SDAR: A Synergistic Diffusion-AutoRegression Paradigm for Scalable Sequence Generation

Shuang Cheng[1,2*], **Yihan Bian**[3*], **Dawei Liu**[1,4*], **Linfeng Zhang**[4], **Qian Yao**[1], **Zhongbo Tian**[1], **Wenhai Wang**[1], **Qipeng Guo**[1], **Kai Chen**[1], **Biqing Qi**[1‡✉], **Bowen Zhou**[1,5✉]

[1]Shanghai AI Laboratory, [2]Zhejiang University, [3]University of Maryland, College Park, [4]Shanghai Jiao Tong University, [5]Tsinghua University

We propose SDAR, a **S**ynergistic **D**iffusion–**A**uto**R**egression paradigm that establishes a new language modeling framework combining the training efficiency of autoregression with the parallel inference capability of diffusion. Instead of costly end-to-end diffusion training, SDAR performs a lightweight paradigm conversion that transforms a well-trained autoregressive (AR) model into a blockwise diffusion model through brief, data-efficient adaptation. During inference, SDAR models generate sequences autoregressively across blocks for global coherence while decoding all tokens within each block in parallel via a discrete diffusion process. Through extensive controlled experiments, we demonstrate that AR models remain substantially more compute-efficient than masked diffusion models, providing a strong foundation for adaptation. Building on this insight, SDAR achieves efficient AR-to-diffusion conversion with minimal cost, preserving AR-level performance while enabling parallel generation. Scaling studies across both dense and Mixture-of-Experts architectures further confirm that SDAR scales without compromise—larger models exhibit increasing robustness to block size and decoding thresholds, yielding greater parallel speedups without loss of accuracy. Beyond efficiency, SDAR also exhibits enhanced reasoning and domain adaptability. Our 30B MoE model surpasses its AR counterpart on challenging scientific reasoning benchmarks such as GPQA and ChemBench, benefiting from local bidirectional context and reduced causal constraints. When combined with test-time scaling strategies such as majority voting and pass@k, SDAR achieves substantial additional gains, indicating strong potential for reinforcement learning optimization. Together, these results establish SDAR as a new and practical language modeling paradigm that unifies the complementary strengths of autoregression and diffusion, enabling scalable, high-throughput inference while preserving the accuracy and reasoning competence of state-of-the-art AR models.

*** Equal contribution** ✉ **Corresponding author** ‡ **Project leader**
Github: https://github.com/JetAstra/SDAR
Huggingface: https://huggingface.co/JetLM

1 Introduction

Large Language Models (LLMs) are primarily built upon the autoregressive (AR) paradigm, modeling sequences from left to right via next-token prediction [1, 18, 20, 7, 43, 44]. This approach imposes a strict, token-level causal inductive bias, which aligns naturally with the sequential structure of natural language. However, the very success of this causal inductive bias creates a fundamental tension. First, the strictly sequential nature of AR decoding—generating token by token—imposes a severe bottleneck on inference speed, precluding parallelization, thereby increasing latency and serving costs at scale. Second, this token-level causal dependency creates a misalignment with tasks that demand non-local or holistic reasoning, particularly in scientific problems where bidirectional attention facilitates the identification of chemical functional groups, DNA sequences, or molecular

formulas. For these tasks, the rigid left-to-right generation process can be detrimental, necessitating substantially greater compute and data to mitigate the limitations of sequential decoding.

As an alternative, masked diffusion language models (MDLMs) have recently become the most widely adopted paradigm among diffusion-based LMs, proposing a fundamentally different philosophy. By treating the entire sequence as a holistic entity to be generated jointly, these models circumvent the strict causal constraints of autoregressive methods, enabling a more flexible generation process with advantages such as arbitrary generation order and parallel decoding [30, 58, 19, 46].

However, MDLMs face two fundamental challenges that limit their training and inference efficiency. First, they typically optimize the Evidence Lower Bound (ELBO) [36, 16, 41], rather than the standard negative log-likelihood. Since NELBO is only a loose upper bound of NLL, this objective is inherently less efficient [40, 3, 14], and empirical studies—both ours and others'—consistently validate a substantial performance gap compared to autoregressive models trained directly on NLL. Second, current open-source implementations [41, 57] incur severe inference costs. The absence of KV caching, coupled with the need to avoid quality degradation, results in an $O(N^3)$ computational complexity per sequence. While some works explore approximate KV caching for diffusion models, it remains unclear whether such methods can resolve the bottleneck without harming other capabilities.

To reconcile the trade-offs between autoregressive and diffusion-based models, a class of hybrid models has been explored to unify the diffusion and autoregressive approaches [21, 3, 13]. The core mechanism involves a block-wise decomposition of the sequence. At a local level, diffusion models handle intra-block generation in parallel, thereby relaxing the strict causal constraints. At a global level, an autoregressive framework models the dependencies between these blocks. This hierarchical strategy advantageously retains the macroscopic causal structure (or "Global AR-ness" [17]) of language, which not only tights the learning objective but also preserves practical functionalities such as variable-length generation and KV-caching. Simultaneously, it alleviates the suboptimal inductive bias of pure AR models on locally complex dependencies, unlocking substantial gains in modeling accuracy and decoding speed.

However, the practical viability of these hybrid architectures is severely hampered by fundamental challenges in training efficiency. The diffusion component, often optimized with slow-converging objectives like the ELBO, incurs a prohibitive computational cost. Empirical studies show that MDLM require substantially more FLOPs to match the performence of their autoregressive counterparts, and often demand prolonged training. Compounding this issue, the hybrid block-wise AR-diffusion strategy itself introduces a significant training overhead, with a composite objective that can nearly double the computational expenditure per instance compared to a pure AR or diffusion strategy [3].

Building on these insights, we design SDAR to reconcile the efficiency of autoregressive training with the parallelism of diffusion-based inference. The key principle is *decoupling* the two phases: we leverage full-scale AR pretraining to ensure stability and efficiency, and then introduce a lightweight adaptation stage that equips the model with block-wise diffusion decoding. This design preserves the practical advantages of AR—such as KV caching, variable-length generation, and strong optimization behavior—while unlocking diffusion's unique benefit of parallel intra-block generation.

To evaluate this paradigm, we conduct two parts of experiments. First, we perform a controlled comparison between AR and masked diffusion language models (MDLMs) to quantify training efficiency under identical computational settings. Based on that, we examine the feasibility of adapting SDAR from both AR and MDLM bases, revealing that AR backbones consistently provide a stronger foundation. Second, we scale SDAR across dense and MoE architectures, systematically analyzing the trade-offs between model size, block size, training cost, and inference speed. Together, these studies provide the first comprehensive assessment of hybrid AR–diffusion language modeling and establish SDAR as a practical recipe for the community.

