ADAFLASH: ADAPTIVE SPECULATIVE DECODING VIA ON-POLICY DISTILLED DIFFUSION DRAFTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding, in which a lightweight draft model first generates a draft sequence that is then verified by the target model, has become a prevalent paradigm for accelerating large language model inference. Recent work such as DFlash further boosts drafting efficiency by leveraging diffusion drafters, whose parallel denoising mechanism enables draft generation in a single forward pass. In this work, we uncover a *central pitfall* of diffusion drafters: the *global dependency*, arising from bidirectional attention and KV injection in diffusion drafters, is a double-edged sword. On one hand, it endows the model with parallel generation and global contextual modeling capabilities; on the other hand, this global dependency introduces *high variance* at both the domain-level and the token-level: acceptance rates fluctuate substantially across different domains, and the acceptance probability varies across draft positions. To tackle this issue, we propose the ADAFLASH framework, comprising two components: (i) an *on-policy distillation* (OPD) algorithm with reverse-KL divergence tailored for diffusion drafters, which delivers stable convergence and continuously adapts the drafter to the deployment distribution; and (ii) an *adaptive length head* that dynamically adjusts the candidate length on the fly, substantially lowering the verification cost of the target model. Experiments demonstrate that ADAFLASH consistently improves the speedup during deployment, with especially significant gains under high-concurrency, achieving up to 66% higher average throughput than previous state-of-the-art methods.

1 INTRODUCTION

Large language models (LLMs) have demonstrated impressive capabilities across a wide range of tasks (Brown et al., 2020; Ouyang et al., 2022). However, this progress comes at a price of growing computational cost: emerging techniques, including chain-of-thought (Wei et al., 2022), multi-step reasoning (Guo et al., 2025), and long-horizon agentic tasks (Xi et al., 2025), substantially increase the burden of inference. To reduce the inference latency, *speculative decoding* (Leviathan et al., 2023; Chen et al., 2023) has emerged as a widely adopted paradigm, in which a lightweight draft model rapidly generates a draft sequence that is then verified in parallel by a larger target model, achieving lossless acceleration (Li et al., 2024b; Fu et al., 2024; Hu et al., 2025).

Beyond standard autoregressive (AR) drafters, growing interest in the community has led several recent works (Christopher et al., 2025; Sandler et al., 2026; Li et al., 2026) to explore diffusion language models (dLLMs) (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025) as non-autoregressive drafters, leveraging their *bidirectional attention* and mask-and-denoise mechanism to generate multiple draft tokens simultaneously. SpecDiff (Christopher et al., 2025) first demonstrated the feasibility of parallel diffusion drafting with few denoising steps; SpecDiff-2 (Sandler et al., 2026) further introduced distillation mechanisms to align the diffusion drafter with the AR verifier. Building on top of these works, DFlash (Chen et al., 2026a) proposed a lightweight block diffusion drafter that generates all k draft tokens through a single forward pass; crucially, it conditions each drafting layer on multi-layer hidden features from the target model, which improves the draft–target alignment and yields high acceptance rates.

In this work, we uncover a central pitfall that the *global dependency* in diffusion drafters is a *double-edged sword*. Specifically, global dependency arises from two components: bidirectional attention, which allows each draft token to attend to all other tokens in the draft block, and KV injection, which

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

feeds the target hidden features into every drafting layer. While this global dependency enables global contextual modeling and one-pass parallel generation, it also introduces *high variance* in draft quality. As illustrated in Figure 1, this variance manifests at two levels: (i) *domain-level* variance: the acceptance rate fluctuates significantly across task domains, much more so than that of AR drafters; and (ii) *token-level* variance: the acceptance probability varies across draft positions, so the viable draft length changes drastically across rounds. Consequently, a static system with a fixed diffusion drafter and a constant candidate length suffers from degraded acceptance rates on out-of-distribution domains and wastes computation on tokens unlikely to be accepted.

To address these challenges, we propose ADAFLASH, an online adaptive framework for diffusion-based speculative decoding with two components: (i) an *on-policy distillation* (OPD) algorithm with reverse-KL and entry-wise divergence clipping, tailored specifically to diffusion drafters, which delivers stable convergence and continuously adapts the drafter to the deployment distribution; and (ii) an *adaptive length head* that dynamically adjusts the draft sequence length, substantially lowering the target model's verification cost with only a marginal decrease in acceptance length. We further introduce infrastructure improvements, including an asynchronous training–inference pipeline and adaptive request scheduling, that enable efficient online adaptation within a serving engine.

Experiments across eight benchmarks and three different foundation models, including both dense and mixture-of-experts architectures, validate the effectiveness of our ADAFLASH. Both components yield significant performance gains, and our method consistently outperforms previous SOTA speculative decoding methods, including EAGLE-3, multi-token prediction, and DFlash, achieving up to a $5.3\times$ speedup over standard autoregressive decoding. The advantage is especially remarkable at high concurrency, where prior methods degrade below the standard AR baseline, while our ADAFLASH sustains acceleration with up to 66% higher average throughput than previous SOTA.

Organization. Section 2 introduces the background and our key observation on high variance issues in diffusion drafters. Section 3 describes our ADAFLASH framework. Section 4 evaluates the effectiveness and efficiency of ADAFLASH. Section 5 concludes the paper. Due to page limits, related work is deferred to Appendix A and additional experiments are deferred to Appendix B.

2 BACKGROUND AND KEY OBSERVATION

This section introduces the background on speculative decoding and diffusion-based drafters, and then presents our key observation regarding the high variance issues of the diffusion drafters.

2.1 SPECULATIVE DECODING AND DIFFUSION DRAFTERS

Speculative Decoding. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates inference of a large target model $p_{\mathbf{v}}$ by leveraging a smaller draft model $q_{\mathbf{w}}$. The draft model rapidly generates k draft tokens, i.e., a *draft sequence*, which is then verified in parallel by the target model.

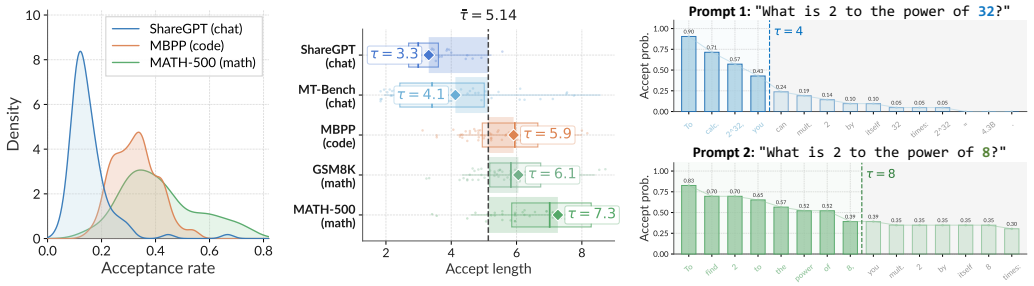
Specifically, each draft token x_i is accepted with probability $\min\{1, p_{\mathbf{v}}(x_i \mid \mathbf{x}_{<i})/q_{\mathbf{w}}(x_i \mid \mathbf{x}_{<i})\}$; all tokens up to the first rejected one are retained, and a corrected token is resampled from the residual distribution at the rejection point. This sampling procedure guarantees that the output distribution is *identical* to that of the target model, enabling lossless acceleration. Hereafter, acceptance length includes one target correction/bonus token. Following Leviathan et al. (2023), its expectation per step is $\mathbb{E}[\tau] = \frac{1-\mathrm{Acc}^{k+1}}{1-\mathrm{Acc}}$, where $\mathrm{Acc} \triangleq \mathbb{E}_{x\sim q_{\mathbf{w}}}[\min\{1, p_{\mathbf{v}}(x \mid \mathbf{x}_{<i})/q_{\mathbf{w}}(x \mid \mathbf{x}_{<i})\}]$ is acceptance rate under the simplified i.i.d. assumption. The acceleration rate is then given by

$$\eta = \frac{\mathbb{E}[\tau]}{\rho k + 1},$$

where k is the draft sequence's length (aka, candidate length), and $\rho \ll 1$ denotes the ratio of the draft model's inference time to that of the target model. This expression shows that the speedup is governed by the inference time ratio ρ and the acceptance rate.

Diffusion Language Models as Drafters. Existing speculative decoding methods predominantly employ AR drafters that *sequentially* generate draft tokens, requiring k sequential forward passes:

$$\textbf{for } i = 1, \ldots, k: \quad x_i \sim q_{\mathbf{w}}(\cdot \mid \mathbf{x}, x_1, \ldots, x_{i-1}).$$



(a) (b) (c)

Figure 1: Illustration of the *high variance* issue in diffusion drafters. (a) Probability density of the acceptance rate across three task domains (chat, code, and math), showing substantial distributional differences. (b) Box plots of the acceptance length over five different datasets. (c) Per-token acceptance probability for two nearly identical prefixes differing by only one token; the acceptance lengths diverge drastically, revealing high token-level variance of diffusion drafters.

Recently, several works (Christopher et al., 2025; Sandler et al., 2026; Li et al., 2026) have explored diffusion language models (dLLMs) (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025) as non-autoregressive drafters. Unlike AR models, dLLMs adopt a *mask-and-denoise* strategy and leverage *bidirectional attention*, enabling simultaneous prediction of multiple masked tokens without the sequential constraint of next-token prediction.

