$
        else
            sample $\hat{y}_{j+i} \sim p_{\mathrm{AR}}(\cdot \mid \mathbf{x}_{1:j+i-1})$ via Eq. 6 and
            set $\mathbf{x}_{1:j+i} \leftarrow \mathbf{x}_{1:j+i-1} \circ \hat{y}_{j+i}$
        end if
        if $\hat{y}_{j+i}$ is EOS then
            break
        end if
    end for
    Update $j \leftarrow \mathrm{length}(\mathbf{x})$
end while
```

is shifted to the parallelizable drafting step, while the AR model is used only for lightweight verification to guarantee exact marginal correctness. The full inference routine is summarized in Algorithm 1.

4. Experiment

In this section, we conduct extensive experiments to evaluate the effectiveness and generalizability of DEER. Our empirical study is organized around the following research questions:

- **RQ1:** How does DEER perform on code generation tasks in terms of inference efficiency and draft acceptance distribution?
- **RQ2:** What is the impact of the Stage *II* and how sensitive is it to hyperparameter choices?
- **RQ3:** How does DEER support batch inference scalability?
- **RQ4:** Does DEER endow dLLMs with new generative capabilities beyond standard denoising?
- **RQ5:** How does DEER perform on mathematical reasoning benchmarks?

4.1. Experimental Settings

Datasets. For code generation, we train our draft model on the OpenCodeInstruct dataset (Ahmad et al., 2025) and evaluate on HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b), LiveCodeBench (Jain et al., 2024), and the Python subset of CodeAlpaca (Lhoest et al., 2021).For the math reasoning setting, we train using data from UltraChat (Ding et al., 2023) and ShareGPT (shareAI, 2023), and evaluate on GSM8K (Cobbe et al., 2021), Math500 (Lightman et al., 2023), and Minerva Math (Lewkowycz et al., 2022).

Models.For code generation experiments, we adopt Open-dLLM (Peng et al., 2025) as our base diffusion model and apply our continued-training procedure to obtain the draft model.For math reasoning, we start with Qwen2.5-0.5B-Instruct (Yang et al., 2024; Team, 2024), modify it with a diffusion decoding head, and then apply our two-stage continued training to derive the corresponding draft model.

Baselines.We compare DEER with state-of-the-art speculative decoding methods for which official training code is publicly available, including Medusa (Cai et al., 2024), Hydra (Ankner et al., 2024), and EAGLE-3 (Li et al., 2025b).

Metrics. Since DEER preserves the original model weights and employs strict rejection sampling to guarantee lossless speculative decoding, we do not report accuracy-based

Table 2. Performance Comparison of Acceleration Methods Across Models (tokens/s), temperature=0, with KV cache

Method	MBPP		CodeAlpacaPy		HumanEval		LiveCodeBench		Mean	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3-4B										
MEDUSA	×1.20	2.07	×1.22	1.85	×1.34	1.94	×1.17	1.89	×1.23	1.94
Hydra	×2.33	2.70	×2.17	2.50	×2.30	2.53	×2.08	2.54	×2.22	2.58
EAGLE3	×2.59	3.37	×2.20	3.11	×2.47	3.26	×2.33	3.50	×2.40	3.31
DEER	×2.86	4.91	×2.58	4.21	×2.97	4.68	×2.67	4.63	×2.77	4.61
Qwen3-8B										
MEDUSA	×1.29	1.93	×1.33	1.98	×1.32	1.97	×1.34	1.99	×1.32	1.97
Hydra	×2.37	2.79	×2.02	2.53	×2.39	2.58	×2.23	2.72	×2.25	2.66
EAGLE3	×2.59	3.31	×2.21	3.25	×2.65	3.87	×2.46	3.38	×2.48	3.45
DEER	×3.00	5.12	×2.35	4.06	×3.30	5.00	×2.67	4.27	×2.83	4.61
Qwen2-7B										
EAGLE3	×2.36	3.32	×2.40	3.27	×2.65	3.34	×2.29	2.94	×2.43	3.22
DEER	×2.45	3.57	×2.50	3.69	×2.79	4.21	×3.06	4.96	×2.70	4.11
Qwen3-14B										
EAGLE3	×2.50	3.52	×2.13	3.45	×2.62	3.72	×2.30	3.48	×2.39	3.54
DEER	×3.18	5.28	×2.43	3.98	×3.59	5.72	×2.73	4.31	×2.98	4.82
Qwen3-30B-A3B										
EAGLE3	×2.22	3.07	×2.06	2.89	×2.41	3.21	×2.14	3.01	×2.21	3.05
DEER	×4.00	4.87	×3.08	4.04	×5.54	6.58	×3.52	4.62	×4.04	5.03

Table 3. Average accepted-token lengths on Qwen3-30B-A3B with and without Stage II refinement.

Benchmark	w/o Refinement	w/ Refinement
MBPP	4.74	4.87
CodeAlpacaPy	3.47	4.04
HumanEval	5.38	6.58
LiveCodeBench	3.87	5.03

metrics. Following standard practice in prior work on speculative decoding, we evaluate using two key metrics:

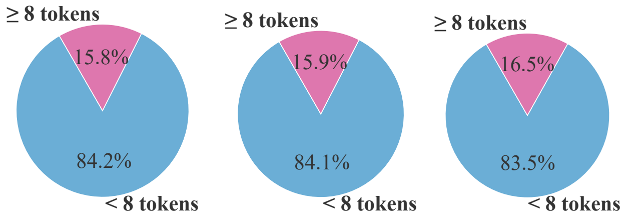
- **Speedup Ratio**. The empirical end-to-end speedup compared with standard autoregressive decoding.
