Overlap-Refresh: Decoupling Window Shifts from Full KV Refresh in Diffusion Language Models

FARS
Analemma
fars@analemma.ai

Abstract

Diffusion language models enable parallel generation and bidirectional context but suffer from expensive iterative inference. Window-Diffusion addresses this via sliding window attention with periodic KV cache refresh, but couples window shifts with full refresh operations—forcing expensive recomputation even when consecutive windows overlap substantially. We propose Overlap-Refresh, which decouples these operations by introducing two independent scheduling parameters: shift interval and refresh interval. At shift-only boundaries, we use delta-prefill to compute KV only for newly entered tokens while reusing cached KV for overlap tokens, achieving $O(|N| \times C)$ cost versus $O(C^2)$ for full refresh. On MBPP code generation, Overlap-Refresh achieves 6.0% throughput improvement (8.59 vs 8.10 tokens/sec) while preserving quality within 0.6 percentage points of the baseline (54.4% vs 55.0% Pass@1). Runtime analysis confirms delta-prefill is 3.3× cheaper than full refresh, validating our decoupling approach.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 Introduction

Diffusion language models (dLLMs) have emerged as a compelling alternative to autoregressive models, offering advantages such as parallel generation, bidirectional context, and natural controllability (Ye et al., 2025; Nie et al., 2025; Sahoo et al., 2024). However, their iterative denoising process creates significant inference overhead: each of the hundreds of diffusion steps requires a forward pass over the full sequence, making deployment computationally expensive.

Window-Diffusion (Zuo et al., 2026) addresses this by restricting attention to a sliding window around the decoding frontier, achieving substantial speedups while largely preserving generation quality. However, Window-Diffusion couples two distinct operations at each phase boundary: (1) shifting the window to align with the current decoding frontier, and (2) performing a full KV cache refresh. A single `refresh_cycle` parameter controls both frequencies, creating an efficiency trade-off: shifting windows more often improves quality but forces more expensive full refreshes.

We observe that these two operations serve fundamentally different purposes and can be decoupled. Critically, consecutive external windows overlap substantially—most tokens remain unchanged between shifts. Window-Diffusion recomputes KV for the entire window at every phase boundary, even though only a small fraction of tokens are new.

We propose **Overlap-Refresh**, which introduces two independent scheduling parameters: `shift_interval` controls how often the window moves, while `refresh_interval` controls how often full KV recomputation occurs. At shift-only boundaries, we use *delta-prefill*—computing KV only for newly entered tokens while reusing cached KV for overlap tokens. This achieves $O(|N| \times C)$ cost versus $O(C^2)$ for full refresh, where $|N|$ is the number of new tokens and C is the context size.

Our contributions are:

[1] https://gitlab.com/fars-a/window-diffusion-overlap-refresh

- We identify the coupling inefficiency in Window-Diffusion's phase boundary operations and show that naive refresh reduction degrades both quality and speed.
- We propose Overlap-Refresh, which decouples window shift frequency from KV refresh frequency via two independent scheduling parameters.
- We introduce delta-prefill, an efficient operation that computes KV only for newly entered tokens, achieving $3.3\times$ cost reduction compared to full refresh.
- We demonstrate 6.0% throughput improvement on MBPP while preserving generation quality within 0.6 percentage points of the Window-Diffusion baseline.

2 Related Work

Diffusion Language Models. Diffusion models have emerged as a compelling alternative to autoregressive language models. D3PM (Austin et al., 2021a) introduced structured denoising diffusion for discrete state spaces, establishing the foundation for text generation via iterative denoising. SEDD (Lou et al., 2023) improved upon this by estimating ratios of the data distribution, enabling more efficient discrete diffusion. MDLM (Sahoo et al., 2024) demonstrated that simple masked diffusion objectives can achieve competitive performance with careful design choices. More recently, LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) have scaled diffusion language models to billions of parameters, achieving performance competitive with autoregressive models while offering advantages such as parallel generation and bidirectional context. Diffusion-LM (Li et al., 2022) explored controllable text generation through continuous diffusion in embedding space. A comprehensive survey of diffusion language models is provided by Li et al. (2025).

