TIDAR: THINK IN DIFFUSION, TALK IN AUTOREGRESSION

Jingyu Liu [*1,2] **Xin Dong** [*1] **Zhifan Ye** [1,3] **Rishabh Mehta** [1] **Yonggan Fu** [1]
Vartika Singh [1] **Jan Kautz** [1] **Ce Zhang** [2] **Pavlo Molchanov** [1]

ABSTRACT

Diffusion language models hold the promise of fast parallel generation, while autoregressive (AR) models typically excel in quality due to their causal structure aligning naturally with language modeling. This raises a fundamental question: can we achieve a synergy with high throughput, higher GPU utilization, and AR level quality? Existing methods fail to effectively balance these two aspects, either prioritizing AR using a weaker model for sequential drafting (speculative decoding), leading to lower drafting efficiency, or using some form of left-to-right (AR-like) decoding logic for diffusion, which still suffers from quality degradation and forfeits its potential parallelizability.

We introduce TIDAR, a sequence-level hybrid architecture that drafts tokens (**T**hinking) **i**n **D**iffusion and samples final outputs (Talking) **A**uto**R**egressively - *all within a single forward pass using specially designed structured attention masks*. This design exploits the free compute density on GPUs, achieving a strong balance between drafting and verification capacity. Moreover, we design TIDAR to be serving-friendly as a standalone model.

We extensively evaluate TIDAR against AR models, speculative decoding, and diffusion variants across generative and likelihood tasks at both 1.5B and 8B scales. Thanks to parallel drafting and sampling as well as efficient exact KV cache support, TIDAR outperforms speculative decoding in measured throughput and surpasses diffusion models like Dream and Llada in both efficiency and quality. Most notably, TIDAR is the first architecture to close the quality gap with AR models while delivering 4.71× to 5.91× more tokens per second.

1 INTRODUCTION

As we move towards Artificial General Intelligence (Bubeck et al., 2023), the remarkable success of large language models (LLMs) can be largely attributed to their ability to harness the massive computational scaling afforded by the explosively increasing power of GPUs (NVIDIA, 2025). So how to make the most of compute resources during both training and testing times becomes increasingly important. Although autoregressive models (Vaswani et al., 2017; Radford et al., 2019) are the prevailing approach, they remain memory-bound during decoding (Dao et al., 2022; Yuan et al., 2024) and cannot fully exploit hardware compute density, particularly at small batch sizes, since they generate only one token per step. In contrast, diffusion language models (dLMs) (Sahoo et al., 2024b; Nie et al., 2025; Ye et al., 2025a) offer the promise of parallel token decoding. However, they typically face a trade-off between quality and parallelizability. In this work, we provide a principled analysis to identify where these models fall short and propose a simple yet effective hybrid architecture that combines the strengths of both paradigms.

Decoding in AR models is memory-bound because the latency is dominated by loading the model weights and KV cache rather than compute (Kwon et al., 2023; Leviathan et al., 2023). If we can decode multiple tokens in a single forward pass, those tokens can share the same loaded weights and KV cache, increasing the compute density without increasing end-to-end latency before transitioning to the compute-bound region. This is precisely why dLMs could provide potential speedup from a systems perspective. Concretely, given a prefix $x_{<t}$ and model function F, an AR model appends a single token slot (last prompt token) and predicts one next token, e.g., $x_{t+1} := F([x_{<t-1}; x_t])$. In a masked diffusion model (e.g., Block Diffusion (Arriola et al., 2025a) with a simplified one-step denoising scenario), instead we predict multiple tokens at once, $x_{t+1}, \ldots, x_{t+k+1} := F([x_{<t}; m_{t+1}, \ldots, m_{t+k+1}])$ (due to no label shifts). If for a given k such that both computations are still memory-bound, the forward time of $F([x_{<t-1}; x_t])$ and $F([x_{<t}; m_{t+1}, \ldots, m_{t+k+1}])$ should be similar. We refer to the extra token slots $(m_{t+2}, \ldots, m_{t+k+1})$ as **free token slots** because carrying them through a single forward incurs minimal to no latency

[*]Equal contribution [1]Nvidia [2]University of Chicago (work done as an intern) [3]Georgia Institute of Technology (work done as an intern). Correspondence to: Jingyu Liu <jingyu6@uchicago.edu>, Xin Dong (Project Lead) <xindong@alumni.harvard.edu>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

increase as validated by a real-world profiling in Figure 1.

However, a well-known tension between parallel decoding and output quality exists for masked diffusion models (Wu et al., 2025b; Feng et al., 2025): the best quality is often achieved when decoding strictly one token per denoising step, while attempting to exploit within-step parallelism tends to degrade quality. Consequently, current open source SOTA dLMs, such as Dream (Ye et al., 2025a) and Llada (Nie et al., 2025), have not yet matched the combined speed–quality profile of strong AR LLMs. To formalize their difference from a modeling perspective, AR models sample from a chain-factorized joint distribution:

$$p_{\text{AR}}(\cdot;\theta) = \prod_i p_\theta^i(x_i|\mathbf{x}_{<i};\theta).$$

A diffusion model, on the other hand, samples from the following distribution:

$$p_{\text{Diff}}(\cdot;\theta) = \mathbb{E}_{\tilde{\mathbf{x}}\sim q(\cdot|\mathbf{x})} \prod_i p_\theta^i(x_i|\tilde{\mathbf{x}})$$

Often the quality is best preserved if we choose to decode one token x_i instead of k tokens $\{x_j, j \in K\}$ per step because in this case $\tilde{\mathbf{x}}$ will have the decoded token as a condition for the next denoising step. Decoding k tokens in one step will further factorize p_{Diff} as a product of marginals:

$$p_{\text{Diff\_Independent\_K}}(\cdot;\theta) = \mathbb{E}_{\substack{\tilde{\mathbf{x}}\sim q(\cdot|\mathbf{x}) \\ s.t. \tilde{\mathbf{x}}_{i\in K}=[m]}} \prod_{i\in K} p_\theta^i(x_t^i|\tilde{\mathbf{x}})$$

The introduced token independence assumption will degrade the quality despite providing more parallelism during sampling. The diffusion sampling procedure would fall back to AR if we restrict the order as strict left-to-right, decode one token per step, and change the masking strategy $q(\cdot|\mathbf{x})$ to uniform suffix masking during training.

Ideally, we hope to compute with p_{Diff} and leverage its independence assumption for parallel sampling. At the same time, we want to get the quality from p_{AR} due to its aligned casual factorization nature to language modeling and more context in the condition.

To this end, we propose TIDAR, a new architecture that enables parallel token computation ("thinking") from the marginal distribution via diffusion, and high-quality sampling ("talking") from the chain-factorized joint distribution via autoregression. Concretely, at each generation step (one forward pass), we partition tokens into three sections: prefix tokens, tokens proposed in the previous step, and tokens pre-drafted for the next step. We reuse the KV cache of prefix tokens from the last step. Tokens proposed from the last step are autoregressively sampled via rejection sampling guided by p_{AR} computed at current step. At the same time, we pre-draft proposals from the p_{Diff} conditioned on all possible prefix outcomes of the rejection sampling. One pre-draft proposal will be chosen from them, and passed to the next step. All of these happen in a single forward pass with a simple and well-designed attention mask. The overhead is almost negligible as long as the tokens residing in the last two sections can be well fitted into the "free token slots". Figure 2 illustrates the architecture in detail.

