REFUSION: A DIFFUSION LARGE LANGUAGE MODEL WITH PARALLEL AUTOREGRESSIVE DECODING

Anonymous authors
Paper under double-blind review

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 32% performance gains and an over 10× speedup on average, but also bridges the performance gap to strong ARMs while maintaining a 1.4× 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 necessitates the use of 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 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.

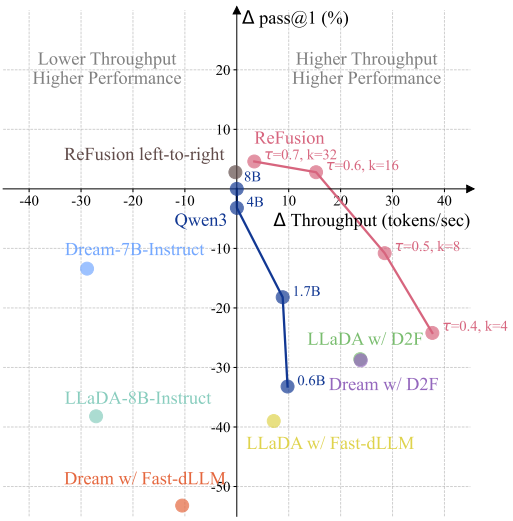

Figure 1: Performance-throughput trade-off on MBPP. We plot pass@1 (%) against throughput (tokens/sec), with both metrics calculated relative to the Qwen3-8B baseline at the origin. The "REFUSION left-to-right" ablation forces serial decoding using the REFUSION model. See §5.3 for details about hyperparameters of REFUSION.

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 32% while being over 10× 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 3.2 absolute points while being 1.4× faster on average. This dual advantage in both performance and speed is further illustrated in Figure 1, where REFUSION (the red line) is the only method to establish a superior performance-efficiency frontier in the top-right quadrant. It significantly outperforms both the Qwen3-series (the blue line) and prior MDM-based methods (lower-left, implying slower and less effective). Furthermore, our controlled experiments confirm these gains are driven by our architectural and training innovations, not initialization and 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., 2025b) 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., 2025b). 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.

3 Preliminary

Autoregressive Models. ARMs are a prominent class of generative models that factorize the joint probability of a sequence $x = (x_1, \ldots, x_L)$ 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 θ are optimized by minimizing the negative log-likelihood: $-\sum_{i=2}^{L} \log P_\theta(x_i \mid x_{<i})$. 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 $x_0 = (x_0^1, x_0^2, \cdots, x_0^L)$ 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: $-\frac{1}{t}\sum_{i=1}^{L} \mathbf{1}(x_t^i = \texttt{[MASK]}) \log P_\theta\left(x_0^i \mid x_t\right)$. 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.

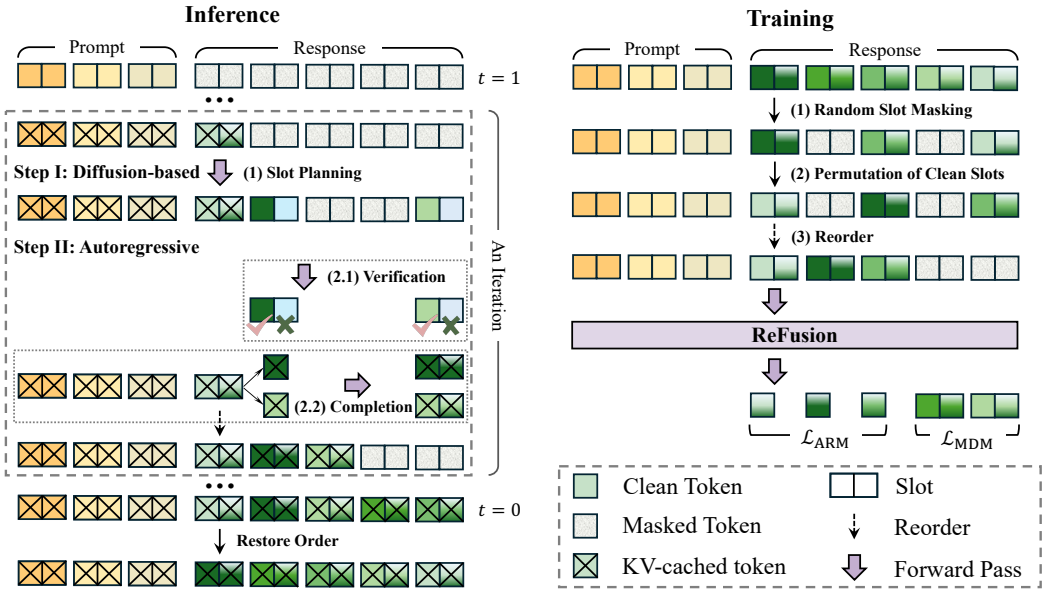

Figure 2: The inference and training process of REFUSION. Inference (Left) proceeds via an iterative "plan-and-infill" loop at the slot level: A diffusion-based step first plans which slots to generate and proposes initial drafts, and a parallel autoregressive step then fills them using a verify-and-complete mechanism. Full KV cache reuse is achieved by reordering generated slots before masked ones after each iteration. Training (Right) mirrors inference by optimizing a hybrid objective that combines an autoregressive loss ($\mathcal{L}_{\text{ARM}}$) on permuted clean slots and a denoising loss ($\mathcal{L}_{\text{MDM}}$) on masked slots.