Building on this line, DFlash (Chen et al., 2026a) introduces a lightweight block diffusion drafter for speculative decoding. Specifically, given a prefix $\mathbf{x}$, the diffusion drafter initializes k masked positions and generates *all* draft tokens through a *single* forward pass:

$$\{x_1, \ldots, x_k\} \sim q_{\mathbf{w}}(\cdot \mid \mathbf{x}, \mathbf{z}),$$

where $\mathbf{z}$ denotes the masked state of the draft block. Since the entire block is generated in one forward pass rather than k sequential ones, the time ratio satisfies $\rho_{\text{dLLM}} \approx \rho_{\text{AR}}/k$, making the per-token drafting overhead roughly k times smaller. This allows diffusion drafters to employ deeper, more expressive architectures without increasing latency, achieving both lower drafting cost and higher acceptance rates than AR drafters (Chen et al., 2026a).

2.2 Key Observation: High Variance in Diffusion Drafters

Despite the advantages of diffusion drafters described above, we uncover a central pitfall: the global dependency in diffusion drafters, arising from bidirectional attention and KV injection, is a *double-edged sword*. Specifically, while it enables global contextual modeling and high-quality one-pass parallel generation, it also introduces *high variance* in draft sequences (see Appendix B.2 for an ablation study). This variance manifests at the following two levels:

- *Domain-level variance*: The acceptance rate of the diffusion drafter fluctuates significantly across task domains (e.g., code, mathematics, dialogue), much more so than that of AR drafters. Because global dependency makes all draft tokens rely on each other and on the same target hidden features, the drafter becomes more sensitive to domain shifts at inference time. As shown in Figure 1(a), the acceptance rate of DFlash varies substantially across domains.
- *Token-level variance*: Within a single candidate sequence, the per-token divergence between the draft and target distributions exhibits considerable heterogeneity. Certain token positions closely match the target distribution, while others deviate substantially, leading to non-uniform acceptance patterns across positions. As shown in Figure 1(c), the per-position acceptance probability fluctuates significantly.

As illustrated in Figure 1, both levels of variance are substantially large for diffusion drafters. These two levels of variance pose distinct challenges for practical deployment. First, domain-level variance causes the acceptance rate to drop on out-of-distribution queries, degrading end-to-end speedup. A fixed, offline-trained diffusion drafter cannot adapt to the shifting query distribution encountered during deployment. Second, token-level variance leads to heterogeneous acceptance lengths across the scenarios of concurrent requests; if a fixed candidate length k is used, requests with short acceptance

lengths incur unnecessary verification cost, wasting target model computation on tokens unlikely to be accepted.

In summary, a static speculative system with a fixed diffusion drafter and a constant verification budget suffers efficiency degradation due to high variance issues. In the next section, we introduce our ADAFLASH framework, which continuously updates the drafter parameters and adaptively adjusts the verification length to handle these two challenges, respectively.

3 OUR APPROACH

In this section, we introduce our ADAFLASH framework, which addresses the two challenges identified in Section 2. We propose two components, on-policy distillation and an adaptive length head, to tackle the high variance in diffusion drafters.

- *On-Policy Distillation* continuously adapts the diffusion drafter to the target distribution during deployment, reducing the performance drop caused by domain-level variance.
- *Adaptive Length Head* dynamically adjusts the verification length in each round based on the predicted acceptance rate, reducing the computational waste caused by token-level variance.

Finally, we describe our *infrastructure improvements* that realize these components using a serving engine, enabling asynchronous training and high-throughput serving.

3.1 ON-POLICY DISTILLATION FOR DIFFUSION DRAFTERS

In this part, we propose an on-policy distillation algorithm to address *domain-level* variance. As demonstrated earlier, the diffusion drafter exhibits a wide range of acceptance lengths across different domains. Fortunately, during deployment, *interactive feedback* is inherently available in speculative decoding: at each round, the target model produces its output distribution over the candidate positions, revealing exactly where the drafter diverges. Leveraging this feedback to update the drafter naturally forms a "draft–feedback–adapt" loop (Liu et al., 2024b; Qian et al., 2026b).

We exploit this feedback to perform *on-policy distillation* (OPD): at each round t, the drafter $q_{\mathbf{w}_t}$ generates a candidate sequence of length k given prefix $\mathbf{x}_{<t}$, and the target model returns its distributions $p_{\mathbf{v}}(\cdot \mid \mathbf{x}_{<t+i})$ at each position $i \in \{1, \dots, k\}$ as supervision signals. Since the drafter is updated using online supervision derived from the target model's responses in the current deployment domain, the training distribution remains consistent with the inference distribution. This avoids the off-policy mismatch inherent in offline distillation and helps the drafter to adapt to the deployment distribution. While this framework applies generally, the distinctive properties of diffusion drafters introduce two specific challenges, each of which we address with a dedicated mechanism detailed below.

Mixture OPD Loss with Reverse-KL. Parallel drafting and bidirectional attention in diffusion drafters tend to produce high-entropy output distributions, i.e., the probability mass is spread broadly (Huang et al., 2022). However, speculative decoding only requires covering the high-probability regions of the target. Standard forward KL distillation (or cross-entropy loss) is mode-covering and forces the drafter to spread mass broadly, which is counterproductive in this setting. We therefore adopt *reverse-KL* as the distillation objective, which penalizes the drafter for placing mass where the target has low probability, encouraging it to concentrate on the target's high-probability modes and directly improving acceptance rates.

To accelerate convergence toward the correct target model's prediction, we additionally include a hard-label cross-entropy loss on the target's top-1 token $x_i^\star = \arg\max_x p_{\mathbf{v}}(x \mid \mathbf{x}_{<t+i})$. The overall loss at round t is $\alpha \ell_{\text{hard}} + (1-\alpha)\ell_{\text{rkl}}$, where $\ell_{\text{hard}} = -1/k \sum_{i=1}^{k} \log[q_{\mathbf{w}}(x_i^\star \mid \mathbf{x}_{<t}, \mathbf{z})]_i$ is the hard-label cross-entropy, $\ell_{\text{rkl}} = 1/k \sum_{i=1}^{k} \mathrm{KL}\big([q_{\mathbf{w}}(\cdot \mid \mathbf{x}_{<t}, \mathbf{z})]_i \,\|\, p_{\mathbf{v}}(\cdot \mid \mathbf{x}_{<t+i})\big)$ is the reverse-KL divergence, and $\alpha \in [0, 1]$ is a mixing coefficient. The reverse-KL term drives the drafter's distribution toward the target's modes, while the hard-label term provides a low-variance gradient signal anchoring the drafter to the correct top-1 prediction.

Entry-wise Divergence Clipping. The reverse-KL at each draft position i is a sum $\sum_{y \in \mathcal{V}} q(y) \log(q(y)/p(y))$ over the entire vocabulary. A few entries y in the vocabulary on which

the drafter assigns non-negligible probability $q(y)$ but the target assigns a much smaller probability $p(y)$ yield a large ratio $q(y)/p(y)$; these entries can dominate the loss and produce large gradients that may mislead the overall update direction, as later illustrated in Figure 3. To address this, we introduce an entry-wise divergence clipping mechanism that bounds each individual term, indexed by draft position i and vocabulary entry y, at a threshold $\delta > 0$ before summation:

$$\ell_{\text{rkl}}^{\text{clip}} = \frac{1}{k} \sum_{i=1}^{k} \sum_{y \in \mathcal{V}} \min \left\{ [q_{\mathbf{w}}(y \mid \mathbf{x}_{<t}, \mathbf{z})]_i \log \frac{[q_{\mathbf{w}}(y \mid \mathbf{x}_{<t}, \mathbf{z})]_i}{p_{\mathbf{v}}(y \mid \mathbf{x}_{<t+i})}, \delta \right\},$$

where the $\min$ is applied entry-wise to each (i, y) pair. This mitigates the outlier gradients produced by the few high-divergence vocabulary entries, while preserving the gradient signal from the remaining entries. The threshold δ controls the trade-off between robustness and convergence speed. Combining both techniques, our final OPD loss is

$$\ell_{\text{OPD}} = \alpha\, \ell_{\text{hard}} + (1 - \alpha)\, \ell_{\text{rkl}}^{\text{clip}}.$$

3.2 Adaptive Length Head

The previous section addresses domain-level variance by continuously adapting the drafter's parameters. We now tackle the complementary challenge of *token-level variance*: existing methods use a fixed candidate length k, yet the per-position acceptance probability of diffusion drafters varies substantially, causing positions with low acceptance probability to waste the target model's computation. To this end, we introduce a lightweight *adaptive length head* that predicts the acceptance ratio on the fly and truncates the candidate sequence to its viable prefix, eliminating wasteful verification of low-quality draft tokens.

Design of the Adaptive Length Head. We attach a lightweight head to the diffusion drafter. At round t, the head receives the same input $\mathbf{h}_t \in \mathbb{R}^{k \times H}$ as the draft model. The head first applies a linear projection followed by SiLU activation, producing per-position features $\mathbf{h}'_i \in \mathbb{R}^{H'}$ for $i \in \{1, \ldots, k\}$, where $H' < H$. These features are then aggregated via mean pooling $\widehat{\mathbf{h}}_t = 1/k \sum_{i=1}^{k} \mathbf{h}'_i$ to form a global representation that captures the overall draft quality. A final linear layer followed by sigmoid activation maps $\widehat{\mathbf{h}}_t$ to a scalar $\widehat{a}_t \in [0, 1]$, representing the predicted acceptance length of the draft sequence. Given $\widehat{a}_t$, we determine the verification length $\widehat{k}_t$ as

$$\widehat{k}_t = \text{clamp}\left(\lfloor \widehat{a}_t \cdot k \rfloor,\ 1,\ k\right).$$

After that, only the first $\widehat{k}_t$ tokens of the draft sequence are sent to the target model for verification. In practice, $\widehat{a}_t$ can be further scaled by a factor $\gamma \in \mathbb{R}^+$ to adapt to system load.