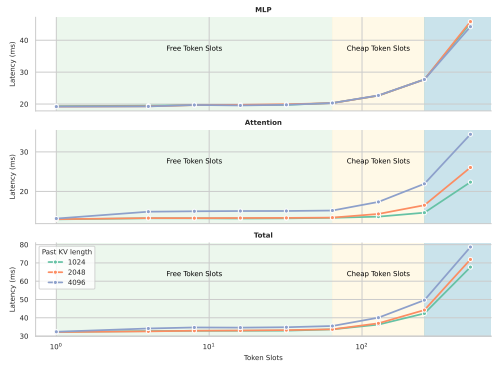

Figure 1. **Latency Scaling over Token Slots:** We plot the latency of Qwen3-32B decoding on NVIDIA H100 with batch size=1 and Flash Attention 2 (Dao, 2023) over different prefix lengths. Latency stays relatively the same with a certain amount of tokens sent to forward (free + cheap token slots), before transitioning to the compute-bound regime. We leverage this characteristic to achieve almost free parallelled drafting and sampling for TIDAR.

Training TIDAR is straightforward and data-efficient, as it employs a structured hybrid attention mask over the input sequence, enabling it to learn and sample from p_{AR} and p_{Diff} within a single model and foward. Consequently, both autoregressive and diffusion losses can be computed on the same data sample. During training, all tokens in the diffusion section are set to mask tokens. This simplifies the masking strategy, strengthens the diffusion loss signal, promotes train-test consistency, and balances the autoregressive and diffusion objectives.

Our contributions can be summarized as:

- We propose a sequence-level hybrid architecture called TIDAR that utilizes the "free token slots" to conduct parallel token drafting via diffusion and sampling via autoregression, which combines the speed and quality advantages from both paradigms.
- We provide a training recipe as well as comprehensive evaluations on both likelihood and generative downstream tasks to demonstrate its superiority over other architectures in certain applications.
- We conducted detailed ablations to prove the effectiveness of our core design choices and flexibility. In addition, we analyze TIDAR from the lens of diffusion and

	Draft Model		Drafting Process	
Model	**Shared w/ Base**	**Drafting Capacity**	**Parallel Decoding**	**Parallel to Verification**
Classic Spec. Decoding (Chen et al., 2023)	✗	Low	✗	✗
APD (Israel et al., 2025)	✗	High (Weak Verifier)	✓	✗
EAGLE-3 (Li et al., 2025) & DeepSeek-V3 (Liu et al., 2024)	✓*	Mid	✗	✗
Apple MTP (Samragh et al., 2025)	✓*	Mid	✗	✓
TIDAR	✓	High	✓	✓

Table 1. **Comparison among Speculative Frameworks:** We compare different speculative frameworks in two major aspects: 1) whether the drafter is a separate module from the base model and has high capacity. 2) if the drafting process has paralleled decoding and is sequential to the verification or sampling process. ✓* means that draft models need extra layers and heads to the base model.

speculative decoding, providing a clear understanding of why our method is appealing.

- We show that for TIDAR 1.5B, we can achieve lossless quality compared to its AR counterpart while generating with 4.71× relative throughput (tokens per second) speedup. For TIDAR 8B, we achieved an impressive 5.91× relative throughput speedup with minimal loss.

2 Background and Related Work

TIDAR model is related to and enjoys the benefits of two lines of works: diffusion language models and speculative decoding. In this section, we connect TIDAR to these two categories and highlight how we understand TIDAR from different perspectives and how it improves upon prior works.

2.1 Diffusion Language Models

Diffusion language models (Austin et al., 2021a; Li et al., 2022; Nie et al., 2025; Ye et al., 2025a; Arriola et al., 2025a; Sahoo et al., 2024b) (dLMs) offer a promising alternative to purely sequential generation of autoregressive models by allowing parallel generation of multiple tokens at each step, offering a path towards significant speed-up in generation. Although theoretically appealing, dLLMs face a trade-off contradiction between parallelizability and generation quality. As exemplified by open-source dLLMs like Llada (Nie et al., 2025) and Dream (Ye et al., 2025a), the best generation quality is often achieved when decoding one token at a time (1 token per model function forward). As reported in APD (Israel et al., 2025), there is a clear trend of declining generation quality with increasing number of tokens to generate in parallel per step. Specifically, the accuracy on GSM8K (Cobbe et al., 2021) drops by 10% when increasing from 1 to 2 tokens per step for Dream-7B with the entropy-base sampling strategy (Ye et al., 2025a). When decoding multiple tokens per step, dLMs typically sample each token independently from the marginal distribution, which introduces an intra-step token independence assumption (Wu et al., 2025a) that can hurt sequence-level coherence and correctness (Feng et al., 2025). Recent works have attempted to address the gap between the generation quality of diffusion models and AR and improve the throughput-quality trade-off. E2D2 (Arriola et al., 2025b) adopts an encoder-decoder architecture that enables disaggregating processing FLOPs by token type, where a large encoder processes clean tokens and a lightweight decoder is tasked with decoding noisy tokens. EDLM (Xu et al., 2025) introduces residual energy-based approach to reduce mismatch between training and sampling distributions, improving generation quality.

Another challenge of scaling up dLM is the lack of support for exact KV caching, as a result of bidirectional attention. Fast-dLLM (Wu et al., 2025b;a) proposes to perform block parallel decoding with the prefix and optionally the suffix being cached during the process of denoising the current block. d-KV cache (Ma et al., 2025), on the other hand, takes a more dynamic route by selectively cache certain tokens step by step in a delayed fashion to minimize quality degradation. Block Diffusion (also known as semi-autoregressive models) (Arriola et al., 2025a) attempts to address this issue by interpolating between discrete diffusion and autoregressive models. Specifically, the model defines an autoregressive probability distribution across blocks and the conditional probability of tokens within a block given previous blocks is specified by a denoising discrete diffusion model. Despite being equipped with exact caching, Block Diffusion still suffers from the same dilemma of quality degradation and intra-block token parallelizability.

2.2 Speculative Decoding

TIDAR is also closely related to speculative decoding (Chen et al., 2023), which accelerates generation by first using a faster draft model to generate a sequence of candidate tokens and then uses a modified rejection sampling strategy (Leviathan et al., 2023) to validate the these tokens against the target distribution of the base model. In order to improve the speed of drafting, a smaller draft model is usually used. However, if degradation in drafting quality is too severe, the overall generation speed can be slowed down because of low acceptance rate of draft. In order to increase both the speed and the quality of drafting, prior works propose to carry the hidden states from the base model to the drafter. The intuition is that the model's hidden states (e.g., the second-to-top layer embedding) carry richer information not only for the next token prediction but also for future tokens. For example, Medusa (Cai et al., 2024) adds extra multiple linear decoding heads on the base model's hidden states to predict future tokens with an efficient tree verification pattern to expand possible paths. EAGLE series (Li et al., 2024a;b; 2025) and DeepSeek-V3 Multi-Token Predic-

tion (MTP) (Liu et al., 2024) use additional autoregressive layers on the base model's hidden states to predict future tokens. However, the drafting process does not fully take advantage of the base model, and the maximal speedup is hindered by the lower drafter capacity. In addition, EAGLE and DeepSeek-V3's MTP modules are still autoregressive and sequential to the base verification. These two factors show that they cannot effectively increase the compute density and release the full power of parallel generation.

If we view TIDAR from the perspective of speculative decoding (summarized comparison in Table 1), one of the main advantages of TIDAR is that it only has a single model and can complete both the drafting and sampling simultaneously in a single forward pass. This induces three merits: 1) The draft model is the base model itself, so it has high capacity by reusing the base model's weights and compute. 2) The drafting process follows a diffusion approach, allowing it to be fully parallelized. This means it can utilize the full computational power across all input mask tokens, rather than being limited to only the last token as in autoregressive MTP methods. 3) The drafting and verification processes are parallelized in a single forward pass, which further eliminates the overhead of the sequential dependency between drafting and verification.

3 METHOD

In this section, we describe the proposed architecture TIDAR. We start from the backbone that enables modeling both the joint distribution (i.e., AR mode) and the marginal distribution (i.e., diffusion mode) in a single model and a single model forward (Section 3.1). We then elaborate how we use the dual modes for paralleled self-speculative generation, which allows the model to draft ("thinking") in diffusion for efficiency and sample ("talking") in autoregression for quality (Section 3.2). Finally, we conclude by discussing the training and inference optimizations (Section 3.3).

