Unlocking Lossless Speedups in LLMs via Discrete Diffusion

Subham Sekhar Sahoo[*,†,1], Lingjie Chen[†,1,2], Khiem Pham[†,1,3], Jonathan Geuter[†,1,4], Chaitanya Dwivedi[1], Varad Pimpalkhute[1], Yash Akhauri[1], Alexander Moreno[1], Mikhail Yurochkin[1], Zhenting Wang[1], Mostafa Elhoushi[5], Nolan Dey[5], Shane Bergsma[5], Joel Hestness[5], John Thickstun[3], Eric Xing[1], Zhengzhong Liu[1]

[1]Institue of Foundation Models, [2]University of Illinois Urbana-Champaign, [3]Cornell Tech
[4]Harvard University [5]Cerebras Systems
[†] Core Contributors [*] Correspondence to subham.sahoo@mbzuai.ac.ae

Large Language Models (LLMs) owe much of their success to next-token prediction (NTP), but their autoregressive (AR) structure requires slow, sequential token generation. To overcome this bottleneck, we introduce diffusion-augmented LLMs, a new class of models that defines an AR model distribution while using diffusion to draw multiple tokens in parallel from that distribution. We decouple the parameters of these models into two sets: AR weights, trained using the standard NTP objective, and lightweight diffusion weights, trained to generate multiple tokens simultaneously. The diffusion weights are learned through a simple Diffusion Distillation phase that adds negligible overhead to existing LLM training pipelines. We also introduce Ψ-Spec, a family of samplers that enables lossless acceleration and inference-time scaling at a fixed context length. Unlike speculative decoding, our method requires no separate draft model. Unlike diffusion LLMs (d-LLMs), it accelerates generation without sacrificing the quality of the underlying AR model. The resulting models, called Uno, can be trained from scratch or built by augmenting existing open-weight AR LLMs. Uno achieves higher throughput than leading speculative-decoding methods at every evaluated batch size and delivers up to 3× speedups over the base AR model, including at the largest batch size supported by the device. Notably, our 8B Uno model outperforms the leading open d-LLM, the 26B DiffusionGemma, and the proprietary Mercury 2 across all evaluated benchmarks in agentic tool use, coding, and long-context reasoning. We release code and checkpoints on:
https://s-sahoo.com/uno

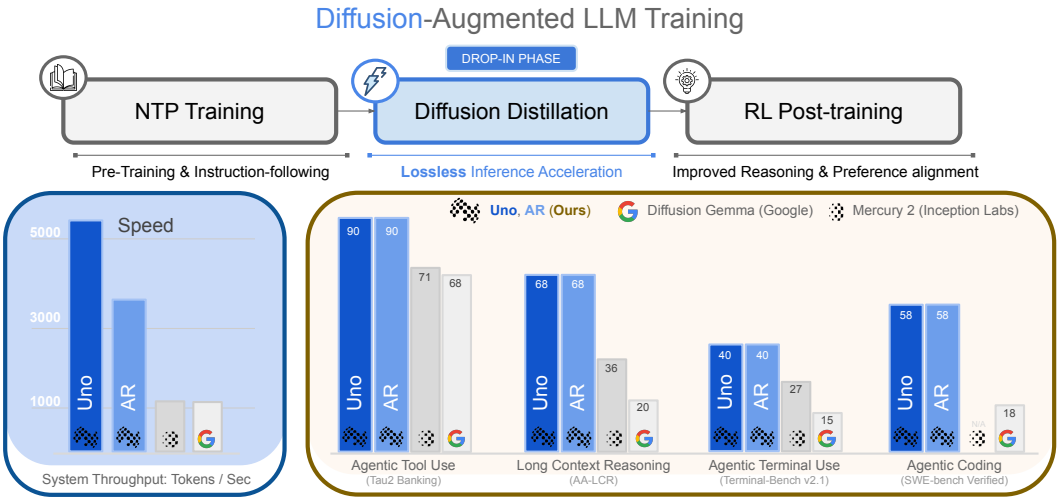

Figure 1: *(Top)* Training overview for diffusion-augmented LLMs. Gray cells indicate AR-weight training, while the blue cell indicates diffusion-weight training. *(Bottom Left)* System throughput of Uno, the base AR model, and the baselines; see Sec. 5.1 for details. *(Bottom Right)* Performance across agentic and long-context reasoning benchmarks.

1 Introduction

Large language models (LLMs) are increasingly showing human-level capabilities across coding, mathematics, and complex reasoning (KimiTeam et al., 2026; GLM-5-Team et al., 2026). These capabilities arise from a simple yet highly effective training objective: next-token prediction (NTP) over massive text corpora (Brown et al., 2020). A key limitation of this objective is that inference is inefficient. By training models to predict only the next token, it produces autoregressive (AR) systems that generate just one token per decoding step. This sequential constraint becomes increasingly costly as reasoning traces grow longer, increasing serving latency and slowing reinforcement-learning (RL) post-training, where rollout generation dominates runtime (Kim et al., 2026).

Sequential decoding is poorly suited to both natural language and modern accelerators. Language contains predictable collocations and formulaic sequences that could be generated together in blocks (Church & Hanks, 1990; Biber, 2009), but standard LLMs cannot exploit this redundancy. Moreover, decoding is often *memory bound*, especially at long context lengths, because moving model weights and key-value states limits inference speed and leaves GPUs underutilized (Verma & Vaidya, 2023). Predicting multiple tokens per step can amortize these memory transfers and better use parallel compute.

Existing approaches only partially address this opportunity. Speculative decoding accelerates generation by verifying tokens proposed by a smaller draft model (Leviathan et al., 2023; Chen et al., 2023), but its gains depend on an efficient, well-aligned drafter and require maintaining a separate model. Discrete diffusion models (Austin et al., 2021a; Sahoo, 2026) support parallel generation natively and have succeeded in biological domains (Sahoo et al., 2024b; Schiff et al., 2025; Tang et al., 2025; Lee et al., 2025), yet leading d-LLMs (d-LLMs) still face a quality-speed tradeoff relative to AR models (Sahoo et al., 2025b, 2026; Bie et al., 2026; DiffusionGemmaTeam et al., 2026). Their speedups also vanish at large inference batch sizes (Wu et al., 2025; Fu et al., 2026; DiffusionGemmaTeam et al., 2026). The (lossy) speedups offered by d-LLMs at low batch sizes can therefore be insufficient in practice, as agentic workloads now predominate in contemporary LLM applications (Google Cloud, 2025; Amazon Web Services, 2026). In these workloads, a single request can trigger parallel agents, branching trajectories, tool calls, and retries, while serving systems batch calls across requests to improve accelerator utilization. **Batch-size-one latency therefore captures a narrow operating regime and may overstate speedups that diminish under concurrency**. Thus, practical acceleration should be evaluated at realistic batch sizes.

Our core idea is simple: define a high-quality AR distribution, then learn to sample multiple tokens in parallel from that same distribution. We realize this idea through *diffusion-augmented LLMs* (Sec. 3), which use a single architecture with two decoupled sets of weights: one governing response quality and the other optimized for generation speed. Unlike multi-token prediction (MTP) approaches (Cai et al., 2024; Gloeckle et al., 2024), which modify the LLM architecture by adding prediction heads for future tokens, our method augments each layer in the LLM with lightweight diffusion weights alongside the standard AR weights. Our training pipeline first trains the AR weights through NTP loss, then freezes them and learns the diffusion weights through *Diffusion Distillation* (Sec. 3) to generate token blocks in parallel with negligible training overhead. Our Ψ-Spec sampler (Sec. 4) performs parallel prediction from the AR distribution. **The resulting model is a drop-in alternative to speculative decoding and self-speculative decoding**, requiring neither a separate draft model nor lossy AR-to-diffusion conversion.

We call our models **Uno** because they unify AR and diffusion weights within one architecture. Across our evaluations, Uno improves the speed-quality frontier over the speculative-decoding methods EAGLE-3 (Li et al., 2026b) and DFlash (Chen et al., 2026), the open-weight d-LLMs Nemotron-Labs-Diffusion (Fu et al., 2026) and DiffusionGemma (DiffusionGemmaTeam et al., 2026), and the proprietary d-LLM Mercury 2 (Ermon, 2026). Its speedup over the base AR model persists at every evaluated batch size, including the largest batch that fits on the device with the AR weights, supporting both low-latency generation and high-throughput serving. Our contributions are:

1. We introduce diffusion-augmented LLMs, which decouple the parameters that determine response quality from lightweight parameters optimized for generation speed, together with a corresponding training pipeline (Sec. 3).

2. We propose Ψ-Spec samplers to sample from diffusion-augmented LLMs. These samplers enable both lossless, AR-verified acceleration and inference-time scaling through additional denoising at a fixed context length; see Sec. 4.
3. We show how to train diffusion-augmented LLMs from scratch (Sec. 5.1) or create them by augmenting an existing open-weight AR LLM (Sec. 5.2).
4. We show that Uno achieves up to a 2× speedup at the largest batch size supported by the base AR model, accelerating both inference and end-to-end RL training (Sec. 5.1).

2 Background

Notation We denote scalar discrete random variables with K categories as 'one-hot' column vectors and define $\mathcal{V} = \{\mathbf{v} \in \{0,1\}^K : \sum_{i=1}^K \mathbf{v}_i = 1\}$ as the set of all such vectors. Define $\text{Cat}(\cdot\,;\,\boldsymbol{\pi})$ as the categorical distribution over K classes with probabilities given by $\boldsymbol{\pi} \in \Delta$, where Δ denotes the $K-1$-simplex. We also assume that the K-th category corresponds to a special `[MASK]` token and let $\mathbf{m} \in \mathcal{V}$ be the one-hot vector for this mask, i.e., $\mathbf{m}_K = 1$. Additionally, let $\mathbf{1} = \{1\}^K$, and $\langle \mathbf{a}, \mathbf{b} \rangle$ and $\mathbf{a} \odot \mathbf{b}$ respectively denote the dot and Hadamard products between two vectors $\mathbf{a}$ and $\mathbf{b}$. We use $\mathbf{x} \in \mathcal{V}^L$ to represent clean data which is a length-L sequence containing no mask tokens, and let $\mathbf{x}^\ell$ denote its ℓ^{th} element where ℓ refers to the token index. Under our notation, each $\mathbf{x}^\ell$ is a one-hot vector. The notation $\mathbf{x}^{m:n}$ denotes the subsequence from index m to index n, inclusive, while $\mathbf{x}^{<\ell} \coloneqq \mathbf{x}^{1:\ell-1}$ denotes the prefix consisting of the first $\ell-1$ tokens. Finally, $[\cdot,\cdot] : \mathcal{V}^m \times \mathcal{V}^n \to \mathcal{V}^{m+n}$ denotes the operator that concatenates sequences of lengths m and n. Colors are used throughout the paper solely for emphasis and carry no mathematical meaning.

2.1 Autoregressive Models

Consider a length-L sequence $\mathbf{x} \in \mathcal{V}^L \sim q_{\text{data}}$ drawn from the data distribution q_{data}. Autoregressive (AR) language models factorize the joint distribution using the chain rule:

$$\log p_\theta(\mathbf{x}) \;=\; \sum_{\ell=1}^{L} \log p_\theta(\mathbf{x}^\ell \mid \mathbf{x}^{<\ell}).$$

Here, $p_\theta : \mathcal{V}^L \to \Delta^L$ is typically implemented with a causal Transformer (Vaswani et al., 2017) with parameters θ. This factorization leads to strong likelihood modeling and enables efficient inference primitives such as KV caching. However, AR generation is intrinsically sequential: ℓ^{th} token can only be generated after the prefix $\mathbf{x}^{<\ell}$ has been generated. As a result, producing an output of length L requires L causal decoding decisions.

Speculative Decoding Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) uses a smaller draft model to autoregressively generate a block of candidate tokens, while a larger base model verifies them. The key observation is that although sampling from the base model is inherently sequential and expensive, evaluating the likelihood of an entire candidate sequence can be performed efficiently in parallel. Speculative decoding exploits this asymmetry by generating candidate tokens with the draft model that can be sampled more efficiently, and verifying them via rejection sampling in a single forward pass, accepting the longest valid prefix. This accelerates inference while exactly preserving the target distribution.

2.2 Discrete Diffusion Language Models

Discrete diffusion corrupts a clean sequence $\mathbf{x} \in \mathcal{V}^L \sim q_{\text{data}}$ into a simple prior $\boldsymbol{\pi}^L$ and learns to reverse the corruption. We use the interpolating forward process from Sahoo et al. (2024a):

$$\mathbf{z}_t^\ell \sim q_t^\ell(\cdot \mid \mathbf{x}; \boldsymbol{\pi}) \coloneqq \text{Cat}\big(\cdot\,;\, \alpha_t \mathbf{x}^\ell + (1-\alpha_t)\boldsymbol{\pi}\big), \tag{1}$$

where $\mathbf{z}_t$ denotes the noisy sequence at time step $t \in [0,1]$, and the noise schedule α_t decreases monotonically from $\alpha_0 = 1$ at clean data to $\alpha_1 = 0$ at the prior. A denoising model $\mathbf{x}_\theta : \mathcal{V}^L \times [0,1] \to \Delta^L$, parameterized by θ, predicts the clean tokens from $\mathbf{z}_t$ and t. Masked diffusion uses $\boldsymbol{\pi} = \mathbf{m}$, while uniform-state diffusion uses $\boldsymbol{\pi} = \mathbf{1}/K$. We use the latter because it natively supports self-correction, few-step generation (Sahoo et al., 2025a), and better inference-time scaling than masked diffusion models (Deschenaux et al., 2026).

Sampling For $0 \le s < t$, let $q_{s|t}$ denote the reverse posterior that maps $\mathbf{z}_t$ to a less noisy state $\mathbf{z}_s$. The Ψ-samplers proposed by Deschenaux et al. (2026) form a family of discrete-diffusion samplers that generalize posterior (ancestral) sampling by incorporating predictor-corrector capabilities, thereby producing higher-quality samples. Their practical token-wise transition kernel is given by

$$[\Psi^{\theta}_{s|t}(\cdot \mid \mathbf{z}_t; \mathbf{x}_\theta(\mathbf{z}_t), \pi)]^{\ell} = \kappa_t q^{\ell}_{s|t}(\cdot \mid \mathbf{z}_t, \mathbf{x}_\theta(\mathbf{z}_t); \pi) + (1-\kappa_t)\Big[\alpha_s q^{\ell}_{0|t}(\cdot \mid \mathbf{z}_t, \mathbf{x}_\theta(\mathbf{z}_t); \pi) + (1-\alpha_s)\pi\Big], \quad (2)$$

where $\kappa_t \in [0,1]$ controls the strength of the correction. The exact functional form of (2) for MDMs and USDMs, see Suppl. A.4.

Discrete Consistency Distillation Discrete Consistency Distillation (DCD) compresses a multi-step uniform-state diffusion process into a few-step generator (Sahoo et al., 2025a). It constructs a deterministic discrete trajectory from $\mathbf{z}_1 \sim \mathbf{1}^L/K$ to $\mathbf{x}$ using the Gaussian diffusion process underlying the uniform-state discrete diffusion process,. For adjacent states $\mathbf{z}_t$ and $\mathbf{z}_s$, with $s < t$, the student distribution $\mathbf{x}^{\ell}_{\boldsymbol{\theta}}(\mathbf{z}_t, t)$ is trained to match the teacher distribution $\mathbf{x}^{\ell}_{\boldsymbol{\theta}_0}(\mathbf{z}_s, s)$; $\forall \ell \in [L]$:

$$\mathcal{L}_{\mathrm{DCD}}(\boldsymbol{\theta}; \boldsymbol{\theta}_0) = \sum_{\ell=1}^{L} \mathrm{D}_{\mathrm{KL}}\big(\mathbf{x}^{\ell}_{\boldsymbol{\theta}}(\mathbf{z}_t, t) \,\|\, \mathbf{x}^{\ell}_{\boldsymbol{\theta}_0}(\mathbf{z}_s, s)\big).$$

Increasing the gap between s and t across distillation rounds teaches the student to take larger denoising steps. Refer Suppl. A.5 for further details.

3 Diffusion-augmented LLMs

We propose a new framework for designing LLMs that decouples generation quality from generation speed by augmenting each layer of a standard AR model with a separate set of diffusion weights dedicated to parallel token generation. We call the resulting model a diffusion-augmented LLM. The key distinction is that each layer contains two sets of weights: AR weights, trained to generate tokens autoregressively and determine response quality, and diffusion weights, trained to generate tokens in parallel. This separation preserves the established AR training pipeline while introducing a dedicated phase for training the diffusion weights to accelerate inference. At generation time, both sets of weights draft tokens in parallel, after which the AR weights alone verify the drafts. Our sampler provably preserves the AR model's output distribution (Sec. 4), enabling lossless acceleration without sacrificing response quality. The framework can be applied to any causal autoregressive neural network, including causal Transformers (Vaswani et al., 2017) and State-Space Models (SSMs; Gu et al., 2020; Gu & Dao, 2023). Because of this separation, the framework can also augment existing open-weight LLMs with parallel generation capabilities, providing lossless speedups without retraining their base parameters (Sec. 5.2).