- **Average Acceptance Length** (τ). The average number of tokens accepted from the draft per drafting–verification cycle, reflecting the effective number of tokens generated in each speculative step.

4.2. Performance on Code Generation (RQ1)

In this section, DEER is used with KV cache enabled.

4.2.1. Overall Efficiency

As shown in Tables 1 and 2, DEER consistently outperforms state-of-the-art speculative decoding methods across all model scales and datasets, in both average acceptance length τ and end-to-end speedup.



(a) Qwen3-4B *(b)* Qwen3-8B. *(c)* Qwen3-14B.

Figure 4. Proportion of short (<8 tokens) and long (≥8 tokens) accepted tokens for different model.

For Qwen3-30B-A3B at temperature = 0, DEER achieves an average acceptance length of 5.03, a 67% increase over EAGLE-3 (τ = 3.05). This demonstrates that speculative decoding can remain highly effective even for modern large models with complex, high-entropy vocabularies. Prior methods are constrained by cumulative left-to-right draft errors, which sharply limit acceptance lengths on such models. In contrast, DEER greatly mitigates this error accumulation through its one-step block-generation mechanism, yielding a substantially more stable verification process.

Across model families, DEER improves τ by 50–120% relative to EAGLE-3, depending on model scale and task. On HumanEval, the acceptance length of Qwen3-30B-A3B under DEER is more than twice that of EAGLE-3 (6.58 vs. 3.21), and this longer acceptance directly translates into faster decoding: DEER achieves up to 2× the speedup of EAGLE-3 under the same setting. Overall, these results

Table 4. Maximum accepted token lengths across models.

Model	EAGLE-3	DEER
Qwen3-4B	8	**32**
Qwen3-8B	8	**32**
Qwen3-14B	8	**32**
Qwen3-30B-A3B	7	**32**

Table 5. Batch inference performance (tokens/s) on HumanEval across different batch sizes.

Method	Batch 2	Batch 4	Batch 8	Batch 16
AR	34.03	32.50	38.35	49.76
DEER	82.97	103.95	159.87	175.66

indicate that controlling draft error accumulation is crucial for achieving robust acceleration on contemporary LLMs, and that DEER provides a practical way to do so.

4.2.2. Acceptance Distribution Mechanism

To understand the performance gains, we examine how DEER reshapes the distribution of accepted token lengths, which directly reflects the model's ability to propose longer drafts for parallel verification. As shown in Figure 4, the probability of accepting drafts longer than 8 tokens consistently exceeds 15% across all tested models, and this probability slightly increases with model scale, from 15.8% for Qwen3-4B to 16.6% for Qwen3-30B-A3B. Table 4 further shows DEER attains a maximum acceptance length of 32 tokens, whereas EAGLE-3 is limited to 7–8 tokens.

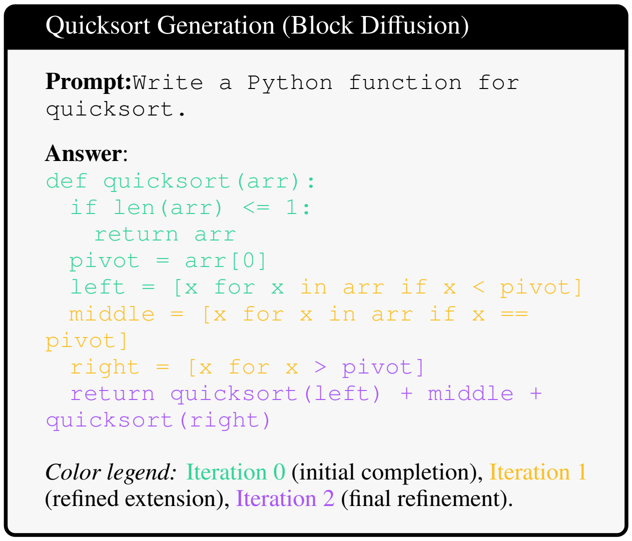
These distributions confirm that the one-step generation in DEER effectively decouples token dependencies within each draft block, substantially reducing the left-to-right uncertainty accumulation that typically constrains speculative decoding. As a result, DEER can reliably produce longer contiguous blocks of accepted tokens, leading to higher throughput and greater acceleration potential than conventional autoregressive drafters.

4.3. Ablation Study and Sensitivity Analysis (RQ2)

In this section, we evaluate the role of Stage *II* in shaping the dLLM into a more reliable draft generator and analyze the sensitivity of its key hyperparameters.

4.3.1. Impact of Stage *II*

We measure the effect of Stage *II* by comparing the average accepted-token lengths of models trained with and without refinement (Table 3). Enabling Stage *II* consistently increase the number of accepted tokens across all four code-generation benchmarks: from 4.74 to 4.87 on MBPP, 3.47 to 4.04 on CodeAlpacaPy, 5.38 to 6.58 on HumanEval, and 3.87 to 5.03 on LiveCodeBench.



Quicksort Generation (Block Diffusion)

Prompt: Write a Python function for quicksort.