3.1 Diffusion-AR Dual-mode Backbone Training

Our goal is to train a model that enables the diffusion and AR dual modes. More importantly, we want to have the dual modes happening in a single model forward in order to exploit the "free token slots". To achieve this, sequence level casual and bidirectional attention hybridization becomes most natural choice. A similar idea has been explored in the context of Block Diffusion models (Arriola et al., 2025a); however, their primary objective is not to support dual-mode operation, but rather to enable KV caching and improve generation quality specifically for the diffusion component. The resulting attention mask of Block Diffusion is intra-block bidirectional and inter-block causal. We make a modification to Block Diffusion by only preserving the last block, which is the decoding block, to be bidirectional and the rest (i.e. prefix) to be causal. The benefits of doing so are two-folds. First, it allows us to compute the chain-factorized joint distribution just like in AR models. This, as we will show in Section 3.2, will allow us to conduct rejection sampling using the joint distribution with high quality guarantee and evaluate likelihood the same way as AR in terms of efficiency (Section 4.2.2). Second, computing next token prediction (NTP) loss on the prefix becomes possible during model pre-training and finetuning (Gat et al., 2025). Note that Block Diffusion cannot compute this loss on the prefix because of the label leakage issue of intra-block bidirectional attention. In addition, the NTP signal is much denser than pure diffusion loss since the latter is only applied to the mask tokens. In our model training, we can leverage the two types of losses simultaneously to better utilize the training data.

In Figure 3 (Left), we visualize the attention masked during training. Similar to Block Diffusion (Arriola et al., 2025a) and Set Block Decoding (SBD) (Gat et al., 2025), we also need to double the sequence length, as a result of appending original input sequence with corrupted tokens. For the causal (autoregressive) section, the target labels are shifted by one position to match NTP objective, while for the bidirectional (diffusion) section, the labels remain aligned with their input positions. For the diffusion part, Block Diffusion (Arriola et al., 2025a) and SBD (Gat et al., 2025) add noise to the input sequence by applying random masks sampled from a special distribution, following the traditional diffusion language modeling approaches (Nie et al., 2025; Ye et al., 2025a; Sahoo et al., 2024a).

We propose a simpler and more effective training strategy by setting all tokens in the diffusion section to mask tokens. This eliminates the hassle of deciding the optimal masking strategy. And more importantly, it allows us to compute the loss for every token in the diffusion part, resulting in three key benefits: (1) The diffusion loss becomes much denser compared to only corrupted tokens like before. (2) Balancing the diffusion loss with the NTP loss becomes much simpler. In traditional diffusion approaches, the number of loss terms varies across samples depending on how many tokens are masked, whereas the next-token prediction loss is always calculated over the same number of tokens. This mismatch makes it difficult to balance the two losses. By masking all tokens, the number of loss terms is consistent across both types of losses (equal to the sequence length), allowing straightforward balancing using a user-defined weighting factor. (3) It allows us to do one-step diffusion during inference, which makes the drafting process more efficient than multi-step denoising. We detail this with ablation studies in Section 4.4.4.

Therefore, the training objective of TIDAR is simplified as:

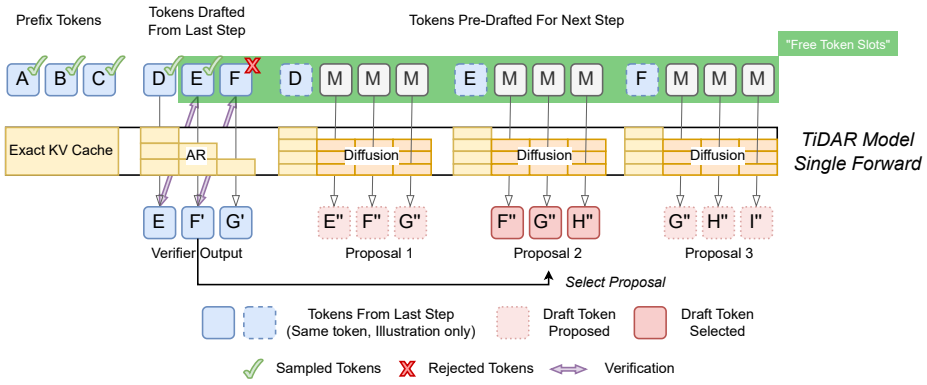

Figure 2. **TIDAR Architecture:** TIDAR uses a single model forward to sample drafted tokens from the last step and pre-draft tokens for the next step in parallel. By switching the attention pattern among different parts of the sequence, TIDAR encodes the clean tokens drafted from last step causally and mask tokens within-block bidirectionally along with the prefix for one-step diffusion pre-drafting. Upon accepting a prefix, the corresponding pre-drafts (proposal) can be selected. The KV cache for tokens forwarded causally will be stored and later evicted if the corresponding tokens are rejected. We illustrate this with a draft length of 3 and an accepted length of 2. Figure 3 shows the exact decoding mask for this example.

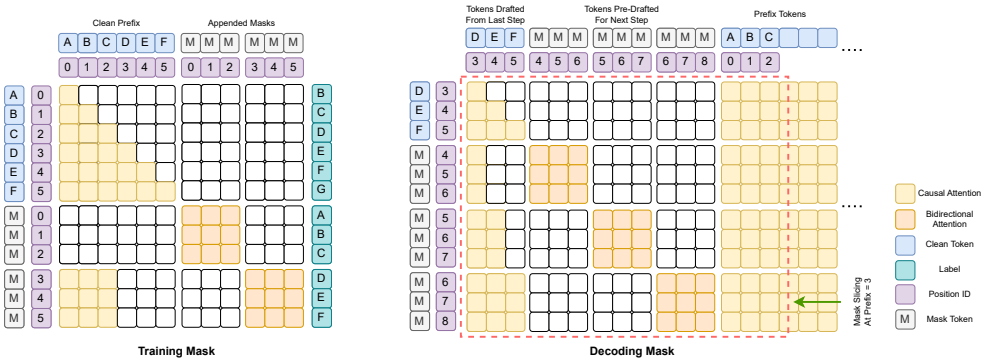

Figure 3. **TIDAR Attention Masks:** (Left) We apply a special training mask (using block length = 3): mask tokens of the same length are appended to the input tokens where the clean input tokens are self-attended causally and mask tokens within-block bidirectionally along with the prefix. During inference parallel decoding, we use a slice of a pre-initialized mask based on the prefix of the current step (Right). To reuse the mask, we reorder the sampling-draft part (tokens drafted from last step and mask positions for next step pre-drafting) and the clean prefix as illustrated with an example prefix length of 3.

$$\mathcal{L}_{\text{TIDAR}}(\theta) = \frac{1}{1+\alpha}\left(\sum_{i=1}^{S-1}\frac{\alpha}{S-1}\cdot\mathcal{L}_{AR}(x_i, x_{i+1};\theta) + \sum_{i=1}^{S-1}\frac{1}{S-1}\cdot\mathcal{L}_{Diff}([mask], x_i;\theta)\right)$$

where $\alpha \in [0,1]$ is the loss balancing factor, $\{x_i\}_S$ is the input sequence with length S, and $\mathcal{L}_{AR}, \mathcal{L}_{Diff}$ are cross-entropy losses with logits calculated at clean and masked sequences using different attention patterns.

3.2 Fully Parallelizable Self-Speculative Generation

Generation efficiency is determined by several factors: the number of forward steps in total, the number of decoded (or sampled) tokens per step, as well as the latency per step. Many existing techniques attempt to optimize some of them such as: diffusion language models (Nie et al., 2025; Ye et al., 2025a; Israel et al., 2025), MTP (Gloeckle et al., 2024; Cai et al., 2024; Liu et al., 2024), and speculative decoding (Leviathan et al., 2023; Li et al., 2024a). However, the result is often a trade-off among latency, quality, and compute density. In this work, we attempt to provide an answer to this global optimization problem by incorporating the diffusion parallelism and AR quality into a single model.

