DFLARE: Scaling Up Draft Capacity for Block Diffusion Speculative Decoding

Anonymous ACL submission

Abstract

Block diffusion speculative decoding accelerates LLM inference by predicting all tokens within a block simultaneously for the target model to verify in parallel. Predicting an entire block at once requires a sufficiently capable draft model and effective utilization of the target model's internal knowledge. However, the state-of-the-art method DFlash constrains all draft layers to share a single fused representation derived from only a few target layers, limiting per-layer expressiveness and hindering further scaling of draft capacity. In this paper, we present DFLARE , which flares out the narrow conditioning bottleneck of DFlash through a lightweight layer-wise fusion mechanism: each draft layer attends to its own learnable combination of a broad set of target layers at negligible overhead, simultaneously injecting richer target knowledge and providing every draft layer with a distinct input. This enhanced per-layer expressiveness enables scaling the draft model to deeper architectures with consistent gains. We further scale training data from 800K to 2.4M samples to fully exploit the enlarged capacity. On six benchmarks spanning mathematical reasoning, code generation, and conversation, DFLARE attains average wall-clock speedups of $5.52\times$ on Qwen3-4B, $5.46\times$ on Qwen3-8B, and $3.91\times$ on GPT-OSS-20B, improving over DFlash by roughly 11%, 8%, and 5% respectively.

1 Introduction

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates LLM inference by employing a lightweight *draft model* to predict multiple future tokens, which the larger *target model* then verifies in a single parallel forward pass. This preserves the target distribution while substantially reducing wall-clock time (Xia et al., 2024). The resulting speedup critically depends on the *acceptance length*—the number of draft tokens accepted per verification step—which reflects how well the draft model approximates the target model (Liu et al., 2023; Zhou et al., 2023). Early approaches predominantly adopt autoregressive draft models (Leviathan et al., 2023; Chen et al., 2023; Li et al., 2024b, 2025c), but their sequential token-by-token generation means that each additional layer directly increases per-token latency, severely constraining the draft model's size and capacity.

Recently, *block diffusion* drafting (Sandler et al., 2025; Li et al., 2025a; Chen et al., 2026) has introduced a new paradigm: instead of generating candidate tokens autoregressively, it predicts all tokens within a block simultaneously through a discrete diffusion process in a single forward pass whose latency grows only marginally with the number of predicted tokens. This fundamentally changes the design space for draft models: since drafting cost is largely decoupled from block size, draft architectures with greater capability become practically viable and desirable for predicting more tokens within a block simultaneously.

To improve draft quality, two complementary strategies are commonly employed: (1) *scaling draft depth* to enhance modeling capacity (Du et al., 2024; Yan et al., 2025), which requires that the acceptance length gain from each additional layer outweighs the extra computational cost it introduces; and (2) *injecting target knowledge* by conditioning the draft model on the internal representations of the target model (Cai et al., 2024; Li et al., 2024b, 2025c), which requires the draft model to effectively leverage the injected information to boost acceptance length. To the best of our knowledge, research along these directions for block diffusion drafting remains preliminary. For instance, the current state-of-the-art method DFlash (Chen et al., 2026) has not fully exploited the potential of either improvement axis above. On the depth axis, DFlash employs only a modest number of draft layers and exhibits diminishing returns when more layers are

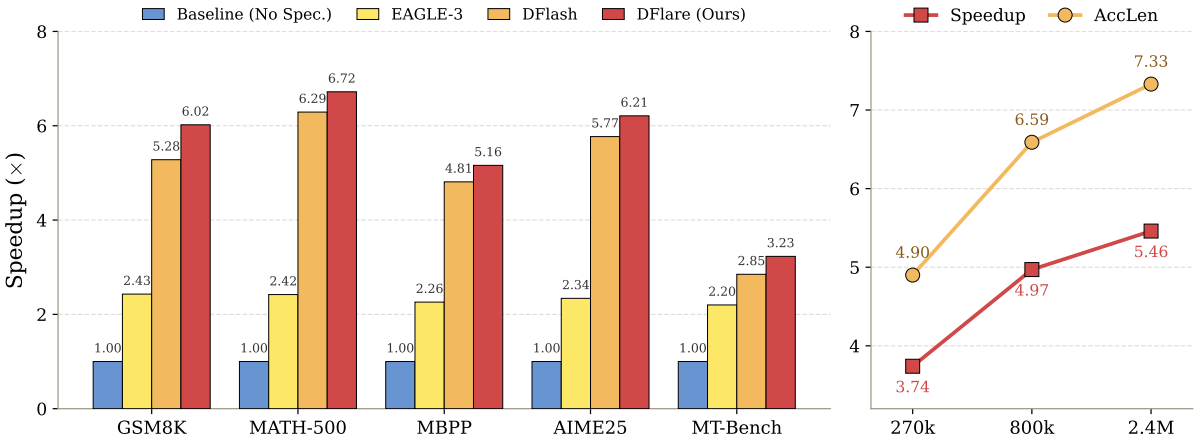

Figure 1: Left: wall-clock speedup (×) of different speculative decoding methods on Qwen3-8B across five benchmarks under greedy decoding; DFLARE consistently achieves the highest speedup, outperforming DFlash by a significant margin on every benchmark. Right: speedup and acceptance length of DFLARE on Qwen3-8B as the training data scales from 270k to 2.4M samples; DFLARE delivers consistent and substantial improvements as more training data becomes available.

added. On the knowledge injection axis, DFlash fuses hidden states from a few target layers into a single representation and shares it across all draft layers. Feeding the *same* fused signal to every layer prevents individual layers from specializing, which also explains why depth scaling saturates: without distinct input, additional layers have little room to contribute new expressiveness. Importantly, the two limitations are tightly coupled, unlocking one requires addressing the other.