Answer:

```
def quicksort(arr):
  if len(arr) <= 1:
    return arr
  pivot = arr[0]
  left = [x for x in arr if x < pivot]
  middle = [x for x in arr if x ==
pivot]
  right = [x for x > pivot]
  return quicksort(left) + middle +
quicksort(right)
```

Color legend: Iteration 0 (initial completion), Iteration 1 (refined extension), Iteration 2 (final refinement).

Figure 5. Illustration of block-diffusion generation. Different colors represent tokens produced at successive denoising iterations, showing that the dLLM can extend partial code blocks without requiring a full-sentence prompt.

The gap between the two settings grows with benchmark difficulty, with the largest reductions on HumanEval and LiveCodeBench (1.20 and 1.16 tokens). This suggests that the refinement stage encourages the dLLM to produce suffixes that are more tightly aligned with the AR teacher, especially in settings with more complex or long-range structure, resulting in more precise and reliable drafts.

4.3.2. Hyperparameter Sensitivity

Stage *II* introduces position-dependent weights parameterized by a scaling factor α, controling how strongly the loss emphasizes the most recent masked tokens. This exponential weighting makes training sensitive to the choice of α.

As shown in Figure 6, when $\alpha = 1.01$, optimization is stable and the loss decreases smoothly. Increasing α to 1.02 yields noticeably noisier training curves with slight upward drift, and setting $\alpha = 1.05$ leads to early divergence. These results indicate that Stage *II* has a relatively narrow stability window: overly aggressive weighting amplifies gradients near the masked boundary and destabilizes optimization. Within the stable regime, however, the refinement stage consistently improves suffix alignment without requiring additional data or extended training.

4.4. Batch Inference Scalability (RQ3)

We evaluate DEER's batch inference performance on HumanEval by measuring throughput (tokens/s) under different

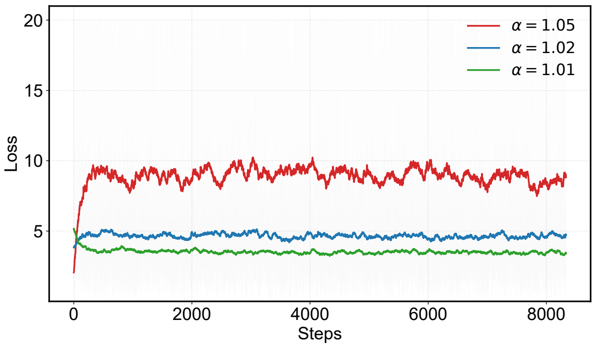


Figure 6. Sensitivity analysis of the exponential weighting factor α during Stage *II*.The figure illustrates how different values of the exponential weighting coefficient $\alpha \in \{1.01, 1.02, 1.05\}$ influence the loss trajectory throughout the Quench Refinement process. Each curve is smoothed using an exponential moving average to highlight the underlying optimization dynamics while suppressing stochastic noise.

Table 6. Performance of acceleration methods on the **Qwen3-30B-A3B** model (temperature = 0.6) across math benchmarks. Reported metrics are speedup (×) and Kendall's τ correlation.

Metric	**EAGLE3**	**DEER**
	Math500	
Speedup	×1.89	×2.12
τ	2.04	2.45
	GSM8K	
Speedup	×1.92	×2.23
τ	2.43	2.70
	Minerva Math	
Speedup	×1.91	×2.02
τ	2.07	2.31
	Mean (across benchmarks)	
Speedup	×1.91	×2.12
τ	2.18	2.47

batch sizes. Since there is currently no mature framework for efficient dLLM+KV-cache deployment, both the baseline autoregressive decoding and DEER are run *without* KV-cache, so the comparison reflects raw model.

Table 5 reports results for batch sizes 2, 4, 8 and 16. Across all settings, DEER yields substantial speedups over the autoregressive baseline and scales well with batch size. For example, at batch size 8, DEER reaches 159.87 tokens/s, nearly 4× the baseline throughput of 38.35 tokens/s.

These results show that improvements in per-step acceptance length translate into batch-level acceleration: one-step draft generation combined with parallel verification allows DEER to better exploit GPU parallelism, especially at larger batch sizes. Modest efficiency gaps at smaller batch sizes are mainly due to fixed speculative-decoding overheads, which become negligible as batch size increases.

4.5. Generative Capabilities (RQ4)

We observe an interesting emergent behavior in dLLMs trained with our DEER: the models are able to perform *reliable block regeneration*. As shown in Figure 5, the model can extend an incomplete code segment purely from its local prefix, generating new tokens in a diffusion-style manner without requiring a full-sentence prompt.

Different colors in Figure 5 indicate tokens produced at different denoising iterations, highlighting how the model incrementally refines and extends the partial block. This illustrates that DEER enables dLLMs to treat block-level continuation as a natural generation mode, even without any architectural modifications such as padding tokens or altered attention patterns.

4.6. Performance on Mathematical Reasoning (RQ5)

Since no pretrained dLLMs are publicly available for mathematical-reasoning tasks, we construct our draft model by converting Qwen2.5-0.5B-Instruct into a diffusion model and training it for 40 epochs on the UltraChat dataset (Ding et al., 2023). This yields only a partially converged dLLM—its standalone generations are not yet semantically reliable—yet DEER still achieves consistent acceleration improvements on mathematical benchmarks.