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. It 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 the grouping of 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*.

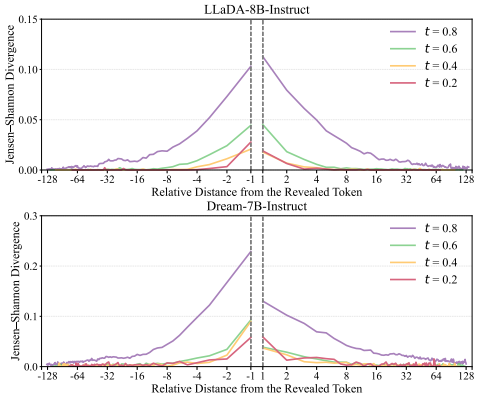

Figure 3: The locality of inter-token dependency in MDMs, with the sign on the x-axis denoting the direction from the revealed token (positive for rightward, negative for leftward).

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 $\tilde{\boldsymbol{S}}_t$ by concatenating already-decoded slots ($\tilde{\boldsymbol{S}}_t^{\text{clean}}$, in generation order) with the masked slots ($\tilde{\boldsymbol{S}}_t^{\text{masked}}$, 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 (Appendix C.2 compares against alternatives). Finally, a batch of slots with scores exceeding a threshold τ 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 draft slot by sampling a token from its distribution at every position, yielding the draft slots $\tilde{\boldsymbol{S}}_t^{\text{draft}}$.

Step II: Autoregressive Slot Infilling. The second step efficiently verifies the draft slots $\tilde{\boldsymbol{S}}_t^{\text{draft}}$ and completes them using the model's autoregressive capability: **(1) Verification.** We concatenate all draft slots into a sequence in their original positional order. The model then performs one forward pass to compute the probability of each token in the sequence, conditioned on the prompt and the already decoded slots. We identify the longest prefix of the concatenated draft where the probabilities of all tokens exceed τ. If this prefix spans one or more complete slots, we accept these slots wholesale and immediately proceed to the next planning iteration, thereby bypassing suffix completion. Otherwise, we determine the longest common prefix length that is successfully verified across all individual draft slots, and truncate each draft slot to this common length. **(2) Completion.** The model then completes any truncated slots by autoregressively sampling the remaining positions in parallel for each slot.

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 §5.4). 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. We also quantify the substantial contribution of confidence-based parallelism to throughput in Appendix C.5.

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, $\boldsymbol{S}_0 = [S_0^1, \ldots, S_0^K]$, each of size k. Then, a corrupted version $\boldsymbol{S}_t$ 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 $\boldsymbol{S}_0$. 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 $\boldsymbol{S}_t^{\text{clean}}$, while keeping the original relative positions of the masked slots to form $\boldsymbol{S}_t^{\text{masked}}$. **(3) Reorder.** The final training instance $\boldsymbol{S}_t$ 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 $\boldsymbol{S}_t^{\text{clean}}$ are trained with a standard ARM loss for next token prediction:

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

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

On the other hand, the masked slots $\boldsymbol{S}_t^{\text{masked}}$ are trained with an MDM objective for denoising[1]:

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

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 $\boldsymbol{S}_t^{\text{masked}}$. The final training objective is a summation of the two losses, balanced by λ:

$$\mathcal{L} = \mathcal{L}_{\text{ARM}} + \lambda \mathcal{L}_{\text{MDM}}. \tag{3}$$

We initialize our model P_θ 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.

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 (∼1.22B-tokens) dataset 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. We evaluate ReFusion 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 assess inference throughput using tokens decoded per second (TPS) with a single A100 GPU and a batch size of 1.

[1] Our per-token normalization, $\frac{1}{k\cdot|\boldsymbol{S}_t^{\text{masked}}|}$, implicitly includes the $\frac{1}{t}$ weighting since $|\boldsymbol{S}_t^{\text{masked}}| \approx tK$, where K is the total number of slots.

