

# REFUSION: A DIFFUSION LARGE LANGUAGE MODEL WITH PARALLEL AUTOREGRESSIVE DECODING

Jia-Nan Li<sup>1,2\*</sup> Jian Guan<sup>2\*</sup> Wei Wu<sup>2†</sup> Chongxuan Li<sup>1†</sup>

<sup>1</sup>Gaoling School of Artificial Intelligence, Renmin University of China <sup>2</sup>Ant Group

{lijianan, chongxuanli}@ruc.edu.cn

{jianguanthu, wuwei19850318}@gmail.com

Code: <https://github.com/ML-GSAI/ReFusion>

Model: <https://huggingface.co/GSAI-ML/ReFusion>

## ABSTRACT

Autoregressive models (ARMs) are hindered by slow sequential inference. While masked diffusion models (MDMs) offer a parallel alternative, they suffer from critical drawbacks: high computational overhead from precluding Key-Value (KV) caching, and incoherent generation arising from learning dependencies over an intractable space of token combinations. To address these limitations, we introduce REFUSION, a novel masked diffusion model that achieves superior performance and efficiency by elevating parallel decoding from the token level to a higher slot level, where each slot is a fixed-length, contiguous sub-sequence. This is achieved through an iterative “plan-and-infill” decoding process: a diffusion-based planning step first identifies a set of weakly dependent slots, and an autoregressive infilling step then decodes these selected slots in parallel. The slot-based design simultaneously unlocks full KV cache reuse with a unified causal framework and reduces the learning complexity from the token combination space to a manageable slot-level permutation space. Extensive experiments on seven diverse benchmarks show that REFUSION not only overwhelmingly surpasses prior MDMs with 34% performance gains and an over 18× speedup on average, but also bridges the performance gap to strong ARMs while maintaining a 2.33× average speedup.

## 1 INTRODUCTION

While autoregressive models (ARMs) (Grattafiori et al., 2024; Yang et al., 2025; Jaech et al., 2024) have achieved impressive progress across a wide range of tasks (Chen et al., 2021; Wei et al., 2022; Lightman et al., 2023; Li et al., 2024), their inference throughput is fundamentally limited by a sequential, left-to-right decoding process that precludes parallelization (Chen et al., 2023; Cai et al., 2024; Zhang et al., 2025). In contrast, masked diffusion models (MDMs) (Nie et al., 2025; Ye et al., 2025) operate via an iterative denoising process with no fixed generation order. This flexibility yields two significant advantages. First, it permits parallel decoding by assuming conditional independence among target tokens: their joint probability, given the context, is assumed to be the product of their individual marginal probabilities (Li et al., 2023). Second, it offers the potential for the model to discover better generation orders than the rigid left-to-right trajectory (Kim et al., 2025).

Despite these theoretical advantages, existing MDMs often suffer from two issues: **(1) Architectural bottlenecks negate efficiency gains from parallelism.** The flexibility of generation orders requires bidirectional attention in MDMs (Vaswani et al., 2017; Devlin et al., 2019), an architectural choice fundamentally incompatible with Key-Value (KV) caching used in ARMs (Radford et al., 2018). That is, each decoding iteration forces a full re-computation of the KV states of the entire context, introducing significant latency and making MDMs significantly slower than ARMs (Feng et al., 2025). **(2) Intractable training complexity hinders coherent parallel generation.** MDMs typically decode multiple tokens with high marginal probabilities in parallel (Nie et al., 2025). However, the

\*Equal contribution.

†Corresponding authors: Wei Wu and Chongxuan Li.

conditional independence assumption frequently fails for these tokens, particularly for nearby tokens, leading to severe incoherence (Huang et al., 2022; Luxembourg et al., 2025; Gwak et al., 2025). For example, in a context where both “at once” and “right now” are valid, an MDM might decode a spurious output “right once” by independently sampling tokens with high individual marginal probabilities but low joint probability. We attribute this failure to an immense learning challenge: modeling a data distribution over an exponential space of possible token combinations is far more demanding than the fixed sequential dependency of ARMs. Consequently, current MDMs often remain undertrained for reliably identifying conditionally independent tokens.

To address these challenges, we introduce REFUSION, a masked diffusion large language model that elevates the parallel decoding process from the traditional token level to a higher slot level, progressively denoising slots in parallel from a fully masked sequence. Specifically, we partition the initially masked sequence into fixed-length, consecutive sub-sequences, i.e., slots, and unfold each decoding iteration into two synergistic steps: first, a diffusion-based planning step identifies a set of weakly dependent slots, and second, an autoregressive infilling step decodes these slots in parallel (Figure 2). This design is guided by a key finding from our pilot study (§4.1): inter-token dependency is highly localized, decaying significantly with distance. Therefore, serializing adjacent tokens within a slot directly mitigates the violation of conditional independence for these strongly-coupled tokens. Furthermore, such a simple design also uniquely achieves two critical benefits simultaneously: (1) By repositioning newly generated slots (Sahoo et al., 2025) to precede masked ones for the next decoding iteration, REFUSION naturally accommodates causal attention that allows the KV states of *all* previously generated tokens to be seamlessly reused; And (2) it reduces the learning complexity from an intractable token combination space to a substantially more manageable slot permutation space. Appendix A.1 compares REFUSION and existing MDMs in detail.

REFUSION’s training process mirrors its inference dynamics. For each training sequence, we randomly mask several slots, permute the clean slots, and reorder the input so that clean slots precede masked ones. The model is then optimized with a hybrid objective that cultivates its dual capabilities: an autoregressive loss on the permuted clean slots for sequential generation, and a denoising loss on the masked slots for context-aware parallel reconstruction. Unlike traditional MDMs which learn only from masked positions, this hybrid objective uses every token for supervision, boosting data efficiency.

Our extensive experiments on seven benchmarks spanning math, code generation, and general-purpose understanding and reasoning demonstrate that REFUSION decisively establishes a new state-of-the-art for MDMs. Compared to LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025), REFUSION achieves an average performance gain of 34% while being over 18× faster in throughput (tokens/sec).

More strikingly, REFUSION consistently challenges and often surpasses strong ARMs. For instance, it outperforms Qwen3-8B (Yang et al., 2025) on GSM8K (Cobbe et al., 2021) and MBPP (Austin et al., 2021) by 3.68 absolute points while being 2.33× faster on average. This dual advantage in both performance and speed is further illustrated in Figure 1, where REFUSION (the red point) pushes the performance-efficiency boundary significantly towards the top-right quadrant. It significantly outperforms both the Qwen3-series (the blue line) and prior MDM-based methods, which are situated

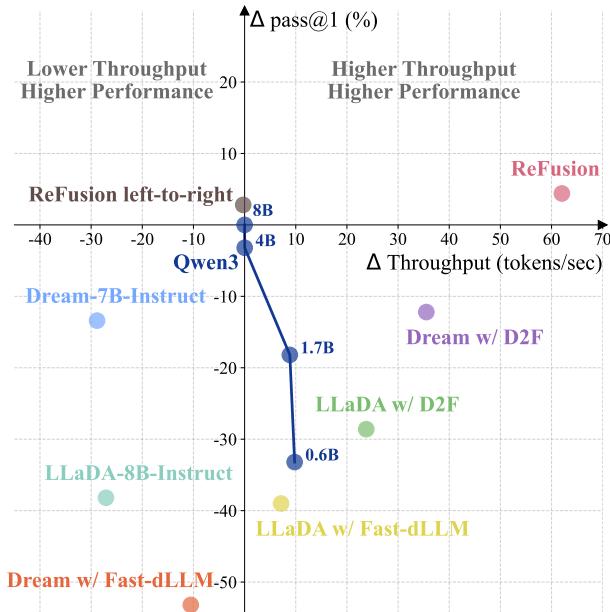


Figure 1: Performance-throughput trade-off on MBPP. We plot pass@1 (%) against throughput (tokens/sec), with both metrics calculated relative to the Qwen3-8B baseline at the origin. The “REFUSION left-to-right” ablation forces serial decoding using the REFUSION model.

in the bottom-left or bottom-right regions, indicating that they lag behind in throughput, performance, or both dimensions. Furthermore, our controlled experiments confirm that these gains are driven by our architectural and training innovations, rather than initialization or data advantages.

Our contributions are summarized as follows:

- I. We propose REFUSION, a generative model integrating inter-slot parallel decoding with intra-slot autoregressive decoding, combining the strengths of autoregRESSive and difFUSION-based modeling.
- II. To the best of our knowledge, REFUSION is the first MDM that achieves full KV cache reuse of every decoded token, while maintaining global generation flexibility and tractable training complexity.
- III. Extensive experiments on seven diverse benchmarks show that REFUSION not only overwhelmingly surpasses all prior MDMs in both performance and speed, but also bridges the performance gap to ARMs while maintaining the efficiency advantage.

## 2 RELATED WORK

MDMs promise to outperform traditional ARMs by offering faster inference through parallel decoding and potentially superior solutions via flexible generation orders (Kim et al., 2025). Recent MDMs such as LLaDA (Nie et al., 2025), the first open-source MDM trained from scratch, and Dream (Ye et al., 2025), initialized from an ARM, have delivered performance on par with ARMs of equivalent scale across diverse tasks, establishing MDMs as a viable research direction.