The remainder of this section develops the framework in three stages. We first describe the roles of the AR and diffusion weights in a Diffusion-augmented LLM (Sec. 3.1). We then present the Diffusion Distillation Phase (Sec. 3.2), which trains the diffusion weights using a one-step block-denoising formulation and associated training objectives. Finally, we discuss the training curriculum for two settings: accelerating inference alone and accelerating both RL rollouts and inference (Sec. 3.3).

3.1 Autoregressive and Diffusion Pathways

Each layer of a diffusion-augmented LLM contains a set of autoregressive (AR) weights and a set of diffusion weights. These two sets of weights follow different training procedures, as described below.

Autoregressive Weights We denote the base AR weights by θ_{AR}. These parameters are trained using the standard LLM recipe of Pre-training (Brown et al., 2020; Hoffmann et al., 2022; Grattafiori et al., 2024), supervised fine-tuning (SFT; Wei et al., 2021; Chung et al., 2024), and RL post-training (Ouyang et al., 2022; Shao et al., 2024; Guo et al., 2025). Given a length-L sequence $\mathbf{x} \in \mathcal{V}^L$, the model $\mathbf{x}_{\theta_{\mathrm{AR}}} : \mathcal{V}^L \to \Delta^L$ is causally masked, so its output at position ℓ observes only the prefix $\mathbf{x}^{<\ell}$, the distribution of the ℓ^{th} token is modeled as:

$$p_{\theta_{\mathrm{AR}}}(\mathbf{x}^{\ell} \mid \mathbf{x}^{<\ell}) = \mathbf{x}^{\ell-1}_{\theta_{\mathrm{AR}}}(\mathbf{x}), \qquad 1 < \ell \le L.$$

We assume that the first token in $\mathbf{x}$ is a special `<BOS>` token that requires no modeling.

Diffusion Weights The diffusion weights θ_Δ are used solely to accelerate generation. As described in Sec. 4, our sampler uses the diffusion pathway to draft tokens and performs rejection sampling against the AR distribution defined by θ_{AR}. We parameterize the diffusion weights as LoRA (Low RAnk) adapters (Hu et al., 2022): each AR weight matrix has an associated LoRA weight. Thus, the diffusion pathway uses $\theta_{\text{AR}} + \theta_\Delta$ to generate draft tokens, whereas the verification pathway uses θ_{AR}, with the two distributions coupled through their shared base parameters. We train only θ_Δ during the Diffusion Distillation Phase while keeping θ_{AR} frozen; see Sec. 3.2.

For effective rejection sampling, the draft distribution must remain closely coupled to the verification distribution. Parameterizing the draft pathway as a LoRA adaptation of the base model promotes this coupling while adding minimal inference-time memory overhead compared with a separate set of full-sized diffusion weights. This design retains the lossless verification of standard speculative decoding without requiring a separate draft model. At the same time, like self-speculative decoding, it tightly couples the drafting and verification pathways, but does so while remaining lossless..

Diffusion training teaches the LLM to refine an entire sequence in parallel. Specifically, given a noisy sequence $(\mathbf{z}_t \in \mathcal{V}^L)_{t\in[0,1]}$ obtained by corrupting a clean sequence $\mathbf{x} \in \mathcal{V}^L$ according to (1), the model is trained to recover $\mathbf{x}$. Throughout this work, the LLM retains the NTP parameterization of standard autoregressive LLMs: the logits at each position predict the subsequent clean token. This parameterization differs from that of conventional diffusion language models (Austin et al., 2021a; Lou et al., 2024; Sahoo et al., 2024b), which typically predict the clean token at the same position. Accordingly, the distribution of the clean token $\mathbf{x}^\ell$ is:

$$p_{\theta_{\text{AR}},\theta_\Delta}(\mathbf{x}^\ell \mid \mathbf{z}_t) = \mathbf{x}^{\ell-1}_{\theta_{\text{AR}},\theta_\Delta}(\mathbf{z}_t), \qquad 1 < \ell \le L.$$

3.2 Diffusion Distillation Phase

We train the diffusion parameters θ_Δ to draft a block of tokens in parallel that the autoregressive (AR) model with parameters θ_{AR} would generate sequentially. Specifically, our goal is to match the single-step diffusion distribution $p_{\theta_{\text{AR}},\theta_\Delta}(\cdot)$ over a token block to the AR distribution $p_{\theta_{\text{AR}}}(\cdot)$ over the same block. Discrete Consistency Distillation (DCD; Sec. 2.2) is well suited to this task because it can distill a multistep diffusion process into a few-step generator while closely approximating the distribution induced by the original process.

In what follows, we first describe how we adapt DCD to approximate the AR distribution through single-step diffusion. We then extend the DCD training objective, namely the single-step distillation loss $\mathcal{L}_{\text{DCD}}$, with another auxiliary loss term: $\mathcal{L}_{\text{TV}}$ to encourage longer diffusion drafts to be accepted by the AR verifier. The resulting objective is:

$$\mathcal{L}(\theta_\Delta;\theta_{\text{AR}},\alpha,\beta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{z}_1\sim\boldsymbol{\pi}^L}\left[\alpha\mathcal{L}_{\text{DCD}}(\theta_\Delta;\theta_{\text{AR}},\mathbf{x},\mathbf{z}_1) + \beta\mathcal{L}_{\text{TV}}(\theta_\Delta;\theta_{\text{AR}},\mathbf{x},\mathbf{z}_1)\right], \tag{3}$$

where $\mathbf{z}_1$ is the fully corrupt sequence. Although $\alpha = 0, \beta = 1$ yields the largest speedups, training first with $\alpha = \beta = 1$ and then switching to $\alpha = 0, \beta = 1$ accelerates convergence. We ablate both loss terms in Sec. 5.2.4. We describe each term below.

One-step Distillation Standard DCD constructs a multistep denoising trajectory by simulating intermediate states of the underlying PF-ODE. At LLM scale, repeatedly constructing and storing these intermediate Gaussian latents is prohibitively expensive. We therefore distill the entire trajectory into a single denoising step that maps a fully corrupted input $\mathbf{z}_1 \sim \boldsymbol{\pi}^L$ directly to the clean sequence $\mathbf{x}$. The frozen base model defines the autoregressive teacher distribution $\mathbf{x}_{\theta_{\text{AR}}}$, and the adapted model defines the student distribution $\mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}$. We train the student to match the teacher directly, eliminating the need to simulate or materialize any intermediate PF-ODE states.

Block Diffusion Recovering an entire long sequence from a fully corrupted input in a single denoising step is prohibitively difficult; we therefore perform one-step denoising blockwise. We partition the clean sequence $\mathbf{x}$ and the fully corrupt sequence $\mathbf{z}_1$ into N blocks of size B. We use $\mathbf{x}^{(b)}$ and $\mathbf{z}_1^{(b)}$ to denote the b^{th} blocks of $\mathbf{x}$ and $\mathbf{z}_1$, respectively. For each block, we train θ_Δ to match the student diffusion distribution over the clean block $\mathbf{x}^{(b)}$, conditioned on the noisy block $\mathbf{z}_1^{(b)}$ and the preceding clean context $\mathbf{x}^{(<b)}$, to the teacher AR distribution over the same block, conditioned on the preceding clean context. We compute the teacher and student predictions in a single forward pass over the concatenated sequence $[\mathbf{x}, \mathbf{z}_1]$. Specifically, we use a block-causal attention mask

that permits causal attention within $\mathbf{x}$ and within each noisy block $\mathbf{z}_1^{(b)}$. Tokens in $\mathbf{z}_1^{(b)}$ additionally attend to all preceding clean blocks $\mathbf{x}^{(<b)}$.

The main challenge is to compute the teacher logits at positions in $\mathbf{x}$ using only θ_{AR}, while computing the student logits at positions in $\mathbf{z}_1$ using both θ_{AR} and θ_Δ. We achieve this using gated LoRA (Samragh et al., 2025), which disables the adapters at clean-sequence positions and enables them at noisy-sequence positions. Consequently, θ_{AR} produces the teacher logits, whereas θ_{AR} and θ_Δ jointly produce the student logits. The resulting blockwise DCD objective is:

$$\mathcal{L}_{\text{DCD}}(\theta_\Delta; \theta_{\text{AR}}, \mathbf{x}, \mathbf{z}_1) = \sum_{b=1}^{N} \sum_{\ell=1}^{B} \mathrm{D}_{\text{KL}} \left(\mathbf{x}_{\theta_\Delta, \theta_{\text{AR}}}^{(N+b,\ell)}([\mathbf{x}, \mathbf{z}_1]) \| \mathbf{x}_{\theta_{\text{AR}}}^{(b,\ell)}([\mathbf{x}, \mathbf{z}_1]) \right).$$

The base parameters θ_{AR} remain frozen throughout fine-tuning, and only the LoRA parameters θ_Δ are updated.

Total Variation Loss Our sampler (Sec. 4) performs rejection sampling against the AR distribution and retains only the longest consecutive prefix of the diffusion prediction. Sampling efficiency therefore depends on the length of this retained prefix. To increase its expected length, we minimize the blockwise Total Variation (TV) distance between the diffusion and the AR distributions following Corollary 3.6 of Leviathan et al. (2023):

$$\mathcal{L}_{\text{TV}}(\theta_\Delta; \theta_{\text{AR}}, \mathbf{x}, \mathbf{z}_1) = \sum_{b=1}^{N} \sum_{\ell=1}^{B} \left| \mathbf{x}_{\theta_\Delta, \theta_{\text{AR}}}^{(N+b,\ell)}([\mathbf{x}, \mathbf{z}_1]) - \mathbf{x}_{\theta_{\text{AR}}}^{(b,\ell)}([\mathbf{x}, \mathbf{z}_1]) \right|.$$

Minimizing this objective increases the probability of consecutive tokens being accepted, and, consequently, increases the expected length of the accepted draft prefix.

3.3 Practical Considerations

The AR weights θ_{AR} and diffusion adapters θ_Δ serve complementary roles and are optimized using separate objectives, making their training order an important practical consideration. In particular, Diffusion Distillation can be performed either after all AR training is complete or before RL post-training so that the resulting adapters can accelerate rollout generation. We therefore consider two training regimes, depending on whether diffusion-based generation is used only during inference or during both RL post-training and inference.

Faster Inference Only If the sole objective is faster inference, we first complete autoregressive pre-training and post-training, or simply initialize them with an Open-weights LLM. We then freeze the AR weights θ_{AR} and train θ_Δ using Diffusion Distillation (Sec. 3.2).

Faster RL Training & Faster Inference We demonstrate in Sec. 5 that our method provides significant generation speedups wrt the base AR model across all batch sizes, and can therefore accelerate RL post-training due to faster rollouts. In this setting, we first apply diffusion distillation after supervised fine-tuning and before RL, as illustrated in Figure 1 (top). The resulting adapters can then accelerate rollouts during RL. Standard RL policy-optimization recipes, including PPO and GRPO-based methods (Ouyang et al., 2022; Shao et al., 2024; Guo et al., 2025; Yu et al., 2026a), update θ_{AR} but not the θ_Δ. One might therefore expect that, as θ_{AR} changes, the draft distribution will drift away from the verifier distribution, reducing the acceptance rate and eroding the speedup. **Surprisingly, we show in Sec. 5.1 that the speedup is retained even as the AR weights are trained using RL.**

4 Ψ-Speculative Sampler

Our goal is to sample from the AR distribution defined by θ_{AR} while drawing multiple tokens in parallel. We therefore introduce the Ψ-Speculative sampler (Ψ-Spec), which uses the diffusion pathway to propose a block of tokens in parallel (Sec. 4.1) and performs rejection sampling against the base AR distribution to accept the longest valid prefix (Sec. 4.2).

4.1 Diffusion Sampler

Let $\mathbf{x} \in \mathcal{V}^L$ denote a partially generated sequence of length L. To generate a block of B tokens, we first append $B-1$ random tokens sampled from the prior, $\mathbf{z}_1 \sim \boldsymbol{\pi}^{B-1}$. Denoising then proceeds from $t = 1$ to $t = 0$, with $\mathbf{z}_t$ denoting the partially denoised sequence at time t. At each step, the denoiser $\mathbf{x}_\theta(\cdot\,;\,\mathbf{x}^{<L}) : \mathcal{V}^B \to \Delta^B$ operates on the block $[\mathbf{x}^L, \mathbf{z}_t] \in \mathcal{V}^B$, while the activations for the preceding sequence $\mathbf{x}^{<L}$ remain in the KV cache. In Sec. 4.1.1, we define the proposal distribution over a block of tokens, and in Sec. 4.1.2, we describe how candidates are sampled from this distribution.

4.1.1 Diffusion Proposal Distribution

Given a noisy block $\mathbf{z}_t$, we use the Ψ-Spec transition in (2) to sample a less noisy block $\mathbf{z}_s$, where $s < t$. Because the denoiser uses the NTP parameterization, the clean-token distributions used to construct $\mathbf{z}_s$ are given by $\mathbf{x}^{1:B-1}_{\theta_{\text{AR}},\theta_\Delta}([\mathbf{x}^L, \mathbf{z}_t])$. The input contains both clean and corrupted tokens, while the diffusion weights are trained only on corrupted tokens. To avoid distribution shift, we compute the logits for the first, clean position using only the base AR weights, and those for the noisy positions using both the AR and diffusion weights. We achieve this in a single forward pass using the gated LoRA technique of Samragh et al. (2025).

Let Ψ_0 denote the distribution induced by the Ψ-sampler at $t = 0$. At the final step, we sample the first token using only the base AR weights:

$$\mathbf{z}_0^1 \sim \Psi_0^{\ell=1}\left(\cdot \mid \mathbf{z}_t; \mathbf{x}^{1:B-1}_{\theta_{\text{AR}}}(\cdot), \pi\right). \tag{4}$$

We sample the remaining $B-2$ tokens from the following joint distribution, discussed in Sec. 4.1.2:

$$\mathbf{z}_0^{2:B} \sim \prod_{\ell=2}^{B} \Psi_0^{\ell}\left(\cdot \mid \mathbf{z}_t; \mathbf{x}^{1:B-1}_{\theta_{\text{AR}},\theta_\Delta}(\cdot), \pi\right). \tag{5}$$

4.1.2 Sampling Candidates

Given the proposal distribution in (5), we construct a candidate set $\mathcal{C} = \{\mathbf{c} : \mathbf{c} \sim \prod_{\ell=2}^{B} \Psi_0^{\ell}\}$. The number of draft candidates determine the tradeoff between acceptance length and verification cost. Sampling more candidates increases the probability of accepting a longer prefix, but also increases the cost of verification. We therefore select the largest candidate set that can be verified without reducing inference throughput, subject to the available serving batch size.

Linear Sampler (System Throughput Optimized) The simplest approach samples each draft token directly from its marginal distribution in (5), producing a single candidate sequence. At high batch sizes, inference can become compute-bound, leaving little spare compute for verifying additional candidates. The linear sampler is therefore well suited to maximizing aggregate system throughput in this regime.

Tree Sampler (Single-user Throughput Optimized) At low batch sizes, inference is typically memory-bound, leaving substantial compute capacity underutilized. We exploit this spare compute by sampling multiple candidates and verifying them in parallel. Specifically, we use the tree-based sampling procedure of Cai et al. (2024), which selects the top K tokens at each position within a block. Rather than evaluating all K^{B-1} candidate sequences, we adopt the candidate-pruning strategy of Li et al. (2026b) which ranks candidates by their log-probabilities and retains the top $V \in \mathbb{Z}^+$ prefixes. Thus, (B, K, V) are the hyperparameters of the tree sampler, corresponding to the block size, branching factor, and prefix budget, respectively.

4.2 One-step Diffusion w/ AR Verification

The central goal of this work is to accelerate generation. We therefore draft an entire block of B tokens using a single diffusion forward pass. In Suppl. B.2, we show that, for single-step diffusion generation, Eqns. (4, 5) reduce to

$$\mathbf{z}_0^1 \sim \mathbf{x}^1_{\theta_{\text{AR}}}(\cdot),$$
$$\mathbf{z}_0^{2:B} \sim p_{\text{draft}} = \prod_{\ell=2}^{B} \mathbf{x}^{\ell}_{\theta_{\text{AR}},\theta_\Delta}(\cdot).$$

From p_{draft}, we sample candidates $\mathcal{C} = \{\mathbf{c} : \mathbf{c} \sim p_{\text{draft}}\}$ and apply the standard speculative-decoding rejection correction (Leviathan et al., 2023), retaining the longest prefix accepted by the base AR model. This preserves the base model's target distribution. For $|\mathcal{C}| > 1$, candidates are verified concurrently as a prefix tree using tree attention (Cai et al., 2024). Because θ_{AR} remains frozen and only the diffusion LoRA adapters are trained, the base AR model provides an unchanged verifier and enables lossless speculative speedups. The complete sampling algorithm is provided in Algo. 1.