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.
Autoregressive Model								
Llama-3-8B-Instruct	35.23 32.07	82.76 44.12	75.13 42.81	25.48 19.73	29.46 42.00	46.34 42.26	53.00 41.68	49.63 37.81
Qwen3-8B	*67.25* 31.42	*90.36* 42.78	81.96 31.20	*83.28* 30.11	*39.06* 30.43	*87.80* 30.95	63.80 30.07	*73.36* 32.42
Masked Diffusion Model								
LLaDA-8B-Instruct	35.80 18.21	85.58 0.03	76.35 27.35	38.78 23.93	32.37 1.99	45.12 12.42	25.60 2.97	48.51 12.41
LLaDA w/ Fast-dLLM	35.02 39.81	82.85 0.86	76.27 73.07	38.58 52.23	28.35 17.54	37.80 62.52	24.80 37.19	46.24 40.46
LLaDA w/ D2F	22.84 44.54	84.13 3.70	39.04 82.59	23.68 59.48	31.25 23.84	36.59 96.90	35.20 53.85	38.96 52.13
Dream-7B-Instruct	40.05 15.98	88.31 0.06	76.42 20.30	46.60 18.99	30.36 1.81	56.71 3.51	50.40 1.23	55.55 8.84
Dream w/ Fast-dLLM	40.36 47.18	86.86 1.42	75.82 61.49	36.76 58.24	31.25 22.96	56.10 49.73	10.60 19.55	48.25 37.22
Dream w/ D2F	38.26 60.64	87.37 14.82	47.99 96.59	24.60 81.59	22.77 25.20	46.95 49.05	35.00 53.95	43.28 54.55
ReFusion	**45.39** 39.38	**89.68** 49.25	**85.60** 40.29	**56.06** 42.40	**35.04** 42.23	**75.61** 46.48	**66.60** 45.34	**64.85** 43.62

Baselines. We compare ReFusion with: (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., 2025c) and D2F (Wang et al., 2025). We implement the baselines based on official hyperparameters.

5.2 Main Results

The main results in Table 1 show: **(1) ReFusion dominates all MDM baselines.** ReFusion consistently outperforms all MDM baselines across all seven benchmarks, often by a substantial margin. For instance, on HumanEval, it achieves 75.61% pass@1, surpassing the next-best MDM (Dream-7B-Instruct) by nearly 19 absolute points. While acceleration methods like Fast-dLLM and D2F[2] improve throughput at a significant performance cost, ReFusion delivers both state-of-the-art performance and competitive efficiency, establishing a new frontier for MDMs. **(2) ReFusion challenges strong ARMs.** More remarkably, ReFusion challenges and often surpasses strong ARMs. On GSM8K and MBPP, for example, it outperforms Qwen3-8B by 3 absolute points while delivering a 1.4× speedup. 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 Analysis of Hyperparameters

We examine the key hyperparameters governing the performance-efficiency trade-off in ReFusion: the verification threshold τ and the slot size k. The threshold τ controls the confidence for both slot selection (planning) and 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 (left & middle), these hyperparameters create a predictable trade-off. **(1) Verification threshold τ:** Lowering τ boosts throughput (TPS) by enabling more aggressive parallelism, but at the cost of reduced performance due to lower token reliability. **(2) Slot size k:** Similarly, smaller slot sizes (k) increase TPS by creating more parallelizable units given a fixed full length, though this speed gain is counteracted by a performance drop.

[2] D2F's low TPS on ARC-C stems from a mismatch between its fixed 32-token block size and the task's short answers (avg. 1∼3 tokens). The latency of generating a full block is incurred for only a few valid tokens.

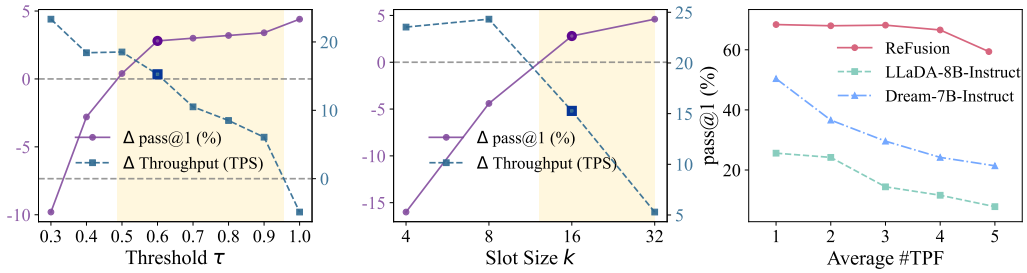

Figure 4: **Left & Middle:** Impact of key hyperparameters on MBPP (0-shot). The plots show the change in **pass@1 (%)** and **throughput (TPS)** of REFUSION relative to Qwen3-8B (horizontal dashed lines at zero). When one parameter is varied, others are held at their default values ($\tau = 0.6, k = 16$). Yellow shaded regions highlight the "sweet spot" where REFUSION surpasses the baseline in both metrics. **Right:** pass@1 on MBPP for REFUSION and baseline MDMs over the average number of tokens generated per forward pass (TPF) under various hyperparameter settings.

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.
Qwen3-8B (Retrained)	52.71 31.32	88.65 30.46	87.57 31.03	67.84 30.73	33.26 26.70	53.66 30.73	59.20 30.16	63.27 30.16
LLaDA (Retrained)	5.12 0.26	70.65 0.03	13.42 1.18	1.38 0.20	0.00 0.15	4.27 0.66	6.20 0.42	14.43 0.41
REFUSION (Retrained)	38.80 35.36	84.64 47.14	79.45 39.67	49.98 44.85	30.36 40.68	62.20 39.54	52.40 38.64	56.83 40.84

More importantly, the shape of this trade-off frontier distinguishes REFUSION from prior MDMs. Figure 4 (right) shows that both LLaDA and Dream suffer a sharp performance decline as parallelism (TPF) increases[3], 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. Expanding on this, Figure 1 presents the complete performance-throughput (TPS) frontier against all baselines, revealing a critical finding: for every baseline model, there exists at least one REFUSION configuration that is simultaneously superior in both performance and throughput.