**Architectural Designs for Efficient MDMs.** Standard MDMs’ reliance on bidirectional attention precludes the use of KV caching. Recent work alleviates this bottleneck through three main strategies. The first strategy approximates KV cache reuse while retaining bidirectional attention. For instance, dLLM-Cache (Liu et al., 2025) reuses slow-changing KV states, while sparse-dLLM (Song et al., 2025) dynamically prunes non-critical KV states. The second strategy mixes bidirectional attention and causal attention. Models like BD3-LMs (Arriola et al., 2025) and Fast-dLLM (Wu et al., 2025a) partition the sequence into consecutive blocks, enforcing a left-to-right order between blocks to enable KV cache reuse, while retaining parallel, bidirectional generation within each block. D2F (Wang et al., 2025) further parallelizes the generation of succeeding blocks, although performance is limited by the lack of inter-block lookahead attention. While sharing the concept of a grouped unit, REFUSION’s “slot” operates fundamentally differently from “block” in these approaches. In the block-based design, a fixed left-to-right inter-block schedule sacrifices any-order flexibility, while intra-block bidirectional attention sacrifices KV caching and risks incoherence. In contrast, REFUSION enables both global any-order generation and full KV cache reuse with a unified causal framework. The final strategy leverages only causal attention, enabling an exact KV cache. Eso-LMs (Sahoo et al., 2025), for instance, dynamically reposition newly generated tokens ahead of masked ones at each step to facilitate caching. However, this strategy introduces an intractable learning objective at a token-level permutation space, which hinders training and leads to significant performance drops.

**Decoding Strategies in MDMs.** A crucial aspect of MDM inference is the strategy used to select which tokens to decode in parallel at each step. Existing approaches generally fall into two categories. The first class leverages confidence heuristics derived from the model’s own distribution, such as top token probability (Nie et al., 2025), low entropy (Ben-Hamu et al., 2025), and probability margins between top candidates (Kim et al., 2025). Some methods further refine these heuristics with position-aware weights and frequency-based calibration (Huang et al., 2025). While simple, these methods rely on the often-unreliable assumption that the model’s confidence scores are perfectly calibrated (Wu et al., 2025a). The second class employs external models for verification, e.g., using a small ARM to validate and extend the longest acceptable prefix (Hu et al., 2025; Israel et al., 2025), or using dedicated reward models to guide generation (Gwak et al., 2025). Although effective, these approaches introduce the overhead of maintaining and querying a separate model. Unlike these methods, REFUSION adopts a unified inference framework that benefits from the parallel efficiency of MDMs without sacrificing the quality assurance of ARMs, all within a single architecture.

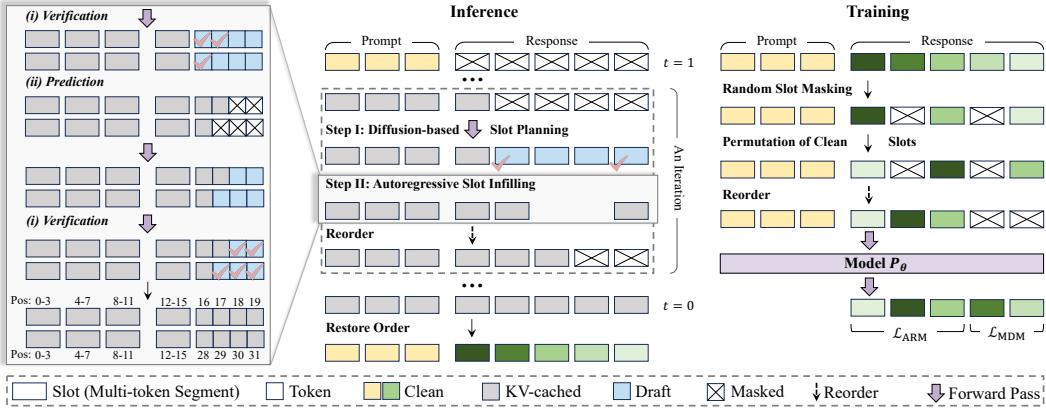


Figure 2: Overview of REFUSION. Left (Inference): An iterative slot-level “plan-and-infill” loop. A diffusion step plans and drafts slots, followed by parallel autoregressive verify-and-predict infilling. Reordering generated slots before masked ones enables full KV cache reuse, while position IDs correspond to their ground-truth indices, invariant to reordering. Right (Training): Mirrors inference, optimizing a hybrid objective of autoregressive loss ( $\mathcal{L}_{\text{ARM}}$ ) on permuted clean slots and denoising loss ( $\mathcal{L}_{\text{MDM}}$ ).

### 3 PRELIMINARY

**Autoregressive Models.** ARMs are a prominent class of generative models that factorize the joint probability of a sequence  $x = (x_1, \dots, x_L)$  by enforcing a strict left-to-right conditional dependency using a causal attention mask. This structure leads to a next-token prediction objective, where the model parameters  $\theta$  are optimized by minimizing the negative log-likelihood:  $-\sum_{i=2}^L \log P_\theta(x_i | x_{<i})$ . During inference, generation is an inherently sequential process requiring  $T$  forward passes to produce a sequence of length  $T$ , resulting in a latency that scales with the sequence length.

**Masked Diffusion Models.** MDMs represent another class of generative models, operate on a “mask-and-denoise” principle. During training, each sample  $x_0 = (x_0^1, x_0^2, \dots, x_0^L)$  is corrupted to  $x_t$  by masking each token with a special token “[MASK]” under probability  $t \sim U(0, 1)$ . The model learns to reconstruct the original context by minimizing the objective:  $-\frac{1}{t} \sum_{i=1}^L \mathbf{1}(x_t^i = [\text{MASK}]) \log P_\theta(x_0^i | x_t)$ . MDM inference proceeds by progressively generating tokens from a fully masked sequence. It requires fewer forward passes than an ARM thanks to parallel decoding, but each pass is drastically more expensive due to its incompatibility with KV caching.

## 4 METHODOLOGY

Traditional MDMs allow a flexible token-level decoding process during inference. We elevate this concept to operate on slots, i.e., a fixed-length, non-overlapping sequence of continuous tokens, denoising them in parallel. Consequently, our approach yields two critical benefits: it enables full KV cache reuse by arranging newly generated slots before masked ones with a causal framework, and it substantially reduces training complexity from the token-level combination space to a more manageable slot-level permutation space. To support non-sequential generation, we build REFUSION upon a standard causal architecture with a key modification: it accepts an explicit, non-contiguous list of position IDs. By applying RoPE (Su et al., 2021) to these absolute position IDs, the model can correctly compute relative distances and attend to all logical predecessors. Figure 2 illustrates the inference and training process.

### 4.1 LOCALITY OF INTER-TOKEN DEPENDENCY

A cornerstone of REFUSION is grouping contiguous tokens into slots for serial generation. This design is motivated by the critical insight that the conditional independence assumption is most prone to failure for nearby tokens, frequently leading to semantic incoherence (Luxembourg et al., 2025).

To formalize this insight and guide our design, we conduct a pilot study to quantitatively investigate *how dependency strength between two tokens correlates with their relative distance*.

Formally, we define the dependency strength between two tokens,  $x_0^i$  and  $x_0^j$ , in a given context  $x_t$ , as the degree to which the presence of  $x_0^j$  influences the model’s prediction of  $x_0^i$ . In practice, we approximate this measurement in a pilot study on the GSM8K test set (Cobbe et al., 2021). For a corrupted sequence  $x_t$ , we first reveal the ground-truth token  $x_0^j$  at a randomly selected masked position  $j$ , and then quantify the influence of this reveal on the prediction at any other masked position  $i$  through the Jensen-Shannon (JS) divergence (Manning & Schutze, 1999) between the distributions before and after this reveal, i.e.,  $p(x_0^i|x_t)$  and  $p(x_0^i|x_t, x_0^j)$ . A higher divergence implies stronger dependency, with zero divergence indicating conditional independence. Using both LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025), we plot the averaged JS divergence against the relative distance between positions  $i$  and  $j$  in Figure 3. The average JS divergence consistently decays as the relative distance increases, and this decay is more rapid in denser contexts (i.e., lower masking ratios  $t$ ). This result directly motivates us to define a slot as a contiguous token sequence, thereby grouping strongly correlated tokens for serial decoding within a slot, in contrast to prior block-based methods that decode nearby tokens within a block in parallel (Arriola et al., 2025).

#### 4.2 SYNERGISTIC DECODING ALGORITHM AT INFERENCE

Armed with the quantitative evidence that inter-token dependency is highly localized, we design the inference algorithm to explicitly leverage this property. The process iteratively generates a final response  $\tilde{r}_0$  from a prompt  $p_0$ , starting with an initial masked sequence  $\tilde{r}_1$ . This sequence is partitioned into  $K$  consecutive slots of  $k$  “[MASK]” tokens each. Each iteration comprises two synergistic steps: (1) diffusion-based slot planning to identify slots that are strongly dependent on the context but weakly interdependent; and (2) autoregressive slot infilling to decode them in parallel.

**Step I: Diffusion-based Slot Planning.** The first step leverages the model’s MDM capability to plan the next decoding slots. At a timestep  $t$ , which is defined as the ratio of remaining masked slots, we construct the input  $\tilde{S}_t$  by concatenating already-decoded slots ( $\tilde{S}_t^{\text{clean}}$ , in generation order) with the masked slots ( $\tilde{S}_t^{\text{masked}}$ , in their original positional order). This ordering naturally enables KV cache reuse. The model then computes a certainty score for each masked slot based on its predictive distribution. While various heuristics exist for certainty score, we adopt a simple yet effective one: the probability of the most likely token at the slot’s first position. Finally, a batch of slots with scores exceeding a threshold  $\tau_{\text{slot}}$  is selected for subsequent infilling. This strategy identifies slots that are strongly constrained by the existing context and weakly interdependent (e.g., distinct function definitions in code generation), making them suitable to parallelize. Furthermore, to accelerate the subsequent autoregressive generation, we adopt a strategy from speculative decoding (Leviathan et al., 2023): for each selected slot, we generate a corresponding draft slot by sampling a token from its distribution at every position, yielding the draft slots  $\tilde{S}_t^{\text{draft}}$ .