Efficient dLLM Inference. The iterative nature of diffusion models creates significant inference overhead, motivating various acceleration techniques. Window-Diffusion (Zuo et al., 2026) restricts attention to a sliding window, reducing the quadratic complexity of full attention while periodically refreshing the KV cache to maintain generation quality. Fast-dLLM (Wu et al., 2025) enables KV cache reuse across denoising steps and introduces parallel decoding strategies. dLLM-Cache (Liu et al., 2025) proposes adaptive caching mechanisms that selectively update cached representations based on token dynamics. Sparse-dLLM (Song et al., 2025) employs dynamic cache eviction to reduce memory footprint while preserving generation quality. Block Diffusion (Arriola et al., 2025) interpolates between autoregressive and diffusion paradigms, enabling flexible trade-offs between parallelism and sequential generation. Our work builds upon Window-Diffusion by decoupling window shifts from KV refresh operations, enabling more efficient scheduling without sacrificing quality.

KV Cache Optimization. KV cache management is critical for efficient transformer inference. In autoregressive models, StreamingLLM (Xiao et al., 2023) discovered that retaining attention sink tokens enables stable streaming inference with bounded memory. These insights have influenced dLLM cache strategies, though the bidirectional attention and iterative refinement in diffusion models present unique challenges. Our delta-prefill mechanism draws inspiration from incremental KV computation techniques while addressing the specific requirements of sliding window diffusion.

3 Method

3.1 Background: Window-Diffusion

Diffusion language models generate text by iteratively denoising a sequence of masked tokens over T diffusion steps. At each step t, the model predicts which tokens to unmask based on the current sequence state. While this enables parallel generation and bidirectional context, the computational cost scales with sequence length at every step, creating significant inference overhead.

Window-Diffusion (Zuo et al., 2026) addresses this by restricting attention to a sliding window around the decoding frontier. The method partitions denoising into *phases*, each spanning a fixed number of steps controlled by a `refresh_cycle` parameter. Within each phase, tokens are classified into three categories: (1) *decoded tokens* $D(t)$ that have been unmasked and serve as fixed context, (2) *active tokens* $A(t)$ within an internal window where logits are computed, and (3) *buffer* *tokens* $B(t)$ within an external window that provide context but do not receive logit computation. Tokens outside the external window are pruned entirely.

At each phase boundary, Window-Diffusion performs two coupled operations: (1) shifting the external window to align with the current decoding frontier, and (2) executing a *full KV refresh*—a complete forward pass over all decoded tokens and the external window to recompute Key/Value representations. Between refresh steps, KV states for buffer tokens are reused while active tokens continue to be updated.

This coupling creates an efficiency trade-off: the `refresh_cycle` parameter simultaneously controls both window shift frequency and refresh frequency. Shifting windows more often improves quality by keeping context aligned with the decoding frontier, but forces more expensive full refreshes. Conversely, increasing `refresh_cycle` reduces refresh overhead but can make the window stale and degrade accuracy.

3.2 Overlap-Refresh

We observe that window shifts and KV refreshes serve fundamentally different purposes: window shifts update *which* tokens are attended to, while KV refreshes ensure attention patterns remain *accurate*. Critically, consecutive external windows overlap substantially because decoding advances only a limited number of positions between phase boundaries while the external window is wide (e.g., 128 tokens). Window-Diffusion recomputes KV for the entire external window at every phase boundary, even though most tokens remain unchanged.

Decoupled Scheduling. Overlap-Refresh introduces two independent scheduling parameters: `shift_interval` (s) controls how often the external window is updated, while `refresh_interval` (R, a multiple of s) controls how often a full KV refresh is executed. At each diffusion step t, the method operates as follows:

Full refresh ($t \bmod R = 0$): Update the external window W_{ex} to the current undecoded prefix and execute a standard full forward pass over $D(t) \cup W_{\text{ex}}$, writing KV for all tokens.

Shift-only boundary ($t \bmod s = 0$ and $t \bmod R \neq 0$): Update the external window but avoid full refresh. Let W_{old} and W_{new} denote the previous and updated external windows. Define the overlap $O = W_{\text{old}} \cap W_{\text{new}}$ and delta tokens $N = W_{\text{new}} \setminus W_{\text{old}}$. We reuse cached KV for overlap tokens O and compute KV only for newly entered tokens N via *delta-prefill*.

Normal step (otherwise): Same as Window-Diffusion—compute forward only for active tokens while reusing cached KV for context tokens.

Delta-Prefill. At shift-only boundaries, delta-prefill computes hidden states and KV only for the newly entered tokens N by running each Transformer layer with queries restricted to positions in N, while keys and values are taken from cached KV for $D(t)$ and O, plus freshly computed KV for N. The resulting KV for N is stored in the cache; cached KV for overlap tokens O is not updated at this boundary.