To jointly address these coupled limitations, we present DFLARE , which flares out the narrow conditioning bottleneck of DFlash through broadening the target-model information flow from a single shared representation into distinct, layer-wise conditioning signals. Specifically, we introduce a *lightweight layer-wise fusion mechanism*: each draft layer learns its own weighted combination over target hidden states via a simple scalar-weighted sum, providing every layer with a distinct conditioning signal that encourages specialization. This differentiated input breaks the saturation ceiling, allowing each additional draft layer to contribute meaningful gains rather than redundant computation. Simultaneously, this mechanism enriches target knowledge injection by drawing from substantially more target layers than DFlash (e.g., 9 vs. 5) with minimal overhead, since the fusion requires only a handful of scalar weights per layer, adding negligible computation. The two improvements reinforce each other: richer per-layer knowledge makes depth scaling effective, while greater depth in turn amplifies the utility of the injected information. To fully exploit the enlarged model capacity, we further scale training data from 800K to 2.4 M samples, providing the supervision needed for the deeper, more expressive model to converge. On six benchmarks spanning mathematical reasoning, code generation, and general conversation, DFLARE attains average wall-clock speedups of 5.52× on Qwen3-4B, 5.46× on Qwen3-8B, and 3.91× on GPT-OSS-20B, improving over DFlash by roughly 11%, 8%, and 5% respectively, as shown in Figure 1 Left.

Our main contributions are as follows:

- **Scaling target knowledge per layer.** We introduce a lightweight layer-wise fusion mechanism that lets each draft layer attend to its own learnable combination of target layers at negligible cost. This allows incorporating far more target layers than prior work (e.g., 9 vs. 5 in DFlash), providing richer conditioning while giving every draft layer a differentiated view to strengthen per-layer expressiveness.
- **Scaling draft depth.** Building on the enhanced per-layer expressiveness, we scale the draft model to more layers and demonstrate that acceptance length improves consistently with depth—confirming that layer-wise fusion effectively unlocks the depth-scaling potential of block diffusion.
- **Scaling training data.** We scale the training corpus from 800K to 2.4 M samples to

match the enlarged capacity, the results are shown in Figure 1 Right. DFLARE keeps improving as more data is supplied, and ultimately attains average wall-clock speedups of $5.52\times$ on Qwen3-4B, $5.46\times$ on Qwen3-8B, and $3.91\times$ on GPT-OSS-20B, improving over DFlash by roughly 11%, 8%, and 5% respectively.

2 Related Work

2.1 Speculative Decoding

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates large language model inference by introducing a *draft-then-verify* paradigm (Xia et al., 2024). Various methods aim to improve draft model quality within this autoregressive framework—for instance, by distilling the target model (Zhou et al., 2023; Liu et al., 2023), utilizing target model features (Li et al., 2024b; Zhang et al., 2025; Du et al., 2024; Li et al., 2025c), or employing tree-based drafting strategies (Miao et al., 2023; Li et al., 2024a; Hu et al., 2025). Another line of work leverages parallel speculative decoding; for example, some approaches employ multiple token prediction heads for parallel prediction (Cai et al., 2024; Gloeckle et al., 2024), while others directly leverage pre-trained diffusion models as draft model (Christopher et al., 2025; Sandler et al., 2025; Li et al., 2025a; Liu et al., 2025). Notably, DFlash (Chen et al., 2026) trains a block diffusion draft model conditioned on the target model's hidden states via KV injection, achieving state-of-the-art acceleration.

2.2 Diffusion Language Models

Diffusion language models generate text by iteratively denoising corrupted sequences (Li et al., 2025b). Early works explored both continuous-space diffusion over token embeddings (Li et al., 2022; Strudel et al., 2022) and discrete-space diffusion with structured transition matrices (Austin et al., 2023; He et al., 2022). Recent advances in discrete diffusion have substantially improved training objectives and scalability through simplified masked diffusion losses and score-based formulations (Sahoo et al., 2024; Shi et al., 2025). Block diffusion (Arriola et al., 2025; Wu et al., 2025b,a) combines this parallelism with autoregressive structure by denoising sequences block-by-block, where tokens within each block are generated in parallel while blocks are produced sequentially.

3 Preliminaries

To facilitate understanding of our DFLARE method, this section presents the key components of DFlash, upon which our approach builds. We first describe how target-model information is extracted and injected into the draft model (Section 3.1), followed by the inference procedure encompassing the draft-and-verification loop (Section 3.2). Together, these two aspects illuminate the architecture of the block diffusion draft model and motivate the specific improvements introduced by our method. Finally, we describe the training procedure of the block diffusion draft model (Section 3.3), which our method also adopts.

3.1 Leveraging target model information

The draft model, with its limited capacity, struggles to precisely approximate the output distribution of the large-scale target model when drafting from scratch. A key insight in recent speculative decoding methods is to leverage the target model's internal hidden states as additional context for the draft model (Li et al., 2024b, 2025c). DFlash (Chen et al., 2026) implements this idea by fusing hidden states from multiple layers of the target model via a fully-connected (FC) layer.

Concretely, let $\mathbf{H}^{(j)} \in \mathbb{R}^{L\times d}$ $(j = 1, \ldots, T)$ denote the hidden state matrix from the j-th uniformly sampled target layer across all L context positions, where d is the hidden dimension. At each position t, the hidden states from all T layers are concatenated and projected through an FC layer to produce a context feature:

$$\mathbf{c}_t = \mathbf{W}_{\text{fc}}[\mathbf{h}_t^{(1)}; \ldots; \mathbf{h}_t^{(T)}] + \mathbf{b}_{\text{fc}}, \quad \mathbf{c}_t \in \mathbb{R}^d, \quad (1)$$

where $\mathbf{h}_t^{(j)} \in \mathbb{R}^d$ is the t-th row of $\mathbf{H}^{(j)}$, $\mathbf{W}_{\text{fc}} \in \mathbb{R}^{d\times Td}$, and $\mathbf{b}_{\text{fc}} \in \mathbb{R}^d$. The context feature $\mathbf{c}_t$ integrates semantic information from different depths of the target model's hierarchy. DFlash directly injects these context features into the Key and Value projections of *every* draft model layer through the KV cache, providing persistent target model conditioning throughout the draft model's depth (detailed in Section 3.2). A notable limitation of this design is that all draft layers receive the *identical* context feature $\mathbf{c}_t$, preventing different draft layers from specializing on different aspects of the target model's representation.