**Step II: Autoregressive Slot Infilling.** The second step efficiently verifies the draft slots  $\tilde{S}_t^{\text{draft}}$  and completes them using the model’s autoregressive capability, while employing a speculative decoding strategy powered by the model’s MDM capability to accelerate the process. **(1) Global Verification.** We first concatenate all draft slots into a single sequence in their original positional order. The model performs a single forward pass to compute the conditional probability for each token, conditioned on the prompt and the already-decoded slots. We identify the longest prefix of this concatenated sequence where all token probabilities exceed  $\tau_{\text{token}}$ . If this prefix spans one or more complete slots, we accept these slots wholesale and immediately proceed to the next planning iteration, thereby bypassing costly suffix completion. **(2) Parallel Iterative Completion.** If global

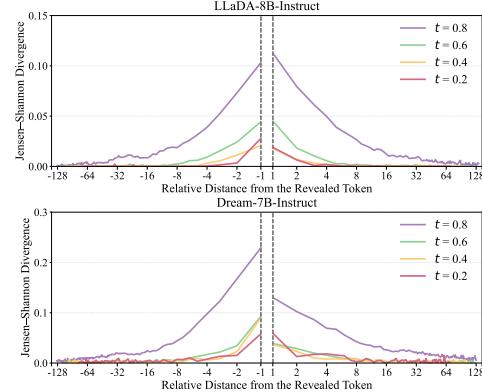


Figure 3: The locality of inter-token dependency in MDMs, with the sign on the x-axis denoting the direction from the revealed token (positive for rightward, negative for leftward).

verification fails to yield complete slots, we revert to processing the draft slots independently via a parallel iterative completion process. In each iteration, we execute two steps: (i) *Verification*: we independently identify the longest valid prefix for each slot where conditional probabilities exceed  $\tau_{\text{token}}$ ; (ii) *Prediction*: we retain this prefix, re-mask the remaining suffix, and predict the masked tokens conditioned on the context and the prefix using the model’s MDM capability. This cycle repeats until all slots are fully completed.

Finally, the newly completed slots are appended to the sequence of decoded slots. Their KV caches are directly concatenated for future iterations, a valid approximation as the lack of inter-slot conditioning during parallel generation has minimal impact on final performance (see §5.5). The plan-and-infill cycle continues until all slots are filled, at which point the final response is constructed by restoring the original slot order. The decoding process is formalized in Appendix A.2.

#### 4.3 TRAINING OF REFUSION

The training procedure for REFUSION is carefully designed to mirror the dynamics of our two-step decoding algorithm. This requires a data construction strategy that simulates the non-sequential, partially-decoded states encountered during generation, and a hybrid training objective that jointly optimizes the model’s planning and infilling capabilities.

**Training Data Construction.** To simulate the partially decoded states encountered during iterative generation, we introduce a three-step strategy to construct training data from each prompt-response pair  $(p_0, r_0)$ . The response  $r_0$  is first partitioned into a sequence of  $K$  slots,  $S_0 = [S_0^1, \dots, S_0^K]$ , each of size  $k$ . Then, a corrupted version  $S_t$  is constructed given a masking ratio  $t \sim U(0, 1)$  as follows: (1) **Random slot masking.** Analogous to token-level masking in traditional MDMs, we randomly select and mask  $\lfloor tK \rfloor$  slots from the original sequence  $S_0$ . Each selected slot is replaced with a block of  $k$  “[MASK]” tokens. (2) **Permutation of clean slots.** Since the generation order of slots is dynamically determined, the model must learn to process context in any arbitrary permutation. To achieve this, we randomly permute the unmasked (clean) slots to form  $S_t^{\text{clean}}$ , while keeping the original relative positions of the masked slots to form  $S_t^{\text{masked}}$ . (3) **Reorder.** The final training instance  $S_t$  is assembled by concatenating the permuted clean slots followed by the masked slots.

**Hybrid Training Objective.** To empower our model with the dual capabilities of global planning and local decoding, we propose a hybrid training objective that learns from every token in the sequence. This approach also provides a significant benefit of data efficiency, which contrasts with traditional MDMs where clean tokens only serve as context and provide no direct supervision.

On one hand, the clean slots  $S_t^{\text{clean}}$  are trained with a standard ARM loss for next token prediction:

$$\mathcal{L}_{\text{ARM}} = -\mathbb{E}_{\substack{(p_0, r_0) \sim \mathcal{D} \\ t \sim U(0, 1)}} \left[ \frac{1}{(k-1) \cdot |S_t^{\text{clean}}|} \sum_{i=1}^{|S_t^{\text{clean}}|} \sum_{j=2}^k \log P_\theta(v_t^{i,j} | p_0, S_{t,<(i,j)}^{\text{clean}}) \right], \quad (1)$$

where  $v_t^{i,j}$  is the  $j$ -th token in the  $i$ -th clean slot,  $S_{t,<(i,j)}^{\text{clean}}$  is the prefix of the token in  $S_t^{\text{clean}}$ .

On the other hand, the masked slots  $S_t^{\text{masked}}$  are trained with an MDM objective for denoising:

$$\mathcal{L}_{\text{MDM}} = -\mathbb{E}_{\substack{(p_0, r_0) \sim \mathcal{D} \\ t \sim U(0, 1)}} \left[ \frac{1}{k \cdot |S_t^{\text{masked}}|} \sum_{i=1}^{|S_t^{\text{masked}}|} \sum_{j=1}^k \log P_\theta(v_0^{i,j} | p_0, S_t^{\text{clean}}, S_{t,\leqslant(i,j)}^{\text{masked}}) \right], \quad (2)$$

where  $v_0^{i,j}$  is the ground-truth token from the original response corresponding to the  $j$ -th token in the  $i$ -th slot of  $S_t^{\text{masked}}$ . The final training objective is a summation of the two losses, balanced by  $\lambda$ :

$$\mathcal{L} = \mathcal{L}_{\text{ARM}} + \lambda \mathcal{L}_{\text{MDM}}. \quad (3)$$

We initialize our model  $P_\theta$  with an off-the-shelf ARM backbone, a strategy validated by prior work (Gong et al., 2025; Ye et al., 2025). Crucially, all tokens retain their original positional indices from  $r_0$  throughout the process. This allows the model to maintain awareness of the relative positions among all tokens, ensuring sequence coherence despite the shuffled input order.

Table 1: Zero-shot performance and throughput (TPS) comparison on multiple benchmarks. Each model displays accuracy/pass@1 (top row) and throughput (TPS, bottom row). Within the MDM category, we highlight the best performance results in **bold** and underline the second best. An *italic* score in the ARM category signifies that it surpasses the best-performing MDM.

| Model                         | MMLU-Pro            | ARC-C               | GSM8K               | MATH                | GPQA                | HumanEval            | MBPP                | Avg.                |
|-------------------------------|---------------------|---------------------|---------------------|---------------------|---------------------|----------------------|---------------------|---------------------|
| <i>Autoregressive Model</i>   |                     |                     |                     |                     |                     |                      |                     |                     |
| <b>Llama-3-8B-Instruct</b>    | 35.23               | 82.76               | 75.13               | 25.48               | 29.46               | 46.34                | 53.00               | 49.63               |
|                               | <i><u>32.07</u></i> | <i><u>44.12</u></i> | <i><u>42.81</u></i> | <i><u>19.73</u></i> | <i><u>42.00</u></i> | <i><u>42.26</u></i>  | <i><u>41.68</u></i> | <i><u>37.81</u></i> |
| <b>Qwen3-8B</b>               | 67.25               | 90.36               | 81.96               | 83.28               | 39.06               | 87.80                | 63.80               | 73.36               |
|                               | <i><u>31.42</u></i> | <i><u>42.78</u></i> | <i><u>31.20</u></i> | <i><u>30.11</u></i> | <i><u>30.43</u></i> | <i><u>30.95</u></i>  | <i><u>30.07</u></i> | <i><u>32.42</u></i> |
| <i>Masked Diffusion Model</i> |                     |                     |                     |                     |                     |                      |                     |                     |
| <b>LLaDA-8B-Instruct</b>      | 35.80               | 85.58               | 76.35               | 38.78               | <u>32.37</u>        | 45.12                | 25.60               | 48.51               |
|                               | <i><u>18.21</u></i> | <i><u>0.03</u></i>  | <i><u>27.35</u></i> | <i><u>23.93</u></i> | <i><u>1.99</u></i>  | <i><u>12.42</u></i>  | <i><u>2.97</u></i>  | <i><u>12.41</u></i> |
| <b>LLaDA w/ Fast-dLLM</b>     | 35.02               | 82.85               | 76.27               | 38.58               | 28.35               | 37.80                | 24.80               | 46.24               |
|                               | <i><u>39.81</u></i> | <i><u>0.86</u></i>  | <i><u>73.07</u></i> | <i><u>52.23</u></i> | <i><u>17.54</u></i> | <i><u>62.52</u></i>  | <i><u>37.19</u></i> | <i><u>40.46</u></i> |
| <b>LLaDA w/ D2F</b>           | 22.84               | 84.13               | 39.04               | 23.68               | 31.25               | 36.59                | 35.20               | 38.96               |
|                               | <i><u>44.54</u></i> | <i><u>3.70</u></i>  | <i><u>82.59</u></i> | <i><u>59.48</u></i> | <i><u>23.84</u></i> | <i><u>96.90</u></i>  | <i><u>53.85</u></i> | <i><u>52.13</u></i> |
| <b>Dream-7B-Instruct</b>      | 40.05               | <i><u>88.31</u></i> | <i><u>76.42</u></i> | <i><u>46.60</u></i> | 30.36               | <i><u>56.71</u></i>  | 50.40               | <i><u>55.55</u></i> |
|                               | <i><u>15.98</u></i> | <i><u>0.06</u></i>  | <i><u>20.30</u></i> | <i><u>18.99</u></i> | 1.81                | 3.51                 | 1.23                | 8.84                |
| <b>Dream w/ Fast-dLLM</b>     | 40.36               | 86.86               | 75.82               | 36.76               | 31.25               | 56.10                | 10.60               | 48.25               |
|                               | <i><u>47.18</u></i> | <i><u>1.42</u></i>  | <i><u>61.49</u></i> | <i><u>58.24</u></i> | <i><u>22.96</u></i> | <i><u>49.73</u></i>  | <i><u>19.55</u></i> | <i><u>37.22</u></i> |
| <b>Dream w/ D2F</b>           | 23.82               | 85.92               | 41.62               | 29.98               | 30.13               | 50.00                | <i><u>51.60</u></i> | 44.72               |
|                               | <i><u>56.43</u></i> | <i><u>59.08</u></i> | <i><u>79.20</u></i> | <i><u>84.14</u></i> | <i><u>49.96</u></i> | <i><u>69.15</u></i>  | <i><u>65.59</u></i> | <i><u>66.22</u></i> |
| <b>REFUSION</b>               | <b>45.94</b>        | <b>89.76</b>        | <b>84.91</b>        | <b>54.22</b>        | <b>35.49</b>        | <b>78.66</b>         | <b>68.20</b>        | <b>65.31</b>        |
|                               | <i><u>52.74</u></i> | <i><u>32.46</u></i> | <i><u>81.24</u></i> | <i><u>81.77</u></i> | <i><u>64.11</u></i> | <i><u>103.90</u></i> | <i><u>92.09</u></i> | <i><u>72.62</u></i> |