We would like to emphasize that the hyperparameters (λ in Eq. 3, τ, and k) do not pose a significant tuning burden. First, τ and k are standard concepts in the field of MDMs (Arriola et al., 2025; Nie et al., 2025; Wu et al., 2025b; Wang et al., 2025; Wu et al., 2025a), while the loss-balancing parameter λ required no tuning (fixed at 1). Second, our analysis in Figure 4 confirms high robustness, revealing a wide "sweet spot" (e.g., $\tau \in [0.5, 0.9], k \in \{16, 32\}$) that offers a generalizable starting point. This hyperparameter complexity is thus comparable to standard ARMs, which also depend on tuning inference parameters like temperature and top-p.

5.4 ABLATION STUDY

Controlled Comparison with the Same Backbone and Data. To isolate the benefits of REFUSION from data or backbone advantages, we conduct a controlled comparison using a smaller 120K data subset randomly sampled from the full 3.7M data due to resource constraints. We fine-tune Qwen3-8B, LLaDA, and REFUSION for 10 epochs using their respective original objectives, with all initialized from Qwen3-8B. This setup ensures that observed differences are attributable solely to the algorithm design. Appendix C.3 discusses the scaling property of REFUSION regarding data size.

Results in Table 2 confirm the architectural superiority of REFUSION. LLaDA suffers a catastrophic performance collapse[4]. While the already highly-optimized Qwen3-8B baseline understandably degrades when retrained on our smaller, open-source dataset, REFUSION still outperforms it by $\sim$9 points on HumanEval while being 1.3$\times$ faster. This result robustly validates that REFUSION's

[3]We use TPF here, rather than TPS, to isolate the algorithmic trade-off from any system-level overheads.

[4]The low TPS for LLaDA in this experiment is due to the fair-comparison re-implementation on the Qwen3-8B backbone, which is computationally heavier (36 layers, $\sim$152K vocab) than LLaDA's original architecture (32 layers, $\sim$126K vocab).

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.
Dream-7B-Instruct	**40.05**	**88.31**	76.42	**46.60**	30.36	56.71	50.40	55.55
	15.98	0.06	20.30	18.99	1.81	3.51	1.23	8.84
REFUSION (Retrained)	34.51	83.11	**79.38**	46.38	**31.47**	**69.51**	**60.00**	**57.77**
	50.93	46.49	41.15	45.69	20.90	65.57	58.49	47.03

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
REFUSION w/ KV Re-computation	45.34	89.68	84.91	55.90	35.04	73.78	66.40
	34.13	36.21	36.38	37.57	29.96	41.02	40.13
REFUSION	45.39	89.68	85.60	56.06	35.04	75.61	66.60
	39.38	49.25	40.29	42.40	42.23	46.48	45.34

architectural innovations are the primary driver of its success, enabling effective learning even from limited data where standard MDMs fail.

Furthermore, we conducted a controlled comparison with Dream-7B-Instruct. Since its training code is not open-sourced, we could not re-train it on Qwen3. 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 benefited from massive pre-training (146.5M samples) plus SFT (1.8M samples), whereas our REFUSION variant was only fine-tuned (3.7M samples) without any pre-training. Despite the pre-training disadvantage, Table 3 shows that REFUSION still achieves a 2.22% average performance gain and a massive 5.32× speedup over Dream. REFUSION significantly excels on reasoning and coding tasks (GSM8K, HumanEval, MBPP). Its lower performance on knowledge-intensive tasks (MMLU-Pro, ARC-C) is expected, as it skipped the pre-training stage that Dream utilized for knowledge injection. These results collectively confirm that REFUSION's architectural advantages are robust across different base models and training setups.

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 $1.1-1.4\times$ 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.

5.5 DIRECT COMPARISON WITH BD3-LM

To further contextualize the advantages of our approach, we conduct a direct comparison against BD3-LM (Arriola et al., 2025), a representative block-based MDM. While the performance of baselines like LLaDA (Nie et al., 2025) in our main results already serves as a strong proxy for the limitations of block-based designs, this head-to-head experiment provides direct empirical validation. We ensure a strictly controlled setting by training both REFUSION and BD3-LM from the Qwen3-8B backbone on our 120K data subset for 10 epochs. Table 5 presents the performance-throughput frontier on MBPP as parallelism (measured in average tokens per forward pass, #TPF) increases.

The results in Table 5 clearly demonstrate REFUSION's superiority. As parallelism increases, REFUSION maintains high performance with a gentle degradation curve while consistently achieving higher throughput. In stark contrast, BD3-LM suffers a precipitous performance collapse, with its

Table 5: Performance-throughput frontier comparison with BD3-LM on MBPP (0-shot). Each cell shows pass@1 (left) and TPS (right, in parentheses) at varying levels of parallelism (#TPF).

