# 🎯 DART: Diffusion-Inspired Speculative Decoding for Fast LLM Inference

Fuliang Liu [1,2]  Xue Li [2]  Ketai Zhao [1]  Yinxi Gao [1]  Ziyan Zhou [1]  Zhonghui Zhang [1]  Zhibin Wang [1]
Wanchun Dou [1]  Sheng Zhong [1]  Chen Tian [1]

## Abstract

Speculative decoding is an effective and lossless approach for accelerating LLM inference. However, existing widely adopted model-based draft designs, such as EAGLE3, improve accuracy at the cost of multi-step autoregressive inference, resulting in high drafting latency and ultimately rendering the drafting stage itself a performance bottleneck. Inspired by diffusion-based large language models (dLLMs), we propose DART, which leverages parallel generation to reduce drafting latency. DART predicts logits for multiple future masked positions in parallel within a single forward pass based on hidden states of the target model, thereby eliminating autoregressive rollouts in the draft model while preserving a lightweight design. Based on these parallel logit predictions, we further introduce an efficient tree pruning algorithm that constructs high-quality draft token trees with $N$-gram-enforced semantic continuity. DART substantially reduces draft-stage overhead while preserving high draft accuracy, leading to significantly improved end-to-end decoding speed. Experimental results demonstrate that DART achieves a $2.03\times$–$3.44\times$ wall-clock time speedup across multiple datasets, surpassing EAGLE3 by 30% on average and offering a practical speculative decoding framework. Code is released at https://github.com/fvliang/DART.

## 1. Introduction

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Sun et al., 2023; 2024b;a; Gao et al., 2025) has emerged as a promising approach to accelerate memory-bound LLM inference, particularly as modern state-of-the-art models scale to hundreds of billions of parameters (Guo et al., 2025; Liu et al., 2024). In speculative decoding, a
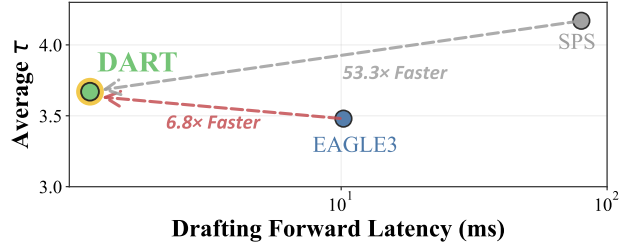


Figure 1. Average Acceptance Length ($\tau$) versus drafting forward latency (ms), averaged across all benchmarks, for speculative decoding of Qwen3-14B on H20-3e GPU. Compared with EAGLE3 and SPS (standard speculative sampling; using Qwen3-1.7B as the draft model with a draft length of 5, set to yield a measurable speedup, whereas both EAGLE3 and DART use a draft length of 8), DART reduces drafting forward latency by up to $6.8\times$ and $53.3\times$, respectively, while preserving relatively high $\tau$, demonstrating a significantly improved drafting efficiency.

lightweight draft model proposes multiple future tokens, which are then verified by the target model to ensure that the final output distribution exactly matches that of standard autoregressive decoding. Through speculative decoding, multiple tokens could be generated in every iteration to significantly improve inference efficiency. The effectiveness of speculative decoding depends critically on the design of the drafter. First, the drafter should achieve high predictive accuracy, most commonly quantified by the *Average Acceptance Length* ($\tau$), i.e., the average number of accepted draft tokens during verification. A higher $\tau$ directly translates to fewer target-model invocations and greater decoding speedups. Second, the draft model itself must be computationally efficient, which could be quantified as drafting latency ($ms$), as excessive drafting overhead can significantly erode the overall benefits of speculative decoding.

Typical speculative decoding's drafter implementation is autoregressive. In vanilla speculative decoding (Leviathan et al., 2023), the drafter is typically a lower-parameter variant from the same model family as the target model. However, vanilla speculative decoding often suffers from high drafting latency, which can account for over 75% and 60% of the total inference time when using Qwen3-1.7B to accelerate Qwen3-14B and Qwen3-32B with draft length 5. In pursuit of lower draft latency, Medusa (Cai et al., 2024)
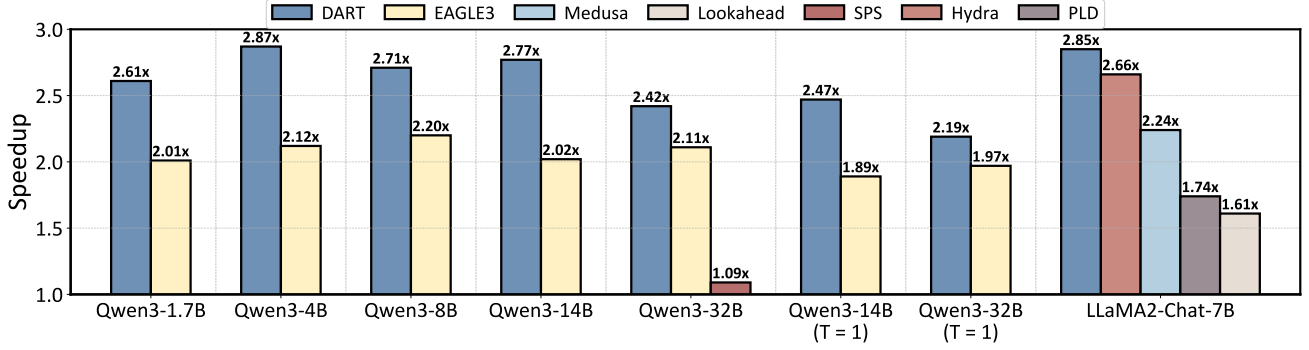
---

[1]State Key Laboratory of Novel Software Technology, Nanjing University [2]Alibaba Group. Correspondence to: Zhibin Wang <wzbwangzhibin@gmail.com>.

*Preprint. January 27, 2026.*

*Figure 2.* Speedup over vanilla autoregressive decoding (batch size = 1), averaged across all datasets. For Qwen3 models, results are reported at temperature $T = 0$ on Qwen3-1.7B, 4B, 8B, 14B, 32B, and additionally at $T = 1$ on Qwen3-14B, 32B; only DART and EAGLE3 are compared on Qwen3, except for Qwen3-32B at $T = 0$, where we also include SPS (using Qwen3-1.7B as the draft model with draft length 5). For LLaMA2-Chat-7B, we compare DART with methods including Medusa, Lookahead, SPS, and PLD.

applies lightweight decoding heads to predict multiple subsequent tokens on the top-layer features of the target model but delivers limited accuracy. EAGLE (Li et al., 2024a;b) improves the accuracy by customizing a dedicated single layer to predict next-feature autoregressively. Subsequent methods (Li et al., 2025b; Zhang et al., 2025) such as EAGLE3 further boost accuracy through reducing inconsistency between training and decoding of draft model.

However, current autoregressive drafter designs introduce limitations. While approaches such as EAGLE3 significantly lower the per-step drafting cost by using a single customized layer, the drafting process is still inherently autoregressive. This sequential dependency forces the drafter to spend nearly 20%–40% of the total inference time, thereby fundamentally limiting the achievable acceleration, causing the drafting stage, especially the drafting forward cost, to emerge as a new bottleneck in speculative decoding. At the other extreme, some speculative decoding approaches, such as Lookahead (Fu et al., 2024), rely solely on $N$-gram or retrieval-based heuristics for drafting. While such methods incur negligible drafting latency, their limited predictive accuracy typically leads to very low average acceptance length $\tau$, resulting in modest end-to-end speedups.

Using diffusion-style parallel generation appears to be a promising direction for overcoming the above limitations. However, directly adopting diffusion-based large language models (dLLMs) as draft models (Christopher et al., 2025; Sandler et al., 2025; Li et al., 2025a; Cheng et al., 2025) suffers from fundamental limitations. For example, DiffuSpec (Li et al., 2025a) uses Dream7B (Ye et al., 2025) as the draft model to accelerate Qwen3-32B and $N$-gram based CPS to give a draft sequence in single forward. However, existing dLLMs like Dream7B are designed for bidirectional context modeling, where the objective is to jointly model and refine tokens across an entire sequence with full contextual access. In contrast, speculative decoding for LLM inference requires predicting a contiguous span of future tokens conditioned strictly on a given prefix, preserving the causal structure of autoregressive generation. This fundamental mismatch in modeling assumptions makes dLLMs inherently unsuitable as drop-in draft models for speculative decoding. Moreover, although dLLMs enable parallel token generation, directly using them as draft models does not translate into lower drafting latency in practice. Because dLLMs such as Dream7B are instantiated as standalone models with substantial parameter, their per-step inference cost remains significantly higher than lightweight, target-coupled drafters such as EAGLE3—often by tens of times in practice. As a result, the overall drafting latency of dLLM-based approaches is often substantially higher, despite their parallel decoding capability. More critically, directly using dLLMs introduces additional practical issues, including tokenizer incompatibility and limited availability.