## 5 EXPERIMENTS

### 5.1 EXPERIMENTAL SETUP

**Implementation Details.** We initialize REFUSION from the Qwen3-8B checkpoint (Yang et al., 2025) and fine-tune it for 4 epochs on a diverse 3.7M-sample dataset ( $\sim 1.22$ B-tokens) covering mathematics, coding, and general instruction-following. For inference, we employ a semi-autoregressive remasking strategy (Nie et al., 2025): the output sequence is partitioned into blocks of size  $b$ , which are decoded serially. Within each block, our plan-and-infill algorithm from §4.2 is applied. Detailed implementation and hyperparameter specifics are provided in Appendix B.1 and B.2, respectively.

**Evaluation Benchmarks and Metrics.** To comprehensively evaluate REFUSION, we test its performance on diverse benchmarks spanning: (1) General-purpose understanding and reasoning: MMLU-Pro (Wang et al., 2024) and ARC-C (Clark et al., 2018); (2) Mathematical and scientific problem-solving: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and GPQA (Rein et al., 2024); (3) Code generation: HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). We use pass@1 for code generation and accuracy for the others. We further assess inference throughput using tokens decoded per second (TPS) with a single A100 GPU and a batch size of 1.

**Baselines.** We evaluate REFUSION against three categories of baselines<sup>1</sup>: (1) ARMs: Llama-3-8B-Instruct (AI@Meta, 2024) and Qwen3-8B (Yang et al., 2025). (2) MDMs: LLaDA-8B-Instruct (Nie et al., 2025), and Dream-7B-Instruct (Ye et al., 2025). (3) State-of-the-art MDM acceleration methods: Fast-dLLM (Wu et al., 2025b) and D2F (Wang et al., 2025). We implement the baseline methods based on their official hyperparameters<sup>2</sup>.

<sup>1</sup>We omit BD3-LMs due to its significantly smaller scale (0.2B parameters). However, its decoding strategy is shared by other baselines like LLaDA and Fast-dLLM, whose performance serves as a proxy. Nevertheless, we provide a comparison with a retrained BD3-LMs under unified settings in §5.3.

<sup>2</sup>Appendix C.1 provides comparisons against a broader set of models.

## 5.2 MAIN RESULTS

The main results in Table 1 highlight two key findings: **(1) REFUSION dominates all MDM baselines.** REFUSION consistently outperforms all MDM baselines in both performance and throughput (TPS) across all seven benchmarks, often by a substantial margin. For instance, on HumanEval, it achieves 78.66% pass@1, surpassing the next-best MDM (Dream-7B-Instruct) by nearly 22 absolute points. While acceleration methods like Fast-dLLM and D2F improve throughput at a significant performance cost, they still fall short of REFUSION’s efficiency. Notably, on MBPP, REFUSION reaches 92.09 TPS, which is  $1.4\times$  faster than the next-fastest MDM (Dream w/ D2F). REFUSION thus delivers both state-of-the-art performance and superior efficiency, establishing a new frontier for MDMs<sup>3</sup>. **(2) REFUSION challenges strong ARMs.** More remarkably, REFUSION challenges and often surpasses strong ARMs. It delivers an average speedup of  $2.33\times$  over Qwen3-8B across all tasks while exhibiting superior performance on several benchmarks. For instance, on GSM8K and MBPP, it outperforms Qwen3-8B by 3.68 absolute points. This demonstrates that our non-autoregressive approach can break the long-standing trade-off between the speed of MDMs and the quality of ARMs (Feng et al., 2025).

## 5.3 CONTROLLED COMPARISON

To isolate the benefits of REFUSION from data or backbone advantages, we adopted a controlled approach: retraining reproducible baselines using unified settings, while retraining REFUSION to match the configurations of non-open-source baselines.

Table 2: Controlled comparison of models initialized from Qwen3-8B and trained on 120K subset.

| Model                       | MMLU-Pro       | ARC-C          | GSM8K          | MATH           | GPQA           | HumanEval      | MBPP           | Avg.           |
|-----------------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| <b>Qwen3-8B (Retrained)</b> | 52.71<br>31.32 | 88.65<br>30.46 | 87.57<br>31.03 | 67.84<br>30.73 | 33.26<br>26.70 | 53.66<br>30.73 | 59.20<br>30.16 | 63.27<br>30.16 |
| <b>LLaDA (Retrained)</b>    | 5.12<br>0.26   | 70.65<br>0.03  | 13.42<br>1.18  | 1.38<br>0.20   | 0.00<br>0.15   | 4.27<br>0.66   | 6.20<br>0.42   | 14.43<br>0.41  |
| <b>BD3-LMs (Retrained)</b>  | 26.11<br>16.82 | 78.24<br>2.19  | 83.55<br>16.35 | 47.32<br>16.90 | 29.02<br>2.82  | 59.15<br>15.59 | 46.80<br>14.09 | 52.88<br>12.11 |
| <b>REFUSION (Retrained)</b> | 42.14<br>40.56 | 84.81<br>29.01 | 80.74<br>53.96 | 51.78<br>77.42 | 31.70<br>46.65 | 70.12<br>58.40 | 58.20<br>67.35 | 59.93<br>53.34 |

**Reproducible Baselines.** We conduct a controlled comparison using a 120K data subset randomly sampled from the full 3.7M dataset due to resource constraints. Specifically, we fine-tune Qwen3-8B, LLaDA, BD3-LMs<sup>4</sup>, and REFUSION for 10 epochs using their respective original objectives, with all models initialized from Qwen3-8B. This setup ensures that observed differences are attributable solely to the algorithm design. Appendix C.3 discusses the scaling properties of REFUSION regarding data size.

Results in Table 2 confirm the architectural superiority of REFUSION. LLaDA suffers a catastrophic performance collapse, and BD3-LMs lags behind REFUSION in both performance and speed. We note that the already highly-optimized Qwen3-8B baseline understandably degrades when retrained on our smaller, open-source dataset. However, under this controlled setting (with data advantages eliminated), REFUSION still outperforms it by  $\sim 16$  points on HumanEval while being  $1.9\times$  faster. This result robustly validates that REFUSION’s architectural innovations are the primary driver of its success, enabling effective learning even from limited data where standard MDMs fail.

**Non-Open-Sourced Baseline.** Furthermore, we conduct a controlled comparison with Dream-7B-Instruct. Since its training code and data processing details are not open-sourced, we cannot retrain it on Qwen3-8B. Instead, we train a REFUSION variant on Dream’s original Qwen2.5-7B backbone. It is crucial to note the significant disparity in training resources: Dream benefits from massive pre-training (580B tokens, 146.5M samples) followed by SFT (1.8M samples), whereas our REFUSION variant is exclusively fine-tuned (3.7M samples) without any pre-training. Despite this disadvantage, Table 3 shows that REFUSION still achieves a 2.23% average performance gain and a massive  $11.05\times$  speedup over Dream. REFUSION significantly excels on reasoning and coding tasks (GSM8K, HumanEval, MBPP). Its lower performance on knowledge-intensive tasks

<sup>3</sup>Appendix C.2 further analyzes the trade-off frontiers of different models.

<sup>4</sup>The block size is set to 8, which is the minimum block size used in the REFUSION experiments.

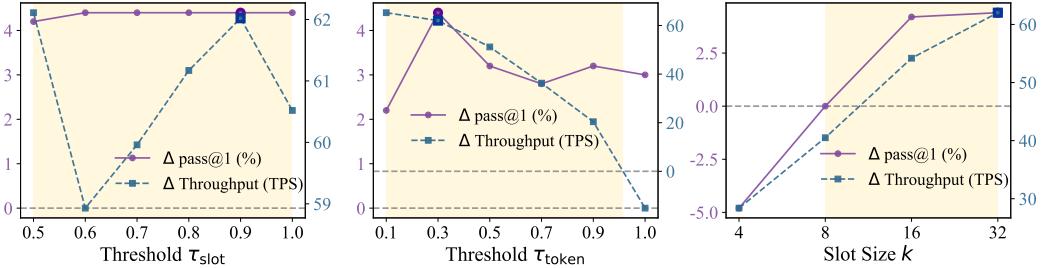


Figure 4: Impact of key hyperparameters on MBPP (0-shot). The plots show the change in **pass@1 (%)** and **throughput (TPS)** of REFUSION relative to Qwen3-8B (horizontal dashed lines at zero). When one parameter is varied, others are held at their default values ( $\tau_{\text{slot}} = 0.9$ ,  $\tau_{\text{token}} = 0.3$ ,  $k = 32$ ). Yellow shaded regions highlight the “sweet spot” where REFUSION surpasses the baseline in both metrics.

(MMLU-Pro, ARC-C) is expected, as it skips the pre-training stage that Dream utilizes for knowledge injection. These results collectively confirm that REFUSION’s architectural advantages are robust across different base models and training setups.