Average #TPF	1	2	3
REFUSION	56.80 (25.83)	56.00 (36.66)	47.40 (42.08)
BD3-LM	51.60 (16.18)	30.20 (29.31)	16.00 (37.96)

pass@1 score plummeting from 51.6% to just 16.0%. This underscores a fundamental weakness of the block-based design: its reliance on intra-block bidirectional attention hinders both generation coherence (leading to the performance collapse) and full KV-caching (reflected in lower TPS). REFUSION's unified causal framework directly overcomes these issues, validating its architectural advantage.

5.6 CASE STUDY

Figure 5 provides a qualitative understanding of how REFUSION solves a programming problem from the MBPP benchmark, revealing two key capabilities: **(1) High degree of parallelism.** The model frequently generates multiple slots concurrently. For instance, at iteration 9, it simultaneously generates three separate slots. This parallel decoding capability, combined with leveraging speculative decoding to retain multiple diffusion-based generated tokens at once, delivers significant acceleration. **(2) Non-linear generation order.** The generation process is markedly non-linear. For example, the model constructs the central "for" loop structure (iteration 7) before initializing a local variable "sum = 1" (iteration 8). 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.

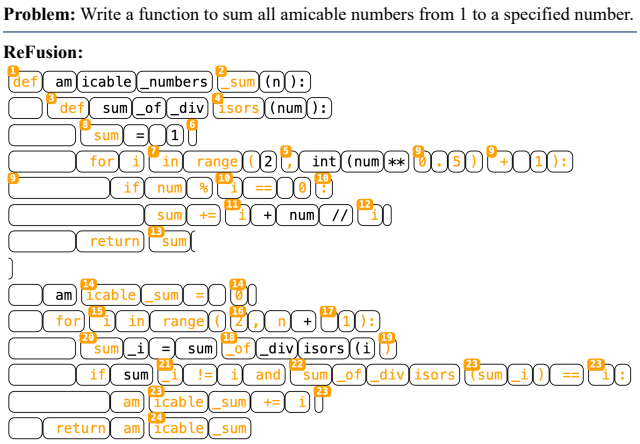

Figure 5: A case study of REFUSION generating a Python function for an MBPP problem. The code is segmented into slots of size $k = 4$. The numbers in the top-left corner of each slot indicate the generation order. The token color indicates the generation source: orange denotes diffusion-based generation, while Black denotes autoregressive generation.

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.

Reproducibility Statement

To ensure the reproducibility of our experimental results, our training and inference code is open-sourced in an anonymized repository `https://anonymous.4open.science/r/ICLR2026-ReFusion`. The specific settings for training and testing are detailed in §5.1 and Appendix B.

References

AI@Meta. Llama 3 model card. 2024. URL `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 Piqueres 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.

Guhao 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.

Shansan 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, Ruifeng 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.

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, Ł ukasz 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/3f5ee243547dee91fbd053c1c4a845aa-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, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm, 2025a. URL https://arxiv.org/abs/2509.26328.

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*, 2025b.

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, 2025c. URL https://arxiv.org/abs/2505.22618.

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

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 Wenhu 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 6: 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
LLaDA	Full Sequence	Bidirectional	Any-Oder	✗	$\sum_{l=1}^{L}\binom{L}{l} \approx 2^L$
BD3-LMs	Intra-block Inter-block	Bidirectional Causal	Any-Order Left-to-Right	✗	$2^k \cdot \frac{L}{k}$
REFUSION	Intra-slot Inter-slot	Causal Causal	Left-to-Right Any-Order	✓	$\frac{L}{k}!$ ($\frac{L}{k}! \ll 2^L$ for large k)

A METHODOLOGICAL DETAILS

A.1 COMPARISON BETWEEN REFUSION AND REPRESENTATIVE MDMS

Table 6 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[5]. 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 ($\frac{L}{k}!$). For a typical sequence length of $L = 4,096$, a slot size of just $k = 8$ is sufficient to ensure $\frac{L}{k}! < 2^L$.

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.

[5]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:

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 $\tilde{\boldsymbol{S}}_t$ for enabling KV cache by concatenating already-decoded clean slots ($\tilde{\boldsymbol{S}}_t^{\text{clean}}$, in generation order) with the remaining masked slots ($\tilde{\boldsymbol{S}}_t^{\text{masked}}$, 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 $\tilde{\boldsymbol{S}}_t^{\text{masked}}$, a draft token $\tilde{d}_t^{i,j}$ is sampled from the model's marginal distribution, conditioned on the leading context:

$$\tilde{d}_t^{i,j} \sim P_\theta(\cdot \mid p_0, \tilde{\boldsymbol{S}}_t^{\text{clean}}, \tilde{\boldsymbol{S}}_{t,\leqslant(i,j)}^{\text{masked}}), \tag{4}$$

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

$$\mathcal{C}(\tilde{S}_t^i) = P_\theta(\tilde{d}_t^{i,1} \mid p_0, \tilde{\boldsymbol{S}}_t^{\text{clean}}, \tilde{\boldsymbol{S}}_{t,\leqslant(i,1)}^{\text{masked}}). \tag{5}$$

The model then selects a batch of slots with scores exceeding a threshold τ 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 using a single autoregressive forward pass. 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:

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

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 τ. If the prefix is long enough to form at least one full slot (i.e., $l \geqslant k$), we accept the first $\lfloor l/k \rfloor$ slots and immediately begin a new planning-infilling iteration, bypassing the costly suffix completion. Otherwise, we find the longest *common* prefix length, $l' \geqslant 1$, that is successfully verified across all individual draft slots. Each slot is then truncated to this length l', and the model proceeds to suffix completion, filling the remaining empty positions in each slot via parallel autoregressive decoding:

$$\tilde{v}_t^{i,j} \begin{cases} = \tilde{d}_t^{i,j}, & \text{if } j \leqslant l' \\ \sim P_\theta(\cdot \mid p_0, \tilde{\boldsymbol{S}}_t^{\text{clean}}, \tilde{v}_t^{i,<j}), & \text{otherwise} \end{cases} \tag{7}$$

where $\tilde{v}_t^{i,j}$ is the finally decoded token at the j-th position of the i-th slot. After infilling each selected slot, the completed slots are moved from $\tilde{\boldsymbol{S}}_t^{\text{masked}}$ to $\tilde{\boldsymbol{S}}_t^{\text{clean}}$. For the subsequent iteration, the KV caches from these parallel-generated slots are concatenated. While this parallel generation forgoes intra-batch conditioning, we observe in our experiments that this has a minimal impact on final performance (see §5.4). 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 $\tilde{\boldsymbol{S}}_0^{\text{clean}}$ back into its original sequence order.

Table 7: Hyperparameter settings for different tasks.

Benchmark	Generation Length	Verification Threshold τ	Slot Size k	Block Size b
MMLU-Pro	512	0.5	16	128
ARC-C	512	0.4	4	16
GSM8K	512	0.7	16	64
MATH	512	0.6	32	64
GPQA	128	0.6	8	16
HumanEval	512	0.6	16	32
MBPP	512	0.6	16	32

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 was approximately 10.68K H20 GPU-hours. We set λ 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 ReFusion inference process, three hyperparameters can be adjusted: the verification threshold τ, the slot size k, and the block size b. Table 7 shows the specific settings used in our evaluation.

C Experiment Results

C.1 Analysis of Generation Length and Latency

To ensure that ReFusion's superior performance is not merely an artifact of generating more tokens than baselines, we present a direct comparison of the average generated token length and total inference latency for representative tasks. The results, shown in Table 8, address the hypothesis that quality gains might stem from quantity.

The data clearly demonstrates that ReFusion's generated outputs are consistently and significantly shorter than those of the ARM baseline (Qwen3-8B) and are either shorter or comparable in length to other MDMs. For instance, on MMLU-Pro, ReFusion generates only 128 tokens, roughly 5$\times$ fewer than Qwen3-8B, while achieving superior performance. This directly refutes the hypothesis that our model's quality gains are achieved by generating longer sequences.

Furthermore, the table highlights ReFusion's dramatic efficiency advantage, with measured latency being substantially lower across all tasks. These results confirm that ReFusion's superior performance-efficiency profile is a direct result of its methodological innovations, enabling it to produce concise and high-quality responses with minimal latency.

Table 8: Comparison of average generated length and total latency (in seconds) across key benchmarks. For each model, the cell shows generated length, with the measured latency in parentheses.

Model	MMLU-Pro	GSM8K	MBPP
Qwen3-8B	654 (20.82s)	300 (9.63s)	53 (1.74s)
LLaDA-8B-Instruct	251 (13.81s)	247 (9.02s)	82 (27.65s)
Dream-7B-Instruct	211 (13.23s)	223 (10.99s)	45 (36.26s)
REFUSION	128 (3.25s)	148 (3.66s)	46 (1.01s)

Table 9: Comparison of certainty score heuristics on zero-shot performance. "Prob. of First Token" is our default method used in Table 1. "Mean Prob. of Slot" is the alternative. Results are highly comparable, validating our design choice.

Method	MMLU-Pro	ARC-C	GSM8K	MATH	GPQA	HumanEval	MBPP
Prob. of First Token	**45.39**	**89.68**	**85.60**	56.06	**35.04**	75.61	66.60
Mean Prob. of Slot	45.18	**89.68**	85.14	**56.14**	33.71	**77.44**	**67.80**

C.2 ANALYSIS OF CERTAINTY SCORE HEURISTIC

A key design choice in our slot planning step is the metric used to compute the certainty score, which determines which slots are selected for parallel generation. In our default implementation, as described in Section A.2, we use the probability of the most likely token at the slot's first position. This choice is motivated by our two-step "plan-and-infill" decoding process. The diffusion-based planning step primarily aims to identify valid starting points for parallel generation. The subsequent autoregressive infilling step is then responsible for coherently completing the rest of the slot, conditioned on this first token. Therefore, the confidence of the initial token serves as an efficient and effective proxy for the overall viability of initiating the slot's generation.