In this paper, we design a customized diffusion-style drafter tailored for speculative decoding and propose DART, a diffusion-inspired drafting approach.

DART is designed according to the following two principles. **First, diffusion-style parallel drafting is only beneficial when explicitly customized for speculative decoding.** Rather than naively adopting conventional dLLMs with bidirectional context modeling, the draft model should be tailored to predict the next few positions conditioned on a given prefix, aligning with the causal requirements of speculative decoding while retaining parallel generation. In this paper, DART adopts an efficient customized training recipe that enables the draft model to learn the distribution of multiple future positions conditioned on the given prefix. **Second, effective drafting should preserve the low-cost design.** To keep the drafting overhead negligible, DART is tightly coupled with the target model: it reuses the target model's hidden states as input and applies only a single lightweight layer to directly produce logits for multiple future positions,
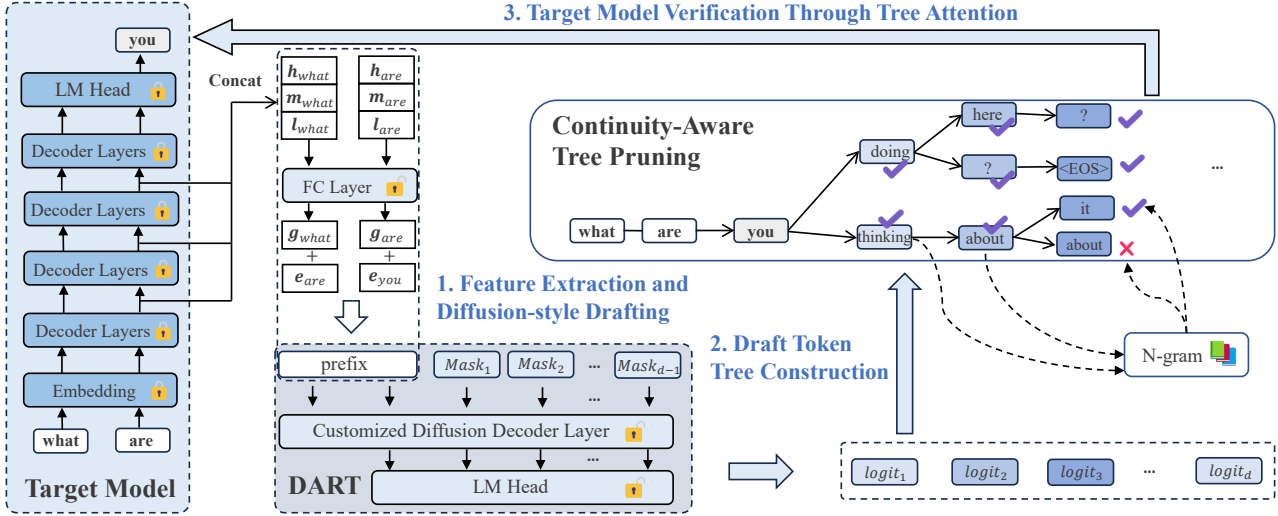
*Figure 3.* Diagram of the DART inference pipeline, illustrating the three substeps of DART's speculative decoding. $\mathbf{l}, \mathbf{m}, \mathbf{h}$ represent the low, middle, and high-level features of the target model, respectively. $\mathbf{e}$ denotes the embedding. Unlocked icon means learnable parameter. After feature extraction, we append $(d-1)$ Mask tokens to the prefix and conduct single forward to get $d$ logits, where the first logit comes from the output of the last position in prefix. We call this "Shifted logits prediction" in Section 3.2. During "Continuity-Aware Tree Pruning", candidate tokens are selected from the corresponding position's predicted logit and a $N$-gram model ensures that the expanded tokens maintain continuity. After getting the final pruned draft token tree, we verify them in target model with Tree Attention.

instead of adopting a drop-in dLLM with prohibitive inference latency. However, these predicted logits implicitly induce an exponentially large combinatorial space of possible token continuations, DART further employs an efficient tree pruning algorithm to construct the final draft token tree for verification, achieving low latency while maintaining high-quality candidates with $N$-gram.

In summary, we propose DART, a diffusion-inspired speculative decoding framework that rethinks the design of draft models under the constraints of drafting latency. Our main contributions are summarized as follows:

- **Lightweight diffusion-style parallel drafting.** We are the first to introduce a draft model that operates directly on the target model's hidden states and predicts multiple future logits in parallel using a single customized layer. This design completely eliminates autoregressive rollout in the drafter and removes the need for complex KV cache management of autoregressive draft model. As shown in Figure 1, DART achieves up to *6.8× faster* drafting forward than autoregressive drafters such as EAGLE3, while remaining relatively high average acceptance length $\tau$.

- **Continuity-aware tree pruning via $N$-gram.** We identify that parallel logits prediction induces an exponentially large combinatorial search space, which cannot be efficiently handled by naive decoding. Instead of using $N$-gram models as standalone draft predictors, which suffer from low acceptance rates, we redesign

$N$-gram as a continuity-aware pruning mechanism to efficiently constrain the draft token tree constructed from parallel logits. This continuity-aware pruning preserves high-quality candidates and substantially improves the average acceptance length $\tau$.

- **Significant end-to-end acceleration.** As shown in Figure 2, extensive experiments across multiple benchmarks demonstrate that DART achieves substantial end-to-end throughput improvements, with up to *2.03–3.44×* speedup over standard autoregressive decoding. Compared to prior speculative decoding methods such as EAGLE3, DART delivers around *30%* higher speedup on average, while further achieving up to *65%* improvement on certain code-centric workloads under the same target model setting. These results validate a new speculative decoding paradigm that combines low drafting latency and high average acceptance length.

## 2. Preliminaries

### 2.1. Speculative Decoding

Let $q_\theta$ denote the target autoregressive language model and $p_\phi$ a lightweight draft model. Standard autoregressive decoding generates one token per forward pass of $q_\theta$, which incurs high inference latency. Speculative decoding accelerates generation by allowing the draft model to first propose a block of $K$ future tokens conditioned on the current prefix. The target model then verifies these draft tokens in parallel and accepts a contiguous prefix of them using a

rejection-based procedure, guaranteeing that the final output distribution exactly matches $q_\theta$. The effectiveness of speculative decoding is commonly measured by the expected number of accepted draft tokens $\tau$, which determines the amortized reduction in target-model invocations. Practical draft models therefore aim to maximize $\tau$ while keeping the drafting overhead minimal.

## 2.2. Diffusion-Based Large Language Models

Diffusion-based large language models (dLLMs) (Nie et al., 2025; Ye et al., 2025; Khanna et al., 2025) generate text by iteratively denoising partially masked sequences, predicting multiple token positions in parallel conditioned on the surrounding context. This masked, non-autoregressive formulation enables holistic sequence modeling and significantly reduces generation steps compared to autoregressive decoding. The strong parallel prediction capability of dLLMs motivates our design in DART, where we seek to eliminate autoregressive rollout in the drafting stage of speculative decoding. However, standard dLLMs are inherently bidirectional and operate on full sequences, which is incompatible with the strictly prefix-conditioned requirement and exactness guarantees of speculative decoding. DART therefore adopts a diffusion-inspired masked prediction mechanism that is explicitly tailored to speculative decoding, rather than performing full-sequence denoising.

## 3. DART

In this section, we will go through the details of DART.

## 3.1. Requirements of Speculative Decoding

The draft model in speculative decoding for autoregressive LLMs is subject to several unique requirements, which fundamentally distinguish DART from standard diffusion-based text generation:

- **Causal attention mask.** Since all prefix tokens are fully accessible and all future positions are predicted in a single forward pass, bidirectional attention for iterative token refinement is unnecessary. DART therefore retains a strictly causal attention mask, both over the prefix and within the masked block.