3.2 Inference of block diffusion draft model

At inference time, the block diffusion draft model and the target model operate in an iterative draft-then-verify loop. Each iteration proceeds in two phases.

Drafting phase. Let the target decode token at position t denote the latest token produced by the target model. The draft model takes this token as the starting point and generates $B-1$ candidate tokens in parallel for positions $t{+}1, \ldots, t{+}B{-}1$. To condition the draft model on target model knowledge, the fused context features $\mathbf{c}_t$ (Eq. 1) computed from all preceding positions—which we refer to as the *target context tokens*—are injected into the KV cache of every draft layer. Let $\mathbf{C}$ denote the matrix stacking the context features $\mathbf{c}_t$ for all L context positions. In each draft layer, the Keys and Values are formed by concatenating the projections from three sources: (1) the target context features $\mathbf{C}$ for L context positions, (2) the target decode token representation $\mathbf{x}_t \in \mathbb{R}^{1\times d}$, and (3) the draft masked-position hidden states $\hat{\mathbf{X}} \in \mathbb{R}^{(B-1)\times d}$:

$$\begin{aligned} \mathbf{K} &= \big[\mathbf{C}\,\mathbf{W}_K \;;\; \mathbf{x}_t\,\mathbf{W}_K \;;\; \hat{\mathbf{X}}\,\mathbf{W}_K\big], \\ \mathbf{V} &= \big[\mathbf{C}\,\mathbf{W}_V \;;\; \mathbf{x}_t\,\mathbf{W}_V \;;\; \hat{\mathbf{X}}\,\mathbf{W}_V\big], \end{aligned} \tag{2}$$

with Queries $\mathbf{Q} = [\mathbf{x}_t; \hat{\mathbf{X}}]\,\mathbf{W}_Q$, where $\mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_Q$ are the standard projection matrices of the draft model. Note that all three sources share the *same* KV projections. The attention is *bidirectional* within the block: every position attends to all other block positions and to the full target context, allowing the model to resolve inter-token dependencies in a single forward pass. The attention output is then processed by the MLP sub-layer (following the standard Transformer block structure). Only the B block positions (i.e., $\mathbf{x}_t$ and $\hat{\mathbf{X}}$) are passed to subsequent draft layers; the target context features $\mathbf{C}$ remain fixed across layers. After the final draft layer, the language modeling head produces logits at each masked position, from which $B-1$ candidate tokens are sampled.

Verification phase. The target decode token together with the $B-1$ candidates are fed into the target model for parallel verification. The target model evaluates all B positions in a single forward pass and accepts the longest prefix whose tokens match its own predictions. In addition to the accepted tokens, the target model generates one extra token at the position immediately following the last accepted token. This newly generated token serves as the target decode token for the next drafting iteration, and the target model's hidden states and KV cache are updated accordingly. This cycle repeats until generation is complete.

3.3 Training for block diffusion draft model

During training, multiple *anchor* positions are randomly sampled from the response. Each anchor serves as the target decode token $\mathbf{x}_t$ of a block, and the subsequent $B-1$ positions are masked as $\hat{\mathbf{X}}$—directly aligned with the inference procedure described in Section 3.2. All sampled blocks are concatenated into a single sequence and processed jointly using a sparse attention mask: tokens attend bidirectionally within the same block and to the corresponding target context features, while attention across different blocks is disallowed. This design enables multiple draft blocks to be trained efficiently within a single forward and backward pass. To improve training efficiency, the draft model shares the token embedding layer and the language modeling head with the target model and keeps them frozen; only the draft Transformer layers are updated.

4 Method

Building upon DFlash (Section 3.1–3.3), we present DFLARE with three improvements that collectively *strengthen the per-layer capacity* of the draft model: (1) a lightweight layer-wise fusion mechanism that replaces the FC projection, allowing each draft layer to receive its own dedicated combination of target hidden states and thereby maximizing the expressiveness of every single layer; (2) heterogeneous KV projections that decouple the representational spaces for draft and target information, granting each layer additional degrees of freedom to independently extract and utilize target knowledge; and (3) a progressive position-weighted training loss that improves training efficiency so the enlarged per-layer capacity is fully exploited. Together, these designs ensure that each draft layer becomes a more powerful computational unit, enabling the model to scale to greater depth with consistent gains. We describe each component below.

4.1 Adaptive Layer Fusion of Target Hidden States

Different layers of the target model encode distinct levels of abstraction—from shallow syntactic