Table 3: Controlled comparison with Dream-7B-Instruct on its native Qwen2.5-7B backbone.

| Model                       | MMLU-Pro     | ARC-C       | GSM8K        | MATH         | GPQA        | HumanEval   | MBPP        | Avg.        |
|-----------------------------|--------------|-------------|--------------|--------------|-------------|-------------|-------------|-------------|
| <b>Dream-7B-Instruct</b>    | 40.05        | 88.31       | 76.42        | 46.60        | 30.36       | 56.71       | 50.40       | 55.55       |
| <b>REFUSION (Retrained)</b> | <b>15.98</b> | <b>0.06</b> | <b>20.30</b> | <b>18.99</b> | <b>1.81</b> | <b>3.51</b> | <b>1.23</b> | <b>8.84</b> |

#### 5.4 ANALYSIS OF HYPERPARAMETERS

We examine the key hyperparameters governing the performance-efficiency trade-off in REFUSION: the slot selection threshold  $\tau_{\text{slot}}$ , the token acceptance threshold  $\tau_{\text{token}}$ , and the slot size  $k$ . The threshold  $\tau_{\text{slot}}$  controls the confidence for slot selection (planning), and  $\tau_{\text{token}}$  governs the confidence for draft acceptance (infilling), while  $k$  defines the granularity of the generation unit. An analysis of other hyperparameters is shown in Appendix C.4.

As illustrated in Figure 4, these hyperparameters create a predictable trade-off. **(1) Slot selection threshold  $\tau_{\text{slot}}$ :** Increasing  $\tau_{\text{slot}}$  improves performance due to higher token reliability. However, throughput (TPS) exhibits a non-monotonic trend. Although a higher threshold reduces slot parallelism, processing fewer slots mitigates synchronization overhead during the Parallel Iterative Completion phase, which can potentially boost throughput. **(2) Token acceptance threshold  $\tau_{\text{token}}$ :** Increasing  $\tau_{\text{token}}$  reduces the number of accepted draft tokens per step, thereby lowering throughput. Performance also follows a non-monotonic trend. While higher thresholds enforce stricter verification, excessive strictness causes frequent truncation and regeneration. This can trap the model in local optima and degrade final performance. **(3) Slot size  $k$ :** A larger slot size  $k$  enhances local coherence and allows more draft tokens to be accepted wholesale during verification. This leads to simultaneous gains in both performance and speed. Collectively, these analyses reveal a robust and wide “sweet spot,” highlighted by the yellow shaded regions in Figure 4, where REFUSION consistently surpasses the Qwen3-8B baseline in both performance and throughput (TPS). This superior operating zone corresponds to a slot selection threshold  $\tau_{\text{slot}} \in [0.5, 1.0]$ , a token acceptance threshold  $\tau_{\text{token}} \in [0.1, 0.9]$ , and a slot size  $k \in \{8, 32\}$ .

#### 5.5 ABLATION ON KV CACHE REUSE

To maximize efficiency, REFUSION directly concatenates the KV caches of parallel-generated slots, bypassing a costly forward pass that would otherwise be needed to contextualize them. To quantify the impact of this approximation, we conduct an ablation study comparing our default model against a variant, “REFUSION w/ KV Re-computation,” which performs this extra forward pass to ensure full contextualization at the cost of speed.

As shown in Table 4, our default approach is consistently  $1.16\text{--}1.33\times$  faster across all benchmarks. Surprisingly, this significant speedup comes at virtually no cost to performance; in fact, accuracy

remains stable and even slightly improves on several benchmarks. We hypothesize this counter-intuitive benefit arises from a form of implicit regularization: by avoiding over-conditioning on potentially flawed parallel drafts, our method mitigates error propagation. This result validates our KV cache reuse strategy not merely as a speed-accuracy trade-off, but as a design choice that simultaneously enhances efficiency and robustness.

Table 4: Ablation regarding our KV cache reuse mechanism. We compare our default REFUSION, which efficiently reuses KV caches by concatenating them after parallel generation, against a variant (w/ KV Re-computation) that recomputes caches for full contextualization at a higher cost.

| Model                                | MMLU-Pro | ARC-C | GSM8K | MATH  | GPQA  | HumanEval | MBPP  |
|--------------------------------------|----------|-------|-------|-------|-------|-----------|-------|
| <b>REFUSION w/ KV Re-computation</b> | 45.56    | 89.76 | 84.38 | 54.18 | 35.49 | 77.44     | 68.20 |
|                                      | 42.80    | 28.03 | 69.42 | 69.20 | 48.51 | 78.00     | 74.45 |
| <b>REFUSION</b>                      | 45.94    | 89.76 | 84.91 | 54.22 | 35.49 | 78.66     | 68.20 |
|                                      | 52.74    | 32.46 | 81.24 | 81.77 | 64.11 | 103.90    | 92.09 |

## 5.6 CASE STUDY

Figure 5 provides a qualitative understanding of how REFUSION solves a programming problem from the MBPP benchmark, highlighting two key capabilities: **(1) High degree of parallelism.** The model frequently generates multiple slots concurrently. For instance, at iteration 8, it simultaneously generates four separate slots, demonstrating its ability to exploit parallel decoding opportunities. Simultaneously, within each slot, the model leverages speculative decoding to accept four tokens in a single pass, significantly accelerating the generation process. **(2) Non-linear generation order.** The generation process is markedly non-linear. For example, the model constructs the central “for” loop structure (iteration 5) before initializing a local variable “sum = 1” (iteration 6). This ability to plan and execute in a parallel, non-monotonic fashion allows REFUSION to construct complex, structured code in a manner that is both efficient and conceptually closer to human problem-solving. Appendix D.1 shows the results of baseline models on the same problem. Furthermore, to facilitate understanding, we provide a detailed visualization of the decoding process in Appendix D.2.

**Problem:** Write a function to sum all amicable numbers from 1 to a specified number.

ReFusion:

```

1 def am_icable_numbers(2 sum(n):
2   3 def sum_div_isors(4 num):
3     5 sum = 1
4     6 for i in 7 range(2, 7 int(num ** 0.5)) + 8 1:
5       9 if num % i == 0:
6         10 sum += i
7         11 if i != num // i:
8           12 sum += num // i
9
10   13 am_icable_sum = 0
11   14 for i in 15 range(2, 15 n + 1 16):
12     17 sum_i = sum_div_isors(i)
13     18 if sum_i != i and sum_div_isors(sum_i) == i:
14       19 am_icable_sum += i
15
16   20 return am_icable_sum <|im_end|>

```

Figure 5: A case study of REFUSION generating a Python function for an MBPP problem ( $k = 4$ ,  $\tau_{\text{slot}} = 0.6$ ,  $\tau_{\text{token}} = 0.3$ ,  $b = 16$ ). The number in the top-left corner of each slot indicates the generation order, while the color intensity within each slot represents the generation time (darker indicates earlier).

## 6 CONCLUSION

In this work, we present REFUSION, a novel generative model that synergizes the strengths of diffusion-based planning and autoregressive infilling to address the long-standing efficiency and coherence challenges in traditional MDMs. This unique design enables full KV cache reuse within a flexible, any-order generation framework, while making the training objective tractable by simplifying the combinatorial complexity of the generation space. Extensive evaluations across seven benchmarks show that REFUSION establishes a new state of the art for MDMs. More strikingly, it bridges the performance gap to strong ARMs, often outperforming them while being significantly faster. Our work demonstrates that by structuring the parallel generation process, it is possible to achieve the throughput potential of MDMs without sacrificing generation quality. Future directions include further scaling of the model and data size, as well as leveraging reinforcement learning to optimize the model’s planning policy for complex, multi-step reasoning tasks.

## REFERENCES

- AI@Meta. Llama 3 model card. 2024. URL [https://github.com/meta-llama/llama3/blob/main/MODEL\\_CARD.md](https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md).
- Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piñeres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL <https://arxiv.org/abs/2502.02737>.
- Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://arxiv.org/abs/2503.09573>.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025.
- 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. *arXiv preprint arXiv:2401.10774*, 2024.
- 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.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.
- Guobao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model. *arXiv preprint arXiv:2502.09622*, 2025.
- Shanshan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. Scaling diffusion language models via adaptation from autoregressive models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=j1tSLYKwg8>.
- 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.

- Daehoon Gwak, Minseo Jung, Junwoo Park, Minho Park, ChaeHun Park, Junha Hyung, and Jaegul Choo. Reward-weighted sampling: Enhancing non-autoregressive characteristics in masked diffusion llms. *arXiv preprint arXiv:2509.00707*, 2025.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*, 2025.
- Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. On the learning of non-autoregressive transformers. In *International conference on machine learning*, pp. 9356–9376. PMLR, 2022.
- Pengcheng Huang, Shuhao Liu, Zhenghao Liu, Yukun Yan, Shuo Wang, Zulong Chen, and Tong Xiao. Pc-sampler: Position-aware calibration of decoding bias in masked diffusion models. *arXiv preprint arXiv:2508.13021*, 2025.
- Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Rui Feng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024. URL <https://arxiv.org/pdf/2411.04905.pdf>.
- Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.
- Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.
- Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. 2024.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org, 2023.
- Jia-Nan Li, Quan Tu, Cunli Mao, Zhengtao Yu, Ji-Rong Wen, and Rui Yan. Streamingdialogue: Prolonged dialogue learning via long context compression with minimal losses. *Advances in Neural Information Processing Systems*, 37:86074–86101, 2024.
- Yifan Li, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. Diffusion models for non-autoregressive text generation: A survey. *arXiv preprint arXiv:2303.06574*, 2023.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.
- Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.
- Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed-dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025.

Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

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.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

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.

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. *arXiv preprint arXiv:2506.01928*, 2025.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction. *arXiv preprint arXiv:2508.02558*, 2025.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fb0d053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fb0d053c1c4a845aa-Paper.pdf).