In summary, our work makes the following contributions:

1. **Fair comparison of AR and MDLM training efficiency.** We conduct controlled experiments comparing autoregressive (AR) models with masked diffusion language models (MDLMs), using the same training data and recommended hyperparameters. This provides the first real-world, meaningful evidence that AR models deliver substantially higher training efficiency than MDLMs under identical computational settings.
2. **Adaptation of SDAR from AR and MDLM bases.** We systematically evaluate the feasibility of adapting SDAR from both AR and MDLM backbones. Using a 100B-token subset of the 1T training corpus, we perform thorough downstream task benchmarks, demonstrating that AR-based adaptation consistently yields superior performance and is therefore the preferred choice for serving as the base model.
3. **Scaling experiments and adaptation recipe.** We extend our study to larger models by adapting state-of-the-art open-source AR models (both dense and MoE variants) using an arbitrary 50B-token open-source dataset. We provide a practical recipe showing that any AR decoder-only model, regardless of its architecture, can be efficiently adapted into SDAR—even without access to its original pretraining data. We argue that this adaptation requires significantly less data than adaptation method such as Dream, making it broadly accessible to the community.
4. **Comprehensive analysis of scaling laws.** We offer detailed empirical analysis of the relationship between model size, block size, downstream task performence and inference speed. Our study spans models from 1.7B to 30B parameters, covering both dense and MoE architectures, and block sizes ranging from 4 to 64. We also show that with minimal additional supervised fine-tuning, block size can be increased from a base SDAR model at negligible cost.
5. **Open-sourced models and inference engine.** We release all trained SDAR models across different sizes(1.7B, 4B, 8B, 30B) and block configurations(from 4 to 64), along with both research-oriented and production-ready inference engines. For research and educational purposes, we provide JetEngine, a lightweight implementation prioritizing code clarity. For industrial applications, we offer a high-performance implementation on the lmdeploy framework with block-oriented optimizations for efficient SDAR inference. Our release also includes SDAR-30B-A3B-Sci, the most powerful diffusion-based reasoning model to date, and the first diffusion model capable of long chain-of-thought generation (see Appendix B), achieving state-of-the-art performance in complex reasoning tasks.

2 Preliminary: Language Modeling Paradigm

We begin by establishing our notation and reviewing the foundational paradigms in language modeling that are pertinent to our work. We consider a vocabulary $\mathcal{V}$ of size V. A token at position ℓ is represented by its index $x^\ell \in \mathcal{V}$, and a sequence of length L is a tuple $x = (x^1, \ldots, x^L)$. For computational purposes, we represent each token x^ℓ by its one-hot vector $\mathbf{x}^\ell \in \{0,1\}^V$. The entire sequence is thus represented by $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^L)$. Finally, we denote the probability simplex over the vocabulary as Δ^V, and use $\mathrm{Cat}(\cdot; p)$ for a categorical distribution with probabilities $p \in \Delta^V$.

2.1 Autoregressive Models

Autoregressive (AR) models model the probability of a sequence x by decomposing the joint distribution into a product of conditional probabilities:

$$\log p_\theta(x) = \sum_{\ell=1}^{L} \log p_\theta(x^\ell | x^{<\ell}) \tag{1}$$

where $x^{<\ell}$ is the sequence of preceding tokens. The models are trained with maximum likelihood estimation on the next-token prediction task. The sequential nature of this process, however, imposes

that generation is iterative; generating a sequence of length L requires L sequential forward passes, which is a major bottleneck for low-latency applications.

2.2 Discrete Masked Denoising Diffusion Models

Discrete masked diffusion models are non-autoregressive generative models that learn to reverse a corruption process [4]. This forward process is typically applied independently to each token. The conditional distribution $q(x_t^\ell | x_0^\ell)$ is a categorical distribution, parameterized by the linear interpolation of the original token's one-hot vector $\mathbf{x}_0^\ell$ and the one-hot vector $\mathbf{m}$ for the special [MASK] token. The interpolation coefficient α_t is governed by a predefined noise schedule, such as a linear one $\alpha_t = 1 - t$ [41, 57]:

$$q(x_t^\ell | x_0^\ell) = \text{Cat}(x_t^\ell; \alpha_t \mathbf{x}_0^\ell + (1 - \alpha_t)\mathbf{m}) \tag{2}$$

The generative model p_θ is trained to reverse this process by minimizing the negative evidence lower bound objective (NELBO). For this absorbing-state formulation, the NELBO simplifies to a reweighted cross-entropy objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_0 \sim p_{\text{data}}, x_t \sim q(x_t|x_0), t \sim U(0,1)} \left[-\frac{1}{t} \sum_{\ell=1}^{L} \mathbf{1}[x_t^\ell = \text{[MASK]}] \log p_\theta(x_0^\ell | x_t) \right] \tag{3}$$

where the indicator function $\mathbf{1}[x_t^\ell = \text{[MASK]}]$ ensures the loss is computed only on tokens replaced by the absorbing state [MASK]. The term $1/t$ is a time-dependent weighting factor derived from the diffusion dynamics under the linear noise schedule $\alpha_t = 1 - t$.

2.3 Blockwise Diffusion Language Models

To enable variable-length generation and the KV cache mechanism for diffusion models, several works have proposed blockwise diffusion frameworks (also known as semi-autoregressive diffusion frameworks) [13, 3, 21]. This hybrid approach partitions a sequence x into B contiguous, non-overlapping blocks, $\{x^1, \ldots, x^B\}$, with each block containing L' tokens. The modeling is autoregressive at the block level while being non-autoregressive within each block:

$$\log p_\theta(x) = \sum_{b=1}^{B} \log p_\theta(x^b | x^{<b}) \tag{4}$$

Each conditional distribution $p_\theta(x^b | x^{<b})$ is modeled by a self-contained diffusion process. When applying the *Discrete Masked Denoising Diffusion* strategy (Section 2.2), the training objective becomes the minimization of the expected conditional NELBO over blocks:

$$\mathcal{L}_{\text{blockwise}}(\theta) = \mathbb{E}_{x \sim p_{\text{data}}, b \sim U[1,B], t} \left[-\frac{1}{t} \sum_{\ell=1}^{L'} \mathbf{1}[x_t^{b,\ell} = \text{[MASK]}] \log p_\theta(x_0^{b,\ell} | x_t^b, x^{<b}) \right] \tag{5}$$

where L' is the block size, x_t^b is the noised sequence for block b, and $x_0^{b,\ell}$ denotes the ℓ-th original token in that block. The model is thus trained to reconstruct the original block x^b from its corrupted version x_t^b, conditioned on the preceding clean blocks $x^{<b}$.

3 Method

The core of SDAR lies in a decoupled paradigm that combines the training efficiency of autoregressive (AR) modeling with the parallel inference capability of diffusion. Instead of performing costly end-to-end block-diffusion training, SDAR leverages a well-trained AR model as a foundation and introduces

a highly efficient adaptation procedure that transforms it into a block-wise diffusion model. This process enables parallel generation while preserving the AR model's strong language understanding and reasoning capabilities. After adaptation, a standard instruction-based supervised fine-tuning (SFT) stage is applied to align the model with human preferences, thereby endowing it with conversational and reasoning abilities at minimal additional cost.

3.1 The SDAR Training Paradigm

Starting from a sufficiently capable AR base model θ_{AR} via conventional NTP pretraining, we subsequently continue training the model to convert the language modeling paradigm from AR to block-wise diffusion. This strategy avoids the prohibitive computational cost of training a block-wise diffusion model from scratch, which is markedly less efficient than AR training.