Online Update of the Length Head. The length head is also continuously updated during deployment. Specifically, at each round t, after the target model verifies the candidate sequence, we obtain the ground-truth acceptance rate $a_t^\star \in [0, 1]$ (i.e., the fraction of accepted tokens). The length head is trained with a mean squared error (MSE) loss:

$$\ell_{\text{len}} = \left(\widehat{a}_t - a_t^\star\right)^2.$$

Since the verification outcome is already available from the speculative decoding pipeline, this supervision signal incurs no additional cost. We remark that the length head's loss is *detached* from the drafter, so that ℓ_{len} updates only the length head parameters and no gradient flows back into the drafter. This ensures that the MSE objective of the length head and the reverse-KL objective of OPD remain fully decoupled, preventing interference during training. During deployment progresses, the length head adapts to the evolving diffusion drafter and the changing distribution, continuously refining its predictions.

Remark 1 (Comparison with DSpark)**.** A concurrent work, DSpark (Cheng et al., 2026), also dynamically adjusts the verification length via a per-position confidence head coupled with a hardware-aware scheduler. Our approach differs in two aspects: (i) DSpark estimates per-position conditional acceptance probabilities and requires post-hoc calibration to correct for compounding errors, whereas our adaptive length head directly predicts the overall acceptance length, offering a simpler formulation that bypasses draft sequence's probability calibration; (ii) DSpark's confidence head is trained offline

and remains fixed during inference, while our adaptive length head continuously evolves on the fly, co-adapting with the evolving diffusion drafter during deployment, with an empirical comparison in Appendix B.2. Beyond these differences, DSpark's hardware-aware scheduling is complementary to our approach and can be integrated into our AdaFlash framework for further improvement.

3.3 Infrastructure Design for Online Adaptation

In previous sections, we have introduced OPD with divergence clipping and the adaptive length head to address the high variance of diffusion drafters. We now describe the infrastructure design that realizes these components efficiently via a serving engine built on SGLang (Zheng et al., 2024). Two requirements must be met: (i) the training process must run asynchronously without blocking inference, and (ii) variable verification lengths must be scheduled efficiently under dynamic GPU memory constraints. We address both with the following system design.

Asynchronous Training–Inference Pipeline. We decouple the system into an *inference server* and a *training server* connected through a shared data buffer. During inference, the speculative decoding pipeline records the on-policy generation trajectories into the buffer, comprising the input prompts and the drafter's responses. The training server asynchronously leverages these trajectories and computes the target model's supervision signals, and updates the drafter and length head parameters using our OPD loss. This decoupling ensures that training never blocks token generation.

Once the training server produces updated weights, the inference server incorporates them via *lightweight hot-reloading*: only the drafter and length head parameters are swapped into GPU memory, while the target model remains untouched. This operation completes between scheduling steps, introducing negligible latency.

Adaptive Request Scheduling. Existing serving frameworks (e.g., SGLang (Zheng et al., 2024)) typically use a fixed batch shape $\text{bs} \times k$ for verification, where all requests share the same candidate length k. To support our adaptive verification lengths, we modify the serving engine to accept variable candidate lengths and pack them into a compact verification batch whose total length equals $\sum_i^N \widehat{k}_i$, thereby reducing computational waste and improving GPU utilization under high concurrency.

Since the per-request verification length varies, the number of requests that can be processed per batch becomes less predictable. To address this, we apply an *exponential moving average* (EMA) to dynamically estimate the expected number of requests to process. Concretely, after each scheduling round, we update the estimate as $\widehat{N} \leftarrow (1-\beta)\,\widehat{N} + \beta\, N$, where N is the number of requests actually processed in the current round and $\beta \in (0,1)$ is the smoothing coefficient. The scheduler uses $\widehat{N}$ to determine how many requests to admit in subsequent rounds. When the actual memory usage still exceeds the budget, the scheduler returns excess requests to the pending queue. This strategy maximizes GPU utilization while preventing out-of-memory failures, and naturally accommodates our AdaFlash framework where the drafter evolves during deployment.

4 Experiments

To comprehensively evaluate the effectiveness of AdaFlash, we conduct experiments across eight datasets including two long-sequence reasoning benchmarks, and three target models, under various concurrency levels.[1] Our experimental evaluation aims to answer the following research questions:

Q1: Does the diffusion drafter suffer from high variance during deployment, and does this affect the overall speedup under different concurrency levels?

Q2: Can AdaFlash improve end-to-end inference throughput by leveraging on-policy distillation and adaptive verification length?

Q3: Is each component of AdaFlash effective? How do the hyperparameter choices (e.g., clipping threshold, mixing coefficient) affect the performance?

[1] Code is available at Anonymous GitHub https://anonymous.4open.science/r/AdaFlash.

Table 1: Performance comparison of different speculative decoding methods on six benchmark datasets using Q3-8B (short for Qwen3-8B) and Q3-30B (short for Qwen3-Coder-30B-A3B) as target models. We report SPEEDUP (relative to standard AR decoding) and average acceptance length τ under four concurrency levels ($C \in \{1, 32, 64, 128\}$). The best results are highlighted in **bold**.

Model	Method	Conc.	MATH				CODE				CHAT		HYBRID		*Avg.*	
			MathQA		GSM8K		OpenCodeInstruct		CodeAlpaca		ShareGPT		Blend			
			Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Q3-8B	EAGLE-3	1	2.45×	4.60	2.55×	4.75	2.20×	4.06	2.37×	4.58	2.09×	3.89	2.36×	4.54	2.34×	4.40
		32	0.72×	4.60	0.70×	4.76	0.68×	4.07	0.67×	4.58	0.61×	3.90	0.70×	4.54	0.68×	4.41
		64	0.46×	4.59	0.43×	4.75	0.44×	4.07	0.40×	4.57	0.40×	3.90	0.45×	4.54	0.43×	4.40
		128	0.34×	4.59	0.35×	4.75	0.33×	4.07	0.32×	4.58	0.30×	3.90	0.33×	4.54	0.33×	4.41
	DFlash	1	4.38×	7.09	3.84×	6.27	4.08×	6.69	3.39×	5.63	2.11×	3.40	3.39×	6.07	3.53×	5.86
		32	1.86×	7.10	1.59×	6.28	1.77×	6.68	1.46×	5.66	1.05×	3.42	1.51×	6.04	1.54×	5.86
		64	1.24×	7.11	1.03×	6.29	1.17×	6.67	0.91×	5.65	0.70×	3.41	0.98×	6.06	1.01×	5.86
		128	0.90×	7.11	0.82×	6.28	0.86×	6.67	0.74×	5.66	0.51×	3.40	0.74×	6.04	0.76×	5.86
	OSD	1	5.13×	9.44	4.33×	7.45	4.78×	8.21	3.65×	6.59	2.18×	3.78	3.60×	6.83	3.95×	7.05
		32	2.16×	9.42	1.80×	7.47	2.03×	8.20	1.55×	6.61	1.08×	3.79	1.57×	6.82	1.70×	7.05
		64	1.45×	9.44	1.16×	7.43	1.34×	8.20	0.97×	6.60	0.72×	3.80	1.07×	6.85	1.12×	7.05
		128	1.05×	9.47	0.92×	7.44	0.97×	8.20	0.78×	6.62	0.53×	3.81	0.76×	6.82	0.83×	7.06
	ADAFLASH	1	**5.32**×	9.83	**4.54**×	7.75	**4.87**×	8.40	**3.75**×	6.80	**2.24**×	3.92	**3.66**×	7.00	**4.06**×	7.28
		32	**2.27**×	9.84	**1.83**×	7.76	**2.03**×	8.39	**1.58**×	6.79	**1.10**×	3.92	**1.61**×	6.97	**1.74**×	7.28
		64	**1.65**×	9.51	**1.32**×	7.28	**1.51**×	8.02	**1.18**×	6.31	**1.04**×	3.61	**1.26**×	6.56	**1.33**×	6.88
		128	**1.34**×	9.51	**1.27**×	7.27	**1.25**×	8.05	**1.11**×	6.30	**0.87**×	3.58	**1.05**×	6.55	**1.15**×	6.88
Q3-30B	EAGLE-3	1	1.79×	4.39	1.56×	4.06	1.96×	5.15	1.93×	4.94	1.32×	3.08	1.79×	4.49	1.73×	4.35
		32	1.33×	4.40	1.15×	4.06	2.85×	5.15	1.71×	4.93	1.15×	3.08	1.49×	4.50	1.61×	4.35
		64	1.03×	4.40	0.86×	4.06	1.44×	5.15	1.33×	4.93	0.91×	3.08	1.16×	4.48	1.12×	4.35
		128	0.71×	4.41	0.63×	4.06	1.15×	5.15	1.10×	4.93	0.60×	3.08	0.77×	4.49	0.83×	4.35
	DFlash	1	2.09×	5.15	1.84×	4.85	2.56×	6.86	2.39×	6.03	1.04×	2.79	2.07×	5.43	2.00×	5.19
		32	1.61×	5.14	1.41×	4.85	3.80×	6.84	2.15×	6.01	1.14×	2.79	1.74×	5.42	1.98×	5.18
		64	1.26×	5.16	1.05×	4.84	1.95×	6.84	1.68×	6.01	0.93×	2.81	1.39×	5.42	1.38×	5.18
		128	0.87×	5.15	0.77×	4.84	1.55×	6.84	1.40×	6.00	0.60×	2.80	0.92×	5.42	1.02×	5.18
	OSD	1	2.79×	8.38	2.25×	6.44	2.96×	8.28	2.58×	7.09	1.30×	3.48	2.24×	6.38	2.35×	6.68
		32	2.04×	8.39	**1.60**×	6.43	4.30×	8.27	2.26×	6.94	1.23×	3.35	1.83×	6.37	2.21×	6.63
		64	1.59×	8.36	1.26×	6.43	2.21×	8.27	1.77×	6.95	0.98×	3.37	1.43×	6.35	1.54×	6.62
		128	1.11×	8.39	0.89×	6.43	1.75×	8.27	1.47×	6.95	0.65×	3.39	0.99×	6.35	1.14×	6.63
	ADAFLASH	1	**2.86**×	8.53	**2.32**×	6.69	**2.99**×	8.46	**2.65**×	7.21	**1.32**×	3.52	**2.27**×	6.56	**2.40**×	6.83
		32	**2.05**×	8.53	1.59×	6.69	**4.32**×	8.43	**2.33**×	7.08	**1.23**×	3.42	**1.85**×	6.53	**2.23**×	6.78
		64	**1.60**×	8.08	**1.32**×	6.42	**2.35**×	7.96	**1.98**×	6.63	**1.08**×	3.12	**1.51**×	6.16	**1.64**×	6.40
		128	**1.57**×	8.11	**1.28**×	6.43	**2.57**×	7.96	**2.32**×	6.59	**1.04**×	3.11	**1.33**×	6.14	**1.69**×	6.39

4.1 EXPERIMENTAL SETUP

In this part, we introduce the experimental setup, including the datasets and contenders.