Table 6 shows that, despite the weakly trained draft model, DEER surpasses EAGLE-3 across all datasets. On Math500, DEER improves speedup from 1.89× $\rightarrow$ 2.12× (+12.2%) and increases the acceptance length from 2.04 $\rightarrow$ 2.45 (+20.1%). Similar gains hold for GSM8K (speedup 1.92× $\rightarrow$ 2.23×, τ 2.43 $\rightarrow$ 2.70) and Minerva Math (speedup 1.91× $\rightarrow$ 2.02×, τ 2.07 $\rightarrow$ 2.31). Averaged over the three datasets, DEER delivers a mean speedup of 2.12×, outperforming EAGLE-3's 1.91×, with an average acceptance length of 2.47, compared to 2.18 for EAGLE-3.

These results demonstrate that DEER generalizes beyond code generation and remains effective even when the underlying dLLM is far from convergence. This suggests that the one-step drafting mechanism produces stable, high-quality proposals for the verifier, enabling reliable speculative decoding in domains where fully trained diffusion-based LLMs are not yet available.

5. Conclusion

We presented **DEER**, a speculative decoding framework that uses a discrete dLLM as the sole drafter, avoiding the left-to-right uncertainty accumulation of autoregressive drafters.

To make dLLMs suitable for prefix-conditioned continuation, we introduced a *Diffusion-to-AR Alignment* pipeline that combines AR-style distillation with a lightweight refinement stage near the prefix boundary. On multiple code-generation benchmarks and model scales, DEER yields longer accepted blocks and consistent speedups, even without KV caching, demonstrating dLLMs as a practical, highly parallelizable alternative for efficient LLM decoding.

References

Ahmad, W. U., Ficek, A., Samadi, M., Huang, J., Noroozi, V., Majumdar, S., and Ginsburg, B. Opencodeinstruct: A large-scale instruction tuning dataset for code llms. 2025. URL https://arxiv.org/abs/2504.04030.

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

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

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. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

Campbell, A., Bortoli, V. D., Shi, J., and Doucet, A. Self-speculative masked diffusions. *CoRR*, abs/2510.03929, 2025.

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

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021.

Chen, Z., Liu, T., Tian, M., Tong, Q., Luo, W., and Liu, Z. Advancing mathematical reasoning in language models: The impact of problem-solving data, data synthesis methods, and training stages. In *International Conference on Learning Representations*, 2025. URL https://api.semanticscholar.org/CorpusID:275907001.

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 *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 12042–12059. Association for Computational Linguistics, 2025.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ding, N., Chen, Y., Xu, B., Qin, Y., Zheng, Z., Hu, S., Liu, Z., Sun, M., and Zhou, B. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of LLM inference using lookahead decoding. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

Gao, Y., Ji, Z., Wang, Y., Qi, B., Xu, H., and Zhang, L. Self speculative decoding for diffusion large language models. *CoRR*, abs/2510.04147, 2025.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. URL https://api.semanticscholar.org/CorpusID:261697361.

Le, H., Sahoo, D., Zhou, Y., Xiong, C., and Savarese, S. Indict: Code generation with internal dialogues of critiques for both security and helpfulness. *Advances in Neural Information Processing Systems*, 37:85546–85582, 2024.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V. V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., and Misra, V. Solving quantitative reasoning problems with language models. *ArXiv*, 2022.

Lhoest, Q., Villanova del Moral, A., Jernite, Y., Thakur, A., von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., Davison, J., Šaško, M., Chhablani, G., Malik, B., Brandeis, S., Le Scao, T., Sanh, V., Xu, C., Patry, N., McMillan-Major, A., Schmid, P., Gugger, S., Delangue, C., Matussière, T., Debut, L., Bekman, S., Cistac, P., Goehringer, T., Mustar, V., Lagunas, F., Rush, A., and Wolf, T. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, November 2021.

Li, G., Fu, Z., Fang, M., Zhao, Q., Tang, M., Yuan, C., and Wang, J. Diffuspec: Unlocking diffusion language models for speculative decoding. *CoRR*, abs/2510.02358, 2025a.

Li, X., Thickstun, J., Gulrajani, I., Liang, P. S., and Hashimoto, T. B. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35:4328–4343, 2022.

Li, Y., Wei, F., Zhang, C., and Zhang, H. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024a.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees. In *Conference on Empirical Methods in Natural Language Processing*, 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*, 2025b.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liu, J., Dong, X., Ye, Z., Mehta, R., Fu, Y., Singh, V., Kautz, J., Zhang, C., and Molchanov, P. Tidar: Think in diffusion, talk in autoregression. 2025. URL https://api.semanticscholar.org/CorpusID:282940200.

Ma, Y., Du, L., Wei, L., Chen, K., Xu, Q., Wang, K., Feng, G., Lu, G., Liu, L., Qi, X., Zhang, X., Tao, Z., Feng, H., Jiang, Z., Xu, Y., Huang, Z., Zhuang, Y., Xu, H., Hu, J., Lan, Z., Zhao, J., Li, J., and Zheng, D. dinfer: An efficient inference framework for diffusion language models. *CoRR*, abs/2510.08666, 2025.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. *ArXiv*, 2025.