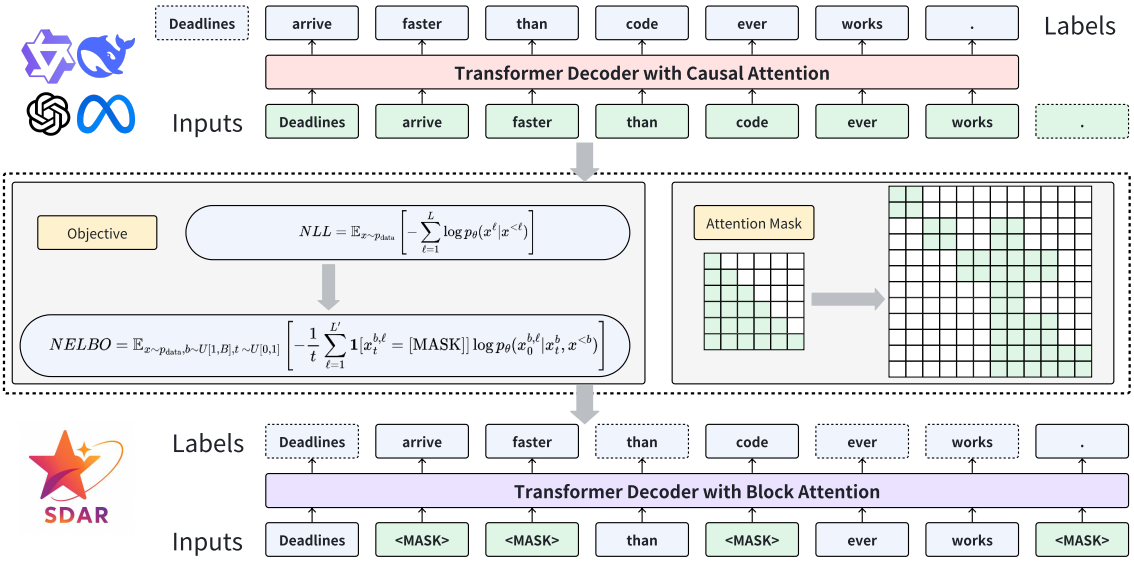

Figure 1: The training paradigm of converting AR to SDAR. First the traditional NTP training is adopted to obtain the AR base model. We then continue training the AR model by modifying the attention mask and replacing the training objective from NLL to NELBO, converting it into a block-wise diffusion model (SDAR), without logits shift or attention mask annealing.

We first perturb the sequence by randomly masking tokens following the MDLM procedure. The entire sequence is perturbed during the pretraining stage, whereas during SFT, the prompt remains clean and the response is perturbed. We partition a sequence x into K non-overlapping blocks of size B, such that $x = (b_1, b_2, \ldots, b_K)$, where each block $b_k = (x_{(k-1)B+1}, \ldots, x_{kB})$. The modeling objective now becomes a hybrid of inter-block autoregression and intra-block parallel diffusion, as defined in the block-wise diffusion preliminary. The probability of a sequence is factorized as:

$$P(x|\theta) = \prod_{k=1}^{K} P(b_k|b_{<k};\theta). \tag{6}$$

Each conditional term $P(b_k|b_{<k};\theta)$ is modeled using a discrete diffusion process. Specifically, during this training stage, the model (initialized with θ_{AR}) learns to denoise a corrupted block b_k^t at diffusion step t, conditioned on the preceding clean blocks $b_{<k}$. The clean blocks adopt block-wise causal attention, while corrupted blocks attend to themselves and all preceding blocks. As illustrated in Figure 1, this can be achieved by concatenating the perturbed and clean sequences into a single input

for the forward pass, with the attention mask modified accordingly [3]. The objective is to optimize the conditional block-wise ELBO instead of log-likelihood, as introduced in the preliminaries:

$$\mathcal{L}_{\text{blockwise}}(\theta) = \mathbb{E}_{t,q(b_k^t|b_k^0)}\left[\log P(b_k^0|b_k^t, b_{<k};\theta)\right]. \tag{7}$$

The conversion process does not involve logits shift or attention mask annealing. Crucially, this conversion is performed on a significantly smaller dataset (e.g., 30 B $\sim$ 50 B tokens, compared to trillions for pretraining). The pretrained AR model provides a powerful initialization, allowing for rapid convergence on the new objective. This step effectively "unlocks" the model's latent ability to perform holistic, non-causal reasoning within local blocks, preparing it for parallel inference.

3.2 Hierarchical Inference and Decoding Strategies

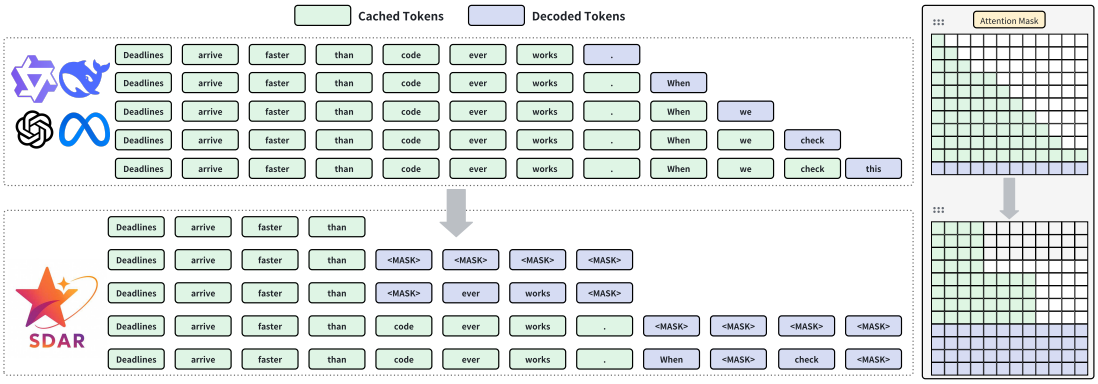

Figure 2: Contrasting inference paradigms between AR and SDAR. SDAR adopts a block-wise causal attention mechanism to enable inter-block causal autoregression and intra-block parallel diffusion. The KV cache from previously generated blocks is reused, while the block currently being decoded does not store KV cache.

The inference process of SDAR is hierarchical: it generates text autoregressively at the block level while employing a parallel diffusion-based mechanism for decoding within each block. This hybrid approach marries the long-range coherence of AR models with the speed of parallel generation.

Given a prompt, sequence generation proceeds by sequentially producing blocks $b_1, b_2, \ldots, b_K$. The generation of each block b_k is conditioned on the entire preceding context, which consists of the initial prompt followed by any previously generated blocks $(b_1, \ldots, b_{k-1})$.

To generate a single block b_k, we initiate a parallel decoding process starting from a sequence of B '[MASK]' tokens, which we denote as the fully-noised state b_k^T. The model then iteratively refines this block over T denoising steps. At each step t (from T down to 1), the model takes the current state of the block b_k^t and the preceding context to predict the logits for the less noisy block b_k^{t-1}:

$$\text{logits}(b_k^{t-1}) \sim P(\cdot|b_k^t, b_{k-1}, \text{prompt};\theta). \tag{8}$$

From the resulting logits, a subset of the masked positions is selected for decoding. For each selected position, a token is then generated from its corresponding probability distribution using a standard decoding strategy (e.g., greedy search, nucleus sampling). The remaining positions are kept as '[MASK]' tokens, forming the input b_k^{t-1} for the next iteration. This denoising loop continues until all positions in the block are filled, at which point the model proceeds to generate the next block, b_{k+1}. To determine which positions to decode at each step, we propose two primary remasking strategies:

Static Low Confidence Remasking. This strategy decodes a fixed number of tokens at each step. For a block of size B and a total of T denoising steps, we select the $\lceil B/T \rceil$ masked positions with the highest prediction confidence (i.e., the highest probability assigned to the most likely token) to be decoded in the current step. This approach is simple and predictable, ensuring the generation of a block completes in a constant number of forward passes.

Dynamic Low Confidence Remasking. This adaptive strategy accelerates decoding by finalizing tokens based on model confidence. At each step, we select any masked position where the prediction confidence exceeds a predefined threshold τ. To guarantee progress, a minimum number of positions (e.g., one, or $\lceil B/T \rceil$) are always selected based on highest confidence if the threshold is not met by a sufficient number of candidates. This allows the model to fill in "easy" parts of a block in fewer steps, reducing the overall number of required forward passes for generation.