- **Limited drafting horizon.** Due to the inevitable distribution mismatch between the lightweight draft model and the target model, speculative decoding typically benefits from predicting only a small number of future tokens (e.g., 8). Since DART is designed to generate only a limited number of draft tokens, it can achieve relatively high accuracy even without iterative bidirectional attention refinement.

- **Positional importance bias.** Earlier draft positions

are substantially more important than later ones, as token acceptance proceeds sequentially from the prefix– an error in an early token immediately terminates the verification process.

These characteristics impose stringent constraints on the design of efficient diffusion-style draft models and motivate specialized architectures that prioritize early-token accuracy, controlled horizon prediction, and strict causal conditioning.

## 3.2. Diffusion-Inspired Drafting Phase

**Draft model architecture.** DART follows the lightweight design principle established by prior speculative decoding methods such as EAGLE3 (Li et al., 2025b). The draft model consists of a single customized Transformer decoder layer that operates on intermediate representations of the target model. Concretely, after a prefilling or verification forward pass of the target model, we extract hidden states from multiple intermediate layers (denoted as $\mathbf{h}, \mathbf{m}, \mathbf{l}$ in Figure 3). These hidden states are concatenated and projected through a fully connected layer to obtain a compact prefix representation $\mathbf{g}_{1:n} \in \mathbb{R}^{n \times k}$. In addition, we incorporate shifted token embeddings $\mathbf{e}_{2:n+1}$, obtained by sampling the next token $\mathbf{t}_{n+1}$ from the output of target model and applying the embedding layer of the target model. The final prefix input to the draft model is formed by concatenating $\mathbf{g}_{1:n}$ and $\mathbf{e}_{2:n+1}$ along the feature dimension, which is denoted as $\mathbf{z}_{1:n}$. To enable parallel prediction of future tokens, DART appends a fixed-length suffix of $d - 1$ $\langle\text{MASK}\rangle$ tokens to the prefix, yielding the following input sequence:

$$\left[\ \mathbf{z}_{1:n},\ \langle\text{MASK}\rangle_{n+1:n+d-1}\ \right].$$

A single forward pass of the draft model yields logits for all future positions simultaneously:

$$\{\boldsymbol{\ell}_{n+1}, \boldsymbol{\ell}_{n+2}, \dots, \boldsymbol{\ell}_{n+d}\},$$

where $\boldsymbol{\ell}_{n+1}$ is read from the output at the last prefix position, and subsequent logits are read from masked positions.

**Shifted logits prediction.** In dLLMs, logits predicted at masked positions are typically interpreted as predictions for the tokens at the same positions. In contrast, DART adopts a *shifted logits prediction* scheme, where the logit produced at each position corresponds to the prediction of the next token. Empirically, we find that this shifted formulation significantly improves the prediction accuracy at the first drafted position. Moreover, it enables more efficient training by allowing all positions to contribute supervision signals.

**Parallel holistic prediction.** DART is inspired by dLLMs, which predict multiple masked positions in parallel. However, DART does *not* perform iterative denoising or bidirectional refinement. Instead, it directly predicts the conditional token distributions of multiple future positions *in a*
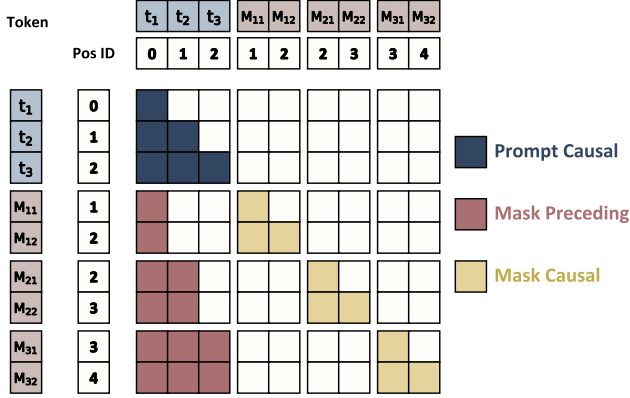
*Figure 4.* Position IDs and attention mask during prefix-share training. The attention mask combines clean data causal attention (Prompt Causal), prefix attention for every mask block (Mask Preceding), and block-inner causal attention (Mask Causal). For clarity of presentation, the figure depicts a simplified example with a prefix length of 3 and mask block length of 2, rather than mask block length of 7 in the actual DART training.

*single step*, strictly conditioned on the given prefix. Through holistic prediction, **sequential latency is eliminated**: the drafting cost is independent of the draft length $d$, requiring only a single forward pass of the lightweight draft model.

### 3.3. Training of the Draft Model

**Prefix-shared masked training.** To closely match the inference-time drafting behavior, the draft model is trained using a *prefix-shared, multi-token prediction objective*. Given a training sequence $x_{1:L}$, we simultaneously construct multiple training instances at different prefix positions within the same sequence. For each prefix position $n$, the model is trained to predict the next $d$ tokens $\{x_{n+1}, \ldots, x_{n+d}\}$ under a shifted prediction scheme, where supervision is applied to both the prefix and masked positions, and all predictions are conditioned only on the prefix $x_{1:n}$.

All such prefix positions are trained jointly within a single forward pass by employing a carefully designed sparse attention mask. Specifically, as illustrated in Figure 4, masked positions corresponding to different prefixes are allowed to attend exclusively to their respective prefixes. Attention among masked positions within the same block is causal to fit the attention mask during inference, while attention across different blocks is explicitly disabled. This *prefix-isolated* attention structure ensures that each masked position learns to model $p_\phi(\cdot \mid x_{1:n})$, without access to future ground-truth tokens. This design enables efficient supervision of multiple future positions across the sequence in parallel, substantially increasing training efficiency while strictly preserving causal conditioning. In practice, the resulting attention pattern is highly sparse. We further leverage *Flex-*

*Attention* (Dong et al., 2024) to exploit this sparsity, significantly reducing memory consumption and accelerating training without altering the underlying model computation. We give the pseudo *Flex-Attention* code of DART's sparse attention mask in Appendix B.

**Annealed KL divergence objective.** Instead of supervising the draft model with discrete one-hot targets, DART optimizes a position-aware KL divergence objective between the draft model predictions and the target model distributions. For each future position $t \in \{1, \ldots, d\}$, we minimize

$$\mathcal{L}_{\text{KL}} = \sum_{t=1}^{d} \lambda_t \, \text{KL}(q_\theta(\cdot \mid x_{1:n+t-1}) \parallel p_\phi(\cdot \mid x_{1:n}, t)),$$

where $\lambda_t = \gamma^{t-1}$, $p_\phi(\cdot \mid x_{1:n}, t)$ denotes the draft model's predicted distribution for the $t$-th future position. This exponentially decaying weighting reflects the increasing uncertainty of longer-horizon predictions and prevents later, noisier targets from dominating the training signal. In practice, we set $\gamma = 0.6$, which consistently yields the best trade-off between early-position accuracy and overall training stability in our ablation studies (Section 5.3). By emphasizing short-horizon predictions while softly regularizing distant ones, this annealed objective substantially improves draft quality at the most critical positions and thereby provides higher-quality candidate distributions for downstream draft tree construction.

### 3.4. What Does DART Output?

Rather than directly producing a draft token tree, DART outputs a set of parallel logits $\{\ell_{n+1}, \ldots, \ell_{n+d}\}$, one for each future position. Each logit encodes a high-quality candidate set for that position conditioning on the prefix. Together, these logits define a compact but expressive *candidate set* that implicitly contains many plausible future continuations.

## 4. Efficient $N$-gram-based Tree Pruning

### 4.1. Implicit Huge Search Space in Parallel Logits

The $d$ parallel logits predicted by DART define a factorized distribution over future positions, which implicitly induce an exponentially large combinatorial space of possible token continuations, corresponding to a full $d$-level token tree. Directly verifying this space is infeasible, and naively combining independently predicted tokens may lead to locally semantically implausible draft sequences. The goal of draft tree construction is therefore to extract a compact, high-quality subset of this implicit tree that can be efficiently verified by the target model.