Tokens-Per-Forward-pass (TPF) During drafting, the first token is generated using the base AR weights and therefore matches the verifier distribution exactly, so it is always accepted. If a subsequent draft token is rejected, the verifier samples a replacement from the renormalized residual distribution, as in standard speculative decoding. Consequently, even an immediate rejection produces two output tokens: the always-accepted first token and the verifier-sampled replacement token. If all B draft tokens are accepted, the verifier additionally samples one token from the logits following the final draft token, producing $B + 1$ output tokens. Because each iteration requires two forward passes, one for drafting and one for verification, the TPF is bounded by $1 \leq \text{TPF} \leq \frac{B+1}{2}$.

We provide support for both the Linear and Tree samplers in Nano-vLLM (Yu, 2025) and SGLang (Zheng et al., 2024). All experiments reported in this paper were conducted using our Nano-vLLM implementation.

4.3 Inference-Time Scaling

Ψ-Spec introduces an additional axis for inference-time scaling in LLMs by allowing the number of denoising steps T to exceed the number of drafted tokens B. Deschenaux et al. (2026) show that, for $0 \leq \kappa_t < 1$, increasing T consistently improves sample quality. The central question is whether this improvement can eventually surpass the quality of AR generation. If so, AR verification should be disabled, since it would constrain the final output to the lower quality level of the AR model and prevent the gains from additional denoising from being realized. **Unlike conventional inference-time scaling methods, this approach allocates additional computation without increasing the context length.** We leave a systematic exploration of this quality-compute tradeoff to future work.

5 Experiments

We train diffusion-augmented LLMs in two settings. In the first, we train the AR weights from scratch on proprietary data and train the diffusion weights on data drawn from the same distribution. In the second, we augment an open-weight AR model, Qwen3-8B (Yang et al., 2025), without access to its original training data. We instead train the diffusion weights on a different data distribution using the open-source OpenThoughts dataset (Guha et al., 2025). Together, these experiments demonstrate the versatility of our method and show that the diffusion and AR weights need not be trained on the same data distribution.

Throughput Analysis Throughput depends strongly on context length. However, existing evaluations (Wu et al., 2025; Fu et al., 2026) measure throughput on downstream tasks whose reasoning traces vary in length. This can misleadingly favor models that generate shorter traces. Following standard LLM evaluation practice[1], we instead evaluate every method using m random input tokens and a fixed output length of n tokens. For diffusion and speculative decoding methods, which draft a block of B tokens per forward pass, we first measure the average number of accepted tokens per forward pass (TPF) across all benchmarks. We then run $\left\lceil \frac{n}{\text{TPF}} \right\rceil$ decoding steps, constraining each method to accept TPF tokens per step on average. This ensures that all methods are evaluated at the same input and effective output lengths. Throughout the paper, we use $m = 1024$ and $n = 8192$ and refer to this setting as the "1K/8K throughput test".

Per-request Throughput vs. System Throughput We report throughput at batch size 1 and at the largest batch size that fits on a single H200 GPU. **Batch-size-1 throughput is less representative of practical serving because agentic workloads often generate concurrent requests even while serving a single user** as described in Sec. 1. We therefore also report system throughput, which is

[1] https://nvidia.github.io/TensorRT-LLM/developer-guide/perf-benchmarking.html

the aggregate token generation rate at the maximum feasible batch size, which better reflects serving capacity and cost efficiency. For each regime, we select the TPF setting that maximizes throughput and use the 1K/8K test described above.

5.1 End to End Diffusion Augmented LLM Training

5.1.1 Setup

Benchmarks We evaluate on both agentic and non-agentic benchmarks. The agentic suite comprises the Telecom, Airline, and Retail domains of τ^3-Bench (Shi et al., 2026), τ^2-Bench (Barres et al., 2025) , Terminal-Bench v2.1 (The Terminal-Bench Team, 2026), and SWE-bench Verified (Jimenez et al., 2024). The non-agentic suite includes AA-Omniscience (Jackson et al., 2025), AA-LCR (Artificial Analysis, 2025), Humanity's Last Exam (HLE; Phan et al., 2025) , GPQA-Diamond (Rein et al., 2024), GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2024), AIME 2024 and 2025 (Balunović et al., 2025), AIME 2026 (Dekoninck et al., 2026), and MBPP (Austin et al., 2021b). We use a context window of $262,144$ with $131,072$ maximum generation tokens, allowing for long prompts on agentic benchmarks, and sampling parameters temp = 1, top-p = 0.95, and top-k = 50.

Metrics For each method, we report average pass@1 over multiple generations and average TPF; see Table 4 for details. We compute throughput using the "1k/8k throughput test," averaging TPF across benchmarks. System throughput is measured using the largest batch size that fits on a single H200 GPU, whereas per-request throughput is measured at batch size 1.

Baselines We compare against Nemotron-Labs-Diffusion-14B (Fu et al., 2026), DiffusionGemma-26B-A4B (DiffusionGemmaTeam et al., 2026), and Mercury 2 (Ermon, 2026). Nemotron-Labs-Diffusion is pretrained autoregressively on 1T tokens and then fine-tuned as a diffusion model on 300B tokens. It also uses masked diffusion with bidirectional attention within draft blocks. We evaluate the largest Nemotron-Labs-Diffusion variant because it performs better on benchmarks than the smaller variants. DiffusionGemma is a sparse 26B mixture-of-experts model with 4B active parameters, fine-tuned from Gemma 4 (DiffusionGemmaTeam et al., 2026). Since **both Nemotron-Labs-Diffusion and DiffusionGemma modify the base AR parameters, they are lossy unlike Uno**. We use the default baseline configurations: Nemotron-Labs-Diffusion uses Linear Self-Speculation with a block size of 32, a maximum generation length of 8192, a thinking-token budget of 6000, and temperatures of 0 for drafting and verification. DiffusionGemma uses a block size of 256, temperature 1, a maximum generation length of 131072, a context length of 162144, and an entropy-bounded diffusion sampler with a bound of 0.1 and thinking enabled. We also compare against Inception Labs' closed-source d-LLM, Mercury 2, whose parameter count is undisclosed.

5.1.2 Uno: Diffusion Augmented LLM (Ours)

We first define the architecture of our model. The weights corresponding to every layer are the the AR weights θ_{AR}. Corresponding to every weight matrix, we introduce rank-128 LoRA adapters with LoRA-α = 256 which serve as the diffusion weights θ_{Δ}. We share more details subsequently.

LLM Architecture We use a dense decoder-only causal Transformer with 36 layers, hidden width $4,096$, SwiGLU-style MLP width $12,288$, 32 query heads, 8 KV heads, head dimension 128, grouped RMSNorm, RoPE ($\theta = 10^7$), a $250,624$-token vocabulary, and a maximum configured context length of $524,288$ tokens. The model contains 6.95B transformer-body parameters plus approximately 2.05B parameters from the large untied input/output vocabulary matrices.

Training AR weights The AR weights are trained on approximately 23T tokens on internal quality data with a staged context-length extension schedule. Pretraining uses 21.9T tokens at an 8K-token context length, with a peak learning rate of 3×10^{-4} under a warmup-stable-decay (WSD) configuration and a 3,000-step warmup. We then extend the context length to 32K, training on an additional 1.1T tokens while linearly decaying the learning rate from 3×10^{-4} to 3×10^{-5}, following a 1,250-step warmup. Next, the context length is increased to 128K and the model is trained on 0.5T tokens

at a constant learning rate of 3×10^{-5}, with a 500-step warmup. Finally, the context length is extended to 512K and trained for 0.3T tokens with a constant learning rate of 3×10^{-5} and a 200-step warmup.

Training Diffusion Weights Diffusion weights are implemented as rank-128 LoRA adapters with LoRA-α = 256. We train them on 7B tokens sampled randomly from the SFT training data while keeping the AR weights frozen. We employ curricula for context length and diffusion block size: 1.8B tokens at context length $16,384$, divided into increasing block sizes of two, four, and eight with 600M tokens each; followed by 5.2B tokens at context length $65,536$ and block size 8. We train on a global batch size of 128 with a WSD learning rate scheduler with 200 warmup steps and a peak learning rate of 5×10^{-5}. In (3), we set $\alpha = 0.01, \beta = 1$. Training takes approximately 60 hours on 8 H200 nodes with 8 GPUs each.

To sample from Uno, we use the Linear Sampler with $B = 4$ to maximize system throughput and the Tree Sampler with $(B, K, V) = (16, 32, 32)$ to maximize single-request throughput at batch size 1.

5.1.3 Results

Ψ-Spec Sampler Configurations We examine how TPF and throughput vary across Linear and Tree sampler configurations in Table 6. For the Linear sampler, TPF increases from 1.8 at $B = 4$ to 2.2 at $B = 8$, but reaches only 2.3 at $B = 16$; $B = 4$ achieves the highest system throughput. For the Tree sampler, with $B = 16$ and $K = 32$, $V = 32$ provides higher per-request throughput than $V = 64$, likely because the larger verification cost reduces throughput.

Uno vs. Base AR Uno qualitatively matches the base AR model while achieving higher throughput at every batch size. In the 1k/8k throughput test, the largest batch size supported by the base AR model is 64, at which Uno is 1.5× faster (see Table 7). This speedup benefits multi-user serving and RL post-training. At batch size 1, Uno is approximately 2.2× faster.

Takeaway 1: Uno matches the qualitative performance of Base AR while delivering over 2× higher throughput across all batch sizes.

Comparison with Open Weights Diffusion Language Models As shown in Table 1, Uno outperforms the open-weight DiffusionGemma and Nemotron-Labs-Diffusion models on all tasks, with larger gains on agentic tasks. Using the TPF values in Table 1, we measure maximum system throughput at the largest feasible batch size and per-request throughput at batch size 1. Despite using full attention in every layer, Uno achieves higher system throughput (with Tree Sampler; $(B, K, V) = (16, 32, 32)$) than both baselines. Notably, DiffusionGemma uses strided attention, with five local sliding-window self-attention layers for every global self-attention layer, yet remains slower at large batch sizes. DiffusionGemma is faster at batch size 1 but has substantially lower accuracy, while Nemotron-Labs-Diffusion trails Uno in both throughput and output quality. Although smaller blocks may improve throughput, further tuning offers no practical benefit given the baselines' lower quality. Also, **note that both baselines are slower than their respective base AR models at large batch sizes** (Fu et al., 2026; DiffusionGemmaTeam et al., 2026).

Takeaway 2: Uno outperforms open-weight d-LLMs on every benchmark, while also achieving the highest system throughput.

Comparison with Proprietary Diffusion Language Models As shown in Table 1, the 8B-parameter Uno outperforms Mercury 2 on all Agentic Tool Use, Agentic Coding, and Long-Context Reasoning benchmarks, trailing only on one Science and Knowledge benchmark, potentially due to differences in model size and training data. More importantly, Mercury 2[2] reports a system throughput of 1,154 tokens/s for a 1K-token input at batch size 10. Uno achieves a maximum system throughput ~ 4.6× higher, despite Mercury 2 running on substantially faster Blackwell GPUs. Moreover, Mercury 2 does not disclose its quantization, so it may also benefit from lower precision, whereas Uno and the other baselines use bfloat16. This result highlights the strength of diffusion-augmented LLMs.

[2] https://artificialanalysis.ai/models/mercury-2?speed=output-speed-by-prompt-type#speed-tabs

Table 1: Accuracy and TPF of Uno, Nemotron-Labs-Diffusion, Mercury 2, and DiffusionGemma on agentic and non-agentic benchmarks. Mercury 2 results are from Artificial Analysis. Results for Nemotron-Labs-Diffusion and DiffusionGemma are computed from their open-source checkpoints (Table 5). For Uno, subscripts report "TPF_1 / TPF_2," where TPF_1 uses the system-throughput-optimal Linear sampler with $B = 4$, and TPF_2 uses $(B, K, V) = (16, 32, 32)$, optimized for per-request throughput. *Reported by Artificial Analysis's live tracker on August 30, 2026.

Model size	Uno (8B)	Mercury 2 N/A	Diffusion-Gemma (26B-A4B)	Nemotron-Labs-Diffusion (14B)
Agentic Tool Use				
τ^3 Banking	$\mathbf{25.8}_{1.8/2.7}$	$9_{\text{N/A}}$	–	–
τ^2 Telecom	$\mathbf{90.1}_{1.7/2.1}$	$71_{\text{N/A}}$	$68.1_{18.8}$	$14.3_{4.8}$
τ^2 Retail	$\mathbf{67.1}_{1.8/2.4}$	–	$65.5_{23.7}$	$5.6_{3.1}$
Terminal-Bench v2.1	$\mathbf{39.6}_{2.1/2.7}$	$27_{\text{N/A}}$	$14.7_{14.1}$	$4.5_{7.5}$
Agentic Coding				
SWE-bench Verified	$\mathbf{68.4}_{2.2/3.1}$	–	$18.7_{5.6}$	$0.8_{1.5}$
Long-Context Reasoning				
AA-LCR	$\mathbf{68.0}_{1.8/2.6}$	$36_{\text{N/A}}$	$19.7_{10.8}$	$7.3_{1.1}$
Science and Knowledge				
Humanity's Last Exam	$\mathbf{18.6}_{1.8/2.8}$	$16_{\text{N/A}}$	$9.2_{14.1}$	$2.6_{7.2}$
GPQA-Diamond	$\mathbf{77.1}_{2.0/4.2}$	$77_{\text{N/A}}$	$70.7_{11.9}$	$40.4_{7.6}$
AA-Omniscience	$14.3_{1.7/3.1}$	$\mathbf{20}_{\text{N/A}}$	$17.7_{9.9}$	$11.0_{11.1}$
Math				
GSM8K	$\mathbf{95.4}_{1.9/2.7}$	–	$95.1_{28.9}$	$93.1_{6.1}$
MATH500	$\mathbf{98.9}_{1.9/2.4}$	–	$92.4_{24.1}$	$89.2_{5.6}$
AIME-24	$\mathbf{93.0}_{1.9/2.5}$	–	$73.7_{16.9}$	$56.7_{4.9}$
AIME-25	$\mathbf{90.7}_{1.8/2.4}$	–	$74.3_{18.9}$	$40.0_{4.5}$
AIME-26	$\mathbf{86.3}_{1.8/2.4}$	–	$70.7_{17.8}$	$46.7_{4.8}$
Coding				
MBPP	$\mathbf{84.1}_{1.8/2.4}$	–	$80.1_{15.5}$	$73.8_{5.3}$
HumanEval	$\mathbf{95.2}_{1.9/3.4}$	–	$95.1_{28.2}$	$84.8_{7.5}$
Avg. TPF	1.9 / 2.7	N/A	17.56	5.41
System Throughput	**5255**	1197*	1136	2794
Per-request Throughput	405	769*	**836**	290

Takeaway 3: Uno beats proprietary d-LLM Mercury 2, on all agentic, coding, and long-context benchmarks and achieves ~ 4.6× higher maximum system throughput, despite running on slower hardware.

Faster RL Training From the supervised fine-tuning (SFT) checkpoint, we trained four experts in mathematics, code generation, tool use, and web search using the DAPO reinforcement learning algorithm. During this stage, we updated only the base AR weights; the diffusion weights trained on the SFT checkpoint remained frozen and were used only to speed up RL rollouts. This yielded up to a 40% end-to-end training speedup, primarily for the mathematics and code experts. Gains were smaller for the tool-use and search experts because tool calls dominated their runtime. We will provide detailed results in the next revision. We then consolidated the four experts into a single model using ISO-Merger (RAM; Yuan et al., 2026), a data-free method that combines specialists trained from a shared base checkpoint without additional rollouts. As shown in Table 8, the diffusion adapters trained on the SFT checkpoint retain their speedups after RL post-training, with only a nominal 6% decrease in TPFs.

5.2 Open Weights LLMs

We show that Diffusion-augmented LLMs can initialize their autoregressive (AR) weights from the open-weight Qwen3-8B model while retaining the speedups enabled by diffusion weights trained on a different data distribution, specifically, the open-source OpenThoughts dataset. Although training Qwen3-8B on this dataset degrades its quality (see Table 9), the diffusion weights trained on the same nevertheless enable lossless speedups on the same benchmarks.

5.2.1 Setup

Benchmarks We evaluate on mathematical reasoning (GSM8K, MATH500, AIME-24, AIME-25, and AIME-26), code generation: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b), and LiveCodeBench v6 (Jain et al., 2025)); Science reasoning: GPQA and GPQA-Diamond (Rein et al., 2024); instruction following: IFEval with prompt-level strict accuracy (Zhou et al., 2023), and General Knowledge: MMLU-Pro (Wang et al., 2024).

Lossless Baselines (Speculative Decoding Methods) Our primary lossless baselines are EAGLE-3 (Li et al., 2026b), which uses an AR drafter, and DFlash (Chen et al., 2026), which uses a diffusion drafter. Both draft models are designed for Qwen3, and we evaluate their open-source checkpoints. EAGLE-3 adds a lightweight 0.40B-parameter drafter that sequentially predicts tokens from fused intermediate features of the target model. DFlash instead adds a 1.05B-parameter diffusion drafter that predicts multiple tokens in parallel, conditioned on target-model features. Thus, DFlash introduces roughly three times as many parameters as our method. Its training is also substantially more expensive. For block size B and sequence length L, DFlash requires a training context length of $B \cdot L$, whereas our method always uses $2 \cdot L$, independent of B. **This makes DFlash significantly more expensive to train than our method.** We evaluate the open-source checkpoints for EAGLE-3 and DFlash; see Table 5. Refer Suppl. C.1.2 for sampler configurations.