Peng, F. Z., Zhang, S., Tong, A., and contributors. Open-dllm: Open diffusion large language models. https://github.com/pengzhangzhi/Open-dLLM, 2025. Blog: https://oval-shell-31c.notion.site/..., Model: https://huggingface.co/fredzzp/open-dcoder-0.5B.

shareAI. Sharegpt-chinese-english-90k bilingual human-machine qa dataset. https://huggingface.co/datasets/shareAI/ShareGPT-Chinese-English-90k, 2023.

Team, Q. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. *CoRR*, abs/2505.22618, 2025.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., and Fan, Z. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Zhang, Y., Ni, B., Chen, X.-S., Zhang, H.-R., Rao, Y., Peng, H., Lu, Q., Hu, H., Guo, M.-H., and Hu, S.-M. Bee: A high-quality corpus and full-stack suite to unlock advanced fully open mllms, 2025. URL `https://arxiv.org/abs/2510.13795`.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C. W., and Sheng, Y. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

A. Details of Experimental Settings

All experiments are conducted on a cluster equipped with eight NVIDIA A100 GPUs with 80 GB of memory each.

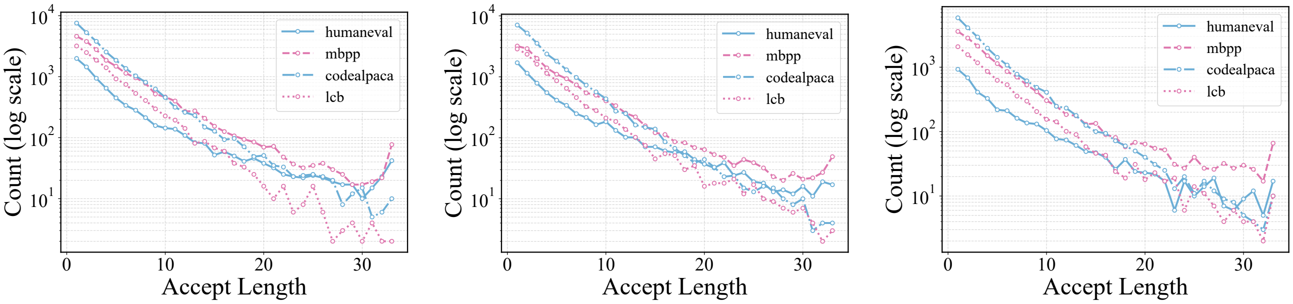
Draft model for code tasks. For code generation benchmarks, we use a 0.5B-parameter diffusion drafter obtained by modifying the open-dCoder checkpoint into a discrete diffusion language model. In Stage *I* (AR-style continuation distillation), we optimize the drafter with the AdamW optimizer, using a learning rate of 1×10^{-4} and training for 1 epoch over the code training corpus. In Stage *II* (prefix-conditioned refinement), we continue training the same drafter with AdamW, a learning rate of 5×10^{-5}, and 1 epoch on a subset of 100k examples.

Draft model for math tasks. For mathematical reasoning benchmarks, we also use a 0.5B-parameter drafter, initialized from Qwen2.5-0.5B. We convert the original autoregressive checkpoint into a diffusion language model and train it on the UltraChat dataset (Ding et al., 2023) for 40 epochs. In Stage *I*, we apply AR-style continuation distillation with AdamW, a learning rate of 1×10^{-4}, and train for 5 epochs. In Stage *II*, we perform the refinement stage with AdamW, using a learning rate of 1×10^{-4} for 1 additional epoch.

B. More Sensitivity Analysis and Experiment

B.1. Details of Accept lenth

As shown in Figure 7, we observe an intriguing pattern in the empirical acceptance-length distribution. For accepted lengths below 30, DEER exhibits an approximately exponential decay, which is consistent with the behavior reported for most speculative decoding methods. However, once the acceptance length exceeds 30 and approaches the maximum range, the probability mass starts to increase again, and this resurgence is quite pronounced. We refer to this phenomenon as the *long-block resurgence effect*. We argue that this effect provides further evidence for our motivation regarding uncertainty accumulation: when later draft tokens are no longer conditioned on earlier draft tokens from the drafter, they are less exposed to left-to-right error propagation and can therefore support substantially longer accepted drafts.

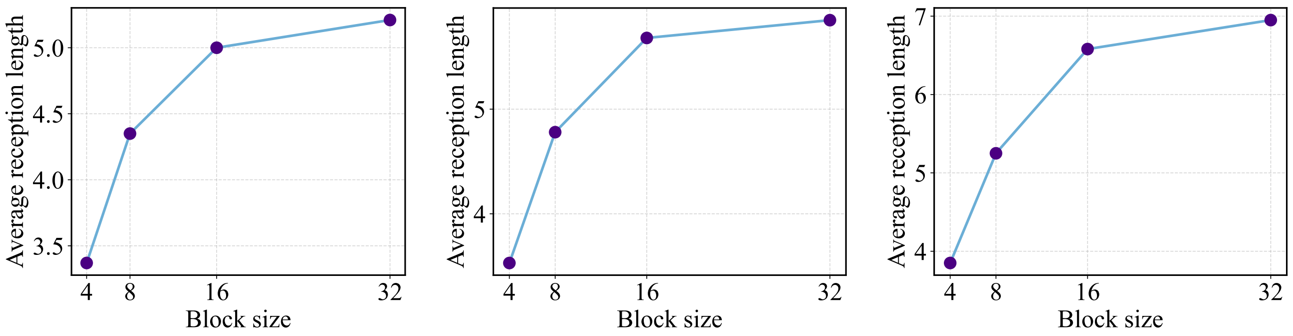


(a) Qwen3-8B *(b)* Qwen3-14B *(c)* Qwen3-30B-A3B

Figure 7. Token-length distribution of accuracy differences across four code benchmarks for Qwen3 models. (a)–(c) correspond to Qwen3-8B, Qwen3-14B, and Qwen3-30B-A3B, respectively. The x-axis denotes the number of tokens and the y-axis shows the frequency (log scale).