**Algorithm 1** Continuity-Aware Tree Pruning

---

**Input** $d$ future-position logits $\{l_i\}_{i=1}^d$; $N$-gram model $g_n$; candidate threshold $\{k_i\}_{i=1}^d$; prefix context $ctx$; tree size $\theta$; beam width $w$

**Output** Final draft token tree $\mathcal{T}$

1: $\mathcal{C} \leftarrow \{(ctx, 0)\}$ ▷ *Active candidate set (sequence, score)*
2: $\mathcal{T} \leftarrow \varnothing$ ▷ *Global token tree*
3: **for** $i = 1$ to $d$ **do**
4:     $\mathcal{C}' \leftarrow \varnothing$
5:     **for all** $(seq, sc) \in \mathcal{C}$ **in parallel do**
6:         $\mathcal{S}_i \leftarrow$ Top-$k_i$ tokens from $l_i$
7:         **for all** $t \in \mathcal{S}_i$ **do**
8:             $s_{\text{logit}} \leftarrow \log(softmax(l)_{i,t} + \epsilon)$
9:             $s_{\text{ng}} \leftarrow g_n(t, seq_{-n:})$ ▷ *N-gram continuity score*
10:            $s' \leftarrow sc + \text{COMBINE}(s_{\text{logit}}, s_{\text{ng}})$
11:            $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{(\text{CAT}(seq, t), s')\}$
12:            $\mathcal{T} \leftarrow \mathcal{T} \cup \{(\text{CAT}(seq, t), s')\}$
13:         **end for**
14:     **end for**
15:     $\mathcal{C} \leftarrow$ Top-$w$ scoring pairs from $\mathcal{C}'$
16: **end for**
17: **return** Top-$\theta$ scoring nodes from $\mathcal{T}$

---

### 4.2. Consistency-constrained Pruning via $N$-gram

To extract a compact and semantically coherent draft tree from the exponentially large search space induced by parallel logits, DART performs consistency-constrained pruning guided by an established $N$-gram model, as shown in Algorithm 1. Starting from the prefix context, candidate tokens at each future position are first locally ranked according to their draft logits. During tree expansion, these candidates are further scored using an $N$-gram continuity score that measures the likelihood of appending a token given the recent $n$-token suffix of the partial sequence. The logit-based score and the $N$-gram score are combined to form a unified ranking criterion, which favors token sequences that are both locally probable under the draft model and globally consistent with surface-level language statistics. This constraint effectively filters out semantically implausible combinations that arise from independently predicted positions, while retaining high-quality candidates. This design is model-agnostic, decoupling the draft model's forward computation from the tree expansion procedure. The detailed hyperparameter setting is given in Appendix D.

## 5. Experiments

### 5.1. Experimental Setup

**Hardware.** All training and inference processes are conducted on a server equipped with 8×NVIDIA H20-3e GPUs (141GB), 90 CPU cores, 900GB of RAM and PyTorch 2.8.0.

**Target models.** We mainly train DART on the Qwen3 model family (Yang et al., 2025), including Qwen3-1.7B, Qwen3-4B, Qwen3-8B, Qwen3-14B and Qwen3-32B. To
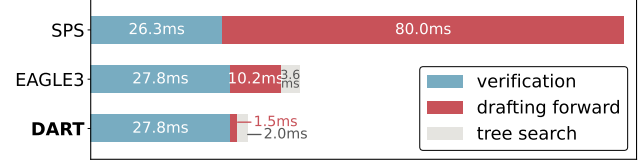


*Figure 5.* Latency of verification, drafting forward and tree search in one draft-verify iteration when accelerating Qwen3-14B using SPS, EAGLE3 and DART. SPS has slightly lower latency in verification because of fewer draft tokens than EAGLE3 and DART.

fairly compare DART with other speculative decoding methods (e.g., Lookahead, PLD (Saxena, 2023), Medusa and Hydra (Ankner et al., 2024)), we additionally train DART on LLaMA2-Chat-7B (Touvron et al., 2023), since these methods are not compatible with the Qwen3 family.

**Benchmarks.** We evaluate DART on a diverse suite of tasks covering instruction following, multi-turn conversation, mathematical reasoning, and code generation: MT-Bench (Zheng et al., 2023), HumanEval (Chen et al., 2021), Alpaca (Taori et al., 2023), Math500 (Lightman et al., 2023), CodeAlpaca (Lhoest et al., 2021), LiveCodeBench (Jain et al., 2024), and MBPP (Austin et al., 2021).

**Implementation.** Unless otherwise specified, all experiments are conducted with a batch size of 1, and experiments on EAGLE3 reuse the weights from Contributors (2025). Other methods such as Medusa reuse their weights from their corresponding open-source repository. We do not compare DART with DiffuSpec or SpecDiff (Christopher et al., 2025) because their implementations are closed-source. Both DART and EAGLE3 use the draft length of 8. We give the details of training and $N$-gram-related settings in Appendix B and C.

**Metrics.** In this paper, we focus on the following performance-related metrics. We do not report generation quality, since DART preserves the exact output distribution of the target model.

- **Throughput.** The actual test end-to-end speedup ratio relative to vanilla autoregressive decoding.

- **Average acceptance length ($\tau$).** The average number of tokens generated per drafting-verification cycle, indicating the number of tokens accepted by the target model's verification.

- **Drafting latency.** The wall-clock time spent in the draft model forward and tree search to propose candidate tokens for each verification cycle. This metric captures the overhead introduced by the draft model and directly impacts the overall end-to-end speedup.

*Table 1.* Speedup ratios of different methods and mean average acceptance lengths $\tau$ on MT-Bench, HumanEval, Alpaca, Math500, CodeAlpaca, LiveCodeBench and MBPP. All Qwen models are from Qwen3 family, for example, Qwen32B represents Qwen3-32B. L2 7B represents LLaMA2-Chat-7B. In SPS, we use the Qwen3-1.7B as the drafter of Qwen3-14B and Qwen3-32B with draft length 5.

| Model | Method | Speedup | | | | | | | Mean | |
| | | Alpaca | Codealpaca | Humaneval | LiveCode | Math500 | MBPP | MT-bench | Speedup | $\tau$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Temperature=0 | | | | | |
| L2 7B | PLD | 1.18× | 1.75× | 1.84× | 1.94× | 1.59× | 1.59× | 1.57× | 1.74× | 1.92 |
| | Lookahead | 1.42× | 1.54× | 1.64× | 1.59× | 1.82× | 1.61× | 1.63× | 1.61× | 1.81 |
| | Medusa | 2.13× | 2.24× | 2.22× | 2.31× | 2.29× | 2.35× | 2.12× | 2.24× | 2.68 |
| | Hydra | 2.72× | 2.89× | 2.60× | 2.59× | 2.69× | 2.62× | 2.48× | 2.66× | 3.55 |
| | **DART** | **2.95×** | **3.03×** | **2.98×** | **2.81×** | **2.84×** | **2.72×** | **2.61×** | **2.85×** | **4.08** |
| Qwen1.7B | EAGLE3 | 1.84× | 1.93× | 1.94× | 2.06× | 2.25× | 2.05× | 1.98× | 2.01× | **3.80** |
| | **DART** | **2.60×** | **2.90×** | **2.79×** | **2.28×** | **2.52×** | **2.64×** | **2.57×** | **2.61×** | 3.60 |
| Qwen4B | EAGLE3 | 2.15× | 2.21× | 2.20× | 1.95× | 2.06× | 2.08× | 2.17× | 2.12× | 3.54 |
| | **DART** | **2.55×** | **3.45×** | **3.25×** | **2.57×** | **2.50×** | **3.06×** | **2.73×** | **2.87×** | **3.87** |
| Qwen8B | EAGLE3 | 2.02× | 2.34× | 2.37× | 2.11× | 2.30× | 2.20× | 2.08× | 2.20× | **3.72** |
| | **DART** | **2.51×** | **3.40×** | **2.80×** | **2.64×** | **2.34×** | **3.09×** | **2.22×** | **2.71×** | 3.61 |
| Qwen14B | SPS | 1.05× | 1.07× | 0.96× | 0.94× | 0.96× | 0.97× | 0.92× | 0.98× | **4.17** |
| | EAGLE3 | 1.69× | 2.08× | 2.27× | 2.08× | 2.32× | 1.99× | 1.71× | 2.02× | 3.48 |
| | **DART** | **2.73×** | **3.44×** | **2.79×** | **2.69×** | **2.50×** | **2.98×** | **2.20×** | **2.77×** | 3.67 |
| Qwen32B | SPS | 1.07× | 1.08× | 1.06× | 1.06× | 1.06× | 1.15× | 1.12× | 1.09× | 3.45 |
| | EAGLE3 | 1.72× | 2.38× | 2.19× | 2.15× | 2.31× | 2.27× | 1.76× | 2.11× | **3.85** |
| | **DART** | **2.03×** | **2.88×** | **2.37×** | **2.42×** | **2.46×** | **2.56×** | **2.24×** | **2.42×** | 3.76 |
| | | | | | Temperature=1 | | | | | |
| Qwen14B | EAGLE3 | 1.62× | 2.01× | 2.17× | 1.92× | 2.10× | 1.81× | 1.61× | 1.89× | 3.38 |
| | **DART** | **2.38×** | **3.10×** | **2.48×** | **2.37×** | **2.29×** | **2.71×** | **1.94×** | **2.47×** | **3.61** |
| Qwen32B | EAGLE3 | 1.57× | 2.30× | 2.08× | 1.97× | 2.15× | 2.17× | 1.58× | 1.97× | **3.67** |
| | **DART** | **1.84×** | **2.78×** | **2.19×** | **2.13×** | **2.16×** | **2.40×** | **1.82×** | **2.19×** | 3.55 |