Lossy Baselines (Diffusion Methods) We also compare against lossy parallel-generation methods. Jacobi Forcing (Hu et al., 2025), SDAR (Cheng et al., 2026), OPDLM (Su et al., 2026), and I-DLM (Yu et al., 2026b) are fine-tuned from Qwen3; Fast-dLLM v2 (Wu et al., 2025) is fine-tuned from Qwen2.5; FLARE (Zhu et al., 2026) is fine-tuned from Qwen3.5; and LLaDA2.1-Flash (Bie et al., 2026) is trained from scratch. We evaluate the open-source checkpoints for Fast-dLLM v2 and SDAR; see Table 5. OPDLM, I-DLM, and FLARE are concurrent works.

Metrics We report average pass@1 accuracy over multiple generations for each benchmark; refer Table 4 for details. For speculative decoding methods, one generation step consists of a draft forward pass followed by a verifier forward pass. We report the number of tokens decoded per step, τ, which comprises one draft and one verify step. However, this metric does not account for differences in drafter size and can therefore be misleading, particularly because the baseline drafters are smaller than ours. We consequently measure throughput using the 1K/8K throughput test described earlier. Following standard practice, we do not report accuracy for lossless methods because any differences arise from sampling randomness and numerical nondeterminism. We report τ under sampler configurations optimized for (1) system throughput and (2) per-request throughput. For each method, we select these configurations through a grid search over sampling hyperparameters; see Table 18. We use thinking mode for all methods. Although DFlash recommends disabling it for greater speedups (Z Lab, 2026), doing so reduces average accuracy from 76.36% to 55.40%; see Suppl. C.7. We therefore retain thinking mode for a fair comparison. When comparing with lossy methods, we instead report tokens per forward pass (TPF). For our method, $\mathrm{TPF} = \tau/2$ because the drafter and verifier are similar in size. For both drafting and verification, we use temperature temp = 1, top-p = 0.95, and top-k = 50 unless otherwise specified. We use a context length of 32,768, the native context length of Qwen3-8B, for all benchmarks and methods.

5.2.2 Uno_{Qwen}: Qwen-based Diffusion Augmented LLM (Ours)

Autoregressive Weights We initialize the AR weights from the open-source Qwen3-8B checkpoint (Yang et al., 2025) and keep them frozen during training.

Training Diffusion Weights For each AR weight matrix, we add a rank-128 LoRA adapter with $\alpha_{\text{LoRA}} = 256$. These adapters introduce 0.35B trainable parameters. We train them for three epochs, corresponding to 14.7B tokens, on OpenThoughts3-1.2M (Guha et al., 2025), using a maximum sequence length of $4{,}096$. We use a block-size curriculum with $B \in \{2, 4, 6, 8, 12, 16\}$, increasing the block size every half epoch. Training uses a global batch size of 64 and a constant learning-rate of 10^{-5} with 2% warmup steps. We set the loss coefficients $\alpha = 0$ and $\beta = 1$. Training takes approximately 32 hours on 4 nodes, each with 8 H200 GPUs. For ablations, we train all models for one epoch with $B = 8$, denoted $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$. Unless stated otherwise, we evaluate this variant using the Linear sampler with $B = 16$.

5.2.3 Results

Comparison with Lossless Speculative Decoding Methods Table 2 compares the average number of tokens per step, τ, for Uno_{Qwen}, EAGLE-3, and DFlash at temp = 1. For each method, we identify the configurations that maximize system and per-request throughput through a grid search over sampling parameters, as detailed in Table 18. At the largest supported batch size, with Linear sampler $B = 4$, all methods achieve their highest system throughput. Uno_{Qwen} exceeds 5700 tokens per second, outperforming DFlash and EAGLE-3 and achieving a $1.6\times$ speedup over the base AR model. This improvement enables faster RL training and offers practical speedups in LLM inference. For the best per-request throughput (batch size of 1), Uno_{Qwen} and EAGLE-3 perform best with tree-sampler configurations $(B, K, V) = (16, 32, 32)$ and $(8, 32, 60)$, respectively, while DFlash performs best with $B = 16$. Uno_{Qwen} substantially outperforms both baselines, achieving a $2.5\times$ speedup over the base AR model. It is also Pareto-dominant in throughput across all batch sizes Fig. 2. Unlike our method, which shares a single KV cache, EAGLE-3 and DFlash maintain separate caches for drafting and verification, resulting in higher peak memory usage.

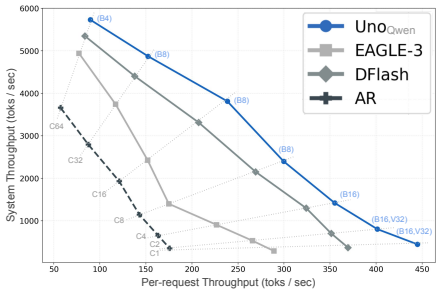

Figure 2: System versus per-request throughput across batch sizes (concurrency C). Uno Pareto-dominates speculative decoding and achieves up to $2.5\times$ speedup over the base AR model. Parentheses indicate the sampler configuration yielding the highest throughput for Uno. Exact throughputs are in Table 18.

Takeaway 4: Uno is strictly faster than DFlash and EAGLE-3 across all batch sizes, with fewer additional parameters and a shared draft-verifier KV cache that reduces peak memory.

Comparison with (Lossy) Diffusion Methods Table 3 compares Uno_{Qwen} with the leading lossy diffusion-based acceleration methods for completeness. Unlike our method Uno_{Qwen}, I-DLM, TiDAR, Jacobi Forcing, FLARE, Fast-dLLM-v2, and SDAR achieve lossy speedups relative to their respective parent models. Despite optimizing for quality first, our method achieves a higher TPF than most methods across numerous benchmarks. Benchmarks highlighted in red indicate accuracy degradation. FLARE supports both lossy and target-lossless sampling, but the paper does not state which produced the results.

5.2.4 Ablation

We ablate the main training choices in Table 12 using $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$ with the Linear sampler and $B = 16$.

Ablation 1: Loss Terms The diffusion loss in (3) combines the distillation loss, $\mathcal{L}_{\text{DCD}}$, and the total variation loss, $\mathcal{L}_{\text{TV}}$. Training with $\mathcal{L}_{\text{TV}}$ alone achieves a TPF of 2.39, outperforming both $\mathcal{L}_{\text{DCD}} + \mathcal{L}_{\text{TV}}$ (2.23) and $\mathcal{L}_{\text{DCD}}$ alone (2.23). Further analysis revealed that the $\mathcal{L}_{\text{DCD}}$ loss was an order of magnitude larger than the $\mathcal{L}_{\text{TV}}$ loss. Consequently, reducing the $\mathcal{L}_{\text{DCD}}$ weight to 0.01 yielded a marginal improvement, increasing the TPF to 2.40.

Ablation 2: Training Curriculum In Suppl. C.6.2, we show that increasing the training block size from 4 to 16 every half epoch improves TPF from 2.65 to 2.71, compared with using a fixed block size of 16 for two epochs.

Ablation 3: Diffusion Weights Configuration In Suppl. C.6.3, we show that distributing diffusion adapters across all layers is more effective than placing the same number of parameters in only a subset of layers. We also ablate the LoRA rank r_{LoRA} and scale α_{LoRA}. Increasing r_{LoRA} from 128 to 256 improves TPF, as shown in Table 12, but also increases inference cost. Across the tested values of $\alpha_{\text{LoRA}}/r_{\text{LoRA}}$, a ratio of 64 performs best.

Table 2: Acceptance lengths (τ), throughput (1K/8K test), peak memory usage, and additional parameter counts for Uno_{Qwen}, EAGLE-3, and DFlash at sampling temp = 1. We report the τ values that maximize system and per-request throughput.

	System Throughput Optimal			Per-request Throughput Optimal		
	Uno_{Qwen} B=4	EAGLE-3 B=4	DFlash B=4	Uno_{Qwen} B=16, V=32	EAGLE-3 B=8, V=60	DFlash B=16
Math						
GSM8K	**3.99**	2.26	2.47	**6.29**	3.83	3.38
MATH500	**4.11**	2.14	2.28	**6.89**	3.52	3.24
AIME-24	**4.08**	2.11	1.96	**6.83**	3.41	2.76
AIME-25	**4.11**	2.13	1.90	**6.81**	3.46	2.56
AIME-26	**4.08**	2.13	1.93	**6.71**	3.49	2.57
Coding						
HumanEval	**3.83**	2.12	2.35	**5.63**	3.71	3.03
MBPP	**3.87**	2.16	2.22	**5.87**	3.65	3.04
LCBv6	**3.77**	2.04	1.76	**5.34**	3.45	2.22
Science and Knowledge						
GPQA	**3.79**	1.99	1.98	**5.49**	3.25	2.57
GPQA-Diamond	**3.78**	1.98	1.92	**5.49**	3.19	2.50
MMLU-Pro	**3.84**	2.03	2.11	**5.70**	3.31	2.80
Instruction Following						
IFEval	**3.48**	1.91	1.92	**4.58**	3.49	2.26
τ	**3.89**	2.08	2.07	**5.97**	3.48	2.74
Throughput (Toks / sec; ↑)	**5733**	4944	5351	**445**	284	370
Peak Memory (GiB; ↓)	**122.2**	130.0	130.1	**118.0**	129.4	129.8
Additional Params (B; ↓)	**0.35**	0.40	1.05	**0.35**	0.40	1.05

6 Related Work

Speculative Decoding Speculative decoding methods such as EAGLE-3 and DFlash also provide lossless speedups but require training a separate draft model that is smaller than the target model. Designing a draft model requires numerous choices about its depth, hidden dimension, MLP expansion, attention heads, parameter sharing, and KV projections for target features. Li et al. (2026a); Sandler et al. (2026) on the other hand use large pretrained dLLMs for drafting and separate AR models for verification. These large drafters incur substantial memory and latency overhead, making these methods slower than DFlash. In contrast, Uno uses a single architecture with distinct drafting and verification pathways. Consequently, Uno maintains a single KV cache, introduces fewer additional parameters, and requires less peak inference memory than conventional speculative decoding. These advantages yield larger speedups at high batch sizes, making Uno a drop-in replacement for methods that use separate draft and target models.

Self-Speculative Decoding Sahoo et al. (2025b) introduced a hybrid AR-diffusion framework in which a single denoising model performs either multi-token diffusion generation or AR generation, depending on the available compute budget. This approach substantially outperforms block diffusion (Arriola et al., 2025). Building on self-speculative decoding (Stern et al., 2018), Liu et al. (2025) extended this framework by using the same denoising model to draft sequences in diffusion mode and verify them via rejection sampling in AR mode, yielding significant qualitative improvements over standalone diffusion generation. However, because this method modifies the base AR model's weights to support diffusion generation, it does not preserve the model's original distribution. Moreover, its speedups are limited to small batch sizes (Fu et al., 2026). In contrast, Uno preserves the base AR model's distribution while providing lossless speedups, making it a drop-in alternative to these methods.

Diffusion Models Large-scale d-LLMs (Nie et al., 2025; Gat et al., 2025; Bie et al., 2026; Fu et al., 2026; DiffusionGemmaTeam et al., 2026) can generate faster than similarly sized AR models

Table 3: Benchmark accuracy (ACC) and TPF for our lossless method, Uno_{Qwen}, and lossy diffusion methods. Entries are reported as ACC_{TPF}. For Uno_{Qwen}, TPF uses temp = 0 and the tree sampler with $(B, K, V) = (16, 32, 60)$. Lossy speedups are taken from the respective papers, except for Fast-dLLM v2 and SDAR, which we evaluated using the provided checkpoints. Accuracy drops relative to the corresponding parent AR model are shaded in red by severity. Jacobi Forcing[J] uses separate math and coding models trained from Qwen2.5-Math-7B-Instruct and Qwen2.5-Coder-7B-Instruct, respectively. [F] Fast-dLLM v2 uses Qwen2.5-7B-Instruct.

		Lossy Speedup Methods							
Benchmark	Uno_{Qwen}	SDAR	TiDAR (Trust Diff)	OPDLM	I-DLM	Jacobi Forcing (MR)[J]	Fast-dLLM v2[F]	FLARE (AR Trust)[L]	LLaDA2.1-Flash (S Mode)
Model size	8B	8B	8B	8B	8B	7B	7B	9B	100B-A5B
Parent AR		Qwen3-8B				Qwen2.5-7B Family		Qwen3.5-9B	N/A
Math									
GSM8K	$96.1_{3.56}$	$91.4_{2.3}$	$80.4_{7.1}$	87.1_{1}	$95.0_{\text{N/A}}$	$91.4_{4.0}$	$83.7_{3.0}$	$93.3_{\text{N/A}}$	–
MATH500	$96.4_{3.92}$	$72.0_{2.8}$	–	71.2_{1}	$96.8_{\text{N/A}}$	–	$61.1_{2.1}$	$95.2_{\text{N/A}}$	–
AIME-24	$76.7_{4.01}$	$10.0_{2.8}$	–	14.7_{1}	$69.6_{\text{N/A}}$	–	$6.67_{2.7}$	$63.3_{\text{N/A}}$	–
AIME-25	$76.7_{3.96}$	$10.0_{2.6}$	–	12.4_{1}	$60.8_{\text{N/A}}$	–	$0.0_{2.6}$	$54.4_{\text{N/A}}$	$63.3_{5.4}$
AIME-26	$73.3_{4.05}$	–	–	–	–	–	–	–	–
Coding									
HumanEval	$94.8_{3.67}$	$75.6_{2.8}$	$57.9_{7.3}$	59.8_{1}	$93.3_{2.6}$	$83.5_{4.1}$	$63.4_{2.5}$	$92.1_{\text{N/A}}$	–
MBPP	$89.0_{4.21}$	$68.1_{1.5}$	$65.4_{10.0}$	48.7_{1}	$92.2_{\text{N/A}}$	$70.4_{2.8}$	$63.0_{4.5}$	$91.1_{\text{N/A}}$	–
LCBv6	$51.4_{3.75}$	16.6	–	9.7_{1}	$45.7_{\text{N/A}}$	–	$10.0_{1.7}$	$49.7_{\text{N/A}}$	$44.1_{6.5}$
Science and Knowledge									
GPQA	$58.0_{4.10}$	–	–	–	$54.9_{\text{N/A}}$	–	$31.9_{\text{N/A}}$	–	–
GPQA-D	$60.9_{4.01}$	40.2	–	36.1_{1}	$55.6_{\text{N/A}}$	–	$21.7_{2.2}$	$71.2_{\text{N/A}}$	$66.7_{4.0}$
MMLU-Pro	$74.8_{3.61}$	$56.9_{1.6}$	–	53.7_{1}	$73.1_{\text{N/A}}$	–	–	$77.4_{\text{N/A}}$	$75.3_{4.4}$
Instruction Following									
IFEval	$86.5_{2.49}$	$61.4_{1.5}$	–	50.1_{1}	$84.7_{\text{N/A}}$	–	$61.4_{1.5}$	$71.4_{\text{N/A}}$	$83.4_{2.2}$

at small batch sizes but generally lag behind them in quality. Their lossy speedups also diminish at larger batch sizes, limiting their applicability. Post-training these models is more expensive because rollouts become slower at large batch sizes and existing RL recipes and objectives require nontrivial modifications (Zhao et al., 2026; Wang et al., 2025). In contrast, Uno provides lossless speedups across batch sizes, uses standard AR post-training algorithms without modification, and substantially accelerates post-training.

Multi-Token Prediction (MTP) Methods such as Medusa (Cai et al., 2024) and the approach of Gloeckle et al. (2024) provide lossless speedups but modify the transformer architecture to predict future tokens. Li et al. (2026b) showed that speculative decoding with a separately trained draft model is faster than these approaches. We further show that Uno outperforms speculative decoding methods and, consequently, these MTP methods. Nevertheless, architectural changes for predicting future tokens are complementary to Uno and could improve its draft acceptance rate. We leave this combination to future work.

Quadratic Samplers Each generation step in our sampler uses two forward passes: one for drafting and one for verification. Quadratic sampling, proposed by Samragh et al. (2025) and used in TiDAR and Nemotron-Labs-Diffusion, can combine these operations into a single pass. For a draft of k tokens, it inserts k masked placeholders after each draft position, yielding k^2 masked positions that represent possible future continuations. This structure verifies the current draft while generating candidates for the next step. Practical speedups require efficient kernels, which we leave to future work.