An alternative and intuitive approach is to use the mean probability of the most likely tokens across all positions within a draft slot. To evaluate this alternative, we conducted a comparative experiment. As shown in Table 9, the performance of the two methods is highly comparable across all seven benchmarks. While using the mean probability yields a slight improvement on HumanEval (+1.83) and MBPP (+1.20), our default first-token-based method performs slightly better on MMLU-Pro, GSM8K, and GPQA. This indicates that both metrics are effective and likely select a significantly overlapping set of high-confidence slots for parallel decoding. Given this parity and the slightly simpler computation of the first-token probability, we retain it as our default method, validating our design choice.

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 collected additional, publicly available data to expand our training set to 14M samples. Figure 6 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 7.

The results reveal a clear and positive scaling trend for both key metrics. Specifically, throughput (TPS, dashed lines) consistently improves as the training data size increases. For instance,

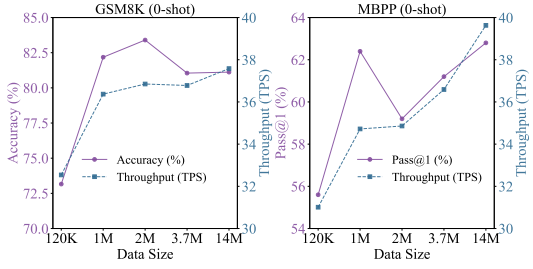

Figure 6: Scaling properties of REFUSION on GSM8K and MBPP. We plot performance (Accuracy/Pass@1, %) and inference throughput (TPS) as a function of training data size.

on MBPP, throughput rises from approximately 31 TPS with 120K samples to over 36 TPS with 3.7M samples, and further to nearly 40 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 a 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. A more pronounced effect is seen on MBPP, where the optimal pass@1 score is achieved with just 1M samples. 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.

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 $b \geqslant k$ must be satisfied, where k is the size of a slot, the fundamental unit for parallel decoding in our method.

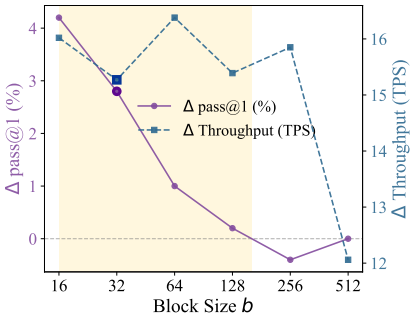

Figure 7: Relative change in ReFusion's **pass@1 (%)** and **throughput (tokens/sec)** compared to Qwen3-8B (horizontal dashed lines at zero) as block size b varies. The yellow shaded region highlights the range of b where ReFusion surpasses Qwen3-8B.

Figure 7 illustrates the impact of block size b on our method's performance and throughput (TPS). The figure shows that performance degrades as b increases, since generating a larger, more complex block in any order is inherently more challenging, although we have mitigated this difficulty through intra-slot serial generation. Throughput (TPS) exhibits a non-monotonic trend, peaking around $b = 64$. This non-monotonic trend is due to computational overhead: while a larger b provides more opportunities for parallelism, it also 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 its performance slightly degrades with larger block sizes, ReFusion's pass@1 decreases by only approximately 4% as b increases from 16 to 512. This robustness to the 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 7, where ReFusion consistently surpasses the Qwen3-8B baseline in both performance and throughput (TPS). This superior operating zone corresponds to a verification threshold $\tau \in [0.5, 0.9]$, a slot size $k \in \{16, 32\}$, and a block size $b \in [16, 128]$.

C.5 Ablation of Confidence-Aware Parallelism

ReFusion already integrates representative MDMs acceleration techniques such as confidence-aware parallelism (Wu et al., 2025c) as a core component of its inference strategy. This is manifested in two key stages: **(1) In the Diffusion-based Slot Planning phase:** We select only those slots whose certainty scores exceed the confidence threshold τ. This ensures that only high-probability slots are considered for parallel generation. **(2) In the Autoregressive Slot Infilling phase:** During

Table 10: Ablation study on the impact of confidence-aware parallelism. Each cell shows accuracy/-pass@1 (left) and throughput in TPS (right, in parentheses).

Method	GSM8K (0-shot)	MBPP (0-shot)	MMLU-Pro (0-shot)
w/ confidence-aware parallelism	85.60 (40.29)	66.60 (45.34)	45.39 (39.38)
w/o confidence-aware parallelism	85.75 (28.95)	68.20 (25.19)	46.15 (28.93)

speculative decoding, we accept the longest prefix of the concatenated draft where the probabilities of all tokens exceed the same threshold τ. This avoids generating low-confidence continuations.

To ablate the impact of this integrated mechanism, we compare our default model against a variant where confidence-aware parallelism is disabled (i.e., always selecting a fixed number of slots with the highest scores, and disabling speculative decoding). As shown in Table 10, incorporating confidence-aware parallelism yields a substantial $\sim$**1.5**$\times$ **speedup** across all tasks, with only a negligible performance drop (0.84% on average). This demonstrates that confidence-aware parallelism is a critical and effective component of ReFusion's efficiency.

D Case Study

D.1 Code Generation Case Study

Table 11 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 8. Specifically, our inference process progressively generates the response through an iterative "plan-and-infill" mechanism.

The model maintains 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.

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

Table 11: Model responses to the same MBPP problem as Figure 5.