## 5.2. Main Results

**Throughput improvement.** We use vanilla autoregressive decoding as the baseline and compare DART against recent lossless speculative decoding methods, including standard speculative sampling (SPS), PLD, Hydra, Lookahead, Medusa and EAGLE3. As shown in Figure 2 and Table 1, DART achieves a 2.03×–3.44× throughput improvement across tasks, outperforming EAGLE3 by 30% on average. Notably, DART yields larger gains on code-related benchmarks. For example, in CodeAlpaca, DART surpasses EAGLE3 by 65% when accelerating Qwen3-14B.

**Relatively high $\tau$.** As shown in Table 1, DART achieves relatively high $\tau$ across all model scales. While DART does not always exceed EAGLE3 in $\tau$, the difference is within 0.2, and thus negligible. More importantly, due to its substantially lower drafting latency, DART converts a comparable $\tau$ into significantly higher throughput. This highlights that minimizing drafting overhead, rather than solely maximizing $\tau$, is critical for achieving superior end-to-end performance.

**Drafting latency reduction.** We focus on the drafting latency of 3 model-based drafters, including SPS, DART and EAGLE3. Methods like PLD and Lookahead do not have a separate model-based drafter, so their drafting latency is not reported. As shown in Figure 5, DART consumes 1.5ms latency to finish the drafting forward and reduces drafting forward latency by 6.8× and 53.3× compared to EAGLE3 and SPS in case of inference of Qwen3-14B. The tree pruning will take another 2ms to get the final draft token tree, which is also negligible latency and more efficient than the tree search of EAGLE3. This enables DART to accelerate LLM inference with negligible drafting overhead.

## 5.3. Ablation Study

**N-gram effectiveness.** The $N$-gram constraint is a key component in DART for pruning the exponentially large search space induced by parallel token prediction. To evaluate its impact on drafting efficiency, we compare average acceptance length $\tau$ with and without $N$-gram pruning across multiple benchmarks. As shown in Table 2, incorporating $N$-gram pruning consistently leads to substantial improvements

*Table 2.* Effect of *N*-gram pruning on drafting efficiency. Reported values are the average number of accepted draft tokens $\tau$ measured under speculative decoding on each benchmark.

| - | HumanEval | Alpaca | Math500 | CodeAlpaca |
|---|---|---|---|---|
| w/o *N*-gram | 3.13 | 2.76 | 2.89 | 3.85 |
| **w/ *N*-gram** | **3.63** (↑0.5) | **3.26** (↑0.5) | **3.37** (↑0.48) | **4.59** (↑0.74) |

in $\tau$ across all evaluated benchmarks. This demonstrates that *N*-gram pruning effectively removes low-quality branches in the draft token tree while preserving high-probability continuations, resulting in more efficient draft construction and higher acceptance rates during verification.

**Shifted logits prediction.** We investigate the effect of shifted logits prediction in the DART inference pipeline. Specifically, we train two DART models based on Qwen3-4B from scratch under identical training configurations: one model adopts shifted logits prediction, while the other uses unshifted logits prediction. Unshifted logits prediction means logits predicted at masked positions are interpreted as predictions for the tokens at the same positions. After training, we evaluate the prediction accuracy at multiple future positions on the same test data. The results are summarized in Table 3. We observe that shifted logits prediction substantially improves the accuracy of the first predicted position, increasing from 57.7% to 71.1%. In addition, modest but consistent accuracy gains are observed for subsequent positions in both hit@1 and hit@10 metrics.

*Table 3.* Accuracy of multiple future positions under shifted and unshifted logits prediction. Hit@k denotes the proportion of cases where the ground-truth token appears among the top-$k$ predictions ranked by model logits. $\alpha$-$k$ denotes the prediction accuracy at the $k$-th future position.

| Position | $\alpha$-1 | $\alpha$-2 | $\alpha$-3 |
|---|---|---|---|
| Unshifted | 57.7% | 38.1% | 25.1% |
| **Shifted** | **71.1%** (↑13.4) | **41.0%** (↑2.9) | **27.6%** (↑2.5) |
| Unshifted(hit@10) | 87.1% | 74.2% | 63.2% |
| **Shifted(hit@10)** | **93.2%** (↑6.1) | **76.7%** (↑2.5) | **65.9%** (↑2.7) |

**Annealing coefficient in DART training.** During DART training, we apply annealed KL divergence objective to prevent the increasing uncertainty of longer-horizon predictions from disturbing overall training stability. Specifically, we introduce an annealing coefficient $\gamma$ to progressively down-weight the supervision on future positions. To identify an appropriate value of $\gamma$, we train multiple DART models for Qwen3-4B from scratch with $\gamma \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, all using shifted logits prediction. The case $\gamma = 1.0$ corresponds to no annealing and serves as the shifted-logits baseline from the previous ablation. Table 4 compares the prediction accuracy at multiple future positions as well as the resulting average accepted length $\tau$. We observe a clear

*Table 4.* Effect of the annealing coefficient $\gamma$ on prediction accuracy and average acceptance length $\tau$ on HumanEval.

| $\gamma$ | $\alpha$-1 | $\alpha$-2 | $\alpha$-3 | $\alpha$-4 | $\alpha$-5 | $\alpha$-6 | $\tau$ |
|---|---|---|---|---|---|---|---|
| 0.5 | 76.2% | 45.0% | 28.7% | 18.4% | 13.0% | 10.6% | 3.59 |
| **0.6** | **75.5%** | **44.7%** | **29.0%** | **19.2%** | **13.8%** | **11.2%** | **3.63** |
| 0.7 | 74.7% | 44.1% | 28.9% | 19.6% | 14.3% | 11.8% | 3.57 |
| 0.8 | 73.8% | 43.2% | 28.8% | 19.8% | 14.8% | 12.2% | 3.54 |
| 0.9 | 72.5% | 42.1% | 28.1% | 19.4% | 14.9% | 12.4% | 3.52 |
| 1.0 | 71.1% | 41.0% | 27.6% | 19.3% | 15.0% | 12.6% | 3.48 |

trade-off: smaller values of $\gamma$ improve accuracy at earlier positions while degrading performance at later positions, whereas larger $\gamma$ favors longer-horizon predictions. Since accuracy at early positions is more critical for speculative decoding, annealing the KL objective leads to a higher average acceptance length $\tau$, which translates linearly into end-to-end decoding throughput. Among all settings, $\gamma = 0.6$ achieves the best balance between early-position accuracy and acceptance length $\tau$, and is therefore used as the default configuration in DART.

### 5.4. Larger Batch Sizes Study

We also conduct experiments with batch sizes larger than 1 with DART and EAGLE3. As shown in Table 5, although throughput improvement decays as the batch size increases due to the more compute-bound brought by larger batch size, DART still gets larger gains than EAGLE3.