We propose a parallel drafting and sampling procedure. The

Algorithm 1 TiDAR Prefill

Require: Prompt $\mathbf{x} \in \mathbb{Z}^L$; draft length K
Ensure: Draft predictions $\mathbf{d}^{(0)} \in \mathbb{Z}^K$; KV cache
$\mathbf{z} \leftarrow [\,\mathbf{x} \parallel [\mathbb{m}]^K\,]$ ▷ Append K mask tokens
$\mathcal{M} \leftarrow \textsc{BuildPrefillMask}(L, K)$
$\mathbf{L},\ \text{KV} \leftarrow \textsc{TiDAR}(\mathbf{z},\ \mathcal{M})$
$\mathbf{d}^{(0)} \leftarrow \arg\max_v \mathbf{L}[L : L+K]$ ▷ Greedy diffusion drafting
$\text{KV} \leftarrow \text{KV}[0 : L]$ ▷ Keep only prompt KV (evict mask KV)
return $\mathbf{d}^{(0)},\ \text{KV}$ ▷ Prefill generates drafts only

Algorithm 2 TiDAR Decode

Require: Previous step draft $\mathbf{d}^{(t)} \in \mathbb{Z}^K$; KV cache; draft length K; mixing weight β
Ensure: Accepted tokens $\mathbf{a}$; new draft $\mathbf{d}^{(t+1)}$; updated KV cache
$\mathbf{z} \leftarrow [\,[\mathbb{m}]^{K\times K} \parallel \mathbf{d}^{(t)}\,]$ ▷ K mask blocks then K draft tokens
$\mathcal{M} \leftarrow \textsc{SlicePrecomputedMask}(\text{KV.len},\ K)$
$\mathbf{L},\ \text{KV}_{\text{new}} \leftarrow \textsc{TiDAR}(\mathbf{z},\ \mathcal{M},\ \text{KV})$ ▷ $\mathbf{L} \in \mathbb{R}^{(K^2+K)\times|V|}$

$\mathbf{L}_{\text{ar}} \leftarrow \mathbf{L}[K^2 : K^2+K]$ ▷ Get AR logits
$\mathbf{L}_{\text{diff}} \leftarrow \mathbf{L}[0 : K^2].\text{view}(K, K, |V|)$ ▷ Get diffusion logits
$\mathbf{L}_{\text{mix}} \leftarrow \beta \cdot \mathbf{L}_{\text{ar}} + (1-\beta) \cdot \mathbf{L}_{\text{diff}}[:, 0]$ ▷ Logit mixing

$\mathbf{a} \leftarrow \arg\max_v(\mathbf{L}_{\text{mix}})$ ▷ Greedy decoding for verifying tokens
$\mathbf{d} \leftarrow \arg\max_v(\mathbf{L}_{\text{diff}})$ ▷ Greedy decoding for draft tokens
$n \leftarrow \max_i\{\mathbf{a}[i-1] = \mathbf{d}[i, 0], i \geq 1\}$ ▷ Rejection sampling

$\mathbf{a} \leftarrow \mathbf{a}[: n]$ ▷ Get accepted prefix
$\mathbf{d}^{(t+1)} \leftarrow \mathbf{d}[n-1, :]$ ▷ Get corresponding draft tokens
$\text{KV} \leftarrow \text{KV}_{\text{new}}[K^2 : K^2+n]$ ▷ Update KV cache for reuse
return $\mathbf{a},\ \mathbf{d}^{(t+1)}, \text{KV}$

method centers around the speculative framework where the model first drafts speculative tokens in parallel from the marginal distribution, which are then rejectively sampled in an autoregressive manner to secure generative quality. During prefill, the model encodes prompt causually and drafts a block of tokens in parallel with a bidirectional attention (mask illustrated in Appendix Figure 8). In the each subsequent decoding step, draft tokens from the last step are rejectively sampled by checking whether they match the prediction from the autoregressive joint distribution computed at current step using causal attention. At the same time, we also pre-draft the next step's tokens in parallel from the marginal distribution using bidirectional attention, conditioned on all possible outcomes of the rejection sampling (as inspired by Apple's MTP work (Samragh et al., 2025). So that no matter how many tokens we accept at the current step, we would be able to get the corresponding drafts for the next step. In Figure 2, we illustrate the generation process. The pseudocode for the prefill and the decode are in in Algorithm 1 and Algorithm 2 respectively.

To further improve the efficiency, we adopt a one-step diffusion drafting (Liu et al., 2025). We found one step is sufficient to produce draft tokens with enough quality to secure high acceptance rate. Correspondingly, we set all tokens in the diffusion section to masks during training as discussed in the above Section 3.1.

3.3 Training and Inference Optimization

As we discuss in Section 3.1, our corrupted sequence is fully masked, which makes loss balancing easier due to the equal variance of the loss scale calculated from the AR and diffusion logits. We choose to set $\alpha = 1$ for most cases and ablate the difference of different α values in Section 4.4.3.

During inference, the number of tokens we send to the model in each forward is the same and has the same attention pattern. Hence, we can initialize one block attention mask of size (q_len[1], q_len + max_sequence_len) and slice the cached mask for each step without recomputing it for Flex Attention (Dong et al., 2024) by reordering the draft and prefix sections. Different from traditional dLMs, TIDAR supports exact KV cache like Block Diffusion. In Figure 2, we showcase the generative process where we first save all the KV cache of tokens which are computed with causal attention, and later on, evict the KV cache if our sampling length is shorter than the draft length. It is worth mentioning that we do not waste any computation by recomputing the KV cache of any token, which makes our method extremely efficient compared to Block Diffusion, SBD, and the cache methods used in pure diffusion (e.g. Fast-dLLMs (Wu et al., 2025b;a) and d-KV Cache (Ma et al., 2025)).

Another main advantage compared to traditional dLMs is that TIDAR has no hyperparameters to tune during inference. But still, we provide some flexibility to accomondate different scenarios as detailed in Section 4.4.2 and 4.4.3.

[1] q_len = block_len · (1 + block_len)

4 Experiments

4.1 Setup

Model initialization and tasks In this paper, we focus on the setting of continual pretraining from AR models (Qwen2.5 1.5B (Yang et al., 2025b), Qwen3 4B, and Qwen3 8B (Yang et al., 2025a)). We include quality evaluations on generative and likelihood tasks, including coding (HumanEval, HumanEval+, MBPP, and MBPP+), math (GSM8K and Minerva Math), factual knowledge (MMLU), and commonsense reasoning (ARC, Hellaswag, PIQA, and Winogrande). We use LM_EVAL_HARNESS (Gao et al., 2024) for all evaluation. The detailed task configurations can be found in the Appendix A. In terms of efficiency, we report the average number of tokens a single model forward (or network function evaluation, NFE for short) can produce, as well as wall-block speedup in terms of token per wall-clock second under the batch size of one.