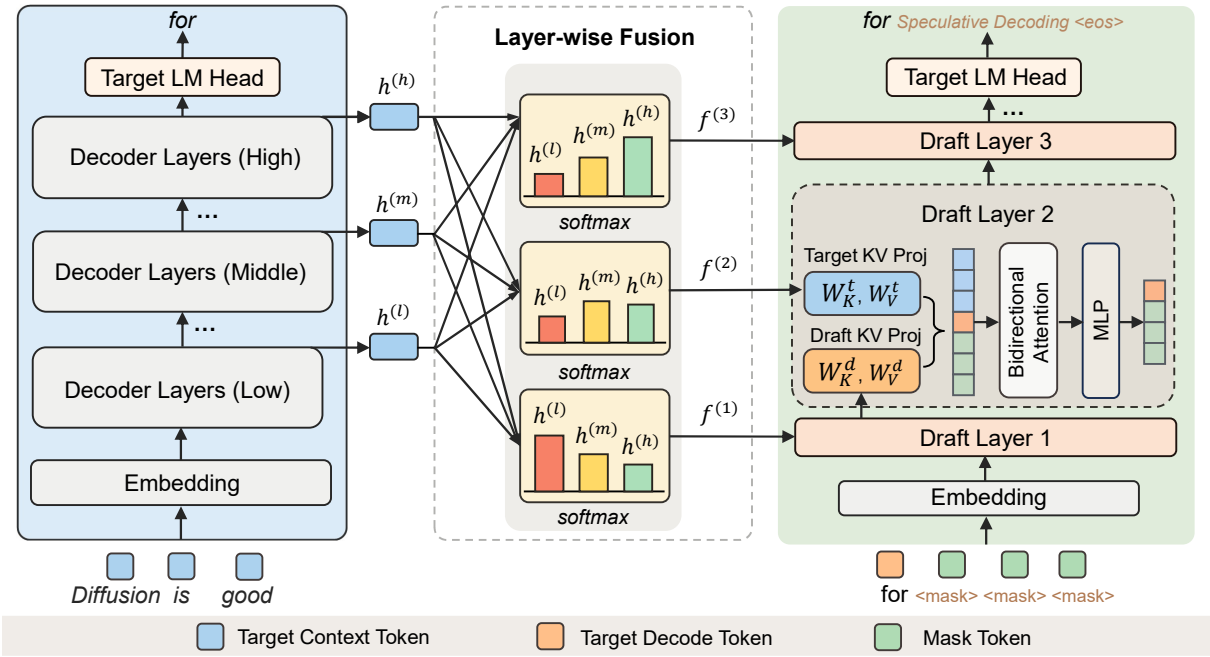

Figure 2: The overview of our DFLARE method. DFLARE utilizes adaptive layer fusion of target hidden states and heterogeneous KV projections to enhance per-layer expressiveness.

patterns to deep semantic representations. To effectively inject this multi-granularity knowledge into the draft model, DFLARE introduces *Adaptive Layer Fusion*: a lightweight, layer-specific mechanism that allows each draft layer to learn its own combination of target hidden states. Unlike prior approaches that project the concatenated target states through an FC layer to produce a single, shared fused context for all draft layers (Eq. 1), our method provides each draft layer with a dedicated view of the target model.

Concretely, let $\mathbf{h}_t^{(j)} \in \mathbb{R}^d$ denote the hidden state at position t from the j-th selected target layer ($j = 1, \ldots, T$). We introduce a learnable fusion weight matrix $\mathbf{W}^{\text{fuse}} \in \mathbb{R}^{D \times T}$, where D is the number of draft layers. For each draft layer i, a layer-specific fused representation is computed via:

$$\boldsymbol{\alpha}^{(i)} = \text{softmax}\big(\mathbf{W}^{\text{fuse}}_{i,:}\big) \in \mathbb{R}^T,$$
$$\mathbf{f}_t^{(i)} = \text{RMSNorm}\left(\sum_{j=1}^{T} \alpha_j^{(i)} \mathbf{h}_t^{(j)}\right), \tag{3}$$

where $\alpha_j^{(i)}$ is the softmax-normalized weight that draft layer i assigns to target layer j. Once training is complete, the softmax-normalized weights $\boldsymbol{\alpha}^{(i)}$ can be precomputed and cached. At inference time, the fusion reduces to a scalar-weighted sum of the T target hidden states per draft layer, introducing virtually no additional latency.

This design offers two key advantages. First, it is *extremely lightweight*: the fusion introduces only $D \times T$ scalar parameters (e.g., 63 for $D{=}7, T{=}9$), incurring only minimal additional overhead when more target layers are incorporated—allowing the draft model to absorb richer knowledge from the target model. Second, it is *layer-specific*: each draft layer learns its own fusion coefficients, enabling each draft layer to receive information suited to its role in the model hierarchy.

4.2 Heterogeneous KV Projections

The fused target context $\mathbf{f}_t^{(i)}$ is injected into each draft layer through the Key-Value (KV) cache, supplying the draft model with information from the target model. In DFlash, the draft model's own hidden states and the injected target context share the same KV projection matrices (Eq. 2). However, these two sources carry different information: the draft hidden states capture the evolving noisy predictions within the diffusion process, while the target context encodes semantic knowledge from the target model. Sharing a single projection forces the model to compromise between two distinct representational spaces. DFLARE addresses this by introducing separate projection matrices for each source. Let $\mathbf{F}^{(i)} \in \mathbb{R}^{L \times d}$ denote the layer-specific fused target context at draft layer i (obtained via Eq. 3), $\mathbf{x}_t \in \mathbb{R}^{1 \times d}$ the target decode token repre-

sentation, and $\hat{\mathbf{X}} \in \mathbb{R}^{(B-1)\times d}$ the draft masked-position hidden states. The Keys and Values are computed and concatenated as:

$$\begin{aligned} \mathbf{K} &= \left[\mathbf{F}^{(i)}\mathbf{W}_K^t \,;\, \mathbf{x}_t\,\mathbf{W}_K^d \,;\, \hat{\mathbf{X}}\,\mathbf{W}_K^d\right], \\ \mathbf{V} &= \left[\mathbf{F}^{(i)}\mathbf{W}_V^t \,;\, \mathbf{x}_t\,\mathbf{W}_V^d \,;\, \hat{\mathbf{X}}\,\mathbf{W}_V^d\right], \end{aligned} \tag{4}$$

where $\mathbf{W}_K^t, \mathbf{W}_V^t \in \mathbb{R}^{d\times d_{\text{kv}}}$ are learnable projections dedicated to the target context, and $\mathbf{W}_K^d, \mathbf{W}_V^d \in \mathbb{R}^{d\times d_{\text{kv}}}$ are independent projections shared by the target decode token and the masked draft positions. The Queries are derived from the block positions: $\mathbf{Q} = [\mathbf{x}_t; \hat{\mathbf{X}}]\,\mathbf{W}_Q$. This heterogeneous design grants the target context and the draft block positions independent representational subspaces in the attention mechanism, enabling the draft model to better extract and utilize the target knowledge. The resulting Keys and Values are then used with the bidirectional attention mechanism described in Section 3.2 to compute the attention output.

4.3 Progressive Position-Weighted Loss

In speculative decoding, errors at early positions within a draft block invalidate all subsequent tokens, making early-position accuracy disproportionately important. DFlash addresses this by applying an exponentially decaying weight $w_k = \exp(-(k-1)/\gamma)$ to the cross-entropy loss at each position k within a block, where γ controls the decay rate. However, DFlash uses a fixed γ throughout training: a small γ concentrates learning on early positions, enabling fast convergence but under-optimizing harder tail positions; a large γ spreads the weight more uniformly but slows convergence on the easy early positions. We resolve this tension with a simple linear warmup:

$$\begin{aligned} \gamma(s) &= \gamma_0 + \frac{s}{S}\,(\gamma_{\max} - \gamma_0), \\ w_k(s) &= \exp\left(-\frac{k-1}{\gamma(s)}\right), \end{aligned} \tag{5}$$

where s is the current training step and S is the total number of steps. In the early phase ($\gamma \approx \gamma_0$, small), the model focuses on mastering the easy first few positions. As γ increases toward $\gamma_{\max}$, the weight distribution flattens and the model progressively shifts its effort to later, harder positions. This curriculum-style schedule ensures that by the end of training, the model has been thoroughly optimized across all block positions, yielding longer accepted sequences without sacrificing early-position accuracy.