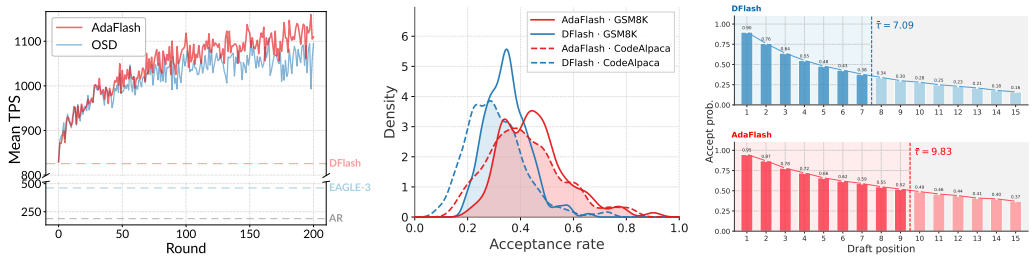
Datasets. We conduct experiments on eight benchmark datasets spanning diverse domains: two math reasoning tasks, *MathQA* (Amini et al., 2019) and *GSM8K* (Cobbe et al., 2021); two code generation tasks, *OpenCodeInstruct* (Ahmad et al., 2025) and *CodeAlpaca* (Chaudhary, 2023); a general-purpose dialogue dataset *ShareGPT*; and a mixed-domain dataset *Blend* that combines samples from the above domains to evaluate cross-domain robustness. We also evaluate on two long-sequence reasoning benchmarks, *MATH-500* (Lightman et al., 2023) and *AIME25* (Zhang & Math-AI, 2025), where the maximum output length is set to 32,768 tokens with thinking mode enabled, to assess performance on extended generation tasks.

Contenders. We compare ADAFLASH with the following methods: *(i) Standard AR* decoding (autoregressive generation without speculative decoding). *(ii) EAGLE-3* (Li et al., 2025b), a state-of-the-art autoregressive drafter that directly predicts tokens from fused multi-layer target-model features and employs tree-structured verification. *(iii) DFlash* (Chen et al., 2026a), the offline diffusion drafter that generates candidate sequences through parallel denoising. *(iv) OSD* (Liu et al., 2024b), which applies online distillation to continuously update the drafter. We apply the OSD forward-KL online-update rule to the DFlash drafter. Due to page limit, we defer the details to Appendix B.1.

Metrics. We evaluate all methods using the following metrics: *(i) Speedup*, defined as the ratio of tokens per second (TPS) of each method to that of standard AR decoding, measuring wall-clock efficiency. *(ii) Average acceptance length* (τ), i.e., how many tokens are produced in one speculative round on average, measuring the quality of the draft model.

4.2 EVALUATION OF OUR APPROACH

In this part, to answer **Q1** and **Q2**, we evaluate our ADAFLASH framework across a range of setups, including different datasets, target models, and concurrency configurations.



(a) (b) (c)

Figure 2: Analysis of ADAFLASH using Qwen3-8B as target model: (a) Mean throughput (TPS) vs. online adaptation rounds on the MathQA dataset, showing continuous improvement over time. (b) Per-sample acceptance rate distributions on GSM8K and CodeAlpaca, both shifted toward higher rates by on-policy distillation. (c) Per-position acceptance probability on the MathQA dataset.

Validating the High Variance of Diffusion Drafters. To answer **Q1**, we first examine whether diffusion drafters exhibit higher variance than autoregressive drafters during deployment, as hypothesized in Section 2. As shown in Table 1, compared with EAGLE-3 in our evaluated benchmarks, diffusion drafter exhibits a wider range of acceptance lengths across domains: for Qwen3-8B, τ ranges from 3.40 on ShareGPT to 7.09 on MathQA, a ratio of approximately $2.1\times$. In contrast, EAGLE-3 exhibits a much narrower range, from 3.89 to 4.75 (ratio $\approx 1.2\times$). This confirms that diffusion drafters exhibit larger cross-domain variation in draft quality, justifying the domain-level variance introduced in Section 2.2. Moreover, this large variance in acceptance length directly translates into wasted verification cost: with a fixed candidate length $k = 15$, the target model verifies all tokens even when only a few are likely to be accepted, degrading throughput under high concurrency. These observations motivate the two components of ADAFLASH: on-policy distillation to adapt the drafter to each deployment domain, and the adaptive length head to reduce verification waste.

Evaluation Results on Different Setups. To answer **Q2**, Table 1 summarizes the speedup and average acceptance length across six datasets, two target models, and four concurrency levels ($C \in \{1, 32, 64, 128\}$). Across all settings, ADAFLASH consistently achieves the highest speedup. For Qwen3-8B at $C = 1$, ADAFLASH attains an average speedup of $4.06\times$, compared to $3.53\times$ for DFlash, $3.95\times$ for OSD, and $2.34\times$ for EAGLE-3. The acceptance length also improves consistently: on MathQA, τ increases from 7.09 (DFlash) to 9.44 (OSD) to 9.83 (ADAFLASH), confirming that our on-policy distillation is more effective than the standard knowledge distillation used by OSD. These improvements generalize across model architectures and scales: on the mixture-of-experts model Qwen3-Coder-30B-A3B, ADAFLASH also consistently achieves the highest speedup.

The advantage of ADAFLASH becomes more remarkable under high concurrency. At $C = 128$ with Qwen3-8B, ADAFLASH maintains a speedup of $1.15\times$, while OSD and DFlash fall below standard decoding, indicating that methods with a fixed verification length waste GPU compute when only a few draft tokens are accepted. In contrast, the adaptive length head reduces the verification length and thus the per-round verification cost. On Qwen3-Coder-30B-A3B at $C = 128$, ADAFLASH achieves an average speedup of $1.69\times$ compared to $1.02\times$ for DFlash, corresponding to 66% higher throughput. Furthermore, Figure 2(a) shows that throughput improves progressively over online adaptation rounds, with ADAFLASH consistently outperforming OSD throughout deployment.

Effectiveness of ADAFLASH. We next examine how ADAFLASH addresses the two types of variance. For domain-level variance, on-policy distillation improves τ on every domain in Table 1, although the gains are uneven (e.g., $+39\%$ on MathQA vs. $+15\%$ on ShareGPT for Qwen3-8B at $C = 1$), since it adapts the drafter to each deployment distribution rather than equalizing acceptance rates across domains, and open-ended dialogue is inherently harder to draft. Figure 2(b) further shows that ADAFLASH shifts the per-sample acceptance rate distributions toward higher rates on both GSM8K and CodeAlpaca. Both distributions also become wider, making a fixed verification length even less suitable. For token-level variance, Figure 2(c) shows that on-policy distillation lifts the acceptance probability at every position, especially at later ones (e.g., from 0.16 to 0.37 at the last position), narrowing the gap between early and late positions and extending the average acceptance length from 7.09 to 9.83.

Table 2: Ablation study of AdaFlash components on the GSM8K dataset using Qwen3-8B as the target model. We evaluate four components: divergence clipping, mixture OPD loss, adaptive length head, and online update, and report the average acceptance length τ and Speedup at concurrency levels $C \in \{1, 128\}$. The best results are highlighted in **bold**.

On-Policy Distillation		Adaptive Length Head		GSM8K		
Divergence Clipping	Mixture OPD Loss	Length Head	Online Update	Conc.	τ	Speedup
		✓	✓	1	7.389	4.17×
				128	6.802	1.18×
✓		✓	✓	1	7.590	4.31×
				128	7.212	1.25×
✓	✓	✓	✓	1	7.751	**4.54**×
				128	7.270	**1.27**×
✓	✓			128	6.594	1.00×
✓	✓	✓		128	7.652	1.21×
✓	✓	✓	✓	128	7.270	**1.27**×

4.3 Additional Experimental Results

Ablation Study. To answer **Q3**, we conduct ablation studies to validate each component's contribution in AdaFlash. As shown in Table 2, we progressively add components and evaluate on GSM8K with Qwen3-8B as the target model. In the upper part of Table 2, we evaluate the on-policy distillation components in our AdaFlash framework while keeping the adaptive length head enabled. With the standard reverse-KL loss and neither of the two components, the drafter achieves moderate speedup but is limited by the distribution mismatch between the drafter and the target model. Introducing entry-wise divergence clipping consistently improves both the acceptance length and speedup across concurrency levels. Further incorporating the mixture OPD loss brings additional gains by introducing hard-label supervision on the target top-1 token. Combining both components, the full configuration achieves the highest acceptance length and speedup.