4 From AR and Diffusion to SDAR

Takeaway

(1) **AR models are more compute-efficient** than masked diffusion objectives, making them the stronger foundational choice under equal budgets.
(2) **SDAR enables efficient AR-to-Block Diffusion conversion**, preserving AR performance while adding parallel decoding.
(3) **AR backbones remain superior post-conversion**, with AR-derived SDAR models consistently outperforming MDLM-based ones.

Before introducing SDAR, we begin by revisiting the fundamental comparison between Autoregressive (AR) and Masked Diffusion Language Models (MDLMs). Under identical architectures, datasets, and training budgets, AR models consistently achieve lower training loss and better downstream performance than MDLMs. This efficiency arises because AR training directly optimizes the exact negative log-likelihood (NLL) through a cross-entropy objective, where every token in the sequence contributes to parameter updates, maximizing the utility of each sample. In contrast, MDLMs approximate this objective via a loose upper bound—the negative evidence lower bound (NELBO)—and rely on random masking, where only the masked subset of tokens contributes to the loss while the remaining unmasked tokens serve merely as context. Consequently, a substantial portion of the computation does not directly improve the model, resulting in slower convergence and higher FLOPs to reach comparable validation perplexity and downstream task performence.

This efficiency gap makes AR the natural foundation for practical large-scale training. However, AR's left-to-right causality restricts parallel decoding, while MDLMs offer flexibility in generation order but lack KV caching and variable-length decoding, limiting real-world usability. Blockwise Diffusion (BD) provides a potential middle ground by enabling intra-block parallelism with block-level KV caching, but pretraining BD from scratch is prohibitively expensive, effectively doubling context length and compute requirements.

To resolve this tension, we propose SDAR, a lightweight conversion method that adapts an existing AR or MDLM backbone into BD through short continued training. SDAR retains AR's superior training efficiency while inheriting BD's parallel decoding capability, thereby combining the best of both paradigms without incurring the cost of full BD pretraining.

Our study therefore asks two controlled questions:

1. Can an AR model be converted into a strong BD model at low cost, gaining parallelism without sacrificing quality?

2. Between AR and MDLM backbones, which provides the stronger initialization for BD adaptation?

4.1 Experimental Setup

Training Setup We pretrain two 2B-parameter models from scratch on an identical 1T-token corpus (general web, STEM, and code), using the same architecture, tokenizer, optimizer, and context length (4096 with packed sequences). The AR model uses variable-length causal attention; the MDLM uses full bidirectional attention, which is standard and effective for diffusion-style training [41].

After pretraining, we perform a 100B-token annealing training phase to obtain four Base models:

1. **AR-2B-Base:** The pre-trained AR model is continually trained with its original autoregressive objective.
2. **SDAR-2B-Base:** The pre-trained AR model is converted to the Block Diffusion paradigm, with a block size of 16. This is our proposed SDAR model.
3. **MDLM-2B-Base:** The pre-trained MDLM is continually trained with its original masked diffusion objective.
4. **MDLM-BD-2B-Base:** The pre-trained MDLM is converted to the Block Diffusion paradigm, with a block size of 16.

Each Base model then undergoes supervised fine-tuning (SFT) on the same 4B-token instruction-following dataset using the same strategy as in its annealing phase, producing four chat variants: AR-2B-Chat, AR-BD-2B-Chat (SDAR-2B-Chat), MDLM-2B-Chat, and MDLM-BD-2B-Chat.

Evaluation Setup We evaluate AR-2B-Chat, AR-BD-2B-Chat, MDLM-2B-Chat, and MDLM-BD-2B-Chat on downstream tasks that probe reasoning, mathematics, and code:

- **Reasoning & Knowledge**: BBH [51] (3-shot), MMLU [22] (5-shot), GPQA-Diamond [45] (0-shot).
- **Mathematics**: MATH [31] (4-shot, CoT), GSM8K [11] (4-shot, CoT).
- **Coding Tasks**: HumanEval [10] (0-shot), MBPP [5] (3-shot).

Our evaluation employs distinct decoding strategies matched to each model architecture. The AR-2B-Chat model uses standard greedy decoding. For our Block Diffusion variants, SDAR-2B-Chat and MDLM-BD-2B-Chat, we utilize a lowest-confidence static decoding method with a block length of 16 and 16 denoising steps. The MDLM-2B-Chat follows the LLaDA [41] protocol, using a lowest-confidence semi-autoregressive remasking approach with hyper-parameters set according to the original publication.

4.2 Summary of Evaluation Results

Table 1: Benchmark Performance of 2B Chat Models. Our proposed SDAR model (**AR-BD**) successfully preserves the high performance of the original AR model, validating our low-cost conversion approach. It also consistently outperforms the MDLM-based counterparts on key reasoning benchmarks. All models are evaluated as chat variants.

Model	BBH	MMLU	MATH	GSM8K	HumanEval	MBPP	GPQA
AR-2B-Chat	35.7	48.7	29.9	61.8	42.1	44.4	26.3
MDLM-2B-Chat	32.2	47.0	12.6	57.9	21.3	27.2	26.3
AR-BD-2B-Chat-b16	35.9	50.9	26.8	59.4	40.0	42.9	28.2
MDLM-BD-2B-Chat-b16	32.9	47.5	23.3	64.7	39.0	33.5	29.8

The evaluation results in Table 1 reveal three key findings.

First, there is a clear performance gap between the autoregressive and masked diffusion baselines. AR-2B-Chat substantially outperforms MDLM-2B-Chat across nearly all benchmarks, particularly in mathematics (MATH: 29.9 vs. 12.6) and coding (HumanEval: 42.1 vs. 21.3), highlighting the superior training efficiency and generalization of the AR paradigm under identical compute budgets.

Second, our proposed conversion strategy successfully preserves the strength of the AR backbone. AR-BD-2B-Chat (SDAR) achieves performance largely on par with AR-2B-Chat, even slightly surpassing it on knowledge-intensive tasks such as MMLU (+2.2) and GPQA (+1.9). The small trade-offs observed on MATH (-3.1) and HumanEval (-2.1) confirm that the conversion incurs negligible quality loss while enabling parallel decoding. This validates our central claim: AR models can be efficiently adapted into the Block Diffusion paradigm at low cost.

Third, converting MDLMs into Block Diffusion also yields significant gains. MDLM-BD-2B-Chat consistently outperforms its own MDLM baseline, with notable improvements on reasoning and coding benchmarks (e.g., MATH nearly doubles from 12.6 to 23.3). However, even after this improvement, MDLM-BD still lags behind AR-BD in most domains, reaffirming that AR is the stronger architectural starting point for Block Diffusion adaptation.

In summary, the results demonstrate that (i) AR baselines are inherently more effective than MDLMs, (ii) AR-BD maintains the performance of AR while adding parallel decoding, and (iii) MDLM-BD benefits from the conversion but cannot close the fundamental gap with AR-based models.

5 Scaling SDAR: Principles and Practice

Takeaway

(1) Low-cost AR to SDAR adaptation works for any modern AR base, delivering **on-par performance with broad applicability.**
(2) Larger SDAR models tolerate bigger blocks and looser decoding thresholds, enabling **higher parallel efficiency without sacrificing performance.**
(3) In SDAR, larger models and higher confidence not only improve quality but also drive faster decoding, **making accuracy the engine of efficiency.**

In Section 4, we conducted a rigorous and fair comparison that established the foundational viability of our SDAR framework. However, those initial experiments were performed on a relatively small scale, utilizing a 1.7B parameter model trained on 1T tokens. To thoroughly investigate the scalability of our approach and develop production-ready models, this section extends our study to a much larger and more capable family of models.