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

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding, 2025b. URL <https://arxiv.org/abs/2505.22618>.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan 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>.

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

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhui Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, et al. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv preprint arXiv:2508.08712*, 2025.

Table 5: Comparison between REFUSION and existing MDMs.  $L$  denotes the generation length and  $k$  denotes the block or slot size.

| Model           | Generation Scope | Attention Mechanism | Generation Order | Full KV Cache Reuse | Number of Distinct Masking Patterns   |
|-----------------|------------------|---------------------|------------------|---------------------|---|
| <b>LLaDA</b>    | Full Sequence    | Bidirectional       | Any-Order        | ✗                   | $\sum_{l=1}^L \binom{L}{l} \approx 2^L$   |
|                 | Intra-block      | Bidirectional       | Any-Order        |                     |   |
| <b>BD3-LMs</b>  | Inter-block      | Causal              | Left-to-Right    | ✗                   | $2^k \cdot \frac{L}{k}$   |
|                 | Intra-slot       | Causal              | Left-to-Right    |                     | $\sum_{i=1}^{L/k} \binom{L/k}{i} \cdot i! = \lfloor (\frac{L}{k})! \cdot e \rfloor - 1$ |
| <b>REFUSION</b> | Inter-slot       | Causal              | Any-Order        | ✓                   | $(\lfloor (\frac{L}{k})! \cdot e \rfloor - 1 \ll 2^L \text{ for large } k)$             |

## A METHODOLOGICAL DETAILS

### A.1 COMPARISON BETWEEN REFUSION AND REPRESENTATIVE MDMs

Table 5 provides a detailed, side-by-side comparison of the architectural and methodological designs of REFUSION against two representative MDMs, LLaDA (Nie et al., 2025) and BD3-LMs (Arriola et al., 2025). This comparison highlights how REFUSION uniquely addresses the fundamental trade-offs between generation flexibility, computational efficiency, and learning complexity.

(1) LLaDA, as a conventional MDM, operates on the entire sequence with a bidirectional attention mechanism. This grants it maximum flexibility, allowing for a fully unconstrained, any-order generation process. However, this design choice incurs two significant penalties. First, the bidirectional attention is fundamentally incompatible with KV caching, resulting in substantial computational overhead at each decoding step. Second, it must learn dependencies across an exponential space of possible masking patterns. For a sequence of length  $L$ , any given training or inference state is defined by a subset of tokens that remain masked. Since each of the  $L$  positions can be either masked or unmasked, the model must, in principle, handle any of the  $2^L$  possible subsets of visible context<sup>5</sup>. This combinatorial space of approximately  $2^L$  distinct masking patterns presents an intractable objective, as the model may not be sufficiently trained on the specific patterns encountered during inference, leading to incoherent parallel generation.

(2) BD3-LMs attempts to mitigate these issues with a hybrid, block-based approach. It enforces a rigid, left-to-right generation order between blocks, which enables KV cache reuse across block boundaries. However, within each block, it retains bidirectional attention and any-order token generation. This design makes a critical compromise. It sacrifices global generation flexibility for discovering optimal generation strategies, which is a key theoretical advantage of MDMs. Furthermore, it still faces the challenges of token-level incoherence and the inability to utilize KV caching for intra-block decoding.

(3) REFUSION introduces a more elegant and unified solution. Generation is structured at the slot level. Within each slot (intra-slot), generation is autoregressive (left-to-right) under a causal attention mask, directly addressing the strong local dependencies between adjacent tokens. Between slots (inter-slot), the model retains the flexibility of any-order generation, enabling it to discover better, non-linear generation paths than the left-to-right order. Crucially, by reordering generated slots to always precede masked ones in the input sequence, REFUSION enables full KV cache reuse for every decoded token, a feature unique among these models. This design simultaneously achieves two critical goals: it combines global generation flexibility with universal computational efficiency, and it drastically reduces the learning complexity from an exponential token-level permutation space to a far more manageable slot-level one ( $\lfloor (\frac{L}{k})! \cdot e \rfloor - 1$ ). For a typical sequence length of  $L = 4,096$ , a slot size of just  $k = 8$  is sufficient to ensure  $\lfloor (\frac{L}{k})! \cdot e \rfloor - 1 < 2^L$ .

In summary, while prior models are forced to trade flexibility for efficiency or vice versa, REFUSION’s innovative slot-based framework is the only approach that concurrently offers global any-order generation, full KV cache reuse, and a tractable training objective.

<sup>5</sup>Notably, due to the bidirectional attention, the model is invariant to the order in which clean tokens are revealed. Therefore, the learning complexity is not permutations ( $L!$ ).

## A.2 INFERENCE FORMALIZATION

In this section, we formalize the two-step decoding iteration as follows. Crucially, standard causal attention is consistently applied throughout the decoding process, and the position IDs for all tokens always correspond to their indices in the correct ground-truth sequence, remaining invariant regardless of this repositioning. By using these consistent position IDs with RoPE (Su et al., 2021), the model correctly perceives relative positions even when the input buffer is reordered.

**Step I: Diffusion-based Slot Planning.** The first step leverages the model’s MDM capability to plan the next decoding slots. At a timestep  $t$  (defined as the ratio of remaining masked slots), we construct the input  $\tilde{S}_t$  for enabling KV cache by concatenating already-decoded clean slots ( $\tilde{S}_t^{\text{clean}}$ , in generation order) with the remaining masked slots ( $\tilde{S}_t^{\text{masked}}$ , in their original positional order). The planning process then generates a draft for all masked slots. This draft serves a dual purpose: providing a basis for scoring each slot for planning, and acting as a speculative guess for the subsequent infilling stage (Leviathan et al., 2023). Specifically, for each position  $j$  in the  $i$ -th slot of  $\tilde{S}_t^{\text{masked}}$ , a draft token  $\tilde{d}_t^{i,j}$  is sampled from the model’s marginal distribution, conditioned on the leading context:

$$\tilde{d}_t^{i,j} \sim P_{\theta}(\cdot | p_0, \tilde{S}_t^{\text{clean}}, \tilde{S}_{t,\leq(i,j)}^{\text{masked}}), \quad (4)$$

where  $\tilde{S}_{t,\leq(i,j)}$  denote the tokens before the position of the target token. This yields a draft version of the masked slots, denoted as  $\tilde{S}_t^{\text{draft}} = \{\tilde{d}_t^{i,j}\}$ . We then quantify the model’s certainty score of  $i$ -th slot  $\tilde{S}_t^i$  in  $\tilde{S}_t^{\text{masked}}$  as the model’s predicted probability of its first token  $\tilde{d}_t^{i,1}$ :

$$C(\tilde{S}_t^i) = P_{\theta}(\tilde{d}_t^{i,1} | p_0, \tilde{S}_t^{\text{clean}}, \tilde{S}_{t,\leq(i,1)}^{\text{masked}}). \quad (5)$$

The model then selects a batch of slots with scores exceeding a threshold  $\tau_{\text{slot}}$  for subsequent infilling. If no slot meets this criterion, the single slot with the globally highest score is selected instead. This strategy identifies slots that are strongly constrained by the existing context and weakly interdependent (e.g., distinct function definitions in code generation), making them suitable to parallelize.

**Step II: Autoregressive Slot Infilling.** The second step verifies and completes the selected draft slots by leveraging the model’s autoregressive modeling capability. To accelerate infilling, we employ a speculative decoding strategy powered by the model’s MDM capability. To achieve this, we first concatenate the slots in their original left-to-right order. The model then calculates the conditional probability of each token, conditioned on all preceding tokens within the newly formed sequence:

$$\mathcal{P}(\tilde{d}_t^{i,j}) = \begin{cases} P_{\theta}(\tilde{d}_t^{i,1} | p_0, \tilde{S}_t^{\text{clean}}, \tilde{S}_{t,\leq(i,1)}^{\text{masked}}), & \text{if } j = 1 \\ P_{\theta}(\tilde{d}_t^{i,j} | p_0, \tilde{S}_t^{\text{clean}}, \tilde{S}_{t,<(i,j)}^{\text{draft}}). & \text{if } j > 1 \end{cases} \quad (6)$$

Next, we verify the draft by identifying the longest prefix of the concatenated sequence, with length  $l$ , where every token’s probability exceeds the threshold  $\tau_{\text{token}}$ . If the prefix is long enough to form at least one full slot (i.e.,  $l \geq k$ ), we accept the first  $\lfloor l/k \rfloor$  slots and immediately begin a new planning-infilling iteration, bypassing the costly suffix completion.

Otherwise, if the accepted prefix fails to cover at least one complete slot (i.e.,  $l < k$ ), we revert to processing the draft slots independently via a **Parallel Iterative Completion** process. We independently refine the draft  $\tilde{S}_t^i = \{\tilde{d}_t^{i,j}\}_{j=1}^k$  for each slot  $i \in \mathcal{I}$  in parallel, where  $\mathcal{I}$  denotes the set of indices for the selected slots. The completion process iterates through two steps until all selected slots are fully completed. In each iteration  $m$ :

(i) *Verification:* For each slot  $i \in \mathcal{I}$ , we identify the longest valid prefix length  $l_i^{(m)}$  where the conditional probability of every token in the prefix exceeds  $\tau_{\text{token}}$ , conditioned on the prompt, the already-decoded clean slots, and the preceding tokens within slot  $i$ :

$$l_i^{(m)} = \max \left\{ \eta \mid \forall j \leq \eta, P_{\theta}(\tilde{d}_t^{i,j} | p_0, \tilde{S}_t^{\text{clean}}, \tilde{d}_t^{i,<j}) > \tau_{\text{token}} \right\}, \quad (7)$$

where  $\tilde{d}_t^{i,<j}$  denotes the prefix within the  $i$ -th slot.