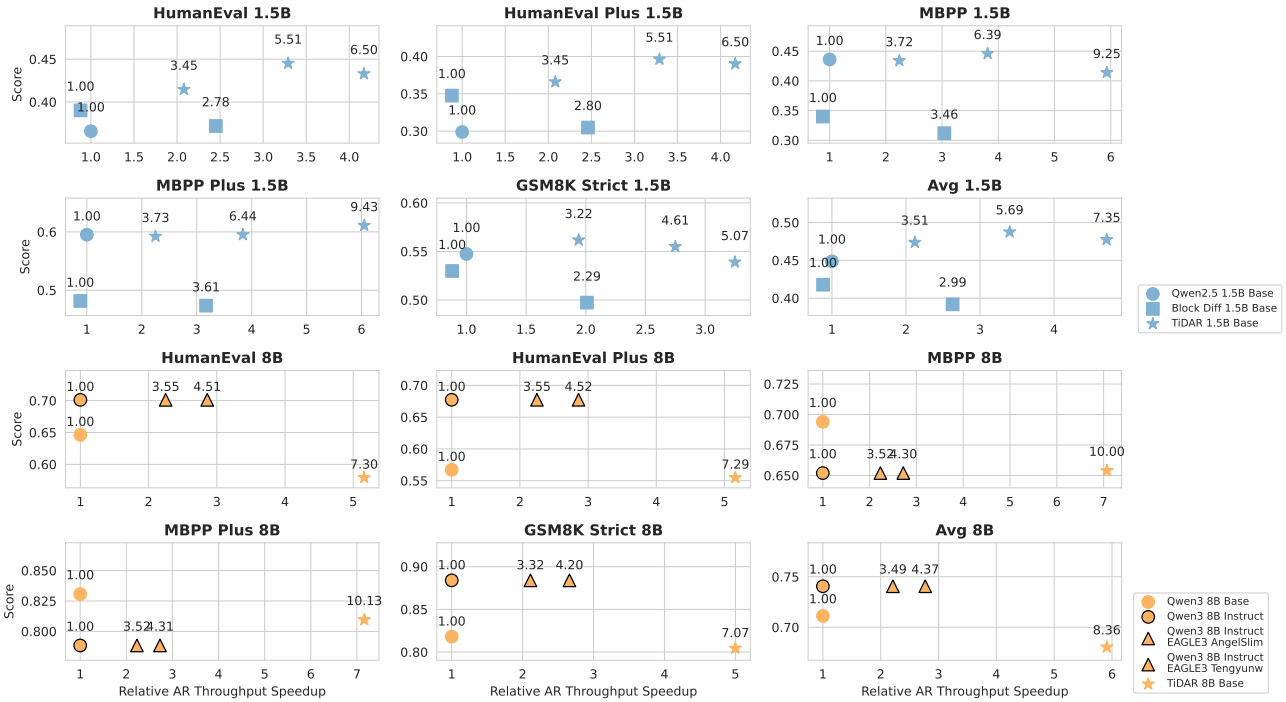

Figure 4. **Efficiency-Quality Benchmarking:** We compare TIDAR on 1.5B and 8B with AR, AR with speculative decoding (EAGLE-3), and Block Diffusion. Points colored the same indicate the same model sizes while markers suggest different methods. On the y-axis we have individual task scores. On the x-axis, we showcase the relative decoding throughput speedup measured in tokens per second, with the baseline being the AR model within the same size group (● Qwen2.5 1.5B Base, ● Qwen3 8B Base and ● Qwen3 8B Instruct). On top of each point, we report the average tokens per NFE. For 1.5B models, we showcase two and three different settings for Block Diffusion (threshold = max, 0.8, illustrated from left to right) and TIDAR (training block size = 4, 8, 16, illustrated from left to right) respectively.

Baseline We include open-sourced AR models such as Llama3.2 (Meta, 2024), SmolLM2 (Allal et al., 2025), Qwen2.5 (Yang et al., 2025b), and Qwen3 (Yang et al., 2025a) with similar model sizes. For diffusion models, we compare ours against Dream (Ye et al., 2025a) and Llada (Nie et al., 2025), as well as Block Diffusion (Arriola et al., 2025a) trained under the same training recipe. *For all Qwen2.5 and Qwen3 models, the results are referring to the base model unless explicitly specified.*

Training and inference We follow (Fu et al., 2025) in terms of the AR-to-Diffusion continual pretrain data recipe and training setup. For TIDAR 1.5B model, we continually pretrain with 50B tokens on Nvidia H100s with a global batch size of 2M tokens (DDP) under block sizes of 4, 8, and 16[2]. For the 8B model, we use 150B tokens instead, with a block size of 16, keeping the rest of the settings the same. We adopt the cosine scheduling with $max\_lr = 1e-5, min\_lr = 3e-6$ and a warm-up step fraction of 1%. The max sequence length is set to 4096 with distributed Adam (Kingma & Ba, 2017) optimizer and we turn on gradient checkpointing for the 8B model. The overall training framework is a modified Megatron-LM (Shoeybi et al., 2020) with Torchtitan (Liang et al., 2025) support. Both training and inference are conducted in the standard BFloat 16 precision. In terms of the main results, we report the performance for TIDAR 1.5B and TIDAR 8B and focus on the use of 1.5B for ablation studies.

[2]We use the model with block size of 16 during main result evaluation and the other two for ablations.

4.2 Main Results

4.2.1 *Generative Task Evaluation*

In this section, we start off by investigating the ability of TIDAR to generate high-quality samples efficiently. We focus on coding and math tasks since their metrics are much more robust. In Table 2, we first compare TIDAR against several AR models and also a popular diffusion variant, Block Diffusion, across two model sizes.

For 1.5B-1.7B size range, TIDAR is highly competitive in terms of quality with an average 7.45 tokens per model forward (NFE). For 8B models, TIDAR incurs very minimal loss with a simple training recipe while increasing the generation efficiency to 8.25 tokens per NFE. We also tested two different modes of generation, namely "trusting AR"

v.s. "trusting diffusion", which shows that for larger models, trusting the predictions from diffusion predictions is beneficial in most cases, especially for math tasks. We will discuss the choice difference more in Section 4.4.3.

Comparing with diffusion LLMs, TIDAR outperforms both public models like Dream and Llada consistently, and also our own Block Diffusion with the same training recipe. Here, we decode one token per NFE since this very often guarantees the best achievable quality for most tasks. We defer the discussion and comparison of different decoding strategies to Section 4.4.2.

In summary, TIDAR strikes a nice balance between generative quality and efficiency due to several designs we mentioned above, making it a highly appealing choice for many critical application scenarios with a stringent latency requirement.

4.2.2 *Likelihood Task Evaluation*

Evaluating model performance based on likelihood for traditional diffusion LLMs (e.g. LLaDA, Dream, MDLM) has been very challenging due to various ways of computing the likelihood. For example, response tokens will be corrupted in the same way as training, and the likelihood is calculated only on the masked positions (Nie et al., 2025), which are averaged over Monte Carlo sample budgets. Although agreement has been achieved on the use of likelihood for multiple-choice questions despite being less efficient, directly comparing models using this metric still remains a subject of debate. TIDAR, on the other hand, alleviates this problem due to the fact that its AR mode naturally supports computing the likelihood in the same way as autoregressive models. We show in Table 3 that 1) this approach is highly aligned with other AR LLMs, making it highly comparable, 2) its performance is competitive and faithful to generative quality (due to autoregressive sampling), and 3) the evaluation is extremely efficient with a single NFE.

4.3 Efficiency Benchmarking

Finally, we conclude the main results with a focus on benchmarking the wall-clock time of TIDAR against AR, AR with EAGLE-3[3] for speculative decoding, and Block Diffusion. We choose these since they are among the most competitive choices across standard autoregressive decoding, speculative decoding, and diffusion generations. All of the models have the exact cache supported and are benchmarked on a single H100 GPU with prompts from downstream generative tasks and a batch size equal to 1.

In Figure 4, we show that TIDAR 1.5B can achieve an average of 4.71x relative speedup to Qwen2.5 1.5B in terms of decoding throughput and TIDAR 8B with an average of 5.91x over Qwen3 8B while maintaining comparable performances. Comparing with Block Diffusion with two settings using different thresholds, we can also achieve better efficiency-quality trade-offs in all tasks we tested. In terms of the SOTA speculative decoding method, EAGLE-3 (Li et al., 2025), we test TIDAR against public weights from AngelSlim[4] and Tengyunw[5], in which we show for the first time that diffusion models can surpass the efficiency gains over speculative decoding. However, we want to emphasize that speculative decoding and our methods should be utilized with different purposes since the former is able to guarantee exactly the same output as the base model. In addition, we show that our raw acceptance rate (T/NFE) is higher than those of EAGLE-3 open weights, and more importantly, our conversion rate (from T/NFE to T/s) is higher, thanks to the parallel drafting and sampling with a single model forward.