To mature SDAR into a production-ready paradigm, our investigation seeks to replace empirical guesswork with a principled understanding of its scaling behavior. This requires answering two fundamental, sequential questions:

- **Intrinsic Efficiency of the SDAR Paradigm**: What is the intrinsic inference speedup of SDAR in downstream tasks, free of any external optimizations? This establishes the core efficiency we seek to scale.
- **The Interplay of Model Scale and Block Size**: How does this efficiency depend on the interplay between model size and 'blocksize'? By modeling this relationship, we aim to derive a predictive scaling law, enabling practitioners to configure larger models optimally and efficiently, thus providing a critical roadmap for future work.

5.1 Scaling Performance of the Qwen3 Family on Downstream Benchmarks

5.1.1 Experimental Setup

Training Setup We investigate scaling properties using the Qwen3 series [55] as our base models, starting from the Qwen3-1.7B, 4B, 8B, and 30B-A3B base models. To adapt these models for downstream tasks, we adopt a two-stage training strategy:

1. **Continued Pre-training (CPT):** We first continue pre-training each base model on a 50B-token subset randomly sampled from the 1T-token corpus described in Section 4, using a context length of 4096 with packed sequences.
2. **Supervised Fine-tuning (SFT):** Following CPT, the models are supervised fine-tuned on a high-quality 4B-token instruction-following dataset.

This procedure is applied to generate our main models, the **SDAR-Chat** series (1.7B, 4B, 8B, 30B-A3B). To establish strong baselines, we also apply the identical CPT and SFT pipeline to produce standard autoregressive counterparts, termed **AR-Chat** series.

Evaluation Setup We conduct a comprehensive evaluation of our models on a suite of downstream benchmarks designed to assess key capabilities in reasoning, mathematics, code generation, and instruction following. The evaluation datasets are organized as follows:

- **Reasoning & Knowledge:** , MMMLU-lite [22] (0-shot), TriviaQA [26] (1-shot), MMLU [22] (5-shot), CMMLU [29] (0-shot), MMLU-Pro [54] (0-shot, CoT), and GPQA-diamond [45] (0-shot).
- **Mathematics:** GSM8K [11] (0-shot, CoT), MATH-500 [31] (0-shot, CoT), MathBench (0-shot, CoT) [33], and the challenging competition-level benchmarks AIME-2024, AIME-2025 [2] and LiveMathBench-Hard (LMB-Hard) [34].
- **Code Generation:** HumanEval [10], HumanEval-X [59], MBPP [5], and LiveCodeBench (LCB) [25], all evaluated in a zero-shot setting.
- **Instruction Following:** IFEval [60] (0-shot).

For our evaluation protocol, we employ greedy static decoding for the SDAR-Chat models, with both the block length and the number of denoising steps set to 4. To ensure a fair comparison, all autoregressive models, including our AR-Chat and external baselines, are evaluated using standard greedy decoding. Our primary baselines are the original Qwen3-Base series, with performance metrics cited directly from the Qwen3 Technical Report [55]. To position our work in a broader context, we also include results for LLaDA and Dream, as reported in [41, 57].

5.1.2 Summary of Evaluation Results

Our comprehensive evaluation reveals several key insights about the scaling behavior and performance characteristics of SDAR models compared to their autoregressive counterparts and other diffusion-based approaches.

Scaling Without Compromise As detailed in Table 2, the SDAR paradigm demonstrates robust scaling properties analogous to traditional AR models. As we increase model size from 1.7B to 30B parameters, performance consistently improves across all benchmark categories, with MMLU scores scaling from 62.9% to 82.8%, for instance. This confirms that the benefits of scale are effectively preserved. Crucially, this scaling does not come at the cost of performance. At the 30B scale, SDAR-Chat achieves parity or surpasses its AR-Chat counterpart on 11 out of 18 benchmarks. This advantage is particularly pronounced in complex reasoning domains. For instance, SDAR models show a notable aptitude for structured generation, outperforming the AR model on key code generation tasks such as

Table 2: Comprehensive performance comparison of AR and SDAR models. † indicates that the models are derived from the Qwen3 Base models by performing the identical CPT and SFT. For each benchmark, the best result within a given scale (e.g., 1.7B, 30B) is shown in **bold**. The colored superscript for SDAR models indicates the performance difference relative to the corresponding AR-Chat baseline.

Model	**AR-Chat Model**				**SDAR-Chat Model**			
scale	Qwen3-1.7B†	Qwen3-4B†	Qwen3-8B†	Qwen3-30BA3B†	SDAR-1.7B	SDAR-4B	SDAR-8B	SDAR-30BA3B
Reasoning, Knowledge & Instruction Following								
ARC-C	81.0	89.5	91.9	93.9	**85.4** (+4.4)	**90.5** (+1.0)	**91.9** (0.0)	93.2 (-0.7)
TriviaQA	45.0	57.9	68.2	66.5	42.6 (-2.4)	57.2 (-0.7)	**69.2** (+1.0)	**75.9** (+9.4)
MMLU	63.8	74.8	77.5	82.2	62.9 (-0.9)	**74.9** (+0.1)	**78.6** (+1.1)	**82.8** (+0.6)
MMLU-Pro	39.0	52.8	56.6	63.8	37.0 (-2.0)	50.9 (-1.9)	**56.9** (+0.3)	61.5 (-2.3)
GPQA-diamond	28.6	31.4	37.0	37.3	**29.8** (+1.2)	**33.0** (+1.6)	**40.2** (+3.2)	36.7 (-0.6)
IFEval	43.3	58.4	60.8	57.7	**43.4** (+0.1)	56.6 (-1.8)	**61.4** (+0.6)	**60.6** (+2.9)
Mathematics								
GSM8K	81.1	90.7	92.8	92.7	80.1 (-1.0)	89.9 (-0.8)	91.3 (-1.5)	91.4 (-1.3)
Math500	62.0	74.1	78.4	76.8	**63.2** (+1.2)	72.8 (-1.3)	**78.6** (+0.2)	**77.8** (+1.0)
MathBench	60.5	70.6	73.5	78.4	**63.6** (+3.1)	**74.7** (+4.1)	**76.9** (+3.4)	**79.3** (+0.9)
AIME-24	7.1	12.9	10.0	15.4	**10.0** (+2.9)	**10.0** (-2.9)	10.0 (0.0)	**16.7** (+1.3)
AIME-25	3.3	5.0	7.5	10.8	2.1 (-5.9)	**7.5** (+2.5)	**10.0** (+2.5)	10.8 (+0.0)
LMB-Hard	9.2	11.6	15.7	15.5	6.6 (-2.6)	6.9 (-4.7)	8.9 (-6.8)	13.7 (-1.8)
Code Generation								
HumanEval	65.9	73.4	80.7	84.8	61.6 (-4.3)	72.8 (-0.6)	78.7 (-2.0)	**87.2** (+2.4)
MBPP	61.9	67.1	75.1	75.1	61.1 (-0.8)	65.4 (-1.7)	72.0 (-3.1)	**71.6** (+3.5)
HumanEval-X	47.0	60.9	63.5	63.5	45.2 (-1.8)	**62.3** (+1.4)	**64.9** (+1.4)	**66.3** (+2.8)
LCB-v6	8.6	10.3	20.5	23.4	5.7 (-2.9)	**13.1** (+2.8)	16.6 (-3.9)	21.7 (-1.7)
Language								
MMMLU-lite	36.7	44.4	55.2	52.9	40.9 (+4.2)	**50.7** (+6.3)	**55.3** (+0.1)	57.2 (+4.3)
CMMLU	60.7	72.6	76.2	82.0	60.2 (-0.5)	71.3 (-1.3)	75.7 (-0.5)	81.0 (-1.0)

HumanEval (+2.4%) and HumanEval-X (+2.8%). They also exhibit consistently strong performance on mathematical benchmarks like Math500 and MathBench. These results establish that the SDAR framework not only scales effectively but also preserves, and in some cases enhances, the core reasoning capabilities of the base models.