(ii) *Prediction:* We retain the valid prefix  $\tilde{d}_t^{i,1:l_i^{(m)}}$ , re-mask the remaining suffix, and predict the masked tokens by sampling from the model’s distribution:

$$\tilde{d}_t^{i,j} \sim P_{\theta}(\cdot | p_0, \tilde{S}_t^{\text{clean}}, \tilde{d}_t^{i,1:l_i^{(m)}}, \tilde{S}_{t, l_i^{(m)} < \cdot}^{\text{masked}}), \quad \forall j \in (l_i^{(m)}, k]. \quad (8)$$

Table 6: Hyperparameter settings for different tasks.

| Benchmark | Generation Length | Slot Selection Threshold $\tau_{\text{slot}}$ | Token Acceptance Threshold $\tau_{\text{token}}$ | Slot Size $k$ | Block Size $b$ |
|-----------|-------------------|---|--|---------------|----------------|
| MMLU-Pro  | 512               | 0.9   | 0.4  | 16            | 128            |
| ARC-C     | 512               | 0.8   | 0.1  | 8             | 8              |
| GSM8K     | 512               | 0.9   | 0.4  | 32            | 128            |
| MATH      | 512               | 0.8   | 0.6  | 32            | 64             |
| GPQA      | 512               | 0.8   | 0.2  | 8             | 16             |
| HumanEval | 512               | 0.9   | 0.4  | 32            | 128            |
| MBPP      | 512               | 0.9   | 0.3  | 32            | 128            |

Table 7: Performance comparison of different models on HumanEval and MBPP.

| Model                         | # Total Params | # Activated Params | System Optimization | Throughput (TPS) | HumanEval (# Shots) | MBPP (# Shots) |
|-------------------------------|----------------|--------------------|---------------------|------------------|---------------------|----------------|
| <i>Autoregressive Model</i>   |                |                    |                     |                  |                     |                |
| Nova Micro                    | -              | -                  | ✓                   | 148              | 79.30 (0)           | 65.40 (3)      |
| GPT 4o Mini                   | -              | -                  | ✓                   | 59               | 88.00 (0)           | 74.60 (3)      |
| Gemini 2.0 Flash Lite         | -              | -                  | ✓                   | 201              | 90.00 (0)           | 75.00 (3)      |
| <i>Masked Diffusion Model</i> |                |                    |                     |                  |                     |                |
| Mercury Coder Mini            | -              | -                  | ✓                   | 1,109            | 88.00 (0)           | 77.10 (3)      |
| Mercury Coder Small           | -              | -                  | ✓                   | 737              | 90.00 (0)           | 76.60 (3)      |
| Gemini Diffusion              | -              | -                  | ✓                   | 1,479            | 89.60 (0)           | 76.00 (3)      |
| Seed Diffusion                | -              | -                  | ✓                   | 1,600            | 82.80 (0)           | 79.40 (3)      |
| LLaDA-MoE                     | 7B             | 1B                 | ✓                   | 884              | 62.20 (0)           | 67.45 (3)      |
| REFUSION                      | 8B             | 8B                 | ✗                   | 98               | 78.66 (0)           | 68.20 (0)      |

After infilling each selected slot, the completed slots are moved from  $\tilde{S}_t^{\text{masked}}$  to  $\tilde{S}_t^{\text{clean}}$ . For the subsequent iteration, the KV caches from these parallel-generated slots are concatenated. While this parallel generation forgoes inter-slot conditioning, we observe in our experiments that this has a minimal impact on final performance (see §5.5). This planning-infilling iteration repeats with an updated timestep  $t$  until no masks remain ( $t = 0$ ), at which point the final response  $\tilde{r}_0$  is formed by sorting  $\tilde{S}_0^{\text{clean}}$  back into its original sequence order.

## B EXPERIMENTAL DETAILS

### B.1 IMPLEMENTATION DETAILS

Our training data comprises 3.7M samples from MAmmoTH (Yue et al., 2023), OpenMathInstruct-2 (Toshniwal et al., 2024), OpenCoder (Huang et al., 2024), SmolLM 2 (Allal et al., 2025), and Tulu 3 (Lambert et al., 2024). For OpenMathInstruct-2, we use its 1M open-source version and remove questions longer than 1,024 tokens as instructed. We use a global batch size of 512, a maximum sequence length of 4,096, and a learning rate of 2e-5. The training is conducted on 16 nodes, each with 8 H20 GPUs, and is accelerated using DeepSpeed ZeRO-2 (Rajbhandari et al., 2020) and Flash-attention-2 (Dao, 2023). The total training cost is approximately 10.68K H20 GPU-hours. We set  $\lambda$  in Eq. 3 to 1. For each training sample, we randomly select a slot size from {4, 8, 16, 32}.

Existing MDMs decode sequences to a predetermined length. Even when an end-of-sequence (EOS) token appears early, the model still expends decoding time on all tokens with higher position IDs. To address this issue, we introduce a mechanism for efficient variable-length generation. Specifically, during training, we pad shorter sequences in a mini-batch with padding tokens and exclude these tokens from the loss computation. During inference, upon generating an EOS token, we dynamically truncate the target length to that token’s position. This prevents the decoding of any tokens with a higher position ID, thereby reducing redundant computation.

### B.2 HYPERPARAMETER SETTING

During the REFUSION inference process, four hyperparameters can be adjusted: the slot selection threshold  $\tau_{\text{slot}}$ , the token acceptance threshold  $\tau_{\text{token}}$ , the slot size  $k$ , and the block size  $b$ . Table 6 shows the specific settings used in our evaluation.

## C EXPERIMENT RESULTS

### C.1 COMPARISON WITH CLOSED-SOURCE AND DIVERSE ARCHITECTURES

To comprehensively demonstrate our advantages, we further compare REFUSION against closed-source and diverse ARM and MDM architectures. As shown in Table 7, REFUSION outperforms several highly optimized ARMs in either speed or performance. Specifically, it achieves  $1.66 \times$  the throughput of GPT-4o Mini and surpasses Nova Micro by 2.8 points on MBPP. Furthermore, REFUSION yields results approaching those of larger and more powerful closed-source MDMs.

### C.2 TRADE-OFF FRONTIER ANALYSIS

To further distinguish REFUSION from prior MDMs, we investigate the trade-off frontiers of different models. Figure 6 shows that both LLaDA and Dream suffer a sharp performance decline as parallelism (tokens generated per forward pass, TPF) increases<sup>6</sup>, indicating a failure to uphold the conditional independence assumption when selecting tokens for parallel decoding. In contrast, REFUSION’s curve is substantially flatter, validating that its training and decoding strategies can more reliably identify conditionally independent tokens.

### C.3 SCALING WITH DATA SIZE

To understand the scaling properties of our model, we investigate the impact of training data size on REFUSION’s performance and efficiency. To this end, we collect additional data of potentially lower quality to expand our training set to 14M samples. Figure 7 illustrates the results of this analysis on GSM8K and MBPP, where we train REFUSION for one epoch on datasets of varying sizes (from 120K to 14M samples) and evaluate it using the same hyperparameters as in Table 6.

The results reveal a clear and positive scaling trend for both key metrics. Specifically, throughput (TPS, dashed lines) improves generally consistently as the training data size increases. For instance, on MBPP, throughput rises from approximately 51 TPS with 120K samples to over 81 TPS with 14M samples. This indicates that as the model is exposed to more diverse data, its internal generation process becomes more efficient, likely due to higher confidence and thus a higher acceptance rate of its parallel drafts, leading to fewer decoding iterations.

Interestingly, the performance scaling (solid lines) is not strictly monotonic, a common phenomenon when training with a fixed epoch count. On GSM8K, accuracy peaks at 2M samples before slightly decreasing at 3.7M. This behavior highlights a trade-off between data breadth and training depth under a constrained computational budget: with a fixed one-epoch schedule, training on a larger dataset potentially leads to under-convergence relative to the dataset’s complexity.

Nevertheless, the consistent rise in throughput coupled with the substantial performance uplift from the 120K baseline suggests that with an increased computational budget (i.e., more training epochs on the larger datasets), performance would likely continue to improve, further unlocking the full potential of our approach.

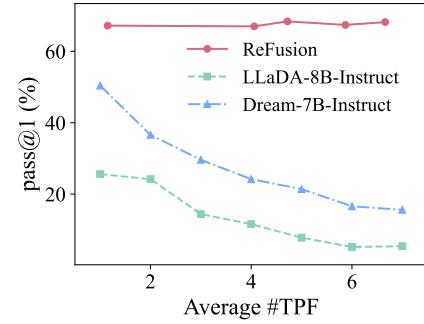


Figure 6: Pass@1 on MBPP for REFUSION and baseline MDMs over TPF.

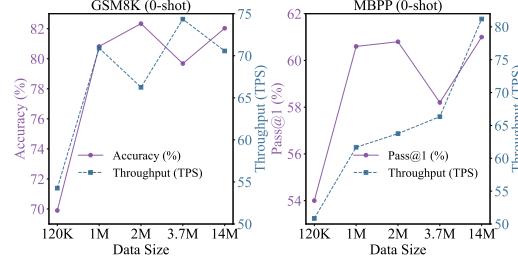


Figure 7: Scaling properties of REFUSION on GSM8K and MBPP. We plot performance (Accuracy/Pass@1, %) and inference throughput (TPS) as a function of training data size.

<sup>6</sup>We use TPF here, rather than TPS, to isolate the algorithmic trade-off from any system-level overheads.

#### C.4 ANALYSIS OF BLOCK SIZE

Our inference strategy is compatible with semi-autoregressive remasking (Nie et al., 2025). Specifically, during inference, the target sequence is partitioned into consecutive blocks of size  $b$ . These blocks are decoded sequentially, while our synergistic decoding algorithm is applied to each block as a single unit. Notably, the constraint  $b \geq k$  must be satisfied, where  $k$  is the size of a slot, the fundamental unit for parallel decoding in our method.