*Table 5.* Throughput improvements under larger batch sizes on the HumanEval benchmark, evaluated on H20-3e device, for EAGLE3 and DART relative to autoregressive decoding (without speculative decoding) when target model is Qwen3-4B.

| Batch Size | 2 | 4 | 8 | 16 | 24 | 32 | 48 | 64 |
|---|---|---|---|---|---|---|---|---|
| EAGLE3 | 1.84× | 1.69× | 1.48× | 1.32× | 1.29× | 1.28× | 1.24× | 1.22× |
| **DART** | **2.16×** | **2.01×** | **1.77×** | **1.57×** | **1.51×** | **1.48×** | **1.47×** | **1.45×** |

## 6. Conclusion

We present DART, a diffusion-inspired speculative decoding framework that enables fast and practical LLM inference. By predicting token distributions for multiple future positions in parallel within a single draft model forward pass, DART eliminates the latency inherent in autoregressive drafting. To transform the parallel logits into coherent draft sequences, we introduce an efficient *N*-gram-guided tree pruning strategy, which preserves semantic continuity without sacrificing parallelism. Experiments show that DART substantially reduces draft-stage overhead and achieves significant end-to-end decoding speedups over prior speculative decoding methods. Overall, DART demonstrates that parallel, distribution-level drafting combined with structured pruning, offers a viable and effective alternative to conventional autoregressive draft designs.

## Impact Statement

This paper proposes work that advances the field of speculative decoding and acceleration of LLMs.

## References

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

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*, 2021.

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. *arXiv preprint arXiv: 2401.10774*, 2024.

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

Chen, M., Tworek, J., Jun, H., Yuan, Q., 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.

Cheng, Z., Yang, G.-W., Li, J., Deng, Z., Guo, M.-H., and Hu, S.-M. Deer: Draft with diffusion, verify with autoregressive models. *arXiv preprint arXiv:2512.15176*, 2025.

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 (Volume 1: Long Papers)*, pp. 12042–12059, 2025.

Contributors, T. A. P. AngelSlim, 6 2025. URL https://github.com/Tencent/AngelSlim.

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. *arXiv preprint arXiv:2305.14233*, 2023.

Dong, J., Feng, B., Guessous, D., Liang, Y., and He, H. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2024.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

Gao, X., Xie, W., Xiang, Y., and Ji, F. Falcon: Faster and parallel inference of large language models through enhanced semi-autoregressive drafting and custom-designed decoding tree. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23933–23941, 2025.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 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.

Khanna, S., Kharbanda, S., Li, S., Varma, H., Wang, E., Birnbaum, S., Luo, Z., Miraoui, Y., Palrecha, A., Ermon, S., et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 1, 2025.

Kong, R., Li, Y., Feng, Q., Wang, W., Ye, X., Ouyang, Y., Kong, L., and Liu, Y. Swapmoe: Serving off-the-shelf moe-based large language models with tunable memory budget. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6710–6720, 2024.

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

Lhoest, Q., Del Moral, A. V., Jernite, Y., Thakur, A., Von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., et al. Datasets: A community library for natural language processing. In *Proceedings of the 2021 conference on empirical methods in natural language processing: system demonstrations*, pp. 175–184, 2021.

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

Li, Y., Wei, F., Zhang, C., and Zhang, H. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, 2024a.

Li, Y., Wei, F., Zhang, C., and Zhang, H. EAGLE-2: Faster inference of language models with dynamic draft trees. In *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. In *Annual Conference on Neural Information Processing Systems*, 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. In *The Twelfth International Conference on Learning Representations*, 2023.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Luo, C., Cai, Z., Sun, H., Xiao, J., Yuan, B., Xiao, W., Hu, J., Zhao, J., Chen, B., and Anandkumar, A. Headinfer: Memory-efficient llm inference by head-wise offloading. *arXiv preprint arXiv:2502.12574*, 2025.

Ma, C., Ye, Z., Zhao, H., Yang, Z., Fu, T., Han, J., Zhang, J., Luo, Y., Wang, X., Wang, Z., et al. Memory offloading for large language model inference with latency slo guarantees. *arXiv preprint arXiv:2502.08182*, 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 preprint arXiv:2502.09992*, 2025.

Olmo, T., Ettinger, A., Bertsch, A., Kuehl, B., Graham, D., Heineman, D., Groeneveld, D., Brahman, F., Timbers, F., Ivison, H., et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Sandler, J., Christopher, J. K., Hartvigsen, T., and Fioretto, F. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding. *arXiv preprint arXiv:2511.00606*, 2025.

Saxena, A. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

Shenggui Li, Yikai Zhu, C. W. F. Y. S. S. Y. W. Y. Z. Y. H. H. Z. Y. Z. Specforge: Train speculative decoding models effortlessly. https://github.com/sgl-project/specforge, 2025.

Sun, H., Chen, Z., Yang, X., Tian, Y., and Chen, B. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024a.

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

Sun, Z., Ro, J. H., Beirami, A., and Suresh, A. T. Optimal block-level draft verification for accelerating speculative decoding. *CoRR*, 2024b.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

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

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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, L., Wang, X., Huang, Y., and Xu, R. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623, 2023.

## A. EAGLE3 Latency Analysis

EAGLE3's drafting process is inherently autoregressive. As shown in Figure 6, EAGLE3's sequential dependency forces the drafter to spend nearly 20%–40% of the total inference time, thereby fundamentally limiting the achievable acceleration.

Qwen3-32B exhibits a lower draft ratio compared to other models in the Qwen3 family. Although both Qwen3-32B and Qwen3-14B share a hidden dimension of 5120—resulting in nearly identical draft model parameters and latency within the EAGLE3 single-layer transformer framework—their verification overhead differs significantly. With 64 transformer layers compared to the 14B model's 40 (Yang et al., 2025), Qwen3-32B incurs nearly double the verification latency. Because drafting time remains constant while verification costs scale with depth, the overall draft ratio for Qwen3-32B is inherently diminished.
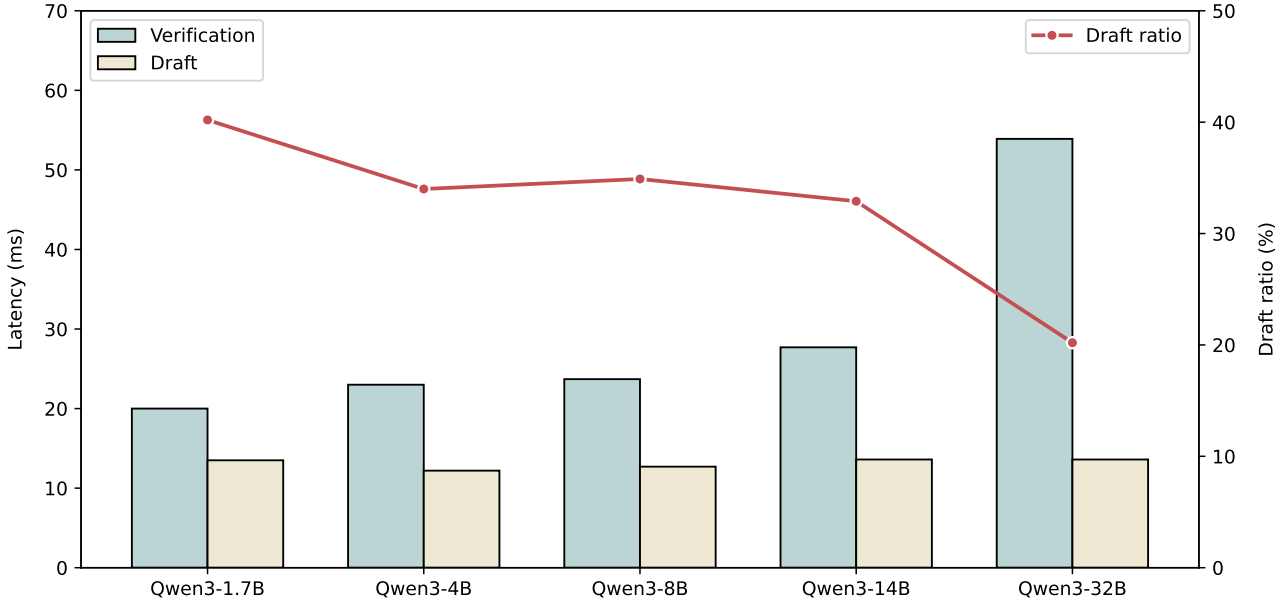


*Figure 6.* Per-round verification latency, drafting latency, and draft ratio across Qwen3 model variants evaluated on H20-3e. Bars (left axis) report per-round verification latency and drafting latency in milliseconds, while the line plot (right axis) shows the draft ratio (%). Results are shown for Qwen3 models ranging from 1.7B to 32B parameters.