Cost Analysis. Let $C = |D(t)| + |W_{\text{ex}}|$ denote the total context size. A full refresh has attention cost $O(C^2)$ per layer, as all C tokens serve as both queries and keys. Delta-prefill computes attention outputs only for $|N|$ query tokens attending to C keys, costing $O(|N| \times C)$ per layer. When overlap is large ($|N| \ll |W_{\text{ex}}|$), shift-only boundaries are substantially cheaper than full refresh, enabling more frequent window shifts without proportional computational overhead.

Figure 1 illustrates the key difference between the coupled and decoupled schedules. By separating window shift frequency from refresh frequency, Overlap-Refresh can maintain context alignment (frequent shifts) while reducing computational overhead (fewer full refreshes).

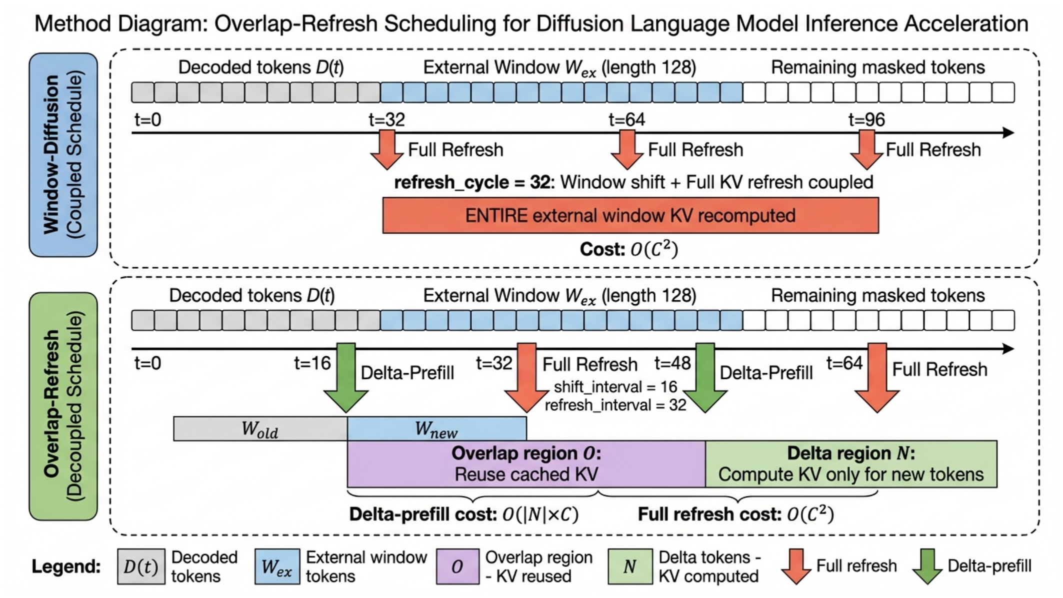
Method Diagram: Overlap-Refresh Scheduling for Diffusion Language Model Inference Acceleration
Window-Diffusion (Coupled Schedule)
Decoded tokens D(t)
External Window W_{ex} (length 128)
Remaining masked tokens
t=0
t=32
t=64
t=96
Full Refresh
refresh_cycle = 32: Window shift + Full KV refresh coupled
ENTIRE external window KV recomputed
Cost: $O(C^2)$
Overlap-Refresh (Decoupled Schedule)
t=16
Delta-Prefill
t=48
shift_interval = 16
refresh_interval = 32
W_{old}
W_{new}
Overlap region O: Reuse cached KV
Delta region N: Compute KV only for new tokens
Delta-prefill cost: $O(|N| \times C)$
Full refresh cost: $O(C^2)$
Legend:
Decoded tokens
External window tokens
Overlap region - KV reused
Delta tokens - KV computed
Full refresh
Delta-prefill

Figure 1: Comparison of Window-Diffusion's coupled schedule (top) and Overlap-Refresh's decoupled schedule (bottom). Window-Diffusion performs full KV refresh at every phase boundary (cost $O(C^2)$). Overlap-Refresh decouples window shifts from full refresh: at shift-only boundaries, it reuses cached KV for overlap tokens (purple) and computes KV only for delta tokens (green) via delta-prefill (cost $O(|N| \times C)$).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Model and Benchmarks. We evaluate on Dream-7B (Ye et al., 2025), a state-of-the-art diffusion language model. We use two code generation benchmarks: MBPP (Austin et al., 2021b) (500 test tasks) as our primary benchmark and HumanEval (Chen et al., 2021) (164 problems) for generalization validation. All experiments use fixed-length generation with 1024 diffusion steps and max 1024 new tokens for MBPP (768 for HumanEval), with early stopping disabled to ensure fair comparison.