In the lower part of Table 2, we ablate the adaptive length head while keeping the full OPD recipe. Without the length head, we use a fixed verification length of 11, matching the average verification length (11.271) of AdaFlash on GSM8K. In this setting, the speedup at high concurrency (C=128) drops to 1×, offering no improvement over standard AR decoding. Adding the length head with fixed parameters substantially restores the high-concurrency speedup, as it truncates the candidate sequence to its viable prefix and eliminates wasteful verification. Enabling online updates of the length head yields further improvement, since the head co-adapts with the evolving drafter, maintaining accurate acceptance-rate predictions throughout deployment. These results confirm that the adaptive length head effectively reduces the verification waste caused by *token-level variance*, and that its online update is essential for sustaining this benefit as the drafter evolves.

5 Conclusion

In this paper, we uncover a fundamental challenge in diffusion-based speculative decoding: the global dependency arising from bidirectional attention and KV injection enables one-pass parallel drafting but also introduces high variance in draft quality, manifesting at both the domain level and the token level. Existing methods rely on fixed, offline-trained diffusion drafters with a constant verification length, which leads to degraded acceptance rates on out-of-distribution domains and wasteful verification of low-quality draft tokens. To this end, we propose AdaFlash, an adaptive framework with two complementary components: (i) on-policy distillation with a mixture of reverse-KL and hard-label cross-entropy, combined with entry-wise divergence clipping, which continuously adapts the drafter to the deployment distribution; and (ii) an adaptive length head that predicts the viable draft length in each round and truncates the draft sequence accordingly, substantially lowering the target model's verification cost. We further introduce infrastructure improvements, including an asynchronous training–inference pipeline and adaptive request scheduling. Experiments across eight benchmarks including two long-sequence tasks, and three target models, covering both dense and MoE architectures, demonstrate that AdaFlash consistently outperforms previous SOTA methods, achieving up to 5.3× speedup over AR decoding, with significant gains under high concurrency.

AI USE STATEMENT

In this work, we used generative AI tools for editing and polishing the manuscript text, and for assisting with writing and debugging experiment code. These tools were primarily used to improve the clarity and fluency of our writing, as well as to accelerate the implementation of training pipelines and evaluation scripts. We have not used generative AI tools for formulating research ideas, designing the core methodology, or generating scientific claims; the identification of the high-variance issue in diffusion drafters, the design of the on-policy distillation algorithm and the adaptive length head, and all experimental analyses are entirely human contributions. We have reviewed all AI-assisted work. Specifically, all polished text was manually reviewed and revised by the authors to ensure accuracy and faithfulness to the intended meaning, and all AI-generated code was verified and tested for correctness before integration into the codebase. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work focuses on inference acceleration techniques for large language models, specifically adaptive speculative decoding with on-policy distilled diffusion drafters. Our research is methodological in nature and does not involve human subjects, private data, or direct deployment in sensitive applications. All benchmark datasets used in our experiments are publicly available and well-established in the community. We do not foresee immediate negative societal impacts arising from this work.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide detailed descriptions of our training procedure, hyperparameters, and evaluation protocols in Section 4 and Appendix B.1. Our source code is publicly available at an anonymous GitHub repository. We believe these resources are sufficient for reproducing the main results reported in this paper.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, pp. 21246–21263, 2024.

Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. Opencodeinstruct: A large-scale instruction tuning dataset for code llms. *ArXiv preprint*, arXiv:2504.04030, 2025.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 2357–2367, 2019.

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, pp. 1877–1901, 2020.

Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. *GitHub repository*, 2023.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, pp. to appear, 2026a.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dLLMs. *ArXiv preprint*, arXiv:2509.26488, 2025.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. Dmax: Aggressive parallel decoding for dllms. *ArXiv preprint*, arXiv:2604.08302, 2026b.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *ArXiv preprint*, arXiv:2607.05147, 2026.

Jacob K. Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 12042–12059, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *ArXiv preprint*, arXiv:2110.14168, 2021.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pp. 16344–16359, 2022.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 14060–14079, 2024.

Yichao Fu, Rui Ge, Zelei Shao, Zhijie Deng, and Hao Zhang. Scaling speculative decoding with lookahead reasoning. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, pp. to appear, 2025a.

Yonggan Fu, Lexington Whalen, Zhifan Ye, Xin Dong, Shizhe Diao, Jingyu Liu, Chengyue Wu, Hao Zhang, Enze Xie, Song Han, Maksim Khadkevich, Jan Kautz, Yingyan Celine Lin, and Pavlo Molchanov. Efficient-dlm: From autoregressive to diffusion language models, and beyond in speed. *ArXiv preprint*, arXiv:2512.14067, 2025b.

Yifeng Gao, Ziang Ji, Yuxuan Wang, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Self speculative decoding for diffusion large language models. *ArXiv preprint*, arXiv:2510.04147, 2025.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *ArXiv preprint*, arXiv:1503.02531, 2015.

Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent Y. F. Tan, and Zhuoran Yang. Banditspec: Adaptive speculative decoding via bandit algorithms. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pp. 24045–24079, 2025.

Lanxiang Hu, Siqi Kou, Yichao Fu, Samyam Rajbhandari, Tajana Rosing, Yuxiong He, Zhijie Deng, and Hao Zhang. Fast and accurate causal parallel decoding using jacobi forcing. *ArXiv preprint*, arXiv:2512.14681, 2025.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. Jetspec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. *ArXiv preprint*, arXiv:2606.18394, 2026.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. On the learning of non-autoregressive transformers. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pp. 9356–9376, 2022.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. In *Proceedings of the 2nd Conference on Language Modeling (COLM)*, 2025.

Haodi Lei, Yafu Li, Haoran Zhang, Shunkai Zhang, Qianjia Cheng, Xiaoye Qu, Ganqu Cui, Bowen Zhou, Ning Ding, Yun Luo, and Yu Cheng. Draft-opd: On-policy distillation for speculative draft models. *ArXiv preprint*, arXiv:2605.29343, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp. 19274–19286, 2023.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 20896–20910, 2026.

Jia-Nan Li, Jian Guan, Wei Wu, and Chongxuan Li. Refusion: A diffusion large language model with parallel autoregressive decoding. *ArXiv preprint*, arXiv:2512.13586, 2025a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7421–7432, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 28935–28948, 2024b.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *ArXiv preprint*, arXiv:2305.20050, 2023.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *ArXiv preprint*, arXiv:2412.19437, 2024a.

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

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 31131–31146, 2024b.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, pp. to appear, 2025a.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, et al. dInfer: An efficient inference framework for diffusion language models. *ArXiv preprint*, arXiv:2510.08666, 2025b.

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

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, pp. to appear, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pp. 27730–27744, 2022.

Rui Pan, Zhuofu Chen, Hongyi Liu, Arvind Krishnamurthy, and Ravi Netravali. Fail fast, win big: Rethinking the drafting strategy in speculative decoding via diffusion llms. *ArXiv preprint*, arXiv:2512.20573, 2025.

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

Yu-Yang Qian, Hao-Cong Wu, Yichao Fu, Hao Zhang, and Peng Zhao. When drafts evolve: Speculative decoding meets online learning. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, pp. to appear, 2026b.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M. Rush, Yair Schiff, Justin T. Chiu, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, pp. 130136–130184, 2024.

Jameson Sandler, Jacob Christopher, Tom Hartvigsen, and Ferdinando Fioretto. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding. In *Proceedings of the 9th Conference on Machine Learning and Systems (MLSys)*, pp. 1128–1147, 2026.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix X. Yu. SpecTR: fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pp. 30222–30242, 2023.

Junxiong Wang, Fengxiang Bie, Jisen Li, Zhongzhu Zhou, Zelei Shao, Yubo Wang, Yinghui Liu, Qingyang Wu, Avner May, Sri Yanamandra, Yineng Zhang, Ce Zhang, Tri Dao, Percy Liang, Ben Athiwaratkun, Shuaiwen Leon Song, Chenfeng Xu, and Xiaoxia Wu. When RL meets adaptive speculative training: A unified training-serving system. *ArXiv preprint*, arXiv:2602.06932, 2026.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion LLMs can do faster-than-AR inference via discrete diffusion forcing. *ArXiv preprint*, arXiv:2508.09192, 2025a.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *ArXiv preprint*, arXiv:2509.06949, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pp. 24824–24837, 2022.

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

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Wenda Xu, Rujun Han, Zifeng Wang, Long T. Le, Dhruv Madeka, Lei Li, William Yang Wang, Rishabh Agarwal, Chen-Yu Lee, and Tomas Pfister. Speculative knowledge distillation: Bridging the teacher-student gap through interleaved sampling. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, et al. Qwen3 technical report. *ArXiv preprint*, arXiv:2505.09388, 2025a.

Jingyi Yang, Guanxu Chen, Xuhao Hu, and Jing Shao. Taming masked diffusion language models via consistency trajectory reinforcement learning with fewer decoding step. *ArXiv preprint*, arXiv:2509.23924, 2025b.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, pp. to appear, 2025c.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. Multi-candidate speculative decoding. *ArXiv preprint*, arXiv:2401.06706, 2024.

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

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. *ArXiv preprint*, arXiv:2412.18910, 2024.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, pp. 62557–62583, 2024.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024.

Zhi-Hua Zhou and Yuan Jiang. Nec4.5: Neural ensemble based C4.5. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):770–773, 2004.

A Related Work

In this section, we discuss the related topics.

Speculative Decoding. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) accelerates large language model inference by pairing a lightweight draft model with a large target model: the draft model proposes a candidate token sequence, which the target model verifies in parallel, accepting all tokens up to the first rejected one and resampling at the rejection point.