Comparison with Concurrent work Concurrent methods such as OPDLM, FLARE, and I-DLM provide lossy speedups relative to the base AR model because they fine-tune its weights; see Table 3. Although I-DLM is lossy, its I-DLM (R-ISD) variant uses gated LoRA. Despite the authors' theoretical argument that its sampler is lossless, we could not reproduce lossless speedups using the released LoRA adapters and sampler implementation. Because this method is compatible with our proposed Ψ-spec sampler using a mask prior, we evaluate the combination and show that it achieves lossless speedups, although it remains substantially slower than our method. We provide a detailed comparison with I-DLM, including an evaluation of its official LoRA checkpoint, in Suppl. C.5.

7 Conclusion

In this work, we introduced diffusion-augmented LLMs, which unify two distinct generation pathways within a single architecture: an autoregressive pathway parameterized by θ_{AR} and a diffusion pathway parameterized by $\theta_{\text{AR}}+\theta_{\Delta}$. We showed that training the diffusion weights requires orders of magnitude fewer tokens than the AR weights. This formulation allows Uno either to be trained from scratch or to be created by augmenting an existing AR model with lightweight diffusion weights for faster generation.

Uno serves as a drop-in alternative to speculative diffusion methods such as EAGLE-3 and DFlash, without requiring a separate draft model, and to self-speculative methods such as TiDAR, without introducing lossy speedups. Across the evaluated batch sizes, Uno achieves up to a 3× speedup over the base AR model, while retaining up to a 2× speedup at the largest batch size supported by that model. These gains accelerate both inference and RL post-training, where rollout generation is a major computational bottleneck. More broadly, diffusion-augmented LLMs expand the design space of AR language models by combining the quality of AR generation with the parallelism of diffusion in a unified architecture.

Acknowledgement We thank A. Feder Cooper, Willis Guo, Rupesh Srivastava, and Matthew Yang for their helpful feedback and insightful discussions.

References

Amazon Web Services. AGENTPERF01-BP01: Define performance-aligned success criteria for agent workloads. AWS Well-Architected Framework, Agentic AI Lens, 2026. URL https://docs.aws.amazon.com/wellarchitected/latest/agentic-ai-lens/agentperf01-bp01.html.

Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

Artificial Analysis. Announcing artificial analysis long context reasoning (AA-LCR), 2025. URL https://artificialanalysis.ai/articles/announcing-aa-lcr.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models, 2021b. URL https://arxiv.org/abs/2108.07732.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating llms on uncontaminated math competitions. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.23281.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Douglas Biber. A corpus-driven approach to formulaic language in English: Multi-word patterns in speech and writing. *International Journal of Corpus Linguistics*, 14(3):275–311, 2009. doi: 10.1075/ijcl.14.3.08bib. URL https://doi.org/10.1075/ijcl.14.3.08bib.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei,

Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. Llada2.1: Speeding up text diffusion via token editing, 2026. URL https://arxiv.org/abs/2602.08676.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=PEpbUobfJv.

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, 2026. URL https://arxiv.org/abs/2602.06036.

Mark Chen et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, 2026.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of machine learning research*, 25(70):1–53, 2024.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990. URL https://aclanthology.org/J90-1003/.

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. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvalddson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with llms, 2026. URL https://arxiv.org/abs/2605.00674.

Justin Deschenaux, Caglar Gulcehre, and Subham Sekhar Sahoo. The diffusion duality, chapter II: \psi-samplers and efficient curriculum. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RSIoYWIzaP.

DiffusionGemmaTeam, Adrien Ali TaÃŕga, James Assiene, Daniele Calandriello, Rahma Chaabouni, JoÃčo Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, Tianlin Liu, Ivan Lobov, Ofir Nabati, JoÃčo Gabriel Oliveira, Nicolas Perez-Nieves, Nastasia Prutianova, Bobak Shahriari, Jean Tarbouriech, Pavel Tyletski, ÃĞaÄ§lar ÃIJnlÃij, Cindy Wu, Glenn Cameron, Jerome Connor, Sertan Girgin, Maarten Grootendorst, Alon Levkovitch, Eliya Nachmani, Omar Sanseviero, Piotr Stanczyk, Quentin Berthet, Andrew Campbell, ClÃl'ment Crepy, Valentin De Bortoli, Arnaud Doucet, Romuald Elie, Alexandre Galashov, Klaus Greff, Alexis Jacq, David Ruhe, Yu-Han Wu, Sebastian Flennerhag, Brendan O'Donoghue, George Scrivener, and Shantanu Thakoor. Diffusiongemma technical report, 2026. URL https://arxiv.org/abs/2608.00146.

Stefano Ermon. Introducing Mercury 2. Inception, 2026. URL https://www.inceptionlabs.ai/blog/introducing-mercury-2.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukic, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. Technical report, NVIDIA, 2026. Technical report.

Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set block decoding is a language model inference accelerator. *arXiv preprint arXiv:2509.04185*, 2025.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Google Cloud. The roi of ai 2025: Measuring the impact of ai and ai agents. Technical report, Google Cloud, 2025. URL https://cloud.google.com/resources/content/roi-of-ai-2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan

Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

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.

Declan Jackson, William Keating, George Cameron, and Micah Hill-Smith. AA-Omniscience: Evaluating cross-domain knowledge reliability in large language models, 2025. URL https://arxiv.org/abs/2511.13029.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Minseo Kim, Minjae Lee, Seunghyuk Oh, Kevin Galim, Donghoon Kim, Coleman Hooper, Harman Singh, Amir Gholami, Hyung Il Koo, and Wonjun Kang. Efficientrollout: System-aware self-speculative decoding for rl rollouts. *arXiv preprint arXiv:2606.18967*, 2026.

Kimi KimiTeam, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Seul Lee, Karsten Kreis, Srimukh Prasad Veccham, Meng Liu, Danny Reidenbach, Yuxing Peng, Saee Paliwal, Weili Nie, and Arash Vahdat. Genmol: A drug discovery generalist with discrete diffusion. *arXiv preprint arXiv:2501.06158*, 2025.

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

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2026b.

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. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

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

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2024.

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

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sMyXP8Tanm.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Subham S. Sahoo. *Foundations of Diffusion Language Models*. PhD thesis, Cornell University, 2026. URL https://www.proquest.com/dissertations-theses/foundations-diffusion-language-models/docview/3355013570/se-2. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2026-06-23.

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 *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=L4uaAR4ArM.

Subham Sekhar Sahoo, Aaron Gokaslan, Christopher De Sa, and Volodymyr Kuleshov. Diffusion models with learned adaptive noise. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=loMa99A4p8.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin T Chiu, and Volodymyr Kuleshov. The diffusion duality. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=9P9Y8FOSOk.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric Xing, John Thickstun, and Arash Vahdat. Esoteric language models: A family of any-order diffusion llms. *arXiv preprint arXiv:2506.01928*, 2025b.

Subham Sekhar Sahoo, Jean-Marie Lemercier, Zhihan Yang, Justin Deschenaux, Jingyu Liu, John Thickstun, and Ante Jukic. Scaling beyond masked diffusion language models. *arXiv preprint arXiv:2602.15014*, 2026.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025.

Jameson Sandler, Jacob Christopher, Tom Hartvigsen, and Ferdinando Fioretto. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding. *Proceedings of Machine Learning and Systems*, 8:1128–1147, 2026.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-torre, Bernardo P de Almeida, Alexander M Rush, Thomas PIERROT, and Volodymyr Kuleshov. Simple guidance mechanisms for discrete diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=i5MrJ6g5G1.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barres. τ-knowledge: Evaluating conversational agents over unstructured knowledge. *arXiv preprint arXiv:2603.04370*, 2026.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32211–32252. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/song23a.html.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Neural Information Processing Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:53208380.

Xingyu Su, Jacob Helwig, Shubham Parashar, Atharv Chagi, Lakshmi Jotsna, Degui Zhi, James Caverlee, Dileep Kalathil, and Shuiwang Ji. Data-efficient autoregressive-to-diffusion language models via on-policy distillation. *arXiv preprint arXiv:2606.06712*, 2026.

Sophia Tang, Yinuo Zhang, and Pranam Chatterjee. Peptune: De novo generation of therapeutic peptides with multi-objective-guided discrete diffusion. *ArXiv*, pp. arXiv–2412, 2025.

The Terminal-Bench Team. Terminal-bench 2.1, may 2026. URL https://www.tbench.ai/news/terminal-bench-2-1.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Shashank Verma and Neal Vaidya. Mastering LLM techniques: Inference optimization. NVIDIA Technical Blog, November 2023. URL https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/. Accessed 2026-08-24.

Guanghan Wang, Gilad Turok, Yair Schiff, Marianne Arriola, and Volodymyr Kuleshov. d2: Improving reasoning in diffusion language models via trajectory likelihood estimation. *arXiv preprint arXiv:2509.21474*, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.01574.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm, 2025. URL https://arxiv.org/abs/2509.26328.

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, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026a.

Xingkai Yu. Nano-vLLM: A lightweight vLLM implementation built from scratch. https://github.com/GeeeekExplorer/nano-vllm, 2025. Software repository.

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, et al. Introspective diffusion language models. *arXiv preprint arXiv:2604.11035*, 2026b.

Xiangchi Yuan, Dachuan Shi, Chunhui Zhang, Zheyuan Liu, Shenglong Yao, Soroush Vosoughi, and Wenke Lee. Behavior knowledge merge in reinforced agentic models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33007–33028, 2026.

Z Lab. Qwen3-8B-DFlash-b16 model card. https://huggingface.co/z-lab/Qwen3-8B-DFlash-b16, 2026. Accessed 2026-08-03.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:56729–56762, 2026.

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. SGLang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

Yuchen Zhu, Jing Shi, Chongjian Ge, Hao Tan, Yiran Xu, Wanrong Zhu, Jason Kuen, Koustava Goswami, Rajiv Jain, Yongxin Chen, et al. Flare: Diffusion for hybrid language model. *arXiv preprint arXiv:2606.01774*, 2026.

Contents

A Background

A.1 Masked Diffusion

MDMs use a masked prior, where $\boldsymbol{\pi} = \mathbf{m} \in \mathcal{V}$ is the one-hot representation of a special [MASK] token. During the forward process (1), tokens either remain unchanged or transition to the masked state $\mathbf{m}$, after which they stay masked. This behavior carries over to the reverse process. The posterior of the reverse process $q^{\text{MDM}}_{s|t}$ for $0 \le s < t < 1$ can be derived using Bayes' Rule, and is given by:

$$[q^{\ell}_{s|t}(\cdot \mid \mathbf{z}_t, \mathbf{x})]^{\text{MDM}} = \begin{cases} \text{Cat}\left(\cdot\,;\, \frac{\alpha_s - \alpha_t}{1-\alpha_t}\mathbf{x}^{\ell} + \frac{1-\alpha_s}{1-\alpha_t}\mathbf{z}^{\ell}_t\right) & \text{if } \mathbf{z}^{\ell}_t = \mathbf{m}, \\ \text{Cat}(\cdot\,;\, \mathbf{x}^{\ell}) & \text{otherwise.} \end{cases} \tag{6}$$

The learned reverse posterior is

$$[p^{\theta}_{s|t}(\mathbf{z}_s \mid \mathbf{z}_t)]^{\text{MDM}} = q^{\text{MDM}}_{s|t}(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x} = \mathbf{x}_\theta(\mathbf{z}_t, t)),$$

where $\mathbf{x}_\theta : \mathcal{V}^L \times [0,1] \to \Delta^L$ is the denoising model. A key limitation is that once unmasked, tokens cannot be remasked. This can create compounding errors during inference, as the denoising model $\mathbf{x}_\theta$ imperfectly models the clean data.

NELBO The likelihood of a sequence under the learned reverse diffusion process is generally intractable to evaluate exactly, as it requires marginalizing over all possible unmasking trajectories. Therefore, typically the negative evidence lower bound (NELBO) is minimized, which is a tractable upper bound on the negative log-likelihood. For continuous-time masked diffusion, the NELBO simplifies to the weighted denoising objective (Sahoo et al., 2024a; Shi et al., 2024; Ou et al., 2025)

$$\mathcal{L}^{\text{MDM}}_{\text{NELBO}} = \mathbb{E}_{t\sim\mathcal{U}[0,1],q_t} \frac{\alpha'_t}{1-\alpha_t} \log\langle \mathbf{x}^{\ell}_\theta(\mathbf{z}_t, t), \mathbf{x}^{\ell}\rangle.$$

A.2 Uniform-State Diffusion Models

We now focus on Uniform-State Diffusion Models (USDMs), which use a uniform prior $\boldsymbol{\pi} = \mathbf{1}/K$ (Austin et al., 2021a; Sahoo et al., 2025a). Unlike masked diffusion, where tokens are fixed once unmasked, USDMs allow continuous updates of all token positions, which naturally supports self-correction at inference. This capability makes them particularly effective for few-step generation (Sahoo et al., 2025a), inference-time scaling (Deschenaux et al., 2026) and guided sampling (Schiff et al., 2025). Given a noisy sequence $\mathbf{z}_t$ at time t, the Markov property defines the reverse posterior $q_{s|t}$ for obtaining a less noisy sequence $\mathbf{z}_s$ $(0 \le s < t)$ by sampling from the following distribution:

$$[q^{\ell}_{s|t}(\cdot \mid \mathbf{z}_t, \mathbf{x})]^{\text{USDM}} = \text{Cat}\left(\cdot\,;\, \frac{K\alpha_t \mathbf{z}^{\ell}_t \odot \mathbf{x}^{\ell} + (\alpha_{t|s} - \alpha_t)\mathbf{z}^{\ell}_t + (\alpha_s - \alpha_t)\mathbf{x}^{\ell} + (1-\alpha_{t|s})(1-\alpha_s)\mathbf{1}/K}{K\alpha_t\langle \mathbf{z}^{\ell}_t, \mathbf{x}^{\ell}\rangle + 1 - \alpha_t}\right). \tag{7}$$

Similar to the posterior for masked diffusion models above, the true reverse posterior (7) depends on the unknown clean sequence $\mathbf{x}$, hence it is approximated as:

$$[p^{\theta}_{s|t}(\mathbf{z}_s \mid \mathbf{z}_t)]^{\text{USDM}} = q^{\text{USDM}}_{s|t}(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x} = \mathbf{x}_\theta(\mathbf{z}_t, t)),$$

where $\mathbf{x}_\theta : \mathcal{V}^L \times [0,1] \to \Delta^L$ is the denoising network parameterized by θ.

NELBO This posterior induces the following NELBO (Sahoo et al., 2025a):

$$\mathcal{L}_{\text{NELBO}}^{\text{USDM}}(q, p_\theta; \mathbf{x}) = -\mathbb{E}_{t\sim\mathcal{U}[0,1],q_t} \sum_{\ell\in[L]} f(\mathbf{z}_t^\ell, \mathbf{x}_\theta^\ell(\mathbf{z}_t^\ell, t), \alpha_t; \mathbf{x}^\ell),$$

where

$$f(\mathbf{z}_t^\ell, \mathbf{x}_\theta^\ell(\mathbf{z}_t^\ell, t), \alpha_t; \mathbf{x}^\ell) = \frac{\alpha_t'}{K\alpha_t}\Bigg[\frac{K}{\bar{\mathbf{x}}_r^\ell} - \frac{K}{(\bar{\mathbf{x}}_\theta^\ell)_r} - \left(\zeta_t \mathbb{1}_{\mathbf{z}_t^\ell=\mathbf{x}^\ell} + \mathbb{1}_{\mathbf{z}_t^\ell\neq\mathbf{x}^\ell}\right)\sum_j \log\frac{(\bar{\mathbf{x}}_\theta^\ell)_r}{(\bar{\mathbf{x}}_\theta^\ell)_j}$$
$$- K\frac{\alpha_t}{1-\alpha_t}\log\frac{(\bar{\mathbf{x}}_\theta^\ell)_r}{(\bar{\mathbf{x}}_\theta^\ell)_i}\mathbb{1}_{\mathbf{z}_t^\ell\neq\mathbf{x}^\ell} - \left((K-1)\zeta_t\mathbb{1}_{\mathbf{z}_t^\ell=\mathbf{x}^\ell} - \frac{1}{\zeta_t}\mathbb{1}_{\mathbf{z}_t^\ell\neq\mathbf{x}^\ell}\right)\log\zeta_t\Bigg].$$

Here, $\bar{\mathbf{x}}^\ell = K\alpha_t\mathbf{x}^\ell + (1-\alpha_t)\mathbf{1}$, $\bar{\mathbf{x}}_\theta^\ell = K\alpha_t\mathbf{x}_\theta^\ell(\mathbf{z}_t, t) + (1-\alpha_t)\mathbf{1}$, α_t' denotes the time derivative of α_t, $r = \arg\max_{j\in[K]}(\mathbf{z}_t^\ell)_j$ is the nonzero entry of $\mathbf{z}_t$, $\zeta_t = \frac{1-\alpha_t}{K\alpha_t+1-\alpha_t}$, and i denotes the index in $\mathbf{x}$ corresponding to 1, that is, $\mathbf{x}_i = 1$.