Positioning SDAR as a State-of-the-Art Paradigm In Table 3, we further contextualize these findings by comparing SDAR with the original Qwen3-Base models and other prominent non-autoregressive architectures like LLaDA [41] and Dream [57]. Our SDAR models substantially outperform these prior diffusion-based approaches across all key benchmarks. For example, our SDAR-8B model achieves 78.6% on MMLU, a significant leap over LLaDA-8B (65.9%) and Dream-7B (69.5%). This substantial performance gap highlights that SDAR represents a major advancement for non-autoregressive generation, making it truly competitive with top-tier AR models for the first time. Furthermore, the SDAR-Chat models demonstrate enhanced instruction-following capabilities over their AR-Chat counterparts (e.g., a +2.9% gain on IFEval at the 30B scale), suggesting the block-wise approach may better reinforce alignment. Taken together, these results validate that our two-stage training strategy provides a robust foundation and positions SDAR as a leading paradigm, not only advancing beyond previous diffusion models but also establishing itself as a powerful and capable competitor to autoregressive generation.

Highly Efficient Adaptation A key advantage of our approach lies in its extremely low adaptation cost. Unlike DiffuLLaMA, which relies on 65B tokens for conversion, or Dream, which requires 580B tokens yet still suffers from substantial degradation relative to their AR bases (LLaMA and Qwen, respectively), SDAR achieves strong performance with only 50B tokens of continued pretraining. This

efficiency stems from the fact that AR pretraining provides dense supervision, so the subsequent block-diffusion adaptation requires only a short training phase to align objectives. We further validate this by experimenting with smaller adaptation budgets of 20B, 30B, and 40B tokens. While these settings also yield functional SDAR models, they result in slightly lower downstream task performance and reduce flexibility for scaling block sizes during subsequent SFT. Importantly, even at these lower budgets, the degradation is far less severe than observed in prior diffusion-based conversions. Moreover, our adaptation relies on relatively low-quality open-source corpora, in contrast to the high-quality synthetic 6T data used in the final stage of Qwen3 pretraining. We therefore believe that with better data quality and distributional alignment, the adaptation phase could operate with an even smaller budget without sacrificing performance.

Table 3: Performance comparison between SDAR and AR models at different scales. Best results within each scale are in bold.

Scale	Model	MMLU	GSM8K	Math500	HumanEval	MBPP	IFEval
AR Models							
1.7B	AR-Chat	63.8	81.1	62.0	65.9	61.9	43.3
	Qwen3-Base	62.6	75.4	43.5	–	55.4	–
30BA3B	AR-Chat	82.2	92.7	76.8	84.8	75.1	57.7
	Qwen3-Base	81.4	91.8	59.0	–	74.4	–
Diffusion Models							
8B	LLaDA	65.9	78.6	37.3	47.6	34.2	59.9
7B	Dream	69.5	81.0	38.7	55.5	58.8	**62.5**
SDAR Models							
1.7B	SDAR-Chat	62.9	80.1	63.2	61.6	61.1	43.4
4B		74.9	89.9	72.8	72.0	65.4	56.6
8B		78.6	91.3	78.6	78.7	72.0	61.4
30BA3B		82.8	91.4	77.8	87.2	71.6	60.6

5.2 Scaling Dynamics: A Trade-off Analysis of Performance, Efficiency, and Model Dimensions

To systematically chart the performance landscape of the SDAR framework, we conduct a multi-dimensional analysis across model scale, architectural hyperparameters, and inference strategies. This factorial experimental design allows us to deconstruct the intricate trade-offs between generative performance and computational efficiency. The resulting analysis provides a principled, empirical guide for configuring and deploying SDAR models to meet specific operational requirements.

5.2.1 Experimental Setup

Our analysis systematically varies three primary factors:

- **Model Scale:** We evaluate all models in the SDAR-Chat family: 1.7B, 4B, 8B, and 30B-A3B.
- **Block Size (B):** We sweep the block size, a core hyperparameter of SDAR, across a range of values from 4 to 64.
- **Decoding Strategy:** We investigate two distinct decoding regimes:

- **Static Decoding:** In this mode, the generation of each block involves a full sequence of B iterative denoising steps, where B is the model's architectural block size. This exhaustive refinement process maximizes output quality, thereby establishing the **performance ceiling** for the given architecture. All dynamic, early-exit strategies are thus evaluated as trade-offs against this quality benchmark.
- **Dynamic Decoding:** To enable a finer-grained trade-off, this mode dynamically adjusts the number of tokens generated per step, k (where $1 \leq k \leq B$), based on a confidence threshold τ. The number of denoising steps equals the chosen block size for that step. We evaluate a range of thresholds, $\tau \in \{0.80, 0.85, 0.90, 0.95\}$, to explore the spectrum between aggressive and conservative parallel generation.

Evaluation Metrics. We assess the models along two axes:

- **Performance:** Model performance is quantified using the comprehensive benchmark suite detailed in Section 5.1.1.
- **Efficiency:** We measure computational efficiency via the **Effective Tokens Per Forward Pass (TPF)**, defined as:

$$\text{TPF} = \frac{\text{Total Generated Tokens}}{\text{Total Forward Passes}}$$

This hardware-agnostic metric directly quantifies the algorithmic speedup of parallel decoding by measuring the average token throughput per inference step. The relative speedup is then benchmarked against a standard autoregressive (AR) model, for which TPF is axiomatically 1.

5.2.2 Performance Landscape Across Model Scale and Block Size

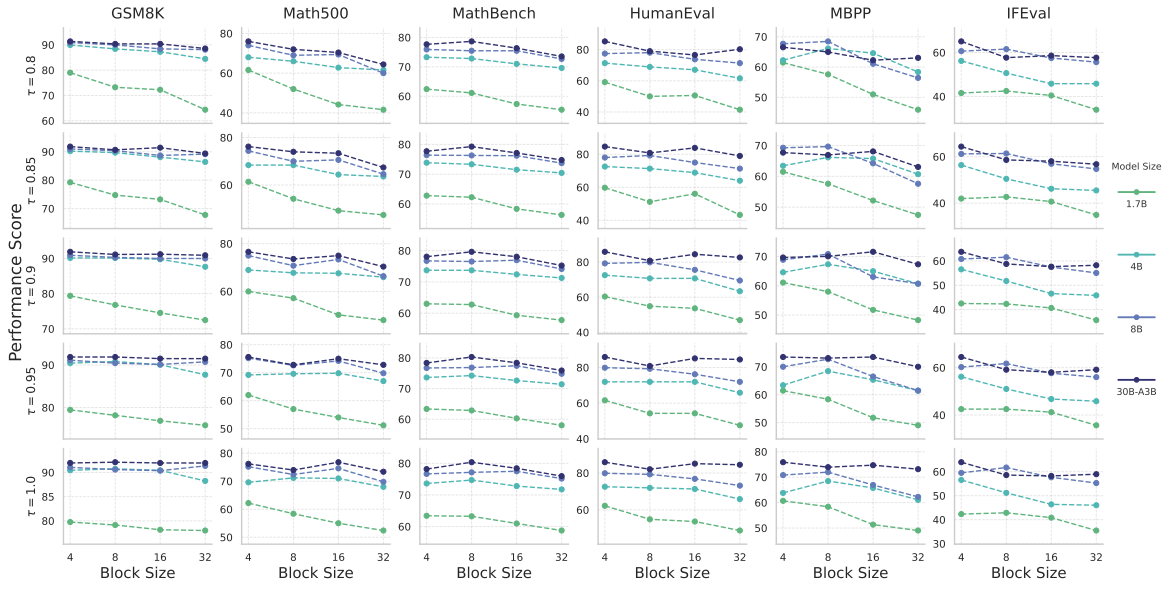

Figure 3: Performance on various benchmarks as a function of architectural block size (B) across different model scales and various confidence thresholds (τ).