A central line of work focuses on improving the draft sequence's quality. EAGLE (Li et al., 2024b) introduces feature-level autoregression with token-conditioned drafting. EAGLE-2 (Li et al., 2024a) extends it with context-aware dynamic draft trees, and EAGLE-3 (Li et al., 2025b) further replaces feature prediction with direct token prediction via multi-layer feature fusion. Lookahead decoding (Fu et al., 2024) avoids a separate draft model entirely by maintaining n-gram pools from the Jacobi decoding trajectory for parallel verification.

On the other hand, SpecTR (Sun et al., 2023) formulates draft selection as an optimal transport problem, while MCSD (Yang et al., 2024) organizes multiple candidate sequences into batches for parallel verification to improve the acceptance rate. JetSpec (Hu et al., 2026) trains a causal parallel draft head over fused target-model hidden states, combining one-forward drafting efficiency with branch-wise causal conditioning to achieve longer acceptance lengths. To adaptively tune hyperparameters during inference, BanditSpec (Hou et al., 2025) casts the selection as a multi-armed bandit problem, and HedgeSpec (Liu et al., 2025) introduces an auxiliary verification step that evaluates all draft models without additional target model queries. Beyond token-level parallelism, Lookahead Reasoning (Fu et al., 2025a) exploits step-level parallelism by having the draft model propose entire reasoning steps, which are then expanded in parallel and verified semantically.

Diffusion Language Models. Diffusion language models (dLLMs) have emerged as a promising alternative to autoregressive generation, and the release of open-source foundation dLLMs has accelerated research in this area. LLaDA (Nie et al., 2025) is a native 8B dLLM trained from scratch; Dream (Ye et al., 2025) is a 7B model initialized from a pretrained AR checkpoint; and LLaDA 2.0 (Bie et al., 2025) scales to 100B parameters via conversion from AR models. Subsequent efforts improve dLLMs in both efficiency and performance (Fu et al., 2025b; Wang et al., 2025a; Chen et al., 2025; 2026b). Fast-dLLM (Wu et al., 2025) introduces a block-wise approximate KV-cache to accelerate dLLM inference, and dKV (Ma et al., 2025a) proposes a delayed KV-cache strategy tailored for the non-autoregressive setting (Gao et al., 2025; Ma et al., 2025b; Yang et al., 2025b). d3LLM (Qian et al., 2026a) improves the efficiency of dLLMs by leveraging distillation. On the other hand, MMaDA (Yang et al., 2025c) applies policy-gradient reinforcement learning to enhance dLLM performance across modalities; ReFusion (Li et al., 2025a) adopts slot-level parallel decoding initialized from Qwen-3-8B; and TraDo (Wang et al., 2025b) incorporates trajectory-aware reinforcement learning for reasoning tasks.

Several recent works explore dLLMs as drafters for speculative decoding. SpecDiff (Christopher et al., 2025) first demonstrated the feasibility of parallel diffusion drafting, showing that a discrete diffusion model with few denoising steps can match the wall-clock speedup of AR drafters by amortizing drafting cost across positions. DiffuSpec (Li et al., 2026) extended this idea by employing pretrained large-scale dLLMs as drafters. SpecDiff-2 (Sandler et al., 2026) then mitigates the draft–verifier mismatch between AR and dLLM via streak-distillation (optimizing expected acceptance length) and test-time self-selection (choosing the best draft from multiple low-cost samples). Building on this line, DFlash (Chen et al., 2026a) proposed a lightweight block diffusion drafter, and conditions each drafting layer on fused multi-layer hidden features from the target model, achieving both low drafting cost and high acceptance rates. Our work uncovers a previously overlooked pitfall of diffusion drafters: the global dependency arising from bidirectional attention and KV injection, while enabling one-pass parallel generation, also introduces *high variance* in draft quality across both domains and token positions, which limits the achievable speedup. Based on this insight, we propose AdaFlash to address this issue through online drafter adaptation and adaptive verification length.

On-policy Distillation. On-policy distillation (Agarwal et al., 2024; Gu et al., 2024; Lu & Lab, 2025; Xu et al., 2025) trains a student model on sequences sampled from its own policy rather than from a fixed dataset or the teacher, thereby eliminating the train–inference distribution mismatch

(exposure bias) that arises when a model is only supervised on offline data. Since the supervision signals are derived from the student's on-policy generations, the student directly corrects the errors it actually encounters during inference, yielding more effective adaptation than offline distillation.

This paradigm also broadens the space of supervision signals. Standard SFT minimizes the cross-entropy against the dataset's ground-truth hard labels, which corresponds to a forward KL objective, and classical knowledge distillation (Zhou & Jiang, 2004; Hinton et al., 2015) similarly matches the teacher's output. In contrast, by operating on student-sampled sequences, on-policy distillation admits more flexible divergences such as the reverse-KL and generalized objectives, as exemplified by GKD (Agarwal et al., 2024), which mitigate the mode-covering pathologies of forward KL and better suit the student's limited capacity.

On-policy distillation is naturally suited for training the draft model in speculative decoding, as the draft model must approximate the target distribution along trajectories it generates itself. DistillSpec (Zhou et al., 2024) shows that the token-level acceptance rate equals $1 - D_{\mathrm{TV}}(p, q)$ and systematically compares FKL, RKL, JSD, and TVD under different data sources, finding that on-policy data is crucial while the best divergence depends on the task and decoding strategy. In this work, we treat the divergence as a design choice and adopt a mixture of hard-label cross-entropy and clipped reverse-KL tailored to the high-entropy outputs of diffusion drafters. OSD (Liu et al., 2024b) applies continuous distillation from the target model to the draft model on live query distributions, updating the draft based on discrepancies identified during the verification stage. Extending this perspective, OnlineSPEC (Qian et al., 2026b) formulates the on-policy update of the draft model as an online learning problem and establishes regret bounds for the evolving draft, thus providing a unified theoretical framework for online speculative decoding. Aurora (Wang et al., 2026) further integrates speculator training and deployment into a single closed-loop system. Draft-OPD (Lei et al., 2026) applies on-policy distillation to speculative drafters including DFlash as an offline post-training stage: it collects target-assisted rollouts on a fixed prompt pool, replays drafting from the verification-exposed error positions, and applies forward KL on accepted tokens and position-decayed reverse KL on rejected ones. In contrast, our OPD continuously updates the drafter on live deployment trajectories.

Dynamic Candidate Length. Adaptive candidate length has been studied in the context of autoregressive speculative decoding. SpecDec++ (Huang et al., 2025) formulates the candidate length selection as a Markov decision process, trains an acceptance prediction head on top of the AR draft model to predict per-token conditional acceptance probabilities, and stops the current speculation round when the cumulative rejection probability exceeds a threshold. AdaEAGLE (Zhang et al., 2024) uses a lightweight MLP to regress the draft length of the AR drafter for the next round. In the context of diffusion-based drafting, DiffuSpec (Li et al., 2026) introduces an adaptive draft-length controller that adjusts the next round's proposal size based on exponential moving averages of recent acceptance and generation lengths. FailFast (Pan et al., 2025) takes a different strategy: it dynamically extends the speculation length when the drafter's token-level confidence exceeds a threshold. Unlike these approaches, our adaptive length head targets diffusion drafters: it directly predicts the acceptance length of the draft sequence to truncate the verification length, and is continuously updated online to co-adapt with the evolving drafter, whereas the predictors of SpecDec++ and AdaEAGLE are trained offline and kept fixed. We also note that FailFast's strategy of dynamically extending the speculation length is orthogonal and complementary to our approach of truncating the verification length.

B Additional Experiments

In this section, we provide additional experimental results.

B.1 Implementation Details

Implementation Details. We use Qwen3-8B, Qwen3-Coder-30B-A3B (Yang et al., 2025a), and Qwen3.5-9B (Qwen Team, 2026) as target models, covering dense and mixture-of-experts architectures of different scales. For the DFlash drafter, we use a candidate length of $k = 15$ (i.e., block size $= 16$) per speculative round. Reported verification lengths include the anchor token. The on-policy distillation uses a mixture loss with mixing coefficient $\alpha = 0.8$, reverse-KL temperature $= 1.0$, and entry-wise divergence clipping threshold $\delta = 0.01$. The drafter is updated with AdamW at learning rate 3×10^{-4}, while the adaptive length head is trained separately at 2×10^{-4} with MSE

Table 3: Ablation study validating the high variance issue in diffusion drafters. All drafters are trained from scratch using Qwen3-4B as the target model with 800K mixed-domain samples for 2 epochs. DFlash variants use candidate length $k = 15$; EAGLE-3 uses tree size 60.

Method	HumanEval		GSM8K		MT-Bench		Cross-Domain	
	τ	Speedup	τ	Speedup	τ	Speedup	Mean(τ)	Std(τ)
DFlash	4.59	3.15×	5.34	3.67×	2.69	1.88×	4.21	1.115
DFlash (causal)	4.41	3.02×	5.15	3.47×	2.64	1.81×	4.07	1.053
EAGLE-3	3.34	1.95×	3.70	2.14×	3.16	1.82×	3.40	0.225



Figure 3: Reverse-KL divergence between the draft and target distributions on a GSM8K response. Each cell is a generated token, shaded by its divergence (darker indicates higher divergence).

loss. In our AdaFlash configuration, the adaptive length head is disabled when the concurrency ≤ 16, in which case the candidate length is fixed at $k = 15$: the adaptive length head is only activated under higher concurrency scenarios to dynamically reduce candidate lengths and lower verification cost. Online training is triggered whenever the replay buffer accumulates 128 on-policy samples; each buffer batch is trained for 2 epochs with batch size 1 and gradient accumulation steps of 2 (effective batch size 2), up to a maximum sequence length of 2,048 tokens. The online adaptation runs in a streaming fashion via an asynchronous training–inference pipeline: an inference worker serves incoming queries on dedicated GPUs while a training worker updates the drafter in the background on separate GPUs. The system is built on top of SGLang (Zheng et al., 2024) for high-throughput serving with continuous batching. We employ mixed-precision training with bfloat16 and use Flash Attention (Dao et al., 2022) to accelerate attention computation. Greedy decoding (temperature $= 0$) is adopted during inference across all experiments unless explicitly stated otherwise.