A.3 Self-speculative Decoding

Self-speculative decoding eliminates the need for a separate draft model by using a single network for both proposal and verification. Stern et al. (2018) first proposed augmenting the base model with auxiliary prediction heads to predict a block of future tokens in parallel, then using the same model to verify these predictions and commit the longest validated prefix. Their exact-match verification rule guarantees equivalence to greedy decoding, rather than exact sampling. Liu et al. (2025) instead convert an AR model into a diffusion model: the model drafts tokens in diffusion mode and verifies them in AR mode, with the attention pattern changing between the two modes while the model weights remain the same in both modes. They combine this approach with the rejection-sampling acceptance rule of Leviathan et al. (2023) and Chen et al. (2023) to recover exact sampling from the base model's distribution. Concurrent work (Zhu et al., 2026; Yu et al., 2026b) builds on the same idea. **Crucially, however, this approach requires modifying the base AR model's weights, so it is not lossless.**

A.4 Ψ-Samplers

Ψ-Samplers (Deschenaux et al., 2026) are predictor-corrector samplers for discrete diffusion models that share the marginals of the ancestral sampler. They interpolate between the reverse posterior and the forward process (1):

$$\Psi_{s|t}(\cdot \mid \mathbf{x}, \mathbf{z}_t; \boldsymbol{\pi}) = \kappa_t q_{s|t}(\cdot \mid \mathbf{z}_t, \mathbf{x}; \boldsymbol{\pi}) + (1-\kappa_t) q_s(\cdot \mid \mathbf{x}; \boldsymbol{\pi}),$$

where $\boldsymbol{\pi}$ is the noise prior (e.g., the masked prior $\boldsymbol{\pi} = \mathbf{m}$ for MDMs or the uniform prior $\boldsymbol{\pi} = \mathbf{1}/K$ for USDMs), $q_{s|t}(\cdot \mid \cdot\,;\, \boldsymbol{\pi})$ is the reverse posterior for MDMs (6) or USDMs (7), q_s is the forward noising process, and $\kappa_t \in [0,1]$ sets the correction strength. The first term *predicts* by denoising the current state; the second *corrects* by re-injecting noise, so earlier decisions can be revised. Setting $\kappa_t = 1$ recovers ancestral sampling.

Since $\mathbf{x}$ is unavailable at generation time, the practical sampler substitutes the denoiser prediction $\mathbf{x}_\theta(\mathbf{z}_t, t)$, giving the token-wise transition

$$\begin{aligned}&\left[\Psi_{s|t}^\theta(\cdot \mid \mathbf{z}_t; \mathbf{x}_\theta(\mathbf{z}_t, t), \boldsymbol{\pi})\right]^\ell \\ &= \kappa_t q_{s|t}^\ell(\cdot \mid \mathbf{z}_t, \mathbf{x}_\theta(\mathbf{z}_t, t); \boldsymbol{\pi}) + (1-\kappa_t)\left[\alpha_s q_{0|t}^\ell(\cdot \mid \mathbf{z}_t, \mathbf{x}_\theta(\mathbf{z}_t, t); \boldsymbol{\pi}) + (1-\alpha_s)\boldsymbol{\pi}\right].\end{aligned}$$

Ancestral samplers can lock in incorrect early predictions, whereas Ψ-samplers correct errors explicitly: for MDMs the corrector can remask decoded tokens, and for USDMs it assigns every token non-zero probability, permitting transitions into states that the denoiser incorrectly scores as low-probability. Sample quality therefore keeps improving with additional steps.

For MDMs, the transition (2) simplifies to

$$\begin{aligned}&\left[\Psi_{s|t}^\theta(\cdot \mid \mathbf{z}_t; \mathbf{x}_\theta(\mathbf{z}_t, t), \boldsymbol{\pi} = \mathbf{m})\right]^\ell \\ &= \begin{cases} \mathrm{Cat}\Big(\cdot; [\alpha_s + \kappa_t(1-\alpha_s)]\mathbf{x}_\theta^\ell(\mathbf{z}_t, t) + (1-\kappa_t)(1-\alpha_s)\mathbf{m}\Big), & \mathbf{z}_t^\ell \neq \mathbf{m}, \\ \mathrm{Cat}\Big(\cdot; \Big[\alpha_s - \frac{\kappa_t\alpha_t(1-\alpha_s)}{1-\alpha_t}\Big]\mathbf{x}_\theta^\ell(\mathbf{z}_t, t) + \Big[1-\alpha_s + \frac{\kappa_t\alpha_t(1-\alpha_s)}{1-\alpha_t}\Big]\mathbf{m}\Big), & \mathbf{z}_t^\ell = \mathbf{m}. \end{cases}\end{aligned} \tag{8}$$

To obtain the marginals for single-step generation, we simply plug in $s = 0$ and $t = 1$ into (8) which gives:

$$
\begin{aligned}
&[\Psi^{\theta}_{s=0|t=1}(\cdot \mid \mathbf{z}_t; \mathbf{x}_\theta(\mathbf{z}_t, t), \boldsymbol{\pi} = \mathbf{m})]^\ell \\
&= \begin{cases} \mathrm{Cat}\Big(\cdot; \left[1 + \kappa_t(1-1)\right] \mathbf{x}^\ell_\theta(\mathbf{z}_1, 1) + (1-\kappa_t)(1-1)\mathbf{m}\Big), & \mathbf{z}^\ell_1 \neq \mathbf{m}, \\ \mathrm{Cat}\Big(\cdot; \left[\alpha_s - \frac{\kappa_t \cdot 0 \cdot (1-1)}{1-0}\right] \mathbf{x}^\ell_\theta(\mathbf{z}_1, 1) + \left[1 - 1 + \frac{\kappa_t \cdot 0 \cdot (1-1)}{1-0}\right] \mathbf{m}\Big), & \mathbf{z}^\ell_1 = \mathbf{m}. \end{cases} \\
&= \begin{cases} \mathrm{Cat}(\cdot; \mathbf{x}^\ell_\theta(\mathbf{z}_1)), & \mathbf{z}^\ell_1 \neq \mathbf{m}, \\ \mathrm{Cat}(\cdot; \mathbf{x}^\ell_\theta(\mathbf{z}_1)), & \mathbf{z}^\ell_1 = \mathbf{m}. \end{cases} \\
&= \mathrm{Cat}(\cdot; \mathbf{x}^\ell_\theta(\mathbf{z}_1))
\end{aligned} \tag{9}
$$

For USDM, the transition (2) simplifies to:

$$
\begin{aligned}
&[\Psi^{\theta}_{s|t}(\cdot \mid \mathbf{z}_t; \mathbf{x}_\theta(\mathbf{z}_t, t), \boldsymbol{\pi} = \mathbf{1}/K)]^\ell \\
&= \mathrm{Cat}\Bigg(\cdot; \frac{1}{K\alpha_t \left\langle \mathbf{z}^\ell_t, [\mathbf{x}_\theta(\mathbf{z}_t, t)]^\ell \right\rangle + 1 - \alpha_t} \Bigg[K\alpha_t \left[\alpha_s + \kappa_t(1-\alpha_s)\right] \mathbf{z}^\ell_t \odot [\mathbf{x}_\theta(\mathbf{z}_t, t)]^\ell \\
&\qquad + \kappa_t(1-\alpha_s)\alpha_{t|s}\mathbf{z}^\ell_t + \left[\alpha_s(1-\alpha_t) - \kappa_t\alpha_t(1-\alpha_s)\right] [\mathbf{x}_\theta(\mathbf{z}_t, t)]^\ell \\
&\qquad + (1-\alpha_s)\left[\kappa_t(1-\alpha_{t|s}) + (1-\kappa_t)\left(K\alpha_t \left\langle \mathbf{z}^\ell_t, [\mathbf{x}_\theta(\mathbf{z}_t, t)]^\ell \right\rangle + 1 - \alpha_t\right)\right] \frac{\mathbf{1}}{K} \Bigg]\Bigg).
\end{aligned}
$$

To obtain the marginals for single-step generation, we simply plug in $s = 0$ and $t = 1$ into (7) which gives

$$
[\Psi^{\theta}_{s=0|t=1}(\cdot \mid \mathbf{z}_t; \mathbf{x}_\theta(\mathbf{z}_t), \boldsymbol{\pi} = \mathbf{1}/K)]^\ell = \mathrm{Cat}(\cdot; \mathbf{x}^\ell_\theta(\mathbf{z}_1, 1)). \tag{10}
$$

A.5 Discrete Consistency Distillation

Sahoo et al. (2025a) show that Uniform-state discrete diffusion emerges from an underlying Gaussian diffusion process (Sohl-Dickstein et al., 2015; Song et al., 2020; Kingma et al., 2021) defined on the one-hot representation $\mathbf{x}^\ell \in \mathcal{V}$: applying an $\arg\max$ projection to the Gaussian latents recovers the discrete process. This correspondence lets us transfer techniques from the continuous to the discrete domain, most notably distillation for few-step generation. Distillation requires deterministic trajectories, namely the probability-flow ODE, which exist for Gaussian but not for discrete diffusion. Since discrete latents are simply $\arg\max$ projections of Gaussian ones, Sahoo et al. (2025a) construct the trajectory in continuous space and map it back to the discrete domain.

Discrete Consistency Distillation (DCD) trains few-step generators on these trajectories. First, the Gaussian probability-flow ODE is constructed on the one-hot vectors and the resulting continuous states are mapped back to tokens via $\arg\max$, yielding a deterministic sequence of discrete latents. The teacher $\mathbf{x}_{\boldsymbol{\theta}_0} : \mathcal{V}^L \times [0,1] \to \Delta^L$ is a neural network with parameters θ_0 that are held fixed, while the student $\mathbf{x}_{\boldsymbol{\theta}} : \mathcal{V}^L \times [0,1] \to \Delta^L$ is a neural network whose parameters θ are trained. Given two adjacent states along this path, a noisier $\mathbf{z}_t$ at time t and a less noisy $\mathbf{z}_s$ at time $s = t - T$, the student takes the noisier state as input and is trained to match the teacher's clean-token distribution at the less noisy one, in the spirit of consistency distillation (Song et al., 2023; Sahoo et al., 2025a):

$$
\mathcal{L}_{\mathrm{DCD}}(\boldsymbol{\theta}; \boldsymbol{\theta}_0) = \sum_{\ell=1}^{L} D_{\mathrm{KL}}\Big(\mathbf{x}^\ell_{\boldsymbol{\theta}}(\mathbf{z}_t^{1:L}, t) \,\Big\|\, \mathbf{x}^\ell_{\boldsymbol{\theta}_0}(\mathbf{z}_t^{1:L},\ s)\Big),
$$

Note that the teacher's parameters $\boldsymbol{\theta}_0$ are not updated during training. Distillation proceeds over numerous rounds, with the step size Δ increased at every round; at the end of each round the trained student parameters θ are copied into the teacher, which then serves as the target for the next round. By taking progressively larger denoising steps while remaining consistent along a shared trajectory, DCD compresses a many-step USDM sampler into a few-step one, making Uniform-state diffusion particularly effective in the low-step regime.

Limitation: DCD is trained on deterministic probability-flow trajectories, which differ from the stochastic denoising trajectories followed at sampling time. This train-test mismatch limits its effectiveness.

B Ψ-Spec samplers

B.1 Diffusion Proposal Distribution Extended

Given a noisy block $\mathbf{z}_t$, we use the Ψ-Spec transition in (2) to sample a less noisy block $\mathbf{z}_s$, where $s < t$. Because the denoiser uses next-token-prediction (NTP) parameterization, the clean-token distributions used to construct $\mathbf{z}_s$ are given by $\mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}^{1:B-1}([\mathbf{x}^L, \mathbf{z}_t])$. The resulting transition is obtained by drawing each token in parallel from its corresponding marginal distribution:

$$\mathbf{z}_s^\ell \sim \begin{cases} \Psi_{s|t}^\ell\left(\cdot \mid \mathbf{z}_t; \mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}^{1:B-1}(\cdot), \pi\right), & \ell > 1 \quad (11) \\ \Psi_{s|t}^{\ell=1}\left(\cdot \mid \mathbf{z}_t; \mathbf{x}_{\theta_{\text{AR}}}^{1:B-1}(\cdot), \pi\right). & \ell = 1 \quad (12) \end{cases}$$

Notice that the logits for the first, clean position use only the base AR parameters θ_{AR}, as shown in (11), whereas the logits at the noisy positions use both the AR weights and the diffusion adapters. This separation prevents distribution shift because the diffusion adapters are trained only to denoise noisy tokens. We compute (11) and (12) in a single forward pass using the gated LoRA technique of Samragh et al. (2025).

B.2 Single-Step Generation

After removing time conditioning from the denoising model and shifting the logits one position to the left to account for its next-token prediction (NTP) formulation in (9) for MDMs and (10) for USDMs, we show that the resulting marginal distribution has the following form. Let Ψ_0 denote the distribution induced by the Ψ-sampler at $t = 0$:

$$\Psi_0^\ell = \begin{cases} \mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}^\ell([\mathbf{x}^L, \mathbf{z}_t]), & \ell > 1 \\ \mathbf{x}_{\theta_{\text{AR}}}^\ell([\mathbf{x}^L, \mathbf{z}_t]). & \ell = 1 \end{cases}$$

B.3 Sampling Algorithm

Algo. 1 shows Uno's complete sampling algorithm for the single-sequence case. Differences to speculative decoding (Leviathan et al., 2023) are written in orange.

Algorithm 1 Lossless Ψ-Spec decoding for USDMs (Linear Sampler)

Require: Prefix $\mathbf{x}$ of length L, target model $\mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}$, block size B

Sample noise $\mathbf{z} \overset{\text{i.i.d.}}{\sim} \prod_1^{B-1} \mathcal{U}[\mathcal{V}]$ ▷ Initialize a block of $B-1$ random tokens
$[\mathbf{q}_0, \mathbf{q}] \leftarrow \mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}([\mathbf{x}^L, \mathbf{z}]; \mathbf{x}^{<L})$ ▷ θ_{AR} apply on $\mathbf{x}^L$; $\theta_{\text{AR}} + \theta_\Delta$ on $\mathbf{z}$
Sample $\mathbf{x}^{L+1} \sim \mathbf{q}_0$ ▷ free clean token from draft step
Sample $\tilde{\mathbf{x}} \sim \mathbf{q}$ ▷ $B-1$ tokens sampled in parallel
$\mathbf{p} \leftarrow \mathbf{x}_{\theta_{\text{AR}}}([\mathbf{x}^{L+1}, \tilde{\mathbf{x}}]; \mathbf{x})$ ▷ AR verifier distribution defined by θ_{AR}
sample $\mathbf{r} \overset{\text{i.i.d.}}{\sim} \prod_1^{B-1} \mathcal{U}[0,1]$
$n \leftarrow \min\left(\left\{i \in [B-1] : r_i > \min\left(1, \frac{\mathbf{p}_i}{\mathbf{q}_i}\right)\right\} \cup \{B\}\right)$
if $n = B$ **then**
    sample $\tilde{\mathbf{z}} \sim \mathbf{p}_B$
else
    sample $\tilde{\mathbf{z}} \sim \text{Norm}([\mathbf{p}_n - \mathbf{q}_n]_+)$
end if
return $[\mathbf{x}, \mathbf{x}^{L+1}, \tilde{\mathbf{x}}^{1:n-1}, \tilde{\mathbf{z}}]$

C Additional Experiments

C.1 Experiment Configs

C.1.1 Benchmarks

We provide additional details about the set of benchmarks we evaluate Uno on in Table 4.

Table 4: Benchmarks used in our Uno evaluation. "Generations" denotes the number of responses sampled per example when computing average pass@1.

Benchmark	Generations	Description
Agentic		
τ^2-Bench	3	Multi-turn tool-use tasks in which an agent interacts with a simulated user and domain APIs while following domain-specific policies.
Terminal-Bench v2.1	3	Realistic and complex tasks that autonomous agents complete in command-line container environments.
SWE-bench Verified	3	Human-validated software-engineering tasks requiring agents to resolve real GitHub issues by modifying their associated repositories.
Long-Context Reasoning		
AA-LCR	3	Open-answer questions requiring information extraction, synthesis, and reasoning over long documents such as reports and legal texts.
Science and Knowledge		
AA-Omniscience	1	Closed-book questions measuring factual recall and calibration across a broad range of economically relevant domains.
Humanity's Last Exam	1	Difficult expert-level questions spanning mathematics, the sciences, the humanities, and other academic disciplines.
GPQA-Diamond	5	Expert-validated, graduate-level multiple-choice questions in biology, physics, and chemistry.
Math		
GSM8K	2	Grade-school mathematics word problems requiring multi-step arithmetic reasoning.
MATH500	2	A 500-problem subset of MATH covering competition-level mathematical reasoning with free-form answers.
AIME 2024–2026	10	Integer-answer problems from the 2024, 2025, and 2026 American Invitational Mathematics Examinations.
Coding		
MBPP	2	Short Python program-synthesis problems specified through natural-language descriptions and test cases.
HumanEval	2	Hand-written Python function-completion problems evaluated using executable unit tests.
Instruction Following		
IFEval	2	Verifiable instruction-following tasks that test compliance with explicit, objectively checkable constraints.