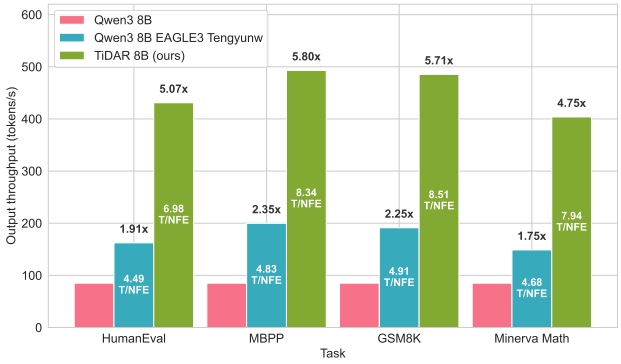

Figure 5. **Decoding TPS in SGLang:** We profile TIDAR 8B in SGLang on a single H100 in BF16 (with CUDA graph disabled), which achieves up to 5.8x decoding decoding TPS than AR, and up to 2.5x higher decoding TPS than EAGLE-3 (Tengyunw).

To project the improvement in pure PyTorch onto state-of-the-art inference engine, we also implement TiDAR in SGLang (Zheng et al., 2024) with FlashInfer (Ye et al., 2025b) (boolean attention mask) as the attention backend. We follow the implementation design of EAGLE framework and compare the decoding TPS of baseline Qwen3, Qwen3 with EAGLE-3 (Tengyunw weight) speculative decoding, and TIDAR 8B in Figure 5. TIDAR consistently achieves gains over AR (up to 5.8x higher TPS) and EAGLE-3 (up to 2.5x higher TPS).

It is worth noting that all of these methods can significantly benefit from further system optimizations such as custom

[3] We use the EAGLE-3 with Qwen3-8B instruct model due to lack of corresponding EAGLE-3 weights for the base model.

[4] https://huggingface.co/AngelSlim/Qwen3-8B_eagle3

[5] https://huggingface.co/Tengyunw/qwen3_8b_eagle3

		Coding				Math		Avg
Model Arch	**Size**	**HumanEval**	**HumanEval+**	**MBPP**	**MBPP+**	**GSM8k**	**Minerva Math**	
Llama3.2	1B	17.68%	14.63%	26.60%	38.89%	5.67%	1.92%	17.57%
SmolLM2	1.7B	0.61%	0.61%	35.40%	47.62%	28.20%	11.28%	20.62%
Qwen2.5	0.5B	27.44%	25.61%	29.60%	44.97%	37.45%	14.48%	29.92%
Qwen2.5	1.5B	35.98%	29.88%	43.60%	59.23%	54.74%	26.40%	41.64%
Qwen3	1.7B	48.17%	41.46%	55.80%	71.43%	66.72%	29.74%	52.22%
Block Diff	1.5B*	39.02%	34.76%	34.00%	48.15%	52.99%	21.56%	38.41%
TIDAR	1.5B*	43.29% (6.50)	39.02% (6.50)	41.40% (9.25)	61.11% (9.43)	53.90% (5.07)	25.48% (7.92)	44.03% (7.45)
Qwen3	4B	57.32%	50.61%	67.00%	80.69%	77.48%	47.10%	63.37%
Qwen3	8B	64.63%	56.71%	69.40%	83.07%	81.80%	52.94%	68.09%
LLaDA	8B	32.32%	27.44%	40.80%	51.85%	70.96%	27.30%	41.78%
Dream	7B	54.88%	49.39%	56.80%	74.60%	77.18%	39.60%	58.74%
Block Diff	4B†	56.10%	51.22%	54.60%	69.84%	82.87%	47.02%	60.27%
TIDAR (Trust AR)	8B‡	55.49% (7.46)	52.44% (7.44)	65.40% (9.96)	79.63% (10.13)	79.83% (6.90)	50.58% (7.48)	63.90% (8.23)
TIDAR (Trust Diff)	8B‡	57.93% (7.30)	55.49% (7.29)	65.40% (10.00)	80.95% (10.13)	80.44% (7.07)	51.64% (7.68)	65.31% (8.25)

Table 2. **Generative Evaluation Results:** We evaluate the generative sample quality of TIDAR over several coding and math tasks. For all diffusion models we decode one token per forward for the best quality and report the average number of tokens (T/NFE) generated by TIDAR in parenthesis. Models initialized from QWEN2.5 1.5B, QWEN3 4B, and QWEN3 8B are super-scripted by $\star$, †, and ‡ respectively.

		Knowledge	Commonsense Reasoning					
Model Arch	**Size**	**MMLU**	**ARC-e**	**ARC-c**	**Hellaswag**	**PIQA**	**Winogrande**	**Avg**
Llama3.2	1B	30.98%	65.28%	36.35%	63.76%	74.43%	63.30%	55.68%
SmolLM2	1.7B	49.99%	77.82%	47.44%	71.44%	77.64%	67.88%	65.37%
Qwen2.5	0.5B	47.65%	64.77%	31.83%	52.25%	70.02%	57.70%	54.04%
Qwen2.5	1.5B	60.96%	75.17%	45.05%	67.90%	76.12%	65.75%	65.16%
Qwen3	1.7B	62.53%	73.32%	44.62%	66.43%	75.63%	64.96%	64.58%
Block Diff	1.5B*	57.94%	74.41%	45.73%	56.26%	70.13%	61.80%	61.05%
TIDAR	1.5B*	58.99%	77.78%	45.39%	65.26%	75.52%	63.61%	64.43%
Qwen3	4B	73.00%	79.00%	52.00%	74.00%	78.00%	72.00%	71.33%
Qwen3	8B	76.93%	81.90%	53.16%	78.59%	79.22%	75.69%	74.25%
Block Diff	4B†	71.53%	81.48%	55.63%	65.48%	74.92%	70.96%	70.00%
Dream	7B	67.00%	82.20%	59.13%	73.73%	75.52%	73.56%	71.86%
LLaDA	8B	65.86%	73.78%	49.15%	71.05%	73.88%	74.66%	68.06%
TIDAR	8B‡	76.57%	84.18%	58.53%	76.36%	80.25%	76.48%	75.40%

Table 3. **Likelihood Evaluation Results:** We compare our method on factual knowledge and common sense reasoning tasks using likelihood estimation. For Llada, Dream, and Block Diffusion, we follow the standard diffusion likelihood evaluation using Monte Carlo (MC) sampling. For our model, we evaluate the likelihood using pure causal mask like AR models, as natively supported by our architecture. Models initialized from QWEN2.5 1.5B, QWEN3 4B, and QWEN3 8B are super-scripted by $\star$, †, and ‡ respectively. Models with *italicized* names are using MC with 128 steps for likelihood evaluation.

kernels, more efficient KV cache management, and request scheduling. Hence, the goal here is to provide a first-hand idea of their performance in native PyTorch.

4.4 Ablation Studies

The objective of this section is to understand more deeply where the performance gains come from and justify the design choices of our architecture, training, and inference.

4.4.1 Pareto Frontier under the Same Training Recipe

The quality and training recipe can drastically impact the performance of LLMs. Therefore, to fairly compare different model architectures, we train all models using the same setting starting from Qwen2.5 1.5B models. In Figure 6, the Pareto Frontier of these trained models provides insight into how TIDAR performs against the base AR, Block Diffusion, and fine-tuned AR models. We can see that with only 50B training tokens, TIDAR can achieve impressive quality while maintaining a high T/NFE compared with Block Diffusion with different thresholds used for parallel decoding. We also notice the remaining quality gap from the fine-tuned AR models, which we believe could be potentially closed with more data due to the fact that TIDAR might require a bit more knowledge due to the initial adaptation phase.