Decoupled Training and Inference Resources. In practical deployments, training and inference are typically provisioned as separate resource pools, which may differ in accelerator types and reside on different nodes. Our asynchronous design naturally fits this setting: the training server runs entirely within the training pool without occupying any inference GPUs. Hence, the serving capacity of the inference pool remains intact.

B.2 Additional Experimental Results

Validating the Variance Issue in Diffusion Drafters. To validate the high variance issue identified in Section 2.2, we ablate the key components of DFlash by comparing three configurations trained from scratch on Qwen3-4B (800K mixed-domain samples, 2 epochs): *(i)* DFlash (bidirectional attention, one-pass parallel generation, KV injection); *(ii)* DFlash (causal), replacing bidirectional with causal attention; and *(iii)* EAGLE-3, an AR drafter that reuses hidden states without KV injection. DFlash variants use candidate length $k = 15$; EAGLE-3 uses tree size 60.

As shown in Table 3, removing bidirectional attention (DFlash causal) reduces $\mathrm{Std}(\tau)$ from 1.12 to 1.05. EAGLE-3, which removes both bidirectional attention and KV injection, achieves the lowest variance ($\mathrm{Std}(\tau) = 0.23$), but at the cost of notably lower acceptance lengths ($\tau = 3.40$ vs. 4.21). These results suggest that bidirectional attention and KV injection, i.e., the two components of global dependency, are both sources of high variance in diffusion drafters, yet they are also key to the strong drafting capability of DFlash. This also indicates that the *primary contributor to DFlash's high acceptance rate is KV injection, rather than the diffusion-based architecture itself.*

Hyperparameter Analysis. To answer **Q3**, we investigate the impact of key hyperparameters on the performance of our AdaFlash framework. We first examine the mixing coefficient α, which interpolates between reverse-KL ($\alpha = 0$) and hard forward KL ($\alpha = 1$). We sweep $\alpha \in$

Table 4: Hyperparameter analysis of our ADAFLASH using Qwen3-8B as target model on GSM8K.

(a) Analysis of the mixing coefficient α

α	τ	TPS	Acc. Rate
0.0	7.590	802.90	0.439
0.2	7.713	811.61	0.448
0.4	7.724	823.85	0.448
0.6	7.732	825.04	0.449
0.8	**7.751**	**835.32**	**0.450**
1.0	7.734	828.85	0.449

(b) Analysis of the div. clipping threshold δ

δ	τ	TPS	Acc. Rate
No Clip	7.684	808.01	0.446
0.003	7.722	739.41	0.448
0.005	7.725	754.73	0.448
0.010	**7.751**	**835.32**	**0.450**
0.015	7.740	824.34	0.449
0.020	7.734	815.76	0.449

(c) Analysis of the adaptive length scale factor γ

γ	Speedup	τ	Verify Len.
0.8	1.257	6.514	9.033
0.9	1.262	6.950	10.162
1.0	**1.270**	7.270	11.271
1.1	1.231	7.522	12.342
1.2	1.202	7.618	13.117
1.3	1.185	7.675	13.674

$\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ on Qwen3-8B with a concurrency of 1 for (a) and (b), and 128 for (c), and report results in Table 4. All three metrics improve monotonically from $\alpha = 0.0$ to $\alpha = 0.8$, confirming that the hard forward KL term provides effective top-1 supervision that the pure reverse-KL objective lacks. Setting $\alpha = 1.0$ causes a slight regression, indicating that the mode-seeking regularization from reverse-KL remains beneficial even when hard-label supervision dominates.

We further ablate the entry-wise divergence clipping threshold δ and report results in Table 4. Most thresholds improve τ and acceptance rate over the no-clipping baseline, confirming that divergence clipping suppresses noisy large-magnitude gradients. However, overly aggressive clipping reduces TPS and acceptance length, suggesting it over-constrains the gradient and slows convergence. The moderate threshold $\delta = 0.01$ achieves the best balance. We also investigate the adaptive length scale factor γ and report results in Table 4. The speedup is relatively stable across the range $\gamma \in [0.8, 1.3]$, varying between $1.19\times$ and $1.27\times$, indicating that the performance is not overly sensitive to this hyperparameter. While larger γ produces longer draft sequences and thus higher acceptance length τ, the increased verification length offsets the gain, leading to slightly lower overall speedup. The best speedup is achieved at $\gamma = 1.0$.

Entry-wise divergence clipping. Unlike AR drafters, diffusion drafters rely on bidirectional attention to predict all draft tokens jointly in a single forward pass, which tends to spread probability mass broadly across the vocabulary. As a result, the reverse-KL at each draft position, which is a sum of per-entry terms over the vocabulary, is dominated by a few entries on which the drafter places non-negligible mass but the target does not. Consequently, as shown in Figure 3, the per-token divergence is highly uneven across the sequence, with a few tokens reaching values orders of magnitude larger than the rest. During on-policy distillation, such excessive high-divergence entries dominate the gradient and can mislead the overall parameter update, destabilizing training. Entry-wise divergence clipping addresses this by bounding each entry's KL contribution at a threshold δ before summation, suppressing outlier gradients while preserving the informative updates from the remaining entries.

Results of ADAFLASH on Qwen3.5-9B. Table 5 reports results on Qwen3.5-9B, which adopts a hybrid Gated DeltaNet architecture. The *MTP* (Multi-Token Prediction) (Liu et al., 2024a) method in the table refers to the native multi-token prediction capability of Qwen3.5. At $C = 1$, ADAFLASH achieves the highest average speedup of $3.58\times$ across all six benchmarks, outperforming DFlash, OSD, and the MTP head. At higher concurrency, ADAFLASH consistently leads on the math and code benchmarks, although its gains on ShareGPT narrow or reverse slightly when $C \geq 32$. We attribute this to an engineering limitation: SGLang's current support for the Gated DeltaNet architecture does not yet allow a fully efficient implementation of variable-length verification scheduling, so the adaptive length head incurs additional overhead under high-concurrency serving on this model. We expect this high-concurrency gap to close as SGLang's support for Gated DeltaNet matures.

Long-Sequence Generation. To evaluate the effectiveness of ADAFLASH on extended reasoning tasks, we conduct experiments on MATH-500 and AIME25 with a maximum output length of 32,768 tokens and thinking mode enabled, using Qwen3-8B as the target model. The results are reported in Tables 6 and 7. On MATH-500, ADAFLASH achieves a speedup of $3.67\times$ at $C = 1$ and $3.52\times$ at $C = 32$, outperforming DFlash ($2.60\times$ and $2.46\times$), OSD ($3.51\times$ and $3.40\times$), and EAGLE-3 ($2.26\times$ and $1.82\times$). On AIME25, ADAFLASH achieves $3.18\times$ at $C = 1$ and $2.77\times$ at $C = 32$. These results indicate that ADAFLASH maintains its advantages in long-sequence generation.

Cross-Domain Generalization. To evaluate whether offline OPD can produce a drafter that generalizes across domains without online adaptation, we compare three settings on Qwen3-8B: *(i)* the offline DFlash drafter, *(ii)* ADAFLASH with in-domain online OPD (training and evaluation on the

Table 5: Performance Comparison on Qwen3.5-9B

Method	Conc.	MATH				CODE				CHAT		HYBRID		*Avg.*	
		MathQA		GSM8K		OpenCodeInstruct		CodeAlpaca		ShareGPT		Blend			
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
MTP	1	2.24×	3.54	2.08×	3.54	2.17×	3.48	1.88×	3.23	1.81×	2.77	2.10×	3.38	2.05×	3.32
	32	1.64×	3.54	1.43×	3.54	1.51×	3.48	1.35×	3.24	**1.42**×	2.77	**1.53**×	3.38	1.48×	3.32
	64	1.29×	3.54	1.19×	3.54	1.29×	3.48	1.16×	3.23	**1.13**×	2.77	**1.25**×	3.37	1.22×	3.32
	128	0.98×	3.54	0.92×	3.54	0.98×	3.49	0.88×	3.23	**0.85**×	2.77	0.95×	3.38	0.93×	3.32
DFlash	1	3.69×	6.92	3.59×	6.54	3.79×	7.77	2.87×	5.20	1.91×	3.33	2.96×	6.20	3.14×	5.99
	32	1.53×	6.92	1.43×	6.54	1.57×	7.77	1.28×	5.21	0.90×	3.34	1.27×	6.23	1.33×	6.00
	64	1.08×	6.92	1.07×	6.53	1.20×	7.77	0.97×	5.20	0.62×	3.34	0.92×	6.20	0.98×	5.99
	128	0.81×	6.92	0.85×	6.53	0.93×	7.77	0.73×	5.20	0.47×	3.33	0.70×	6.22	0.75×	6.00
OSD	1	4.26×	8.62	4.30×	8.28	4.31×	9.35	3.11×	6.11	1.97×	3.68	3.14×	7.02	3.52×	7.18
	32	1.72×	8.63	1.67×	8.28	1.72×	9.31	1.34×	6.10	0.92×	3.65	1.33×	7.04	1.45×	7.17
	64	1.20×	8.62	1.27×	8.27	1.34×	9.33	1.02×	6.09	0.63×	3.62	0.96×	7.05	1.07×	7.16
	128	0.91×	8.63	1.01×	8.27	1.02×	9.33	0.79×	6.11	0.47×	3.63	0.74×	7.04	0.82×	7.17
ADAFLASH	1	**4.33**×	8.88	**4.32**×	8.42	**4.39**×	9.63	**3.24**×	6.39	**1.99**×	3.88	**3.23**×	7.25	**3.58**×	7.41
	32	**1.87**×	7.87	**1.76**×	7.55	**1.78**×	8.29	**1.51**×	5.94	1.08×	3.70	1.49×	6.60	**1.58**×	6.66
	64	**1.44**×	6.61	**1.41**×	6.71	**1.47**×	6.97	**1.33**×	4.84	1.04×	2.98	1.23×	5.17	**1.32**×	5.55
	128	**1.16**×	6.65	**1.24**×	6.72	**1.25**×	7.04	**1.10**×	4.83	0.83×	2.99	**0.99**×	5.18	**1.09**×	5.57

Table 6: Long-sequence generation results on MATH-500 dataset (32K output length, Qwen3-8B, thinking mode enabled).