5 Experiments

5.1 Setup

Models and Evaluations We conduct experiments on Qwen3-4B and Qwen3-8B (Team, 2025) and GPT-OSS-20B (Agarwal et al., 2025) pre-trained models. We evaluate tasks in three categories: Math: GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), and AIME25 (Zhang and Math-AI, 2025); Code: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021); Chat: MTBench (Zheng et al., 2023). More details are presented in Section A.2.

Datasets To provide a diverse set of training data, we collect a mixture of around 2.4M samples from NVIDIA Nemotron Post-Training Dataset V2 (NVIDIA, 2025), CodeAlpaca (Chaudhary, 2023) and Step-3.5-Flash-SFT (Huang et al., 2026). More details are presented in Section A.5.

Implementation For DFLARE draft models, we set the number of draft layers to 7 and use a block size of 16. The target hidden features are extracted from 9 layers uniformly selected between the second layer and the third-to-last layer of the target model (8, 8, 7 for GPT-OSS). More details are presented in Section A.1 and Section A.2.

5.2 Main Results

Table 1 reports the main results across six benchmarks spanning mathematical reasoning, code generation, and general conversation, evaluated on three target models: Qwen3-4B, Qwen3-8B, and GPT-OSS-20B. DFLARE consistently outperforms DFlash across all benchmarks, target models, and temperature settings. Under greedy decoding, DFLARE improves the average acceptance length by 15.5% on Qwen3-4B, 14.7% on Qwen3-8B, and 5.8% on GPT-OSS-20B, with corresponding wall-clock speedup gains of 10.6%, 8.1%, and 5.4% respectively. The improvements vary across domains. Mathematical reasoning benchmarks benefit the most, code generation benchmarks also show substantial improvements. Conversational tasks exhibit relatively lower absolute acceptance lengths due to the diversity of open-ended generation, yet DFLARE still delivers the largest *relative* gain on MT-Bench. For the substantially larger GPT-OSS-20B target, both DFlash and DFLARE attain lower absolute acceptance lengths than on the Qwen3 targets, which is expected given the larger representational gap the draft model must

Model	Method	MATH						CODE				CHAT		Avg.	
		GSM8K		*MATH500*		*AIME25*		*HumanEval*		*MBPP*		*MT-Bench*			
Temperature = 0		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Q3-4B	EAGLE3	2.56	3.76	2.57	3.70	2.46	3.58	2.52	3.58	2.41	3.51	2.39	3.57	2.49	3.62
	DFlash	5.31	6.51	6.20	7.80	5.78	7.31	4.62	6.65	4.89	6.13	3.13	4.40	4.99	6.47
	DFLARE	**6.00**	**7.93**	**6.76**	**8.97**	**6.31**	**8.35**	**5.41**	**7.17**	**5.30**	**7.08**	**3.33**	**5.30**	**5.52**	**7.47**
Q3-8B	EAGLE3	2.43	3.53	2.42	3.55	2.34	3.51	2.46	3.67	2.26	3.31	2.20	3.25	2.35	3.47
	DFlash	5.28	6.53	6.29	7.88	5.77	7.11	5.32	6.54	4.81	5.97	2.85	4.30	5.05	6.39
	DFLARE	**6.02**	**7.88**	**6.72**	**8.95**	**6.21**	**8.12**	**5.42**	**7.08**	**5.16**	**6.86**	**3.23**	**5.09**	**5.46**	**7.33**
GPT-20B	DFlash	3.75	4.54	3.96	4.86	3.85	4.93	3.49	4.25	3.44	4.20	3.74	5.20	3.71	4.66
	DFLARE	**4.06**	**4.98**	**4.20**	**5.19**	**3.88**	**4.98**	**3.71**	**4.55**	**3.72**	**4.58**	**3.87**	**5.32**	**3.91**	**4.93**
Temperature = 1		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Q3-4B	EAGLE3	2.37	3.71	2.43	3.57	2.14	3.26	2.36	3.57	2.30	3.48	2.22	3.49	2.30	3.51
	DFlash	4.80	6.00	5.22	6.75	**4.02**	5.18	4.85	6.04	4.50	5.61	2.69	4.03	4.35	5.60
	DFLARE	**5.38**	**7.14**	**5.33**	**7.34**	3.84	**5.20**	**4.87**	**6.43**	**4.75**	**6.39**	**3.00**	**4.77**	**4.53**	**6.21**
Q3-8B	EAGLE3	2.23	3.44	2.20	3.40	2.11	3.11	2.31	3.54	2.12	3.25	1.99	3.06	2.16	3.30
	DFlash	4.83	5.96	4.99	6.39	3.80	4.82	**4.52**	5.54	4.26	5.29	2.57	3.79	4.16	5.30
	DFLARE	**5.34**	**7.07**	**5.25**	**7.18**	**3.84**	**5.24**	4.41	**5.76**	**4.31**	**5.77**	**2.81**	**4.37**	**4.33**	**5.90**
GPT-20B	DFlash	2.95	3.64	2.99	3.76	2.04	3.10	2.63	3.20	2.58	3.13	1.76	2.76	2.49	3.26
	DFLARE	**3.07**	**3.83**	**3.08**	**3.91**	**2.05**	**3.14**	**2.67**	**3.28**	**2.69**	**3.30**	**1.81**	**2.89**	**2.56**	**3.39**

Table 1: Main results on Qwen3-4B, Qwen3-8B and GPT-OSS-20B across mathematical reasoning, code generation, and conversation benchmarks under greedy and stochastic decoding. Speedup denotes wall-clock speedup ratio and τ denotes acceptance length. Best results in each section are in **bold**.