C.1.2 Uno_{Qwen}, EAGLE-3, DFlash Sampler Configs for Qwen3-8B

At temp = 1, we use top-p = 0.95 and top-k = 50; temp = 0 denotes greedy decoding. Uno_{Qwen} uses linear blocks of size $B \in \{4, 8, 16\}$ or, for tree verification, $B = 16$, tree size $V = 60$, and candidate top-$K = 32$. DFlash uses blocks $B \in \{4, 8, 16\}$ and we evaluate both thinking-enabled and thinking-disabled variants (see Suppl. C.7). Linear EAGLE-3 $B \in \{4, 8, 16\}$ draft tokens; its tree configuration uses seven draft steps, top-$k = 10$, and at most $V = 60$ draft tokens.

C.2 Uno RL

Starting from the supervised fine-tuning (SFT) checkpoint, four specialized experts were trained for mathematics, code generation, tool use, and browse search using the DAPO reinforcement learning algorithm (Yu et al., 2026a). The Math expert was trained for 2,560 steps and generated 28.9B response tokens, while the Code expert was trained in two stages of 600 and 1,500 steps, producing

Table 5: Open-source model checkpoints used in our evaluations.

Model	Checkpoint
DiffusionGemma	https://huggingface.co/google/diffusiongemma-26B-A4B-it
Nemotron-Labs-Diffusion	https://huggingface.co/nvidia/Nemotron-Labs-Diffusion-14B
EAGLE-3	https://huggingface.co/AngelSlim/Qwen3-8B_eagle3
DFlash	https://huggingface.co/z-lab/Qwen3-8B-DFlash-b16
Fast-dLLM v2	https://huggingface.co/Efficient-Large-Model/Fast_dLLM_v2_7B
SDAR	https://huggingface.co/JetLM/SDAR-8B-Chat-b16

6B and 11.94B response tokens, respectively. Both experts used a learning rate of 5×10^{-7}, 16 groups, and 32 prompts per step, yielding 512 sequences per step across 32 nodes per expert. Math training used maximum context lengths of 32,768 and 65,536 tokens, while Code training supported contexts up to 65,536 and 128K tokens. The Tool-use expert completed 39 steps with 128 prompts per step and generated 1.4B tokens, including tool-call tokens. The Browse Search expert completed 59 steps with 64 prompts per step and generated 8.4B tokens, including tool-call tokens. For both Tool-use and Browse Search, the maximum trajectory length was 128K tokens, covering model-generated tokens as well as tool outputs. Finally, the four experts were consolidated into a single model using ISO-Merger (RAM; Yuan et al., 2026), a data-free approach that combines specialists trained from a shared base checkpoint without requiring additional rollouts, gradient updates, or distillation.

C.3 Uno Evaluations

Table 6: We report TPFs for a subset of the evaluations using different sampler configurations for the Uno model. Among all configurations, the Linear sampler with $B = 4$ achieves the highest system throughput at a batch size of 64, while the Tree sampler with $(B, K, V) = (16, 32, 32)$ achieves the highest per-user throughput (batch size 1).

Dataset	Linear B4	Linear B8	Linear B16	Tree B16,K32,V32	Tree B16,K64,V64	Tree B16,K64,V32
GSM8K	1.9	2.3	2.4	2.7	2.6	2.8
AIME-24	1.8	2.1	2.2	2.5	2.5	2.4
AIME-25	1.8	2.2	2.3	2.4	2.6	2.6
AIME-26	1.8	2.1	2.1	2.4	2.5	2.4
MATH500	1.8	2.1	2.2	2.4	2.5	2.5
HumanEval	1.8	2.6	2.8	3.4	2.5	3.4
AA-LCR	1.8	2.1	2.2	2.6	2.5	2.4
TPF	1.8	2.2	2.3	2.6	2.5	2.6
Highest Sys. Throughput	**5190**	5034	3445	2665	1573	2709
Highest Per-req. Throughput	275	319	345	**379**	339	371

Table 7: DG, Uno, NLD, Mercury2. *As reported by Artificial Analysis' live tracker on August 30, 2026.

Method	Max. Sys. Throughput (toks / sec)	Max. Per-req. Throughput (toks / sec)
Uno (Ours)	5255	383
AR (Ours)	3577	176
DiffusionGemma	1136	836
Nemotron-Labs-Diffusion	2794	290
Mercury-2	1197*	769*

Table 8: TPFs for the SFT checkpoint and its RL-post-trained counterpart, evaluated using the diffusion weights trained for the SFT checkpoint. **Even after extensive RL post-training, the SFT diffusion adapters retained their speedup**, with only a 6% reduction in TPFs.

	SFT	Post-Trained
Math		
GSM8K	2.66	1.95
MATH500	2.27	1.93
AIME-24	2.17	1.93
AIME-25	2.26	1.97
AIME-26	2.22	1.83
Coding		
HumanEval	1.97	2.34
MBPP	2.35	2.19
Science and Knowledge		
GPQA-Diamond	2.03	1.97
Instruction Following		
IFEval	2.12	2.50
Other Benchmarks		
HLE	2.14	2.15
AA-Omniscience	2.57	2.34
TPF	2.25	2.10

C.4 Qwen Finetuned on OpenThoughts

Table 9 compares Qwen3-8B with a variant fine-tuned on OpenThoughts across a subset of benchmarks. Fine-tuning on OpenThoughts substantially degrades performance, reducing accuracy by up to 15% points, depending on the benchmark. Nevertheless, diffusion weights trained on OpenThoughts successfully accelerate generation despite the distribution mismatch between OpenThoughts and the data used to train the AR weights.

Table 9: This table compares Qwen3-8B with a variant fine-tuned on OpenThoughts across a subset of benchmarks.

	Qwen3-8B	Qwen-finetuned
Math		
GSM8K	$96_{\pm 0.2}$	$95.2_{\pm 1.1}$
MATH500	$96.2_{\pm 0.3}$	$94.8_{\pm 0.0}$
AIME-24	$77.7_{\pm 3.0}$	$68.3_{\pm 17.6}$
AIME-25	$70.7_{\pm 4.2}$	$53.3_{\pm 0.0}$
AIME-26	$67.7_{\pm 3.9}$	$61.7_{\pm 5.9}$
Coding		
HumanEval	$94.4_{\pm 1.0}$	$77.8_{\pm 4.6}$
MBPP	$88.7_{\pm 0.4}$	$78.1_{\pm 0.4}$
LCBv6	$50.9_{\pm 1.2}$	$42.5_{\pm 0.8}$

C.5 Uno_{Qwen} vs I-DLM (R-ISD) Comparison

In this section, we compare I-DLM (R-ISD), the lossless variant of I-DLM, with our method, Uno_{Qwen}.

In the first experiment, we train I-DLM (R-ISD) with a rank-128 LoRA adapter for one epoch on the OpenThoughts dataset, using exactly the same training configuration as $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$. For inference,

we use the sampler provided in the official I-DLM repository[3]. All results for I-DLM (R-ISD) in Table 10 are evaluated with a block size of $B = 16$. The results show that the I-DLM sampler degrades accuracy on numerous benchmarks relative to the Qwen AR model, as highlighted in red, whereas $\text{Uno}^{\text{lep}}_{\text{Qwen}}$ does not. Upon inspecting the I-DLM codebase, we find that its sampler performs greedy drafting[4], but its rejection-sampling verification procedure is not adjusted accordingly. **As a result, the sampler does not preserve the claimed losslessness property**.

I-DLM (R-ISD) uses LoRA adapters and retains the transformer's causal attention pattern. It is therefore compatible with our Ψ-Spec sampler using the masked prior $\boldsymbol{\pi} = \mathbf{m}$. In the second set of experiments, we evaluate I-DLM (R-ISD) with Ψ-Spec. As shown in Table 10, this combination remains lossless. We observe the same result when evaluating the I-DLM (R-ISD) adapters released in the official repository. However, both variants provide considerably lower speedups than our method.

Table 10: I-DLM and $\text{Uno}^{\text{lep}}_{\text{Qwen}}$ evaluation with the linear sampler at $B = 16$ and temp = 1. We compare the I-DLM checkpoints (Ours and Official) under Ψ-spec samplers with $\boldsymbol{\pi} = \mathbf{m}$; accuracy is reported with 95% confidence intervals across seeds, using two seeds for MMLU-Pro and ten seeds for all other benchmarks.

Sampler		I-DLM sampler		Ψ-spec sampler					
Model	Qwen3 AR	I-DLM (R-ISD) (ours)		I-DLM (R-ISD) (ours)		I-DLM (R-ISD) (official)		Uno (ours)	
	Acc. (%)	Acc. (%)	TPF	Acc. (%)	TPF	Acc. (%)	TPF	Acc. (%)	TPF
Math									
GSM8K	$96.0_{\pm 0.2}$	$96.0_{\pm 0.2}$	2.14	$96.0_{\pm 0.2}$	1.95	$96.0_{\pm 0.2}$	2.23	$96.0_{\pm 0.2}$	2.66
MATH500	$96.2_{\pm 0.3}$	$96.3_{\pm 0.5}$	2.36	$96.1_{\pm 0.4}$	2.06	$95.9_{\pm 0.3}$	2.06	$96.2_{\pm 0.3}$	2.97
AIME-24	$77.7_{\pm 3.0}$	$77.3_{\pm 2.9}$	2.33	$76.7_{\pm 4.2}$	2.03	$76.7_{\pm 2.8}$	1.80	$77.7_{\pm 2.8}$	2.85
AIME-25	$70.7_{\pm 4.2}$	$63.0_{\pm 3.6}$	2.38	$68.7_{\pm 4.8}$	2.06	$70.7_{\pm 5.1}$	1.79	$69.7_{\pm 3.2}$	2.92
AIME-26	$67.7_{\pm 3.9}$	$68.7_{\pm 4.7}$	2.34	$64.3_{\pm 3.2}$	2.02	$66.7_{\pm 3.2}$	1.78	$67.0_{\pm 3.3}$	2.84
Coding									
HumanEval	$94.4_{\pm 1.0}$	$93.7_{\pm 0.9}$	2.30	$95.0_{\pm 0.6}$	1.77	$94.5_{\pm 0.5}$	1.98	$94.3_{\pm 0.7}$	2.40
MBPP	$88.7_{\pm 0.4}$	$87.8_{\pm 0.4}$	2.19	$88.7_{\pm 0.7}$	1.80	$88.7_{\pm 0.5}$	1.93	$88.8_{\pm 0.4}$	2.47
LCBv6	$50.9_{\pm 1.2}$	$48.9_{\pm 1.5}$	2.05	$50.4_{\pm 1.4}$	1.68	$51.0_{\pm 1.4}$	1.60	$50.1_{\pm 1.4}$	2.30
Science and Knowledge									
GPQA	$56.8_{\pm 0.7}$	$57.2_{\pm 1.1}$	2.01	$56.1_{\pm 1.1}$	1.66	$56.6_{\pm 1.1}$	1.71	$57.0_{\pm 1.0}$	2.35
GPQA-Diamond	$59.5_{\pm 0.9}$	$58.4_{\pm 0.5}$	2.02	$59.4_{\pm 1.4}$	1.67	$60.6_{\pm 1.2}$	1.69	$59.7_{\pm 1.1}$	2.35
MMLU-Pro	$74.7_{\pm 0.7}$	$74.6_{\pm 0.5}$	2.07	$74.7_{\pm 0.2}$	1.75	$75.1_{\pm 0.9}$	1.84	$74.7_{\pm 0.7}$	2.46
Instruction Following									
IFEval	$85.8_{\pm 0.7}$	$85.3_{\pm 0.7}$	1.87	$85.9_{\pm 0.8}$	1.65	$85.7_{\pm 0.6}$	1.85	$86.0_{\pm 0.7}$	1.96
Average	$76.6_{\pm 0.4}$	$75.6_{\pm 0.6}$	2.17	$76.0_{\pm 0.7}$	1.84	$76.5_{\pm 0.5}$	1.86	$76.4_{\pm 0.4}$	2.55

Additionally, Table 11 provides TPF for the I-DLM LoRA checkpoint evaluated on block sizes $B = 4$, $B = 8$, and $B = 16$. The results show a significant increase in TPF from $B = 4$ to $B = 8$, while TPF seems to saturate beyond $B = 8$.

[3] https://github.com/Introspective-Diffusion/I-DLM/commit/a23c1a12ef997c7f3ad616b25bcfb62db39ded68

[4] https://github.com/Introspective-Diffusion/I-DLM/blob/a23c1a12ef997c7f3ad616b25bcfb62db39ded68/inference/sglang/sglang/srt/dllm/algorithm/idlm_blockN.py#L575

Table 11: TPFs for the official I-DLM (R-ISD) model with our Ψ-spec samplers across block sizes B.

	B=4	B=8	B=16
Math			
GSM8K	1.87	2.20	2.23
MATH500	1.80	2.04	2.06
AIME-24	1.68	1.80	1.80
AIME-25	1.66	1.81	1.79
AIME-26	1.67	1.78	1.78
Coding			
HumanEval	1.75	1.96	1.98
MBPP	1.73	1.93	1.93
LCBv6	1.54	1.60	1.60
Science and Knowledge			
GPQA	1.61	1.71	1.71
GPQA-Diamond	1.59	1.69	1.69
MMLU-Pro	1.69	1.83	1.84
Instruction Following			
IFEval	1.73	1.86	1.85
Average	1.69	1.85	1.86

C.6 Uno_{Qwen} Ablations

C.6.1 LoRA Rank and Loss Ablations

In Table 12, we ablate the LoRA rank of $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$, as well as various weightings of KL and TV loss terms. We see that average TPF increases from 2.39 to 2.47 when increasing the LoRA rank from 128 to 256, while the number of trainable parameters increases from 349M to 698M. It turns out that a loss weighting of $0.01\times$KL plus $1\times$TV slightly outperforms a pure TV loss objective. Note that a factor of 0.01 for the KL term amounts to a KL loss term only about $10\times$ smaller in magnitude than the TV term, as the KL loss is naturally around $10\times$ larger than the TV loss.

C.6.2 Training Block Size Curriculum

Here we ablate two block-size curricula for Uno_{Qwen} training. Table 13 shows the TPF attained by two different three-epoch block-size curricula, both initialized from a model trained on block size 2 for 0.5 epochs, and block size 4 for another 0.5 epochs (written as $0.5@2$, $0.5@4$). The first curriculum is our default curriculum used for Uno_{Qwen}, which continues at $0.5@6$, $0.5@8$, $0.5@12$, and $0.5@16$; the second one continues at $2@16$. Average TPF drops from 2.71 to 2.65 when employing the second curriculum, which shows that incrementally increasing block size during training is beneficial over consistently training at a large block size.

Table 12: TPF results for one-epoch $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$ checkpoints trained with TV-only, KL-only, and weighted KL+TV objectives. The TV-only objective is evaluated using rank-128 LoRA adapters (the default Uno_{Qwen} configuration, highlighted in blue) and rank-256 adapters, while all other objectives use rank-128 adapters. All adapters are applied to every projection and use $\alpha_{\text{LoRA}}/r = 2$. All checkpoints are evaluated using the linear sampler with $B = 16$ and temp = 1.

	TV only		Loss ablations (r=128)				
	r=128 α=0, β=1	r=256 α=0, β=1	KL only α=1, β=0	KL + TV α=1, β=1	0.1 KL + TV α=0.1, β=1	0.05 KL + TV α=0.05, β=1	0.01 KL + TV α=0.01, β=1
Math							
GSM8K	2.47	2.53	2.27	2.30	2.39	2.42	2.46
MATH500	2.73	2.82	2.52	2.56	2.65	2.70	2.71
AIME-24	2.65	2.75	2.47	2.51	2.62	2.66	2.70
AIME-25	2.76	2.80	2.55	2.56	2.66	2.72	2.75
AIME-26	2.81	2.82	2.53	2.51	2.59	2.67	2.68
Coding							
HumanEval	2.26	2.37	2.04	2.07	2.30	2.23	2.25
MBPP	2.33	2.40	2.13	2.14	2.26	2.29	2.32
LCBv6	2.17	2.22	1.94	1.97	2.08	2.13	2.17
Science and Knowledge							
GPQA	2.21	2.27	1.99	2.03	2.15	2.19	2.20
GPQA-Diamond	2.22	2.26	2.00	2.03	2.15	2.18	2.21
MMLU-Pro	2.31	2.36	2.11	2.14	2.24	2.28	2.30
Instruction Following							
IFEval	1.79	1.99	2.08	1.93	2.08	2.08	2.02
Average TPF	2.39	2.47	2.22	2.23	2.35	2.38	2.40

Table 13: TPF for two three-epoch TV-only block curricula. The notation $e@B$ denotes e epochs of training at block size B. Both models first train on $0.5@2$, $0.5@4$. The standard curriculum then progressively increases the block size, whereas the fixed-$B = 16$ curriculum trains for the remaining two epochs entirely at $B = 16$. Models are evaluated with the linear sampler at $B = 16$ and temp = 1.