## B. Training Details

### B.1. Training Implementation

The code of training DART is based on SGLang's open-source repository SpecForge (Shenggui Li, 2025).

### B.2. Training Setup

We train the DART models of Qwen3-family and LLaMA2-Chat-7B under our carefully designed recipe. Unless otherwise specified, all training adopt a context length of 6400 and draft length of 8. We use the AdamW (Loshchilov & Hutter, 2017) optimizer, with gradient clipping of 0.5 and beta values ($\beta_1$, $\beta_2$) set to (0.9, 0.999). The learning rate is set to 2e-5, and training is conducted for 3 epochs. DART is optimized using a cosine annealing learning rate schedule with linear warmup. The more detailed hyperparameters (e.g., number of heads, hidden dimension) of the customized Transformer decoder layer in DART keeps aligned with the target model. Training is conducted on 8 × NVIDIA H20-3e GPUs.

**Trainable parameters** As shown in Figure 3, the trainable parameters in DART include the fully connected (FC) layer (whose output dimensionality matches the hidden size of the target model), decoder layer, and the LM head. DART shares the embedding layer with the target LLM, which remains frozen during training. In addition, the mask representation is also treated as a set of trainable parameters and is initialized with random values. This design choice follows the practice adopted in dLLM (Nie et al., 2025).

**Training data.**    Our training data is drawn from ShareGPT and UltraChat (Ding et al., 2023). After filtering, the combined dataset contains approximately 280K examples. We use the same data sources as EAGLE3, ensuring consistency in the training corpus. DART takes the hidden states of the target model as input. Specifically, we select the outputs of the 1st, $(\texttt{num\_layers}/2 - 1)$-th, and $(\texttt{num\_layers} - 4)$-th transformer layers as inputs to DART, following the same layer selection strategy as EAGLE3.

**Training with Flex Attention.**    We give the pseudo code of DART's sparse attention mask with Flex Attention.

```
def forward_flexattn_dart(
        hidden_states,          # hidden states of target LLM
        attention_mask,         # attention mask, where 1=valid, 0=pad
        position_ids,           # position ids of tree structure
        draft_len,              # draft length d
        rotary_emb,             # RoPE module (details omitted)
    ):
    B, q_len, _ = hidden_states.shape
    prompt_len = q_len // (draft_len + 1)
    Q, K, V = q_proj(hidden_states), k_proj(hidden_states), v_proj(hidden_states)
    Q, K = apply_rope(Q, K, rotary_emb, position_ids, q_len)
    key_cache, value_cache = K, V
    seq_lengths = attention_mask.sum(dim=-1)
    create_block_mask_func = compile_friendly_create_block_mask
    flex_attention_func    = compile_friendly_flex_attention
    mask_mod = generate_dart_mask(
        seq_lengths=seq_lengths,
        prompt_len=prompt_len,
        draft_len=draft_len,
    )
    block_mask = create_block_mask_func(
        mask_mod=mask_mod,      # function(b,h,q_idx,kv_idx)->bool
        B=B,
        H=1,                    # rely on broadcast over heads
        Q_LEN=q_len,
        KV_LEN=key_cache.shape[-2],
        device=Q.device,
    )
    attn_out = flex_attention_func(
        query=Q,
        key=key_cache.contiguous(),
        value=value_cache.contiguous(),
        block_mask=block_mask,
        enable_gqa=True,
    )
    out = o_proj(reshape_back(attn_out))   # -> (B, q_len, dim)
    return out
```

```
def generate_dart_mask(seq_lengths, prompt_len, draft_len):
    def prompt_causal(b, h, q_idx, kv_idx):
        in_prompt = (q_idx < prompt_len) and (kv_idx < prompt_len)
        causal    = (q_idx >= kv_idx)
        valid     = (q_idx < seq_lengths[b]) and (kv_idx < seq_lengths[b])
        return in_prompt and causal and valid
    def draft_view_prompt(b, h, q_idx, kv_idx):
        is_draft      = (q_idx >= prompt_len)
        kv_in_prompt  = (kv_idx < prompt_len)
        draft_group = (q_idx - prompt_len) // draft_len
        valid_group = (draft_group < seq_lengths[b])
        allowed_prompt = (kv_idx <= draft_group)
        return is_draft and kv_in_prompt and valid_group and allowed_prompt
    def draft_internal_causal(b, h, q_idx, kv_idx):
        is_draft_q = (q_idx >= prompt_len)
        is_draft_k = (kv_idx >= prompt_len)
        q_group = (q_idx - prompt_len) // draft_len
        k_group = (kv_idx - prompt_len) // draft_len
        same_group = (q_group == k_group)
        causal     = (q_idx >= kv_idx)
        valid      = (q_group < seq_lengths[b])
        return is_draft_q and is_draft_k and same_group and causal and valid
    mask_mod = or_masks(prompt_causal, or_masks(draft_view_prompt, draft_internal_causal))
    return mask_mod
```

## C. *N*-gram Details

### C.1. *N*-gram Construction

We implement the *N*-gram using a trie data structure to achieve high retrieval performance. The construction relies on open-source dataset Dolma 3 Mix (Olmo et al., 2025), which covers multiple fields and types. The *N*-gram is built based on the tokenizer of Qwen3 family or LLaMA2 family. To balance retrieval efficiency and memory overhead during inference,

we adopt a *3*-gram model in DART, which consists of approximately 1.3 billion tree nodes and occupies about 43.5 GB of disk space.

### C.2. *N*-gram Usage at Inference

During inference, the *N*-gram model is queried to provide scores for candidate tokens given the preceding context. As mentioned in Algorithm 1, there are $k_i$ possible token candidates at position $i$ given a preceding context. To efficiently retrieve the scores for all $k_i$ candidate tokens with the same preceding context, we leverage the structural properties of trie. Specifically, we first locate the node corresponding to the preceding context. From this node, we can directly access all its child nodes, each representing a candidate token. This approach allows us to retrieve scores for all $k_i$ candidates in a single traversal, significantly reducing the number of required lookups compared to querying each candidate token individually.

To support this efficient retrieval mechanism, we implement the *N*-gram trie in `C++` and load it into CPU RAM before inference. During execution, the *N*-gram trie occupies roughly 100 GB of CPU RAM. While this memory footprint may appear large, it is well aligned with modern inference servers, which typically provision hundreds of gigabytes of system memory and often leave substantial CPU RAM underutilized during LLM inference (Ma et al., 2025; Luo et al., 2025; Kong et al., 2024). Moreover, the footprint is shared across multiple inference processes. Thus, the increased RAM usage represents a practical design trade-off to achieve low-latency *N*-gram retrieval, rather than a fundamental limitation of our approach. As shown in Figure 7, the average latency of *N*-gram retrieval is around 6 $\mu$s per query after warmup, which is negligible compared to the overall inference time.
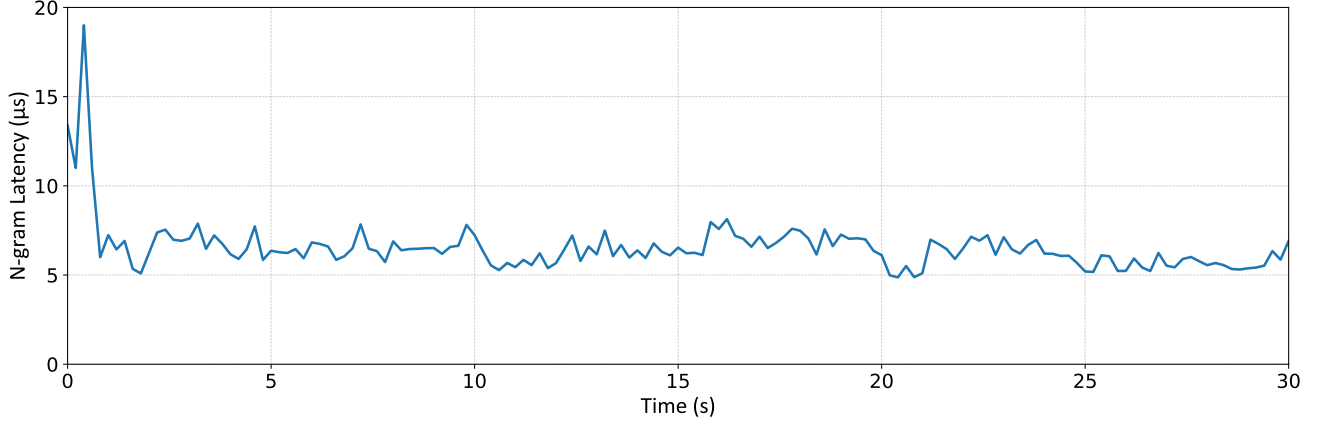


*Figure 7.* *N*-gram retrieval latency over time. The timeline is divided into 200 ms intervals, and the average latency per query is recorded for each interval. After the warmup period, the average latency stabilizes around 6 $\mu$s per query after warmup.