Setting	GSM8k		Humaneval		MtBench		Avg	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFLARE	**3.80**	**4.94**	3.55	4.62	2.76	4.00	**3.37**	**4.52**
− Softmax	3.70	4.85	3.45	4.53	2.77	4.02	3.31	4.47
− KVProj	3.77	4.90	3.50	4.54	2.78	3.99	3.35	4.48
− Loss	3.72	4.84	**3.56**	**4.63**	2.73	3.93	3.34	4.47
+ FC	3.74	4.87	3.53	4.59	**2.78**	**4.06**	3.35	4.51

Table 2: Ablation on model structure design with Qwen3-4B as target model. Best results in each column are in **bold**.

bridge; nevertheless, DFLARE retains a consistent advantage over DFlash across every benchmark. Under stochastic decoding, acceptance lengths decrease across all methods as sampling introduces additional mismatch between draft and target distributions, but DFLARE maintains clear margins of 10.9%, 11.3%, and 3.9% in average acceptance length on Qwen3-4B, Qwen3-8B, and GPT-OSS-20B respectively, demonstrating the robustness of the proposed method.

6 Analysis

6.1 Number of Draft Layers

We investigate how the number of draft model layers affects acceptance length and end-to-end speedup under our proposed architecture. To enable a fair per-layer comparison between DFLARE and DFlash, we train both methods on the full DFlash training dataset and evaluate draft models with 5, 6, and 7 layers under otherwise identical settings. The results are visualized in Figure 3 Left.

DFLARE consistently outperforms DFlash in both acceptance length and wall-clock speedup at every layer configuration, confirming that our layer-wise fusion mechanism strengthens the expressiveness of each individual draft layer. More importantly, the two methods exhibit qualitatively different scaling behaviors as draft depth increases. For DFLARE , both acceptance length and speedup improve steadily from 5 to 7 layers—each additional layer yields a meaningful gain, indicating that the model continues to benefit from increased depth. In DFLARE , each draft layer attends to its own dedicated combination of target hidden states, ensuring that every additional layer introduces genuinely new representational capacity. The consistent scaling of acceptance length with depth confirms that our layer-wise fusion effectively unlocks the depth scaling potential inherent to block diffusion drafting.

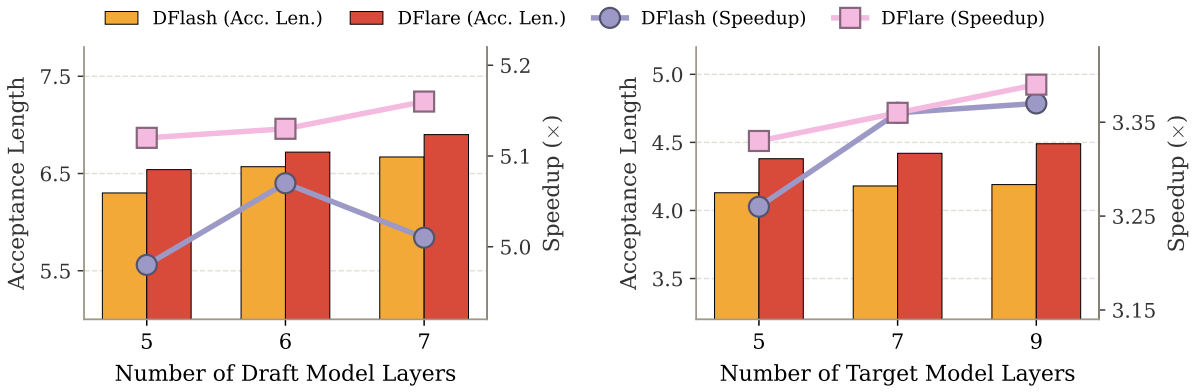

Figure 3: The impact of draft model layers (Left) and target model layers (Right) on the performance of DFlash and DFLARE . The target model is Qwen3-4B.

6.2 Number of Target Hidden Features

We investigate how the number of fused target layers T affects performance, and compare behavior of DFLARE and DFlash as more target information is incorporated. Both methods are evaluated with $T \in \{5, 7, 9\}$ target layers under otherwise identical settings. The results are visualized in Figure 3 Right. DFLARE demonstrates a clear ability to effectively leverage additional target layers. As T increases from 5 to 9, acceptance length improves steadily and consistently, indicating that our lightweight layer-wise fusion successfully extracts complementary information from deeper target layers and translates it into higher-quality draft predictions. The speedup also improves monotonically, confirming that the richer target knowledge leads to longer accepted sequences that more than compensate for any marginal increase in per-step cost. In contrast, while increasing from 5 to 7 target layers provides a modest improvement, further scaling to 9 layers yields almost no additional acceptance length gain on DFlash. The divergence between the two methods highlights the advantage of DFLARE 's layer-wise fusion design.

6.3 Model Structures Design

We ablate the key architectural choices of DFLARE to validate our structure design. Specifically, we consider four variants: (1) $-$**Softmax**: removing the softmax normalization on the per-layer fusion weights in Section 4.1, leaving the learned combination unnormalized; (2) $-$**KVProj**: removing the heterogeneous KV projections described in Section 4.2, so that the draft model's own hidden states and the fused target context share the same KV projection matrices; (3) $-$**Loss**: removing the progressive position-weighted loss strategy described in Section 4.3; and (4) $+$**FC**: appending an additional fully-connected layer after the layer-wise fusion to further transform the fused representation before injection. Table 2 reports the results. Among the ablations, removing softmax normalization causes the most significant degradation, indicating that constraining the fusion weights is essential for stable. Removing the heterogeneous KV projections also leads to a consistent drop in both acceptance length and speedup. Furthermore, the removal of the progressive position-weighted loss also results in a performance drop, demonstrating that our specialized loss design effectively guides the model to achieve a higher acceptance length. Interestingly, adding an extra FC layer after fusion does not improve acceptance length on average. Overall, the DFLARE configuration achieves the best overall acceptance length and speedup, confirming that all design choices are well-motivated.