4.4.2 Comparing TIDAR *with Other Decoding Strategy*

Common strategies used in dLMs include generating a fixed number of tokens per NFE (Nie et al., 2025), dynamic number of NFEs based on entropy (Ye et al., 2025a) or

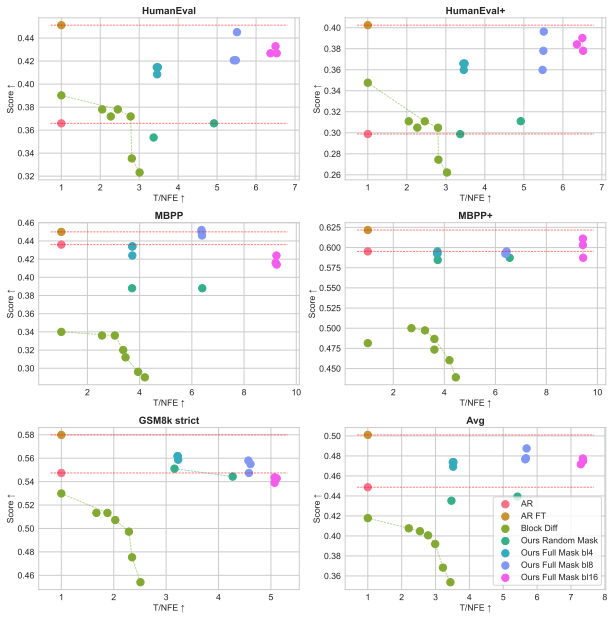

Figure 6. **Pareto Frontier of Different Architectures with the Same Recipe:** We report the performance-efficiency trade-offs on 1.5B scale among AR model, fine-tuned AR model, Block Diffusion under different decoding thresholds, and TIDAR using different drafting lengths. Our model achieves the best Pareto Frontier compared to Block Diffusion and AR and is approaching the quality of fine-tuned AR with 7x more tokens per NFE.

confidence-based decoding (Wu et al., 2025b), and also within-block left-to-right decoding. Many prior works have studied the benefits of these and, here, we provide a comprehensive comparison in Table 4 against our method. We demonstrate that our method leverages both the efficient parallelism from diffusion but also high quality from autoregression, with an added benefit of not requiring to tune any hyperparameters during decoding.

4.4.3 *Sampling with AR v.s. Diffusion Prediction*

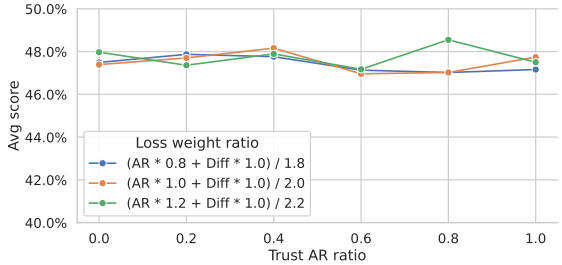

Figure 7. **Trusting AR v.s. Diffusion Outputs for Sampling:** TIDAR is trained to be highly balanced in the sense that no matter whether we trust the prediction from the diffusion or AR parts, the autoregressive sampling, along with the high drafting model capacity, can guarantee quality under almost the same speedup.

Strategy	Avg T/NFE	HumanEval Avg	MBPP Avg	GSM8k
Confidence Max	1.00	34.45%	43.92%	53.07%
	2.00	23.78%	22.19%	38.06%
Left to right (AR)	1.00	36.28%	46.51%	53.37%
	2.00	21.95%	18.61%	41.32%
Confidence > 0.9	2.63	32.01%	42.50%	51.40%
Confidence > 0.8	3.06	28.96%	39.28%	47.54%
Confidence > 0.7	3.42	27.74%	33.90%	43.44%
Confidence > 0.6	3.81	22.56%	26.47%	37.60%
TIDAR (4 drafts)	3.47	38.42%	50.96%	55.87%
TIDAR (8 drafts)	5.49	39.94%	52.13%	54.74%
TIDAR (16 drafts)	6.97	41.16%	51.26%	53.90%

Table 4. **Comparing Different Decoding Strategies:** We showcase the superiority of our proposed parallel draft and sampling process (on full mask models) compared to using standard confidence/negative entropy based decoding, as well as the autoregressive decoding schemes.

As shown in Figure 2, using no label shifts for the diffusion part makes it possible to pre-draft without waiting for AR's results at the same step while keeping the sequence consistent (i.e. the first mask would predict F'' instead without knowing the actual value of E when it is label shifted, which will degrade the quality). However, this comes at the cost that the AR outputs and every first position in the block will be predicting the same position (e.g. E to E'', F' to F'', G' to G''), leading to a potential decision of choosing what to use for verification.

Therefore, we propose to apply an aggregate function to mix their logits before sampling the token as $\arg\max_i\{\beta * logits_i^{ar} + (1-\beta) * logits_i^{diff}, i \in |V|\}$, which then gets compared against the proposed draft token. This can also be understood as whether we "trust" the AR or the diffusion outputs more when it comes to sampling.

In Figure 7, we vary the value of β (over different loss balancing factor α) and see consistent performance. This shows that our model is well trained so that no matter what logits we choose to use for verification, the quality is preserved because in the ideal case (i.e. well-trained), these two outputs will be strictly the same. This also indicates that it is the autoregressive rejection sampling that guarantees the quality-speedup trade-offs rather than the AR knowledge.

4.4.4 *How Useful is the Full Mask Strategy?*

To take advantage of the fact that the model does not see partial clean inputs during inference due to one-step diffusion drafting, we corrupt the inputs to the diffusion part to be full masks during training. This not only promotes better train-test behavior consistency but also provides richer diffusion loss signals and easier loss balancing after normalization. In Table 5, we demonstrate the effectiveness of this masking strategy, which shows a consistent quality improvement especially in coding tasks. We also want to note that this enables the flexibility of deciding which module's predictions

to trust for autoregressive rejection sampling as discussed in Section 4.4.3.

Masking	Draft	HumanEval Avg	MBPP Avg	GSM8k	Avg
Random	4	32.62% (3.37)	48.63% (3.72)	55.11% (3.16)	45.45% (3.42)
	8	33.85% (4.92)	48.77% (6.48)	54.43% (4.27)	45.68% (5.22)
Full	4	38.42% (3.46)	50.96% (3.72)	55.87% (3.23)	48.42% (3.47)
	8	39.94% (5.49)	52.13% (6.40)	54.74% (4.58)	48.94% (5.49)

Table 5. **Quality-efficiency Improvement from Full Masking:** Turning random corruption strategy to full masking on the model, we substantially improve both efficiency and quality due to less train-test discrepancy and richer diffusion loss signals. Average T/NFE is shown in the parenthesis.

5 Discussion and Limitations

Scaling Behavior of Batch Size Although we focus on batch size = 1 efficiency benchmarking, it does not mean that TIDAR cannot handle large batch size. When the batch size increases, TIDAR follows a similar diminishing return as speculative decoding methods because the overall workflow hits the compute-bound faster than normal AR, as analyzed in (Liu, 2025). However, we can adjust the block (draft) length during decoding in a zero-shot manner to accommodate different compute profiles.

Training and Inference Block Size Adjustability The design of TIDAR allows changing the block size during inference to be different from training. The accuracy is preserved due to paralleled verification but the potential speedup gain can vary. With block size interpolation (i.e. shrinking the size), TIDAR would lower the maximal acceptance length but have a lower forward cost in return. The actual choice of the inference block size, therefore, could be adjusted based on the serving compute-memory profile.

Training Cost Analysis Although TIDAR employs a full fine-tuning from AR conversion that leads to more expressive multi-token prediction than speculative decoding (in terms of number of tokens accepted per forward), we want to note that the cost for adaptation is relatively cheap. Quantitatively, TIDAR 8B's continual pretrain only consumes around 0.5% tokens of the pretraining stage according to the technical report from Qwen3 (Yang et al., 2025a). We leave other efficient fine-tuning methods like PEFT (Han et al., 2024; Hu et al., 2021) for future works.