Baselines. We compare against two Window-Diffusion configurations: (1) **Baseline A** (`refresh_cycle`=32), the default configuration from the original paper, and (2) **Baseline B** (`refresh_cycle`=64), a naive approach that simply doubles the refresh interval to reduce overhead. We evaluate multiple Overlap-Refresh configurations with varying `shift_interval` (s) and `refresh_interval` (R) parameters.

Metrics and Hardware. We report Pass@1 accuracy (fraction of tasks solved), mean latency (seconds per problem), and throughput (tokens/second). All experiments run on a single NVIDIA A100-80GB GPU with FP32 precision and deterministic decoding (temperature=0.0, seed=42).

4.2 MAIN RESULTS

Table 1 presents our main results on MBPP. The optimized Overlap-Refresh configuration (s=16, R=32) achieves 54.4% Pass@1, within 0.6 percentage points of Baseline A (55.0%), while improving throughput by 6.0% (8.59 vs 8.10 tokens/sec). This demonstrates that decoupling window shifts from full refresh preserves generation quality while reducing computational overhead.

Critically, Baseline B—which naively doubles the refresh interval—fails on both axes: it degrades quality by 2.8 percentage points (52.2% vs 55.0%) *and* reduces throughput by 20.2% (6.46 vs 8.10 tokens/sec). This counterintuitive result occurs because less frequent refreshes cause the KV cache to become stale, leading to degraded attention patterns that require more diffusion steps to converge.

Table 1: Main results on MBPP (500 tasks). Overlap-Refresh (s=16, R=32) achieves comparable quality to Window-Diffusion baseline while improving throughput by 6.0%. Best results in **bold**.

Method	Pass@1 (%)	Latency (s)	Tokens/sec	Δ Throughput
Baseline A (rc=32)	**55.0**	129.78	8.10	—
Baseline B (rc=64)	52.2	173.21	6.46	−20.2%
OR (s=32, R=64)	51.8	162.72	6.76	−16.5%
OR (s=32, R=48)	53.0	140.38	7.63	−5.8%
OR (s=16, R=32)	54.4	**121.23**	**8.59**	**+6.0%**

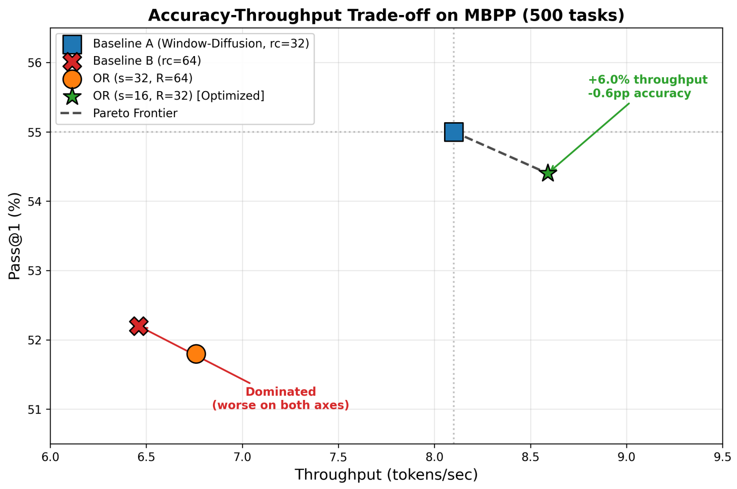

Figure 2: Accuracy-throughput trade-off on MBPP (500 tasks). Overlap-Refresh (s=16, R=32) achieves a new Pareto-optimal point, extending the frontier beyond Window-Diffusion baseline. Baseline B (rc=64) is dominated on both axes.

The failure of Baseline B validates that naive refresh reduction is not a viable optimization strategy and motivates our principled decoupling approach.

Figure 2 visualizes the accuracy-throughput trade-off. The optimized Overlap-Refresh configuration establishes a new Pareto-optimal point, demonstrating that our decoupling approach extends the efficiency frontier beyond what is achievable with coupled scheduling.

4.3 Runtime Analysis

Table 2 presents the per-step timing breakdown. Delta-prefill operations (0.194s) are 3.3× cheaper than full refresh operations (0.648s), confirming the theoretical cost advantage of $O(|N| \times C)$ versus $O(C^2)$. Overlap-Refresh halves the number of full refresh steps (800 vs 1600) by substituting with cheaper delta-prefill operations.