7 Conclusion

We presented DFLARE , a method that scales up draft model capacity for block diffusion speculative decoding along three axes: target knowledge injection, draft model depth, and training data. By enabling each draft layer to learn its own fusion over target hidden states, DFLARE unlocks effective depth scaling and benefits consistently from additional training data. On six benchmarks spanning mathematical reasoning, code generation, and conversation, DFLARE attains average wall-clock speedups of $5.52\times$ on Qwen3-4B, $5.46\times$ on Qwen3-8B, and $3.91\times$ on GPT-OSS-20B, improving over state-of-the-art DFlash by roughly 11%, 8%, and 5% respectively.

Limitations

Our work has two main limitations. First, the training cost of DFLARE is high due to the large draft model and the scaled training corpus. However, since training is a one-time cost while the resulting draft model is deployed for all subsequent inference requests, the cumulative time savings during serving can quickly amortize the additional training investment. Second, increasing the training data is likely to yield further gains in acceptance length, yet we were unable to explore this direction due to computational resource constraints. We leave the investigation of even larger-scale training to future work.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2023. Structured denoising diffusion models in discrete state-spaces. *Preprint*, arXiv:2107.03006.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, pages 5209–5235.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

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

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Jacob K Christopher, Brian R Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. 2025. Speculative diffusion decoding: Accelerating language generation through diffusion. *Preprint*, arXiv:2408.05636.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

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

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. 2024. Glide with a cape: A low-hassle method to accelerate speculative decoding. *Preprint*, arXiv:2402.02082.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. 2022. Diffusionbert: Improving generative masked language models with diffusion models. *Preprint*, arXiv:2211.15029.

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

Shijing Hu, Jingyang Li, Zhihui Lu, and Pan Zhou. 2025. Bridging draft policy misalignment: Group tree optimization for speculative decoding. *arXiv preprint arXiv:2509.22134*.

Ailin Huang, Ang Li, Aobo Kong, Bin Wang, Binxing Jiao, Bo Dong, Bojun Wang, Boyu Chen, Brian Li, Buyun Ma, and 1 others. 2026. Step 3.5 flash: Open frontier-level intelligence with 11b active parameters. *arXiv preprint arXiv:2602.10604*.

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

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. 2025a. Diffuspec: Unlocking diffusion language models for speculative decoding. *Preprint*, arXiv:2510.02358.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. 2025b. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. 2022. Diffusion-lm improves controllable text generation. *Preprint*, arXiv:2205.14217.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. *Preprint*, arXiv:2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*.

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

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. 2025. Tidar: Think in diffusion, talk in autoregression. *Preprint*, arXiv:2511.08923.

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

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, and 1 others. 2023. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*.

NVIDIA. 2025. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model. *Preprint*, arXiv:2508.14444.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. *Preprint*, arXiv:2406.07524.

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

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. 2025. Simplified and generalized masked diffusion for discrete data. *Preprint*, arXiv:2406.04329.

Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, and Rémi Leblond. 2022. Self-conditioned embedding diffusion for text generation. *Preprint*, arXiv:2211.04236.

Hunyuan AI Infra Team. 2026. Angelslim: A more accessible, comprehensive, and efficient toolkit for large model compression. *arXiv preprint arXiv:2602.21233*.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

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

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Siyuan Yan, Mo Zhu, Guo qing Jiang, Jianfei Wang, Jiaxing Chen, Wentai Zhang, Xiang Liao, Xiao Cui, Chen Zhang, Zhuoran Song, and Ran Zhu. 2025. Scaling laws for speculative decoding. *Preprint*, arXiv:2505.07858.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2025. Learning harmonized representations for speculative sampling. *Preprint*, arXiv:2408.15766.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. *Preprint*, arXiv:2312.07104.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*.

A Appendix

A.1 Training Implementation

The draft models are optimized for 6 epochs using AdamW with a learning rate of 6e−4 , a gradient clipping threshold of 1.0, and a cosine schedule with a warmup ratio of 0.04. We train on our training data mixture with a maximum sequence length of 3072 tokens; for each sequence, 512 anchor positions are randomly sampled. For the progressive position-weighted loss introduced in Section 4.3, we set the initial decay parameter to $\gamma_0 = 4.5$ and increase γ by 1 after each training epoch. For GPT-OSS-20B, we instead keep γ fixed at 4 throughout training, since its block size is only 8 and the loss-weight schedule has a much smaller effect on later-position tokens in such a short block. Our main results use the full 2.4 M-sample training corpus with a global batch size of 64. For ablation studies conducted on UltraChat and ShareGPT, as well as for the data-scaling comparison using the 800K-sample Nemotron and CodeAlpaca mixture adopted by DFlash, we use a global batch size of 32 while keeping all other optimization hyperparameters unchanged. For every dataset, we keep only the original prompts and regenerate the responses with the corresponding target model using a sampling temperature of 0.6; the draft model is then trained to predict only these regenerated response tokens, so that its training distribution matches the target model's own output distribution.

A.2 Baselines

We compare DFLARE with the vanilla autoregressive decoding (baseline) and autoregressive speculative decoding method EAGLE3 (Li et al., 2025c) and state-of-the-art method DFlash (Chen et al., 2026). For EAGLE3 on Qwen3 models, we use the checkpoints released by AngelSlim (Team, 2026). For DFlash, we use the official checkpoints released by DFlash team. For each task, we assess the performance of the draft models using average acceptance length τ and end-to-end decoding speedup over the autoregressive baseline. We conduct all experiments on NVIDIA H20 GPUs unless otherwise specified.

A.3 Experiments compute resources

DFLARE draft models trained on the full 2.4 M-sample corpus used 32 GPUs with distributed data parallelism. Under this setting, training the draft model for the Qwen3-8B target takes approximately 160 GPU-hours per device (i.e., $\sim$ 160 h wall-clock on 32 GPUs), the draft model for Qwen3-4B takes approximately 100 h, and the draft model for GPT-OSS-20B takes approximately 90 h. Inference-time evaluation is conducted on a single GPU for each target model, consistent with the deployment scenario of speculative decoding. A complete pass over the full evaluation benchmark takes approximately 4h on a single GPU for the Qwen3-8B target.