Figure 3 and Figure 4 chart the generative performance of SDAR models as a function of their architectural block size (B) and the dynamic decoding confidence threshold (τ). Our analysis reveals several critical scaling dynamics.

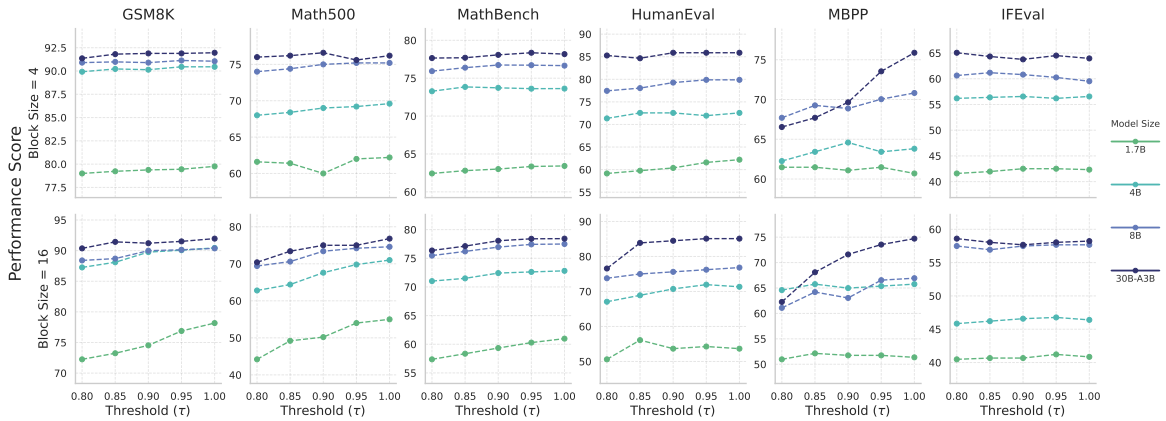

Figure 4: Performance as a function of the dynamic decoding confidence threshold (τ) for block sizes $B = 4$ and $B = 16$.

First, we observe a clear interaction between model scale and robustness to parallelization. Smaller models (1.7B, 4B) exhibit significant sensitivity to increases in block size, with performance degradation becoming pronounced for $B > 4$. In stark contrast, larger models (8B, 30B-A3B) demonstrate remarkable resilience, maintaining stable performance across a much wider range of block sizes. This unveils a **virtuous cycle of scaling**: larger, more capable models tolerate larger block sizes without a commensurate performance drop, directly enabling them to achieve substantially greater parallelization and efficiency. This synergy, where improved capability naturally translates into improved computational throughput, underscores the profound potential of the SDAR framework at scale.

Furthermore, the relationship between performance and block size is not purely monotonic. While small block sizes ($B = 4$) yield performance nearly indistinguishable from their autoregressive counterparts, and larger block sizes ($B = 32$) often introduce task-dependent trade-offs, we identify an interesting inflection point. For intermediate block sizes ($B \in \{8, 16\}$), performance is not only maintained but, on certain complex reasoning tasks such as GSM8k, MathBench, and HumanEval, can even surpass that of the smaller-block configurations. This suggests that a moderately larger generation context can, in some cases, provide a beneficial inductive bias.

The analysis of the confidence threshold τ (Figure 4) reinforces these scaling laws. As expected, performance consistently improves as τ approaches 1.0 (transitioning from dynamic to static decoding). More importantly, larger models display a higher tolerance for more aggressive (i.e., lower τ) decoding strategies. For the 30B-A3B model, performance remains nearly saturated for $\tau \geq 0.90$ across most tasks and block sizes. This dual robustness—to both larger architectural block sizes and more lenient decoding thresholds—is a key finding, indicating that the benefits of scale compound to enable more aggressive parallelization without sacrificing quality. Notably, MBPP shows a stronger sensitivity to τ compared to other benchmarks because it targets base model evaluation, for which our models after SFT are less suited, resulting in larger performance fluctuations as τ changes.

5.2.3 Efficiency Dynamics and the Performance-Speedup Interplay

We now turn to the efficiency axis, quantified by Effective Tokens Per Forward Pass (TPF), which measures the algorithmic speedup over standard autoregressive decoding.

Figure 5 presents a clear and encouraging result: TPF scales predictably and monotonically with the block size B. This relationship holds true across all model scales, tasks, and decoding thresholds, confirming that a larger architectural block size serves as a reliable lever for increasing computational throughput. A second, more profound trend also emerges: **larger models consistently achieve higher**

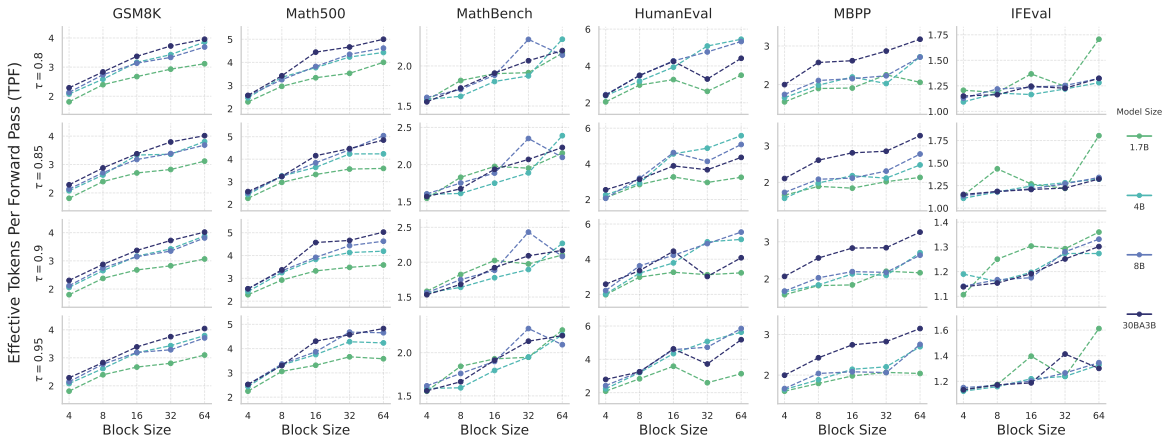

Figure 5: Algorithmic speedup, measured in Effective Tokens Per Forward Pass (TPF), as a function of architectural block size (B) across different model scales and various confidence thresholds (τ).