B.2. Sensitivity Analysis of block size

As shown in Figure 8, the average acceptance length consistently increases with larger block sizes, while the growth rate gradually slows down, indicating a trade-off between longer blocks and the corresponding computation overhead. Furthermore, the scaling of the backbone models significantly enhances the acceptance behavior. Across block sizes from 4 to 32, Qwen3-14B improves acceptance length by **5%–14%** over Qwen3-8B, while Qwen3-30B-A3B delivers further gains of **14%–33%**. We attribute this improvement to stronger output determinism in larger base models, which enables DLLM to more effectively fit deterministic token patterns. Therefore, we expect our acceleration method to yield even greater benefits when deployed on larger-scale LLMs.



(a) Qwen3-8B

(b) Qwen3-14B

(c) Qwen3-30B-A3B

Figure 8. Block-size distribution of average acceptance lengths for Qwen3 models evaluated on the HumanEval benchmark. Subfigures (a)–(c) correspond to Qwen3-8B, Qwen3-14B, and Qwen3-30B-A3B, respectively. The x-axis represents the block size, while the y-axis denotes the average acceptance length.

Table 7. Comparison of drafter sizes between EAGLE-3 and DEER under matched target models.

Target model	Target params (B)	Method	Drafter type	Drafter params (M)
Qwen3-8B	8.0	EAGLE-3	AR draft head	400
		DEER	discrete DLLM	470
Qwen3-14B	14.0	EAGLE-3	AR draft head	610
		DEER	discrete DLLM	470
Qwen3-30B-A3B	30.0	EAGLE-3	AR draft head	140
		DEER	discrete DLLM	470

Note: Drafter parameter counts for EAGLE-3 are taken from the original paper. DEER uses a 0.5B-parameter discrete diffusion model as the drafter for all target models.

C. Additional Analysis of Drafter Size

One potential concern is that the gains of DEER might stem from using a much larger drafter than prior speculative decoding methods such as EAGLE-3, rather than from the proposed drafting strategy itself. To examine this factor, we compare the parameter scales of the drafters used by EAGLE-3 and DEER under the same target backbones. The results are summarized in Table 7.

As shown in Table 7, the drafter sizes of EAGLE-3 and DEER are of the same order of magnitude (hundreds of millions of parameters) and remain well below the corresponding target model sizes. For the 8B backbone, DEER uses a slightly larger drafter than EAGLE-3 (470M vs. 400M), whereas for the 14B backbone EAGLE-3 employs an even larger drafter (610M vs. 470M). For the 30B backbone, DEER again uses a larger drafter (470M vs. 140M), but the drafter still accounts for only a small fraction of the 30B target. Under these matched-capacity regimes, DEER nevertheless achieves substantially longer average acceptance lengths (see Table 2) and higher maximum accepted block sizes (see Table 4) than EAGLE-3. This indicates that our improvements primarily come from the discrete diffusion drafting mechanism and the proposed alignment pipeline, rather than from simply scaling up the drafter model.

D. Training Cost Comparison

We further compare the fine-tuning cost of different speculative decoding drafters. In particular, we report the *approximate* training time for Medusa, Hydra, EAGLE-3, and DEER on the Qwen3-8B backbone, and attempt to scale the same configurations to Qwen3-14B. The results are summarized in Table 8.

As shown in Table 8, all four methods can be trained on Qwen3-8B within a comparable range of GPU hours. However, when moving to the Qwen3-14B backbone, both Medusa and Hydra encounter out-of-memory (OOM) errors under their default publicly released configurations in our setup, even after standard tuning of batch size and sequence length. Consequently,

Table 8. Approximate fine-tuning time of different speculative decoding drafters on Qwen3 backbones.

Target model	Method	Training time (GPU hours)	Status
Qwen3-8B	Medusa	768	succeeds
	Hydra	1800	succeeds
	EAGLE-3	696	succeeds
	DEER	240	succeeds
Qwen3-14B	Medusa	–	OOM (default config)
	Hydra	–	OOM (default config)
	EAGLE-3	1440	succeeds
	DEER	240	succeeds

Note: GPU hours are approximate wall-clock measurements under our training setup using the official or default configurations released for each method. For Qwen3-14B, Medusa and Hydra run out of memory (OOM) under their default settings in our environment, so we only report training times for EAGLE-3 and DEER.

we only report training times for EAGLE-3 and DEER on Qwen3-14B, and exclude Medusa and Hydra from 14B-scale efficiency comparisons.

E. Correctness Proof of the DEER Speculative Decoding Algorithm

In this section, we provide a formal proof that DEER is *lossless*, i.e., it produces exactly the same output distribution as sampling directly from the target autoregressive (AR) model p_{AR}.