Long context extension We believe that our model is not intrinsically limited in long context capability compared to standard AR models. Since our current implementation requires doubling the sequence length with appended mask tokens during training, we defer the exploration of efficient long context extension methods (e.g. context parallelism specifically designed for TIDAR) for future work.

System optimization We show that even writing in native PyTorch with Flex Attention and SGLang with FlashInfer boolean attention mask, TIDAR already achieved substantial throughput improvement. We believe that writing custom attention kernels and scheduling algorithms can maximize the use of the "free token slots" specific to the serving hardware in use as shown in Figure 1.

6 Conclusions

We introduce TIDAR, a sequence-level hybrid architecture that drafts (thinks) in diffusion and samples (talks) in autoregression in a single model forward with a specially designed attention mask. Taking advantage of the efficiency gains from parallel one-step diffusion and the quality guaranteed by autoregressive sampling, TIDAR achieves impressive efficiency and quality trade-offs. We show in a diverse set of downstream tasks that TIDAR 1.5B and 8B models can output with an average of 7.45 and 8.25 tokens per NFE, which translates to $4.71\times$ and $5.91\times$ more tokens per second compared to AR while maintaining competitive quality. We push the limit of "free token slots" on modern GPUs and, for the first time, show it is possible to beat speculative decoding for latency-critical applications. We believe that this will significantly motivate future research on hybrid LLM architecture and inference.

REFERENCES

Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z., Han, J., Sahoo, S. S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025a.

Arriola, M., Schiff, Y., Phung, H., Gokaslan, A., and Kuleshov, V. Encoder-decoder diffusion language models for efficient training and inference, 2025b. URL https://arxiv.org/abs/2510.22852.

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

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

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

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

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

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

Feng, G., Geng, Y., Guan, J., Wu, W., Wang, L., and He, D. Theoretical benefit and limitation of diffusion language model. *arXiv preprint arXiv:2502.09622*, 2025.

Fu, Y., Whalen, L., Ye, Z., Dong, X., Diao, S., Liu, J., Wu, C., Zhang, H., Xie, E., Han, S., Khadkevich, M., Kautz, J., Lin, Y. C., and Molchanov, P. Efficient-dlm: From autoregressive to diffusion language models, and beyond in speed, 2025. URL https://arxiv.org/abs/2512.14067.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Gat, I., Ben-Hamu, H., Havasi, M., Haziza, D., Reizenstein, J., Synnaeve, G., Lopez-Paz, D., Karrer, B., and Lipman, Y. Set block decoding is a language model inference accelerator. *arXiv preprint arXiv:2509.04185*, 2025.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Han, Z., Gao, C., Liu, J., Zhang, J., and Zhang, S. Q. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2403.14608.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Israel, D., Broeck, G. V. d., and Grover, A. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

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

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

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

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024b.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

Liang, W., Liu, T., Wright, L., Constable, W., Gu, A., Huang, C.-C., Zhang, I., Feng, W., Huang, H., Wang, J., Purandare, S., Nadathur, G., and Idreos, S. Torchtitan: One-stop pytorch native solution for production ready llm pre-training, 2025. URL https://arxiv.org/abs/2410.06511.

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

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

Liu, X. *Efficient LLM System with Speculative Decoding*. PhD thesis, University of California, Berkeley, 2025. URL https://escholarship.org/uc/item/2cm0c15n. ProQuest ID: 32402435.

Liu, Y., Cao, Y., Li, H., Luo, G., Chen, Z., Wang, W., Liang, X., Qi, B., Wu, L., Tian, C., et al. Sequential diffusion language models. *arXiv preprint arXiv:2509.24007*, 2025.

Ma, X., Yu, R., Fang, G., and Wang, X. dkv-cache: The cache for diffusion language models, 2025. URL https://arxiv.org/abs/2505.15781.

Meta, A. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. *Meta AI Blog. Retrieved December*, 20:2024, 2024.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

NVIDIA. Nvidia gpus, 2025. URL https://www.nvidia.com/. General-purpose GPU.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Sahoo, S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024a.

Sahoo, S. S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J. T., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models, 2024b. URL https://arxiv.org/abs/2406.07524.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Samragh, M., Kundu, A., Harrison, D., Nishu, K., Naik, D., Cho, M., and Farajtabar, M. Your llm knows the future: Uncovering its multi-token prediction potential, 2025. URL https://arxiv.org/abs/2507.11851.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wu, C., Zhang, H., Xue, S., Diao, S., Fu, Y., Liu, Z., Molchanov, P., Luo, P., Han, S., and Xie, E. Fast-dllm v2: Efficient block-diffusion llm, 2025a. URL https://arxiv.org/abs/2509.26328.

Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding, 2025b. URL https://arxiv.org/abs/2505.22618.

Xu, M., Geffner, T., Kreis, K., Nie, W., Xu, Y., Leskovec, J., Ermon, S., and Vahdat, A. Energy-based diffusion language models for text generation, 2025. URL https://arxiv.org/abs/2410.21357.

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

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report, 2025b. URL https://arxiv.org/abs/2412.15115.

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

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., and Ceze, L. Flashinfer: Efficient and customizable attention engine for llm inference serving, 2025b. URL https://arxiv.org/abs/2501.01005.

Yuan, Z., Shang, Y., Zhou, Y., Dong, Z., Zhou, Z., Xue, C., Wu, B., Li, Z., Gu, Q., Lee, Y. J., et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

A Evaluation Task Configuration

Task Category	Task Name	Number of Few-Shots	Gen Length	Score Metric
Coding	HumanEval (Chen et al., 2021)	0	512	Pass@1
	HumanEval+ (Liu et al., 2023)	0	512	Pass@1
	MBPP (Austin et al., 2021b)	3	512	Pass@1
	MBPP+ (Liu et al., 2023)	3	512	Pass@1
Math	GSM8K-CoT (Cobbe et al., 2021)	8	256	strict-match
	Minerva Math (Lewkowycz et al., 2022)	4	512	exact-match
Factual	MMLU (Hendrycks et al., 2020)	5	/	Acc
Commonsense	ARC-Easy (Clark et al., 2018)	0	/	Acc
	ARC-Challenge (Clark et al., 2018)	0	/	Acc Norm
	Hellaswag (Zellers et al., 2019)	0	/	Acc Norm
	PIQA (Bisk et al., 2020)	0	/	Acc Norm
	Winogrande (Sakaguchi et al., 2021)	5	/	Acc

Table 6. **Benchmark Configuration for All Tasks**

In Table 6, we detail the benchmarking configuration, including the number of few-shot prompts, generation length and performance metric, for each of tasks we adopted in the work. For HumanEval and HumanEval Plus, we apply standard post-processing for the all models. The LM_EVAL_HARNESS version we use is 0.4.8.

B Inference Prefill Mask

We apply the same technique of reusing the initialized attention mask for different prompt lengths across samples. This is achieved by reordering different parts of the input and slicing the big mask. The mask is illustrated in Figure 8.

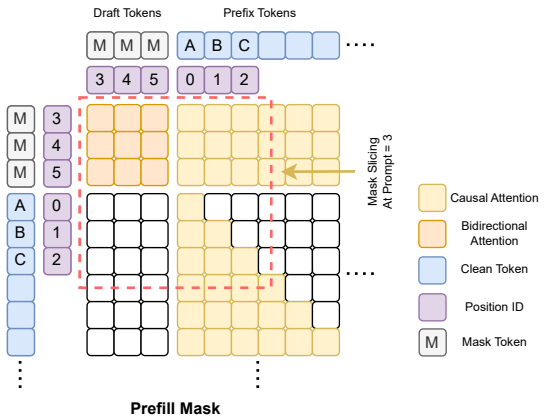

Figure 8. **Prefill Attention Mask:** We initialize the mask at the model initialization with (max_seq_len + block_size, max_seq_len + block_size) and slice it based on the current sample length.