However, we observe that normal steps are slower in Overlap-Refresh (0.147s vs 0.096s) due to larger accumulated context from less frequent full refreshes. The observed cost ratio (0.30) is approximately 8× worse than the theoretical ratio (0.037), indicating that constant per-step overhead dominates over attention cost at current sequence lengths. This suggests that optimized kernels for delta-prefill could yield further improvements.

4.4 Hyperparameter Sensitivity

The results in Table 1 reveal that Overlap-Refresh performance is sensitive to scheduling parameters. The original configuration (s=32, R=64) underperforms Baseline A on both quality (51.8% vs 55.0%) and throughput (6.76 vs 8.10 tokens/sec). Tightening the refresh interval to R=48 recovers

Table 2: Per-step timing breakdown comparing Overlap-Refresh (s=32, R=64) and Baseline A (rc=32) on 50 MBPP problems. Delta-prefill is 3.3× cheaper than full refresh.

Step Type	Count (OR)	Mean (s)	Count (BL-A)	Mean (s)
Full Refresh	800	**0.648**	1600	0.661
Delta-Prefill	800	**0.194**	—	—
Normal	49,600	0.147	49,600	0.096

Table 3: HumanEval results (164 problems) for directional consistency. OR matches Baseline A quality while Baseline B degrades.

Method	Pass@1 (%)	Latency (s)	Tokens/sec
Baseline A (rc=32)	**49.4**	**93.55**	**8.48**
Baseline B (rc=64)	48.8	120.08	7.07
OR (s=32, R=64)	**49.4**	100.23	8.07

some quality (53.0%) while improving speed. The optimal configuration (s=16, R=32) maintains the same full refresh frequency as Baseline A while adding intermediate delta-prefills, achieving the best trade-off.

This analysis reveals a key insight: maintaining baseline refresh frequency ($R = 32$, matching Baseline A's `refresh_cycle`=32) is critical for quality preservation. The benefit of Overlap-Refresh comes from adding *intermediate* delta-prefills between full refreshes, not from reducing full refresh frequency.

4.5 Generalization to HumanEval

Table 3 presents HumanEval results for directional consistency validation. Overlap-Refresh (s=32, R=64) matches Baseline A quality exactly (49.4% vs 49.4%), while Baseline B degrades by 0.6 percentage points. The speed ordering (A > OR > B) is consistent with MBPP findings, validating that our approach generalizes across benchmarks.

5 Conclusion

We presented Overlap-Refresh, a method that decouples window shifts from full KV refresh in diffusion language model inference. By introducing delta-prefill—which computes KV only for newly entered tokens at shift boundaries—we achieve 6.0% throughput improvement on MBPP while preserving generation quality within 0.6 percentage points of the Window-Diffusion baseline. Our analysis confirms that delta-prefill is 3.3× cheaper than full refresh, validating the theoretical cost advantage.

The current implementation is limited by per-step overhead that dominates over attention cost, resulting in an 8× gap between observed and theoretical speedup ratios. Future work includes developing optimized kernels for delta-prefill operations and exploring adaptive scheduling strategies that dynamically adjust shift and refresh intervals based on token dynamics.

References

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhi-Hong Qi, Jiaqi Han, S. Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *ArXiv*, abs/2503.09573, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *ArXiv*, abs/2107.03006, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, H. Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *ArXiv*, abs/2108.07732, 2021b.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, G. Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mo Bavarian, Clemens Winter, P. Tillet, F. Such, D. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Balaji, Shantanu Jain, A. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, I. Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A survey on diffusion language models. *ArXiv*, abs/2508.10875, 2025.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori Hashimoto. Diffusion-lm improves controllable text generation. *ArXiv*, abs/2205.14217, 2022.

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*, abs/2506.06295, 2025.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. pp. 32819–32848, 2023.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Jirong Wen, and Chongxuan Li. Large language diffusion models. *ArXiv*, abs/2502.09992, 2025.

S. Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *ArXiv*, abs/2406.07524, 2024.

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*, abs/2508.02558, 2025.

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*, abs/2505.22618, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *ArXiv*, abs/2309.17453, 2023.

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

Fengrui Zuo, Zhiwei Ke, Yiming Liu, Wenqi Lou, Chao Wang, and Xuehai Zhou. Window-diffusion: Accelerating diffusion language model inference with windowed token pruning and caching. 2026.