E.1. A One-Step Draft–Then–Verify Lemma

We first analyze a single decoding position. Fix a time step $j+i$ and a realized prefix $\mathbf{x}_{1:j+i-1}$. For convenience, define the target conditional distribution at position $j+i$ as

$$p(x) \ \triangleq \ p_{\mathrm{AR}}\big(x_{j+i} = x \mid \mathbf{x}_{1:j+i-1}\big),$$

and let

$$P(x) \ \triangleq \ P\big(x_{j+i} = x\big)$$

be an *arbitrary* proposal distribution over the same vocabulary. We only require the standard support condition

$$p(x) > 0 \ \Rightarrow \ P(x) > 0 \quad \text{for all } x, \tag{7}$$

i.e., the proposal never assigns zero probability where the target is positive.

Following the classical speculative decoding analysis (Leviathan et al., 2023), we define the pointwise overlap

$$m(x) \ \triangleq \ \min\big(p(x), P(x)\big),$$

its total mass

$$\gamma \ \triangleq \ \sum_x m(x),$$

and the *residual* distribution

$$p_{\mathrm{res}}(x) \ \triangleq \ \frac{p(x) - m(x)}{1-\gamma} \quad \text{for } 1-\gamma > 0, \tag{8}$$

with any arbitrary definition (e.g., $p_{\mathrm{res}} = p$) in the degenerate case $\gamma = 1$ (note that then the residual branch is never used).

Lemma E.1 (One-step lossless speculative sampling)**.** *Consider the following sampling procedure for the token at position $j+i$ given $\mathbf{x}_{1:j+i-1}$:*

1. *Draw a draft token $Y \sim P(\cdot)$.*

2. *Define the acceptance probability*

$$\alpha(Y) \;=\; \min\!\Big(1, \tfrac{p(Y)}{P(Y)}\Big) \;=\; \frac{m(Y)}{P(Y)}. \tag{9}$$

3. *Draw* $U \sim \mathrm{Uniform}(0,1)$ *independently.*
   - *If* $U \le \alpha(Y)$*,* accept *the draft and set* $Z = Y$.
   - *Otherwise,* reject *the draft and set* $Z \sim p_{\mathrm{res}}(\cdot)$ *as in* (8).

Then, for every token a in the vocabulary,

$$\mathbb{P}\big[Z = a \mid \mathbf{x}_{1:j+i-1}\big] \;=\; p(a),$$

i.e., the final token Z has exactly the target distribution $p(\cdot)$ *at position* $j + i$.

Proof. Fix any token a. The probability that the procedure outputs a can be decomposed into two disjoint events: (i) a is drafted and accepted, and (ii) the draft is rejected and a is obtained from the residual distribution:

$$\mathbb{P}[Z = a] = \underbrace{\mathbb{P}[\text{accept and } Z = a]}_{\text{draft accepted}} + \underbrace{\mathbb{P}[\text{reject}]\,\mathbb{P}[Z = a \mid \text{reject}]}_{\text{draft rejected}}.$$

For the first term, using (9) we have

$$\mathbb{P}[\text{accept and } Z = a] = \mathbb{P}[Y = a]\,\alpha(a) = P(a)\,\frac{m(a)}{P(a)} = m(a).$$

For the rejection probability,

$$\mathbb{P}[\text{reject}] = 1 - \sum_x \mathbb{P}[Y = x]\,\alpha(x) = 1 - \sum_x P(x)\,\frac{m(x)}{P(x)} = 1 - \sum_x m(x) = 1 - \gamma.$$

Conditioned on rejection, Z is drawn from p_{res} in (8), so

$$\mathbb{P}[Z = a \mid \text{reject}] = p_{\mathrm{res}}(a) = \frac{p(a) - m(a)}{1 - \gamma}.$$

Putting everything together,

$$\begin{aligned}\mathbb{P}[Z = a] &= m(a) + (1-\gamma)\,\frac{p(a) - m(a)}{1-\gamma}\\ &= m(a) + p(a) - m(a)\\ &= p(a),\end{aligned}$$

which proves the claim. □

Lemma E.1 shows that for *any* proposal $P(x_{j+i})$ satisfying the support condition (7), the draft–then–verify step with acceptance probability $\alpha(\cdot)$ and residual distribution $p_{\mathrm{res}}(\cdot)$ is exactly lossless.

E.2. Instantiating the Proposal with the DEER Drafter

We now instantiate Lemma E.1 with the proposal used in DEER. At position $j + i$, given the prefix $\mathbf{x}_{1:j+i-1}$, the target conditional is

$$p(x) = p_{\mathrm{AR}}\big(x_{j+i} = x \mid \mathbf{x}_{1:j+i-1}\big),$$

while the DEER drafter proposes tokens according to the diffusion model

$$P(x) = q_\theta\big(x_{j+i} = x \mid \mathbf{x}_{1:j}\big).$$

Our training in Section 3.1 ensures that whenever

$$p_{\mathrm{AR}}\big(x_{j+i} = x \mid \mathbf{x}_{1:j+i-1}\big) > 0,$$

we also have

$$q_\theta\big(x_{j+i} = x \mid \mathbf{x}_{1:j}\big) > 0,$$

so the support condition (7) holds.