## D. Tree Pruning Algorithm Details

This section provides the detailed hyperparameter settings and implementation specifics of the continuity-aware tree pruning algorithm described in Algorithm 1. The pruning procedure is implemented in `C++` for efficiency, and parallelized using `OpenMP` along both the batch dimension and the candidate expansion process.

At each future position $i \in \{1, \ldots, d\}$, we select the top-$k_i$ tokens from the corresponding draft logits $l_i$. In all experiments, we set a uniform candidate threshold $k_i = 25$ for all positions. The active candidate set is maintained using beam pruning with beam width $w = 20$, which controls the maximum number of partial sequences retained at each depth.

The global draft token tree $\mathcal{T}$ is bounded by a maximum size $\theta = 59$, and only the top-$\theta$ scoring nodes are preserved for downstream verification. This constraint ensures a balance between drafting diversity and verification efficiency.

For each candidate extension, we compute a combined score consisting of a logit-based likelihood term and an *N*-gram continuity score. Specifically, the logit score is defined as

$$s_{\text{logit}} = \log(softmax(l)_{i,t} + \epsilon),$$

*Table 6.* Throughput and relative speedup across batch sizes evaluated on H20-3e (141G) device when target model is Qwen3-4B.

| Batch size | Throughput (tokens/s) | | | Speedup | |
|---|---|---|---|---|---|
| | Baseline | EAGLE3 | DART | EAGLE3 | DART |
| 2 | 105.8 | 194.6 | **228.7** | 1.84× | **2.16×** |
| 4 | 172.1 | 291.1 | **346.3** | 1.69× | **2.01×** |
| 8 | 243.3 | 359.0 | **431.1** | 1.48× | **1.77×** |
| 16 | 309.4 | 408.6 | **484.4** | 1.32× | **1.57×** |
| 24 | 338.3 | 436.4 | **510.9** | 1.29× | **1.51×** |
| 32 | 346.0 | 443.2 | **512.9** | 1.28× | **1.48×** |
| 48 | 364.6 | 450.9 | **538.5** | 1.24× | **1.47×** |
| 64 | 370.0 | 453.3 | **538.7** | 1.22× | **1.45×** |

*Table 7.* Throughput and relative speedup across batch sizes evaluated on H20-3e (141G) device when target model is Qwen3-8B.

| Batch size | Throughput (tokens/s) | | | Speedup | |
|---|---|---|---|---|---|
| | Baseline | EAGLE3 | DART | EAGLE3 | DART |
| 2 | 94.3 | 182.1 | **203.5** | 1.93× | **2.16×** |
| 4 | 160.6 | 246.5 | **268.6** | 1.53× | **1.67×** |
| 8 | 229.6 | 290.9 | **314.8** | 1.26× | **1.36×** |
| 16 | 297.2 | 327.8 | **343.3** | 1.09× | **1.14×** |
| 24 | 329.4 | 335.7 | **353.5** | 1.01× | **1.06×** |
| 32 | 339.7 | 351.4 | **353.6** | 1.02× | **1.04×** |
| 48 | 354.2 | 352.9 | **366.4** | 0.99× | **1.03×** |
| 64 | 360.3 | 356.4 | **366.1** | 0.98× | **1.01×** |

where $l_{i,t}$ is the value of the logit for token $t$ at position $i$, and the *N*-gram score $s_{ng}$ is defined as

$$s_{ng} = \log(\Pr(t \mid \text{context}) + \epsilon),$$

where $\Pr(t \mid \text{context})$ is the conditional probability of token $t$ provided by the pretrained *N*-gram model, and $\epsilon$ is a small constant for numerical stability.

The final score increment is computed via the COMBINE function:

$$\text{COMBINE}(s_{\text{logit}}, s_{ng}) = (w_{\text{logit}} \cdot s_{\text{logit}} + w_{ng} \cdot s_{ng}) \cdot w_{\text{level}},$$

where the *N*-gram weight is fixed to $w_{ng} = 0.5$, and the logit weight decays with tree depth (*level* refers to the depth in the draft tree):

$$w_{\text{logit}} = 0.9^{\text{level}}.$$

This design prioritizes *N*-gram continuity at deeper levels because of the low accuracy of logit of later positions. The level weight $w_{\text{level}}$ is defined as

$$w_{\text{level}} = (\text{level} + 1)^{-0.7},$$

which encourages longer draft sequences by favoring nodes at greater depths.

In practice, we implement the pruning algorithm in `C++` and parallelize the it with `OpenMP`. We also bind both *N*-gram and worker threads to the same NUMA node to avoid expensive cross-node memory access overheads.

Overall, the proposed pruning strategy efficiently constructs a compact yet high-quality draft token tree by jointly considering model confidence, token continuity, and structural constraints.

## E. Larger Batch sizes Study Details

We provide additional results on the performance of DART and EAGLE3 under larger batch sizes. Specifically, we report throughput and relative speedup for EAGLE3 and DART on Qwen3-4B and Qwen3-8B, as summarized in Tables 6 and 7.

*Table 8.* Speedup ratios of different methods and mean average acceptance lengths $\tau$ on MT-Bench, HumanEval, Alpaca, Math500, CodeAlpaca, LiveCodeBench and MBPP, evaluated on a single NVIDIA A100 GPU. All Qwen models are from Qwen3 family, for example, Qwen4B represents Qwen3-4B.

| Model | Method | Speedup | | | | | | | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Alpaca | Codealpaca | Humaneval | LiveCode | Math500 | MBPP | MT-bench | Speedup | $\tau$ |
| Temperature=0 | | | | | | | | | | |
| Qwen1.7B | EAGLE3 | 1.91× | 2.26× | 2.32× | 2.28× | 2.46× | 2.19× | 2.09× | 2.21× | **3.80** |
| | **DART** | **2.31×** | **3.12×** | **2.88×** | **2.57×** | **2.94×** | **2.82×** | **2.50×** | **2.74×** | 3.60 |
| Qwen4B | EAGLE3 | 2.20× | 2.29× | 2.29× | 2.16× | 2.21× | 2.17× | 2.23× | 2.22× | 3.54 |
| | **DART** | **2.44×** | **3.61×** | **2.83×** | **2.69×** | **2.57×** | **3.23×** | **2.42×** | **2.83×** | **3.87** |
| Qwen8B | EAGLE3 | 2.16× | 2.49× | 2.46× | 2.24× | 2.49× | 2.40× | 2.10× | 2.33× | **3.72** |
| | **DART** | **2.52×** | **3.50×** | **3.07×** | **2.84×** | **2.60×** | **3.23×** | **2.06×** | **2.83×** | 3.61 |
| Qwen14B | EAGLE3 | 1.76× | 2.19× | 2.40× | 2.32× | 2.54× | 2.14× | 2.01× | 2.19× | 3.48 |
| | **DART** | **2.61×** | **3.40×** | **2.80×** | **2.83×** | **2.60×** | **3.21×** | **2.47×** | **2.84×** | **3.67** |

# F. DART Performance in A100

Given that speculative decoding is sensitive to hardware-specific balances between memory and compute, we further explore DART's cross-platform generalizability on NVIDIA A100-40G GPU. Under evaluation using a single A100-40G GPU, DART exhibits strong and consistent performance across the Qwen3 family (1.7B, 4B, 8B, and 14B). Specifically, DART achieves roughly 30% speedup over EAGLE3 (see Table 8), highlighting the hardware-agnostic efficiency of our proposed design.

# G. LLM Usage

Large language models were used only for limited editorial assistance, such as improving grammar and clarity. They did not contribute to research ideas, experimental design, analysis, or conclusions. All content was written and verified by the authors, who take full responsibility for the accuracy of the manuscript.