Figure 8 illustrates the impact of block size  $b$  on our method’s performance and throughput (TPS). The figure reveals that both metrics exhibit non-monotonic trends as  $b$  increases. The trend in performance arises from a trade-off between the planning horizon and modeling complexity. Specifically, while a larger  $b$  allows the model to cover longer semantic units in a single plan to enhance coherence, it simultaneously imposes a greater challenge in generating larger, more complex blocks in an arbitrary order. Similarly, the non-monotonic trend in throughput (TPS) is attributable to computational overhead. Although a larger  $b$  provides more opportunities for parallelism, it forces the model to process a longer sequence containing many “padded” (i.e., yet-to-be-generated) positions. This significantly increases the latency of each decoding step, which eventually diminishes and then reverses the throughput gains observed with larger block sizes.

Although performance fluctuates slightly with larger block sizes, REFUSION’s pass@1 decreases by only approximately 0.2% relative to its peak as  $b$  increases from 32 to 512. This robustness to block size highlights the model’s ability to leverage strong diffusion-based planning to select the most appropriate slots for decoding across a wide range. Collectively, these analyses reveal a robust and wide “sweet spot,” highlighted by the yellow shaded regions in Figures 4 and 8, where REFUSION consistently surpasses the Qwen3-8B baseline in both performance and throughput (TPS). This superior operating zone corresponds to a block size  $b \in [32, 128]$ .

## D CASE STUDY

### D.1 CODE GENERATION CASE STUDY

Table 8 showcases responses to the same MBPP problem as Figure 5 from different models. REFUSION’s ability to globally plan the overall structure via a diffusion-based process while locally infilling details autoregressively results in better-structured, high-quality code, demonstrating its superiority over existing MDMs.

### D.2 STEP-BY-STEP VISUALIZATION OF INFERENCE

In order to facilitate the understanding of our inference method, we show a step-by-step decoding process in Figure 9. Specifically, our inference process progressively generates the response through an iterative “plan-and-infill” mechanism.

The model maintains standard token-wise causal attention throughout the entire process. Each decoding iteration operates as a two-stage cycle: First, the planning stage predicts drafts for all masked slots in parallel and selects a subset of high-quality slots based on confidence scores. Second, the infilling stage treats the selected slots as a batch and completes them autoregressively to ensure local coherence.

To enable full KV cache reuse, the newly decoded slots are physically moved to the front of the remaining masked slots after infilling. Crucially, while the tokens’ physical position may change due to this repositioning, their position IDs remain invariant, always corresponding to their indices in the correct ground-truth sequence. By utilizing these consistent position IDs with RoPE (Su et al., 2021), the model accurately perceives the relative positions of all tokens, even when the input buffer is reordered.

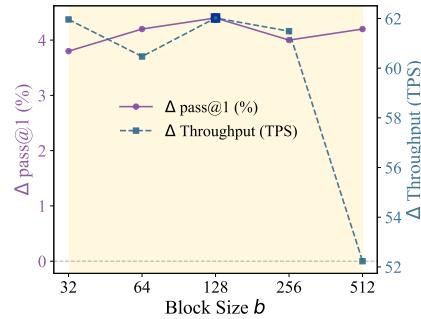


Figure 8: Relative change in REFUSION’s **pass@1 (%)** and **throughput (tokens/sec)** compared to Qwen3-8B (horizontal dashed lines at zero) as block size  $b$  varies. The yellow shaded region highlights the range of  $b$  where REFUSION surpasses Qwen3-8B.

Table 8: Model responses to the same MBPP problem as Figure 5.

|                            |   |
|----------------------------|---|
| <b>Problem</b>             | You are an expert Python programmer. Your task is to write a single Python function to solve the problem described below, and here is your task: Write a function to sum all amicable numbers from 1 to a specified number. Your code should pass these tests:<br><pre>assert amicable_numbers_sum(999)==504 assert amicable_numbers_sum(9999)==31626 assert amicable_numbers_sum(99)==0</pre>  |
| <b>LLaDA-8B-Instruct</b> X | <pre>def amicable_numbers_sum(limit):     def is_perfect_square(n):         root = int(n**0.5)         return root * root == n      def sum_divisors(n):         total = 0         for i in range(1, n):             if n % i == 0:                 total += i         return total      def is_amicable(a, b):         return sum_divisors(a) == b and sum_divisors(b) == a      amicable = set()     for i in range(1, limit + 1):         if is_perfect_square(i):             amicable.add(i)         for j in range(i + 1, limit + 1):             if is_amicable(i, j):                 amicable.add(j)      return sum(amicable)</pre> |
| <b>Dream-7B-Instruct</b> X | <pre>def amicable_numbers_sum(n):     def is_amicable(num):         divisors = 1         for i in range(2, num):             if num % i == 0:                 divisors += i             if i != num // i:                 divisors += num // i         return divisors == num      total = 0     for i in range(2, n):         if is_amicable(i):             total += i     return total</pre>   |
| <b>REFUSION</b> ✓          | <pre>def amicable_numbers_sum(n):     def sum_divisors(num):         sum = 1         for i in range(2, int(num**0.5) + 1):             if num % i == 0:                 sum += i                 if i != num // i:                     sum += num // i         return sum      amicable_sum = 0     for i in range(2, n + 1):         sum_i = sum_divisors(i)         if sum_i != i and sum_divisors(sum_i) == i:             amicable_sum += i     return amicable_sum</pre>   |

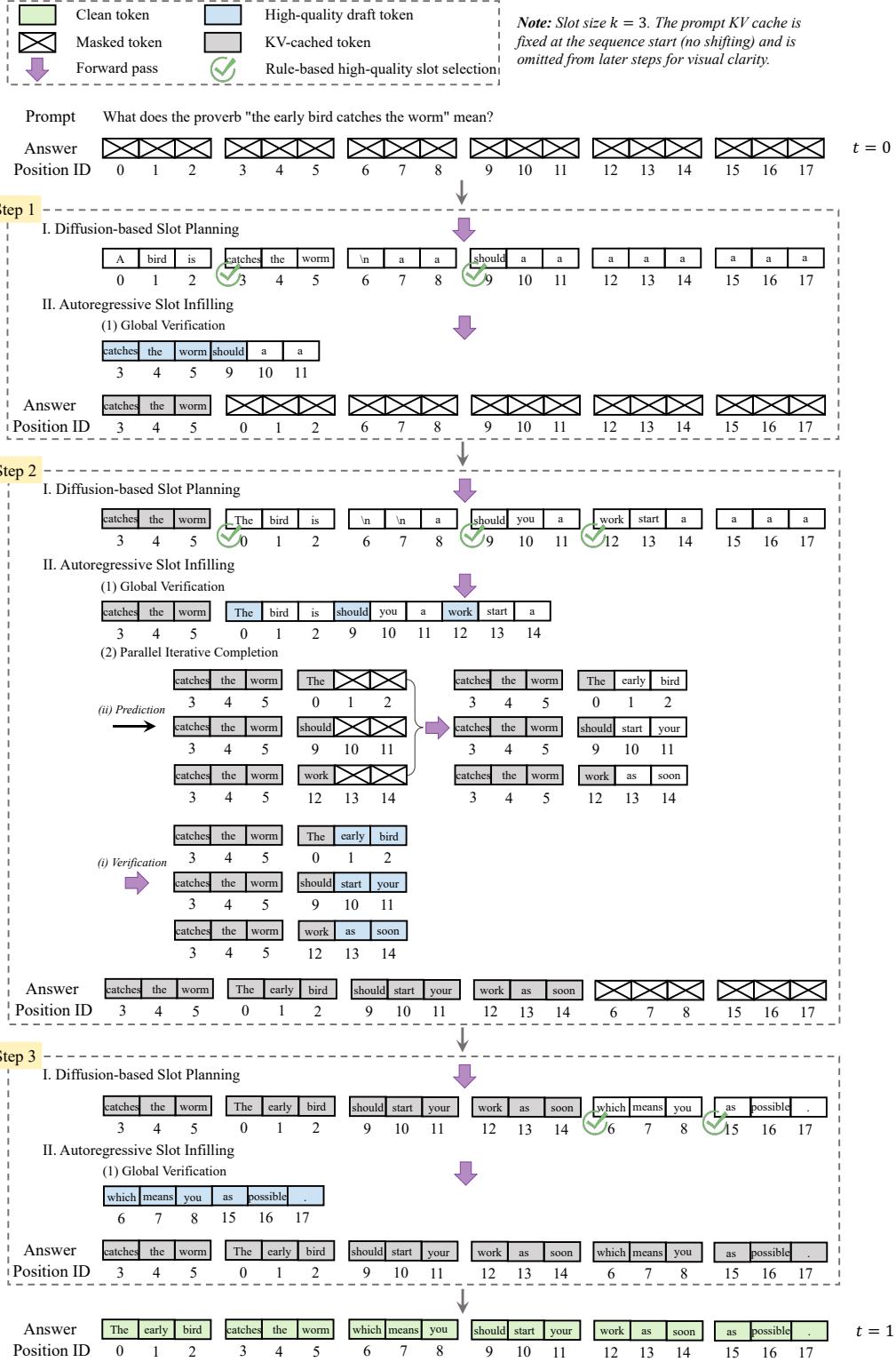


Figure 9: Visualization of the REFUSION inference mechanism.

## E LIMITATIONS

A primary limitation of our current framework is the **immutability of generated slots**. Once the tokens within a slot are generated via diffusion-based planning and autoregressive infilling, they

are considered final and cannot be remasked or refined in future iterations. This design choice, while simplifying the process, precludes the model from correcting potential errors made within a completed slot.

A promising direction for future work would be to introduce a re-masking mechanism at the sub-slot level. For instance, after infilling a slot, the model could verify the generated tokens and preserve only a high-confidence prefix, while re-masking the lower-quality suffix. This would allow for iterative refinement but would necessitate a more complex inference logic, potentially involving dynamic adjustments of slot sizes to handle these newly masked, smaller segments. Developing an efficient strategy for such dynamic, fine-grained refinement remains a key challenge for future research.