	$0.5@2$, $0.5@4$, $0.5@6$, $0.5@8$, $0.5@12$, $0.5@16$	$0.5@2$, $0.5@4$, $2@16$
Math		
GSM8K	2.83	2.81
MATH500	3.13	3.10
AIME-24	3.18	3.04
AIME-25	3.14	3.00
AIME-26	3.03	3.02
Coding		
HumanEval	2.58	2.49
MBPP	2.60	2.58
LCBv6	2.37	2.37
Science and Knowledge		
GPQA	2.45	2.45
GPQA-Diamond	2.47	2.46
MMLU-Pro	2.57	2.56
Instruction Following		
IFEval	2.20	1.95
Average	2.71	2.65

C.6.3 LoRA Adapter Ablations

We ablate which projection matrices in the transformer layer to apply LoRA weights to in Table 14. By default, we apply LoRA to all projections (this includes the Q, K, V, O matrices from the attention layer, as well as the MLP gate, up, and down projections); in addition, we ablate only applying LoRA to the attention layers (i.e., Q, K, V, and O projections); only to the Q and K projections of the attention layer; only to the Q projections; and only to the O projections, which is the setup recommended in Fu et al. (2026). In each setting, we adapted the rank to keep parameter parity at 349M LoRA parameters. All models for this ablation were trained on one epoch at block size eight. Table 14 shows that at fixed α_{LoRA} / rank = 2, applying LoRA to all projections work best. Consequently, we apply LoRA all to projections for our Uno models.

Table 14: TPF for $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$ LoRA target-projection ablations with $\alpha_{\text{LoRA}}/r_{\text{LoRA}} = 2$. Ranks are chosen to approximately maintain LoRA parameter parity. All checkpoints are evaluated with the linear sampler at B=16 and temp = 1.

	All projections	Q, K, V, O projections	Q, V projections	Q projection	O projection
Rank r	128	364	729	1184	1184
LoRA α_{LoRA}	256	728	1458	2368	2368
α_{LoRA}/r	2	2	2	2	2
Math					
GSM8K	2.47	2.45	2.32	2.20	2.43
MATH500	2.73	2.68	2.55	2.40	2.69
AIME-24	2.65	2.67	2.48	2.35	2.72
AIME-25	2.76	2.70	2.54	2.39	2.71
AIME-26	2.81	2.64	2.46	2.32	2.68
Coding					
HumanEval	2.26	2.23	2.16	2.11	2.33
MBPP	2.33	2.31	2.22	2.11	2.33
LCBv6	2.17	2.15	2.05	1.95	2.15
Science and Knowledge					
GPQA	2.21	2.19	2.10	2.02	2.20
GPQA-Diamond	2.22	2.21	2.11	2.01	2.21
MMLU-Pro	2.31	2.29	2.20	2.09	2.30
Instruction Following					
IFEval	1.79	2.00	1.88	1.73	1.78
Average	2.39	2.38	2.26	2.14	2.38

C.6.4 Varying LoRA α/r

For LoRA with rank r_{LoRA} and a projection $W_{\text{base}} \in \mathbb{R}^{n\times m}$ of the base model, both LoRA training and inference use

$$W = W_{\text{base}} + \frac{\alpha_{\text{LoRA}}}{r_{\text{LoRA}}} BA,$$

where $B \in \mathbb{R}^{n\times r}$, $A \in \mathbb{R}^{r\times m}$, and $\alpha_{\text{LoRA}} > 0$ is a hyperparameter that controls the overall contribution of the LoRA adapter. In Table 15 we ablate the ratio $\alpha_{\text{LoRA}}/r_{\text{LoRA}}$ across values between 2 and 256 or both one-epoch and three-epoch models. All three-epoch models in this ablation were trained on the block size curriculum described in Suppl. C.6.2.

The table shows that the optimal ratio differs depending on the training horizon, and $\alpha_{\text{LoRA}}/r_{\text{LoRA}} = 16$ is optimal at three epochs, while $\alpha_{\text{LoRA}}/r_{\text{LoRA}} = 64$ is optimal at one epoch. Consequently, we pick $\alpha_{\text{LoRA}}/r_{\text{LoRA}} = 16$ for Uno_{Qwen}, while for Uno, we set $\alpha_{\text{LoRA}}/r_{\text{LoRA}} = 64$ which we observed to be preferable.

Table 15: TPF for one-epoch and three-epoch, TV-only $\text{Uno}^{\text{1ep}}_{\text{Qwen}}$ models with rank-128 LoRA adapters applied to all projections, across LoRA scaling values. All checkpoints are evaluated with the linear sampler at $B = 16$ and temp = 1; averages are unweighted across the 12 benchmarks.

(a) One epoch of training.

	$\alpha_{\text{LoRA}} = 256$	$\alpha_{\text{LoRA}} = 512$	$\alpha_{\text{LoRA}} = 1024$	$\alpha_{\text{LoRA}} = 2048$	$\alpha_{\text{LoRA}} = 8192$	$\alpha_{\text{LoRA}} = 16384$	$\alpha_{\text{LoRA}} = 32768$
α_{LoRA}/r	2	4	8	16	64	128	256
Math							
GSM8K	2.47	2.53	2.63	2.66	2.76	2.76	2.72
MATH500	2.73	2.79	2.89	2.97	3.04	3.02	2.98
AIME-24	2.65	2.77	2.85	2.85	3.00	2.92	2.90
AIME-25	2.76	2.83	2.91	2.92	3.02	3.01	2.90
AIME-26	2.81	2.72	2.77	2.84	2.93	2.93	2.82
Coding							
HumanEval	2.26	2.39	2.36	2.40	2.55	2.45	2.51
MBPP	2.33	2.39	2.43	2.47	2.56	2.51	2.46
LCBv6	2.17	2.21	2.26	2.30	2.32	2.31	2.25
Science and Knowledge							
GPQA	2.21	2.26	2.31	2.35	2.41	2.41	2.37
GPQA-Diamond	2.22	2.27	2.32	2.35	2.40	2.40	2.36
MMLU-Pro	2.31	2.36	2.41	2.46	2.51	2.50	2.45
Instruction Following							
IFEval	1.79	2.01	2.13	1.96	2.01	2.27	1.84
Average	2.39	2.46	2.52	2.55	2.63	2.62	2.54

(b) Three epochs of training.

	$\alpha_{\text{LoRA}} = 2048$	$\alpha_{\text{LoRA}} = 8192$	$\alpha_{\text{LoRA}} = 16384$
α_{LoRA}/r	16	64	128
Math			
GSM8K	2.83	2.83	2.78
MATH500	3.13	3.12	3.06
AIME-24	3.18	3.02	2.94
AIME-25	3.14	3.08	3.00
AIME-26	3.03	3.05	2.97
Coding			
HumanEval	2.58	2.49	2.44
MBPP	2.60	2.56	2.51
LCBv6	2.37	2.34	2.30
Science and Knowledge			
GPQA	2.45	2.43	2.41
GPQA-Diamond	2.47	2.47	2.39
MMLU-Pro	2.57	2.55	2.50
Instruction Following			
IFEval	2.20	2.00	2.13
Average	2.71	2.66	2.62

C.7 DFlash Thinking Mode Ablation

In Table 16 we evaluate DFlash (Chen et al., 2026) with thinking mode enabled and disabled, at temp = 1. Our results show that while disabling thinking mode significantly increases tokens per step, it deteriorates accuracy from around 76% to roughly 55%. This quality difference does not contradict losslessness: enabling thinking changes the chat template and hence the target-model distribution that the speculative decoder preserves. In light of these results, we subsequently enable thinking mode for DFlash (Z Lab, 2026).

Table 16: **DFlash thinking ablation** at temp = 1, top-p = 0.95, and top-k = 50. Accuracy in percent. AL denotes average acceptance length, i.e., tokens per verification step.

	DFlash, B=4				DFlash, B=16			
	No thinking		Thinking		No thinking		Thinking	
	Acc. (%)	AL	Acc. (%)	AL	Acc. (%)	AL	Acc. (%)	AL
Math								
GSM8K	93.93	3.15	95.91	2.47	93.56	5.75	95.83	3.38
MATH500	84.00	3.19	96.60	2.28	82.80	6.23	96.20	3.24
AIME-24	23.33	2.95	76.67	1.96	30.00	5.41	76.67	2.76
AIME-25	20.00	2.95	60.00	1.90	16.67	4.88	73.33	2.56
AIME-26	16.67	3.01	63.33	1.93	20.00	4.76	66.67	2.57
Coding								
HumanEval	87.20	3.68	94.51	2.35	87.20	10.10	94.51	3.03
MBPP	67.40	3.10	89.60	2.22	67.40	5.17	90.00	3.04
LCBv6	25.14	2.64	49.71	1.76	24.00	4.90	49.71	2.22
Science and Knowledge								
GPQA	43.53	2.57	57.59	1.98	44.42	3.61	55.80	2.57
GPQA-Diamond	43.43	2.50	61.11	1.92	44.44	3.70	58.08	2.50
MMLU-Pro	66.45	2.70	74.75	2.11	66.53	4.12	74.83	2.80
Instruction Following								
IFEval	85.21	1.84	86.14	1.92	87.80	2.59	84.66	2.26
Average	54.69	**2.86**	**75.49**	2.07	55.40	**5.10**	**76.36**	2.74

C.8 Extended Comparison vs EAGLE-3 and DFlash

By default, we evaluate DFlash with block sizes $B \in \{8, 16\}$. For EAGLE-3, we use a linear block size of $B = 8$, as well as its standard tree configuration with depth $D = 7$ (corresponding to block size $B = 8$), top-$k = 10$, and a verification budget of $V = 60$ tokens. These EAGLE-3 tree parameters follow the official settings provided in the paper and repository.

We provide extended evaluation results for Uno_{Qwen}, EAGLE-3, and DFlash across various sampler settings in Table 17.

Table 17: Tokens per step across samplers for Uno$_{\text{Qwen}}$, EAGLE-3, and thinking-enabled DFlash. Values at both temp = 0 and temp = 1 are unfiltered.

(a) Uno$_{\text{Qwen}}$ linear and tree samplers.

	temp = 0					temp = 1				
	Linear B=4	Linear B=8	Linear B=16	Tree B=16, V=32	Tree B=16, V=60	Linear B=4	Linear B=8	Linear B=16	Tree B=16, V=32	Tree B=16, V=60
Math										
GSM8K	4.14	5.52	6.28	7.02	7.27	3.99	5.14	5.67	6.29	6.49
MATH500	4.27	5.96	7.09	7.67	8.01	4.11	5.46	6.26	6.89	7.12
AIME-24	4.26	5.89	6.95	7.83	8.55	4.08	5.49	6.35	6.83	6.81
AIME-25	4.25	5.77	6.73	7.67	8.36	4.11	5.45	6.28	6.81	7.07
AIME-26	4.20	6.10	7.36	7.82	8.61	4.08	5.31	6.05	6.71	6.86
Coding										
HumanEval	4.31	6.26	7.70	8.74	9.28	3.83	4.67	5.16	5.63	5.78
MBPP	4.42	6.55	8.34	9.16	9.45	3.87	4.86	5.20	5.87	5.96
LCBv6	4.29	6.03	7.09	8.16	8.20	3.77	4.51	4.74	5.34	5.53
Science and Knowledge										
GPQA	4.35	6.07	7.15	8.16	8.52	3.79	4.64	4.91	5.49	5.65
GPQA-Diamond	4.37	6.08	7.30	8.46	8.64	3.78	4.64	4.95	5.49	5.65
MMLU-Pro	4.19	5.67	6.57	7.48	7.69	3.84	4.75	5.14	5.70	5.90
Instruction Following										
IFEval	4.20	5.57	6.83	7.29	7.85	3.48	4.36	4.40	4.58	4.53
Average	4.27	5.96	7.11	7.95	8.37	3.89	4.94	5.42	5.97	6.11

(b) EAGLE-3 linear and tree samplers.

	temp = 0							temp = 1						
	Linear B=4	Linear B=8	Linear B=16	Tree B=8, V=32	Tree B=8, V=60	Tree B=16, V=32	Tree B=16, V=60	Linear B=4	Linear B=8	Linear B=16	Tree B=8, V=32	Tree B=8, V=60	Tree B=16, V=32	Tree B=16, V=60
Math														
GSM8K	2.43	2.68	2.73	3.82	4.07	3.78	4.24	2.26	2.48	2.49	3.62	3.83	3.57	3.83
MATH500	2.26	2.46	2.40	3.50	3.80	3.49	3.80	2.14	2.27	2.28	3.27	3.51	3.29	3.52
AIME-24	2.23	2.42	2.49	3.41	3.62	3.48	3.70	2.11	2.25	2.23	3.22	3.46	3.25	3.41
AIME-25	2.23	2.41	2.54	3.51	3.73	3.51	3.72	2.13	2.27	2.26	3.31	3.49	3.28	3.46
AIME-26	2.28	2.49	2.58	3.63	3.75	3.63	3.71	2.13	2.26	2.27	3.28	3.51	3.28	3.49
Coding														
HumanEval	2.87	3.44	3.70	4.52	5.07	4.94	5.21	2.12	2.36	2.28	3.39	3.61	3.43	3.71
MBPP	2.82	3.52	3.58	4.80	5.02	4.97	5.34	2.16	2.31	2.30	3.39	3.65	3.40	3.65
LCBv6	2.64	3.08	3.04	4.27	4.47	4.41	4.65	2.04	2.14	2.15	3.22	3.38	3.22	3.45
Science and Knowledge														
GPQA	2.48	2.77	2.98	3.85	4.14	4.04	4.20	1.99	2.09	2.10	3.06	3.25	3.05	3.25
GPQA-Diamond	2.49	2.77	2.93	3.82	4.17	3.87	4.09	1.98	2.07	2.07	3.00	3.19	2.99	3.19
MMLU-Pro	2.35	2.60	2.62	3.68	3.93	3.69	3.95	2.03	2.14	2.14	3.11	3.27	3.10	3.31
Instruction Following														
IFEval	2.10	3.06	3.39	3.66	3.29	4.32	4.60	1.91	2.22	2.14	3.16	3.04	3.07	3.49
τ	2.43	2.81	2.91	3.87	4.09	4.01	4.27	2.08	2.24	2.23	3.25	3.43	3.24	3.48

(c) Thinking-enabled DFlash at block sizes B=4, B=8, and B=16.

	temp = 0			temp = 1		
	B=4	B=8	B=16	B=4	B=8	B=16
Math						
GSM8K	2.54	3.18	3.60	2.47	3.04	3.38
MATH500	2.35	2.99	3.49	2.28	2.82	3.24
AIME-24	2.04	2.48	3.10	1.96	2.22	2.76
AIME-25	2.00	2.30	2.85	1.90	2.23	2.56
AIME-26	1.97	2.37	3.03	1.93	2.19	2.57
Coding						
HumanEval	2.39	3.12	4.16	2.35	2.71	3.03
MBPP	2.28	2.96	3.87	2.22	2.66	3.04
LCBv6	1.92	2.25	2.77	1.76	1.94	2.22
Science and Knowledge						
GPQA	2.12	2.55	3.23	1.98	2.28	2.57
GPQA-Diamond	2.09	2.46	3.13	1.92	2.19	2.50
MMLU-Pro	2.23	2.70	3.26	2.11	2.47	2.80
Instruction Following						
IFEval	2.10	2.71	2.58	1.92	2.23	2.26
τ	2.17	2.67	3.26	2.07	2.42	2.74

Table 18: Median per-request throughput (tok/s/stream) for the 1K-input/8K-output Qwen3-8B workload at temp = 1. Each value is the median of three fresh repetitions; "–" denotes a resident-capacity-infeasible point.

	Uno					EAGLE-3					DFlash			AR
Con.	$B4$	$B8$	$B16$	$B16, V32$	$B16, V60$	$B4$	$B8$	$B16$	$B16, V32$	$B16, V60$	$B4$	$B8$	$B16$	–
System throughput														
1	305	366	416	445	435	204	198	161	289	284	287	331	369	176
2	566	703	768	803	730	401	377	312	531	504	538	645	702	352
4	1086	1205	1420	1411	1221	784	737	585	906	771	1058	1188	1296	653
8	1906	2397	2384	2161	1630	1381	1292	1038	1399	1019	1874	2073	2153	1144
16	3247	3814	3436	2798	1869	2429	2233	1693	1746	1118	3075	3315	3007	1933
32	4586	4873	4086	2958	1964	3745	3308	2228	1881	1171	4407	4246	3396	2796
64	5733	5600	4112	3138	2004	4944	3965	2580	1969	–	5351	4900	3592	3662
128	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Per-user throughput (tokens/s/user)														
1	305	366	416	445	435	204	198	161	289	284	287	331	369	176
2	283	351	384	401	365	200	189	156	266	252	269	322	351	176
4	271	301	355	353	305	196	184	146	227	193	264	297	324	163
8	238	300	298	270	204	173	161	130	175	127	234	259	269	143
16	203	238	215	175	117	152	140	106	109	70	192	207	188	121
32	143	152	128	92	61	117	103	70	59	37	138	133	106	87
64	90	87	64	49	31	77	62	40	31	–	84	77	56	57
128	–	–	–	–	–	–	–	–	–	–	–	–	–	–