Problem

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:

```
assert amicable_numbers_sum(999)==504
assert amicable_numbers_sum(9999)==31626
assert amicable_numbers_sum(99)==0
```

LLaDA-8B-Instruct ✗

```
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)
```

Dream-7B-Instruct ✗

```
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
```

REFUSION ✓

```
def amicable_numbers_sum(n):
    def sum_of_divisors(num):
        sum = 1
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                sum += i + num // i
    return sum

    amicable_sum = 0
    for i in range(2, n + 1):
        sum_i = sum_of_divisors(i)
        if sum_i != i and sum_of_divisors(sum_i) == i:
            amicable_sum += i
    return amicable_sum
```

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.

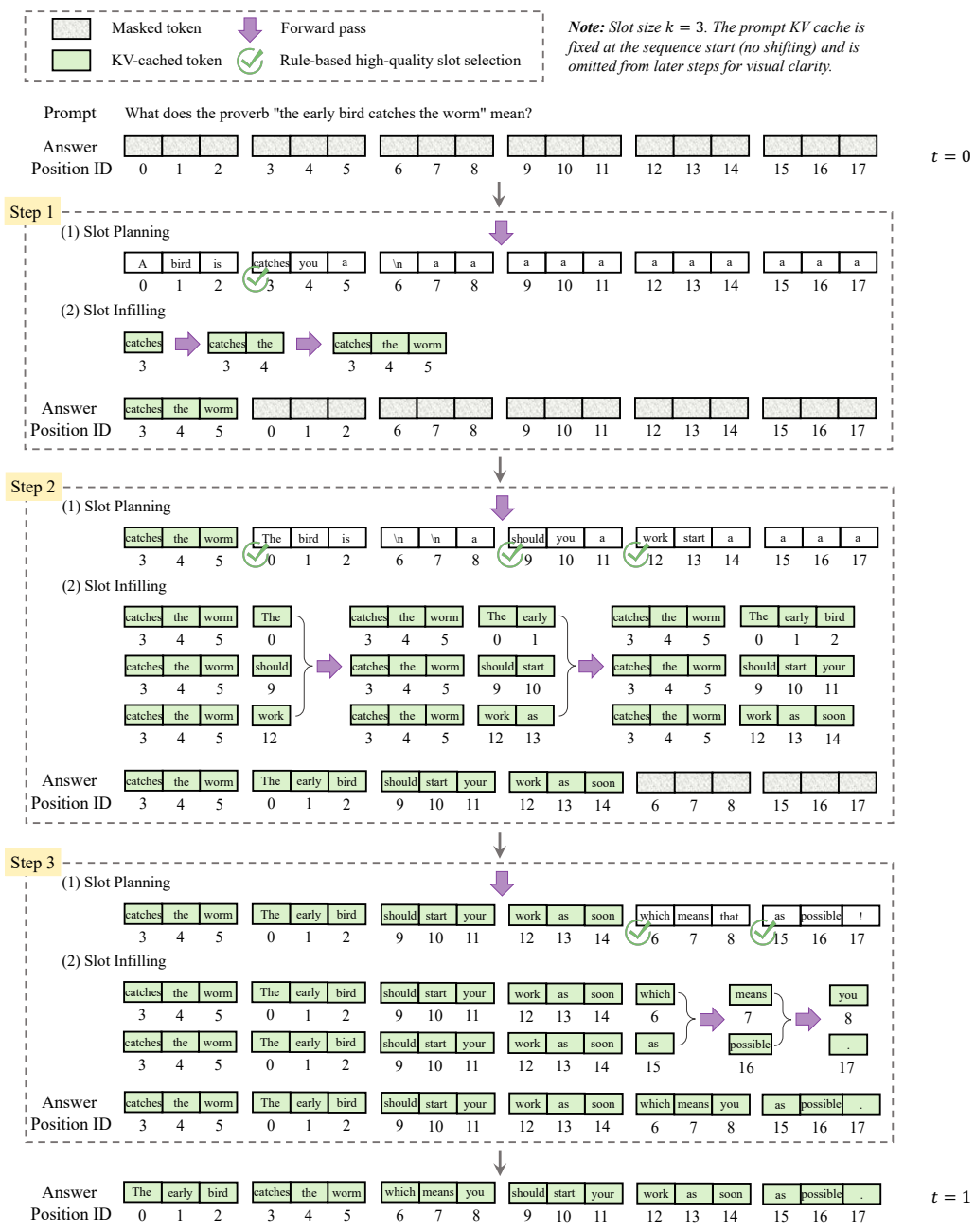

Figure 8: Visualization of the ReFusion inference mechanism.

F The Use of Large Language Models

In the interest of complete transparency, we wish to clarify the use of AI assistance in the preparation of this manuscript. The core research ideas, including the conception of the ReFusion model, the design of the training and inference algorithms, all experimental setups, and the analysis of the results were developed exclusively by the human authors. We utilized a Large Language Model for the limited purpose of linguistic refinement. This involved polishing certain sentences and paragraphs to improve grammatical correctness, clarity, and overall flow. This usage was restricted to editing and did not extend to research ideation, content generation, or experimental analysis.