Method	Conc.	Speedup	τ	Acc. Rate
EAGLE-3	1	2.26×	4.25	0.105
	32	1.82×	4.37	0.109
DFlash	1	2.60×	3.97	0.198
	32	2.46×	4.04	0.203
OSD	1	3.51×	5.28	0.285
	32	3.40×	5.31	0.287
ADAFLASH	1	**3.67**×	5.37	0.291
	32	**3.52**×	5.14	0.276

Table 7: Long-sequence generation results on AIME25 dataset (32K output length, Qwen3-8B, thinking mode enabled).

Method	Conc.	Speedup	τ	Acc. Rate
EAGLE-3	1	2.30×	4.16	0.102
	32	1.68×	4.26	0.105
DFlash	1	1.89×	3.32	0.155
	32	1.54×	3.36	0.157
OSD	1	3.01×	5.21	0.281
	32	2.52×	5.24	0.283
ADAFLASH	1	**3.18**×	5.32	0.288
	32	**2.77**×	4.98	0.265

same benchmark), and *(iii)* ADAFLASH trained offline via OPD on PERFECTBLEND, a mixed-domain dataset distinct from each test benchmark, and evaluated with fixed weights without further online updates. Table 8 reports results on four benchmarks (MathQA, GSM8K, OpenCodeInstruct, and CodeAlpaca) at concurrency $C \in \{1, 32, 64, 128\}$. In-domain ADAFLASH generally achieves the best speedup and acceptance length, confirming the benefit of adapting the drafter to the deployment domain. A minor exception is GSM8K, where the cross-domain drafter slightly outperforms the in-domain one, likely because the limited size of the GSM8K training set restricts the volume of on-policy samples for effective in-domain adaptation. Overall, the cross-domain drafter with fixed weights outperforms the offline DFlash baseline across all settings by leveraging mixed-domain OPD training, yet still lags behind in-domain online adaptation on most benchmarks. This confirms that on-policy updates are necessary to adapt the drafter to the current domain.

Training From Scratch. Beyond online adaptation of a pre-trained drafter, ADAFLASH can also train a diffusion drafter entirely from scratch, which is useful when no off-the-shelf DFlash drafter is available for a given target model. To demonstrate this, we use Qwen3-1.7B as the target model, train ADAFLASH from scratch on the GSM8K training split (7.4K samples) for 10 epochs, and evaluate it on the GSM8K test split (1.3K samples). As shown in Table 9, the from-scratch drafter trained with ADAFLASH achieves an average acceptance length of 4.554 and a 1.84× speedup over AR decoding, substantially outperforming EAGLE-3 (using its publicly released pre-trained weights from Hunyuan AI Infra Team (2026)), which attains an acceptance length of 3.586 and a 1.34× speedup under the same setting. These results confirm that ADAFLASH is effective not only for online adaptation but also for training diffusion drafters from scratch.

Table 9: From-scratch training using Qwen3-1.7B as target model on GSM8K dataset. The number in parentheses denotes the draft budget.

Method	Avg. TPS	τ	Speedup
AR Decoding	455.86	1.000	1.00×
EAGLE-3 (16)	611.58	3.586	1.34×
ADAFLASH (16)	**839.55**	**4.554**	**1.84**×

Different Temperature. We evaluate ADAFLASH under greedy decoding ($T = 0$) and stochastic sampling ($T = 1$) on four benchmarks at $C \in \{1, 32\}$ with Qwen3-8B. Table 10 reports the results.

Table 8: Cross-domain generalization results on Qwen3-8B. We compare offline DFlash with our ADAFLASH (trained on PERFECTBLEND), a mixed-domain dataset distinct from each test benchmark, and evaluated with fixed weights without further online adaptation. In-domain ADAFLASH results are reported in Table 1.

Method	Conc.	MATH				CODE				*Avg.*	
		MathQA		GSM8K		OpenCodeInstruct		CodeAlpaca			
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFlash	1	4.38×	7.09	3.84×	6.27	4.08×	6.69	3.39×	5.63	3.92×	6.42
	32	1.86×	7.10	1.59×	6.28	1.77×	6.68	1.46×	5.66	1.67×	6.43
	64	1.24×	7.11	1.03×	6.29	1.17×	6.67	0.91×	5.65	1.09×	6.43
	128	0.90×	7.11	0.82×	6.28	0.86×	6.67	0.74×	5.66	0.83×	6.43
ADAFLASH (cross-domain)	1	4.88×	8.37	4.65×	8.10	4.11×	6.98	3.43×	5.93	4.27×	7.35
	32	2.07×	8.39	1.84×	8.11	1.76×	6.96	1.48×	5.95	1.79×	7.35
	64	1.55×	7.91	1.37×	7.79	1.38×	6.65	1.13×	5.53	1.36×	6.97
	128	1.29×	7.93	1.35×	7.76	1.15×	6.64	1.07×	5.54	1.22×	6.97

Table 10: Effect of decoding temperature on ADAFLASH (Qwen3-8B). We evaluate greedy decoding ($T = 0$) and sampling with $T = 1$ on four benchmarks at concurrency $C \in \{1, 32\}$.

Method	Conc.	MathQA		GSM8K		OpenCodeInstruct		CodeAlpaca		*Avg.*	
Temperature = 0		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFlash	1	4.38×	7.09	3.84×	6.27	4.08×	6.69	3.39×	5.63	3.92×	6.42
	32	1.86×	7.10	1.59×	6.28	1.77×	6.68	1.46×	5.66	1.67×	6.43
AdaFlash	1	**5.32**×	9.83	**4.54**×	7.75	**4.87**×	8.40	**3.75**×	6.80	**4.62**×	8.20
	32	**2.27**×	9.84	**1.83**×	7.76	**2.03**×	8.39	**1.58**×	6.79	**1.93**×	8.20
Temperature = 1		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
DFlash	1	3.32×	5.85	3.48×	5.70	3.59×	5.82	2.97×	5.03	3.34×	5.60
	32	1.52×	5.84	1.38×	5.74	1.58×	5.82	1.32×	5.03	1.45×	5.61
AdaFlash	1	**3.56**×	6.75	**3.80**×	6.69	**3.85**×	6.67	**3.05**×	5.61	**3.56**×	6.43
	32	**1.62**×	6.74	**1.45**×	6.73	**1.66**×	6.58	**1.34**×	5.62	**1.52**×	6.42

As expected, sampling reduces the average acceptance length compared to greedy decoding ($\tau \approx 6.43$ vs. 8.20 at $C = 1$), since the target model's output distribution becomes more diffuse under $T = 1$ and can reduce their distributional overlap, thereby lowering the acceptance probability. Nevertheless, ADAFLASH retains a substantial speedup under sampling: $3.56\times$ at $C = 1$ and $1.52\times$ at $C = 32$ on average, demonstrating that the on-policy distillation and adaptive length head remain effective across decoding regimes.

Comparison with DSpark. DSpark (Cheng et al., 2026) employs a per-position confidence head that estimates the acceptance probability at each draft position and truncates the sequence once it drops below a threshold, whereas our adaptive length head directly predicts the expected acceptance length for the entire sequence (Section 3.2). We compare the two approaches on GSM8K (both using DFlash's backbone, only offline fine-tuning the confidence head/length head) using Qwen3-8B at concurrency $C \in \{64, 128\}$, with results reported in Table 11. Both methods substantially reduce the verification length from $k+1 = 16$ to approximately 6–7 tokens, confirming the benefit of dynamic length adjustment under high concurrency. However, our adaptive length head consistently achieves a higher acceptance length with a shorter verification length: at $C = 64$, $\tau = 5.164$ vs. 5.071 and verification length 6.448 vs. 6.673, yielding a speedup of $1.23\times$ compared to $1.21\times$; at $C = 128$, $\tau = 5.150$ vs. 5.074 and verification length 6.437 vs. 6.673. This improvement can be attributed to our formulation: directly predicting the overall expected acceptance length avoids the compounding errors inherent in sequential per-position probability estimation, providing a more accurate estimation of the quality of the draft sequence.

Table 11: Comparison of the adaptive length head with DSpark's confidence head on GSM8K using Qwen3-8B. The confidence threshold is 0.94 and $\gamma = 1.0$.

Method	τ	Verify Len.	Speedup
Concurrency = 64			
DFlash ($k=15$)	6.291	16.000	1.03×
DSpark's Head	5.071	6.673	1.21×
Adaptive Len Head	**5.164**	**6.448**	**1.23**×
Concurrency = 128			
DFlash ($k=15$)	6.282	16.000	0.82×
DSpark's Head	5.074	6.673	1.01×
Adaptive Len Head	**5.150**	**6.437**	**1.03**×