Plugging these p and P into Lemma E.1, the acceptance probability takes the familiar form

$$\alpha_i = \min\left(1,\ \frac{p_{\mathrm{AR}}(\hat{y}_{j+i} \mid \mathbf{x}_{1:j+i-1})}{q_\theta(\hat{y}_{j+i} \mid \mathbf{x}_{1:j})}\right), \tag{10}$$

which is exactly Eq. (5) in the main text. When the draft token at position $j+i$ is rejected, Lemma E.1 tells us that it should be replaced by a sample from the corresponding residual distribution

$$p_{\mathrm{res},j+i}(x) = \frac{p_{\mathrm{AR}}(x \mid \mathbf{x}_{1:j+i-1}) - \min\Big(p_{\mathrm{AR}}(x \mid \mathbf{x}_{1:j+i-1}), q_\theta(x \mid \mathbf{x}_{1:j})\Big)}{1 - \sum_{x'} \min\Big(p_{\mathrm{AR}}(x' \mid \mathbf{x}_{1:j+i-1}), q_\theta(x' \mid \mathbf{x}_{1:j})\Big)}.$$

Conceptually, this replacement step corresponds to Eq. (6) in the main text:

$$\hat{y}_{j+i} \sim p_{\mathrm{AR}}(\cdot \mid \mathbf{x}_{1:j+i-1}), \tag{11}$$

together with the standard residual construction of speculative decoding (Leviathan et al., 2023). Under this interpretation, Lemma E.1 directly implies that for every position $j+i$,

$$\mathbb{P}\big[x_{j+i} = x \mid \mathbf{x}_{1:j+i-1}\big] = p_{\mathrm{AR}}\big(x_{j+i} = x \mid \mathbf{x}_{1:j+i-1}\big),$$

i.e., the conditional distribution of the final token matches the AR model exactly.

E.3. Sequence-Level Losslessness of DEER

Finally, we extend the one-step result to the whole generated sequence.

Theorem E.2 (Losslessness of DEER decoding)**.** *Let p_{AR} denote the target autoregressive model. Consider running DEER decoding with drafter q_θ and acceptance/resampling rules given by Eqs.* (5) *and* (6). *Then for any finite sequence $\mathbf{x}_{1:T}$,*

$$\mathbb{P}_{\mathrm{DEER}}(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p_{\mathrm{AR}}\big(x_t \mid \mathbf{x}_{1:t-1}\big),$$

i.e., DEER produces exactly the same joint distribution over outputs as direct autoregressive sampling from p_{AR}.

Proof. We proceed by induction on t.

Base case. For $t = 1$, there is no history, and DEER samples from the target model by construction, so

$$\mathbb{P}_{\mathrm{DEER}}(x_1) = p_{\mathrm{AR}}(x_1).$$

Inductive step. Assume that for some $t \geq 2$, the joint distribution over the first $t-1$ tokens generated by DEER matches that of p_{AR}:

$$\mathbb{P}_{\mathrm{DEER}}(\mathbf{x}_{1:t-1}) = \prod_{s=1}^{t-1} p_{\mathrm{AR}}(x_s \mid \mathbf{x}_{1:s-1}).$$

Conditioned on any realized prefix $\mathbf{x}_{1:t-1}$, the one-step analysis in Lemma E.1 (instantiated as above) implies

$$\mathbb{P}_{\mathrm{DEER}}(x_t \mid \mathbf{x}_{1:t-1}) = p_{\mathrm{AR}}(x_t \mid \mathbf{x}_{1:t-1}).$$

Therefore,

$$\begin{aligned}\mathbb{P}_{\text{DEER}}(\mathbf{x}_{1:t}) &= \mathbb{P}_{\text{DEER}}(\mathbf{x}_{1:t-1})\,\mathbb{P}_{\text{DEER}}(x_t \mid \mathbf{x}_{1:t-1}) \\ &= \left(\prod_{s=1}^{t-1} p_{\text{AR}}(x_s \mid \mathbf{x}_{1:s-1})\right) p_{\text{AR}}(x_t \mid \mathbf{x}_{1:t-1}) \\ &= \prod_{s=1}^{t} p_{\text{AR}}(x_s \mid \mathbf{x}_{1:s-1}),\end{aligned}$$

which completes the induction. □

Theorem E.2 formally establishes that DEER is a *lossless* decoding scheme: it achieves acceleration by using the diffusion drafter q_θ to propose blocks of tokens, while provably preserving the exact output distribution of the target autoregressive model p_{AR}.

F. KV cache of DLLM

At present, the community lacks mature support for diffusion language models (DLLMs) with KV caching in mainstream inference frameworks. Consequently, for batch size $B > 1$ our implementation cannot yet be integrated with popular systems such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024). Nevertheless, there has been rapid progress on enabling KV cache for DLLMs. Fast-dLLM (Wu et al., 2025) is among the earliest attempts to explore KV caching for diffusion-based LMs, and its design is particularly well aligned with block diffusion architectures. More recently, dInfer (Ma et al., 2025) proposes a complete and efficient KV cache mechanism for dLLMs together with a dedicated high-throughput inference engine. We expect these techniques to be gradually incorporated into mainstream inference frameworks such as vLLM and SGLang. Once such integration becomes available, DEER can naturally leverage these KV cache implementations and is expected to exhibit substantial advantages in batched inference scenarios.