A.4 Performance on SGLang

To evaluate the practical acceleration in realistic serving scenarios, we deploy DFLARE on SGLang (Zheng et al., 2024) and measure end-to-end throughput under varying concurrency levels on an H20 GPU. As shown in Table 3, DFLARE consistently achieves the highest throughput across all concurrency levels on both GSM8K and HumanEval. As concurrency increases and the system becomes more compute-bound, the advantage of DFLARE over DFlash grows more pronounced, demonstrating that our method scales favorably under higher load.

A.5 Detailed Results of the Ablation Study

In this section, we present the underlying data for the figures shown in Section 6.1 and Sections 6.2. We use ShareGPT and UltraChat200K (Ding et al., 2023) to do ablation study for comparison with Baseline Models. For the ablation study of number of draft layers, we use NVIDIA Nemotron Post-Training Dataset V2 (NVIDIA, 2025), CodeAlpaca (Chaudhary, 2023) which is the full DFlash training dataset. To prevent test set leakage, we removed all samples from the training set that had a 32-gram or higher overlap with the test set.

A.6 Scaling Training Data

Detailed results on **Qwen3-8B** across different training data volumes in Figure 1.

A.7 Software Dependencies

Our codebase is implemented in Python, primarily relying on `torch` (v2.9.1), `transformers` (v4.57.1), `datasets` (v4.8.4), `sglang` (v0.5.6), and `numpy` (v2.4.3).

Task	Method	Concurrency				
		1	4	8	16	32
GSM8K	Baseline	162.5	610.8	1,058.8	1,046.9	1,053.7
	DFlash	596.6	1,048.1	1,057.6	1,045.5	1,068.6
	DFLARE	**642.6**	**1,764.8**	**1,815.2**	**1,828.7**	**1,809.3**
HumanEval	Baseline	163.6	620.9	1,158.2	1,273.7	1,212.0
	DFlash	**561.1**	1,142.9	1,178.7	1,174.9	1,149.9
	DFLARE	554.5	**1,538.6**	**1,650.2**	**1,672.1**	**1,796.9**

Table 3: Throughput (tokens/s) on SGLang with H20 GPU for Qwen3-8B under varying concurrency levels. Best results are in **bold**.

Method	Layers	MATH						CODE				CHAT		Avg.	
		GSM8K		*MATH500*		*AIME25*		*HumanEval*		*MBPP*		*MT-Bench*			
Temperature = 0		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
DFlash	5 Layers	3.63	4.47	3.50	4.28	3.39	4.14	3.45	4.25	3.11	3.84	2.78	3.78	3.31	4.13
	7 Layers	3.70	4.55	3.54	4.34	3.45	4.19	3.51	4.33	3.17	3.88	**2.81**	3.80	3.36	4.18
	9 Layers	3.73	4.59	3.53	4.33	**3.52**	4.30	3.53	4.34	3.11	3.81	2.78	3.78	3.37	4.19
DFLARE	5 Layers	3.72	4.83	3.49	4.52	3.38	4.36	3.50	4.55	3.14	4.07	2.73	3.92	3.33	4.38
	7 Layers	3.70	4.82	3.51	4.53	3.49	4.48	3.53	4.59	**3.19**	**4.16**	2.73	3.92	3.36	4.42
	9 Layers	**3.80**	**4.94**	**3.57**	**4.67**	3.51	**4.52**	**3.55**	**4.62**	3.17	**4.16**	2.76	**4.00**	**3.39**	**4.49**

Table 4: Detailed ablation study results on Qwen3-4B about number of target layers across mathematical reasoning, code generation, and conversation benchmarks. Speedup denotes wall-clock speedup ratio and τ denotes acceptance length. Best results in each category are in **bold**.

Method	Layers	MATH						CODE				CHAT		Avg.	
		GSM8K		*MATH500*		*AIME25*		*HumanEval*		*MBPP*		*MT-Bench*			
Temperature = 0		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
DFlash	5 Layers	5.30	6.57	6.17	7.70	5.97	7.40	4.61	5.71	4.66	5.85	3.14	4.58	4.98	6.30
	6 Layers	5.38	6.82	6.27	8.03	6.30	7.89	4.66	5.92	4.70	6.04	3.11	4.71	5.07	6.57
	7 Layers	5.43	7.06	6.31	8.26	5.98	7.78	4.60	5.98	4.65	6.12	3.11	4.80	5.01	6.67
DFLARE	5 Layers	5.42	6.85	6.35	8.03	6.16	7.68	**4.73**	5.88	**4.81**	6.10	**3.22**	4.77	5.12	6.55
	6 Layers	5.56	7.13	6.32	8.19	6.23	7.96	4.71	6.03	4.77	6.21	3.16	4.79	5.13	6.72
	7 Layers	**5.58**	**7.29**	**6.37**	**8.41**	**6.36**	**8.27**	4.70	**6.16**	4.80	**6.38**	3.15	**4.88**	**5.16**	**6.90**

Table 5: Detailed ablation study results on Qwen3-4B across different draft layer settings across mathematical reasoning, code generation, and conversation benchmarks. Speedup denotes wall-clock speedup ratio and τ denotes acceptance length. Best results in each category are in **bold**.

Method	Data	MATH						CODE				CHAT		Avg.	
		GSM8K		*MATH500*		*AIME25*		*HumanEval*		*MBPP*		*MT-Bench*			
Temperature = 0		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
DFLARE	270K	4.40	5.70	4.00	5.19	3.75	4.81	3.87	4.98	3.58	4.63	2.84	4.09	3.74	4.90
	800K	5.35	6.94	6.23	8.23	5.97	7.74	4.65	6.01	4.59	6.03	3.02	4.61	4.97	6.59
	2.4M	**6.02**	**7.88**	**6.72**	**8.95**	**6.21**	**8.12**	**5.42**	**7.08**	**5.16**	**6.86**	**3.23**	**5.09**	**5.46**	**7.33**

Table 6: Detailed results on **Qwen3-8B** across different training data volumes. Speedup denotes wall-clock speedup ratio and τ denotes acceptance length. Best results in each category are in **bold**.