TPF. This phenomenon is attributable to the lower predictive entropy and higher confidence inherent in more capable models. For a given block, a larger model is more likely to make confident predictions that surpass the threshold τ, thus generating more tokens in parallel. This points to a promising avenue for future optimization: **techniques that reduce a model's predictive entropy, such as knowledge distillation [49] or reinforcement learning-induced "entropy collapse" [12], could directly translate into greater decoding efficiency.**

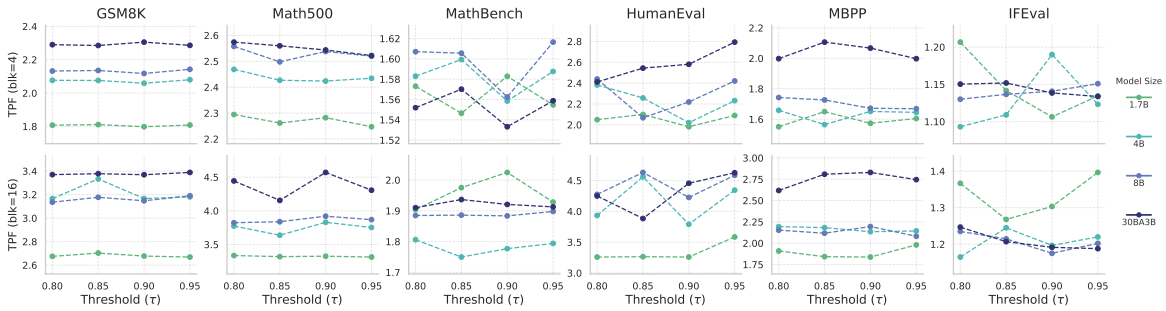

Figure 6: Algorithmic speedup (TPF) as a function of the dynamic decoding confidence threshold (τ) for block sizes $B = 4$ and $B = 16$.

However, the interaction between TPF and the confidence threshold τ, depicted in Figure 6, reveals a more complex dynamic. Contrary to the naive intuition that a lower threshold would always yield higher TPF, our results show a non-monotonic and task-dependent relationship. This is because decoding is a sequential Markov process where the quality of one generated block conditions the next. An overly aggressive (low) threshold may lead to the acceptance of a low-quality, high-entropy block. This error propagation increases the uncertainty for subsequent steps, lowering their predictive confidence and, consequently, reducing the number of tokens generated in parallel.

This exposes our central finding on the trade-off calculus: **in the SDAR framework, performance and efficiency are not independent axes but are deeply intertwined.** Striving for higher generative quality—by using a more conservative threshold or a more capable model—can, paradoxically, lead to improved computational efficiency by maintaining a low-entropy, high-confidence state throughout the decoding process. This reframes the role of model quality entirely. **In the SDAR framework, high**

confidence and low entropy are not outcomes to be balanced against speed, but are themselves the very engine of acceleration.

5.2.4 Deployment-Ready Efficiency Analysis on Industrial Inference Engines

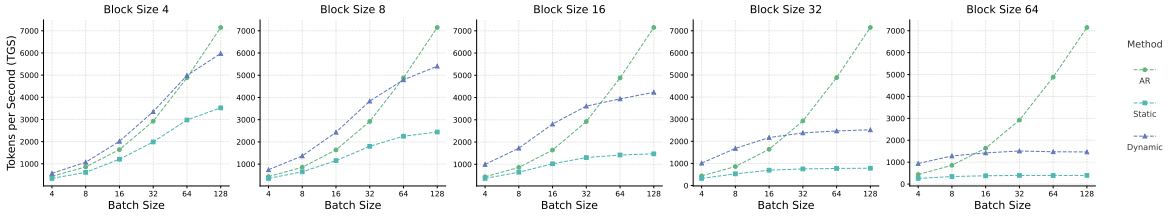

Figure 7: Throughput scaling characteristics (tokens per second) across varying batch sizes for autoregressive (AR) decoding versus SDAR with different block sizes ($B \in \{4, 8, 16, 32, 64\}$). Results shown for SDAR-8B-Chat under both static and dynamic decoding regimes.

While theoretical speedup metrics provide valuable insights into algorithmic efficiency, the translation of these gains into real-world deployment scenarios remains a critical question. We therefore conducted a comprehensive analysis of SDAR's performance on industrial-grade inference infrastructure, addressing two fundamental questions:

1. What absolute throughput can SDAR achieve on production-ready inference engines?
2. How effectively does the theoretical TPF translate to real-world inference speed?

Experimental Setup Our experimental setup employed the LMDeploy inference engine on a single NVIDIA H200 GPU, using SDAR-Chat-8B models with block sizes ranging from 4 to 64. All dynamic decoding experiments utilized a confidence threshold of $\tau = 0.9$, which our earlier analysis showed preserves quality parity with standard autoregressive generation. We measured throughput across batch sizes from 4 to 512 on the Math-500 benchmark.

Analysis Framework To formalize our analysis, we define: t_f (time per forward pass), α (batch size), β (block size), and γ (average forward passes per block). For standard autoregressive decoding:

$$\mathrm{TGS}_{\mathrm{AR}} = \frac{\alpha}{t_f} \tag{9}$$

$$t_f = \frac{\alpha}{\mathrm{TGS}_{\mathrm{AR}}} \tag{10}$$

With our inference engine requiring an additional forward pass for KV cache computation, static SDAR decoding throughput becomes:

$$\mathrm{TGS}_{\mathrm{static}} = \frac{\alpha\beta}{(\beta+1)t_f} \tag{11}$$

$$t_f = \frac{\alpha\beta}{(\beta+1)\mathrm{TGS}_{\mathrm{static}}} \tag{12}$$

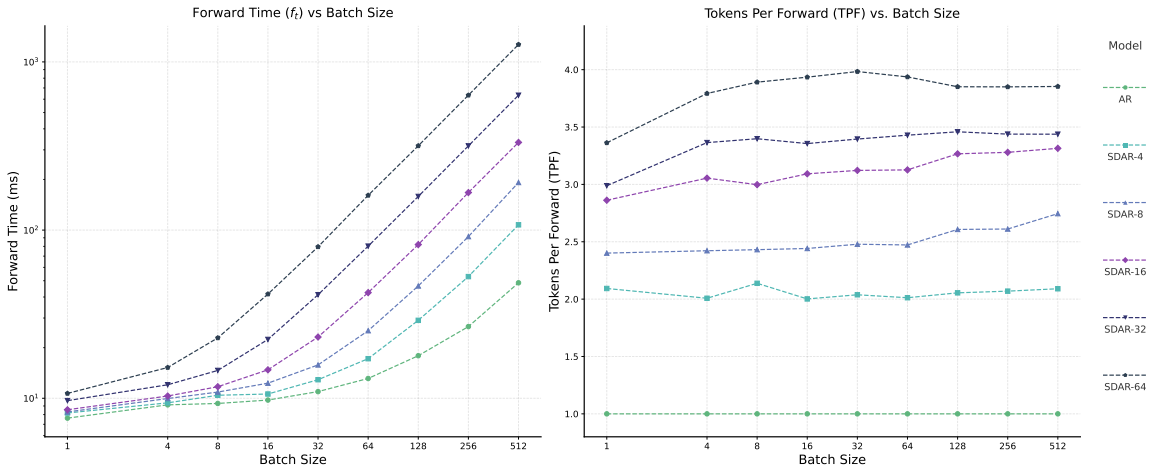

Figure 8: **Left:** Single forward pass time (t_f) as a function of batch size for AR and SDAR models with varying block sizes. The linear relationship at higher batch sizes indicates compute-bound operation. **Right:** Realized Tokens Per Forward Pass (TPF) calculated from actual inference engine throughput, demonstrating the practical manifestation of theoretical speedup across different block sizes.

For dynamic decoding with confidence-based forward passes:

$$\text{TGS}_{\text{dynamic}} = \frac{\alpha\beta}{(\gamma+1)t_f} \tag{13}$$

$$t_f = \frac{\alpha\beta}{(\gamma+1)\text{TGS}_{\text{dynamic}}} \tag{14}$$

$$\text{TPF} = \frac{\beta}{\gamma} = \frac{1}{\frac{\alpha}{\text{TGS}_{\text{dynamic}} \times t_f} - \frac{1}{\beta}} \tag{15}$$

Analysis Figure 7 reveals several critical insights about SDAR's real-world performance characteristics. First, our SDAR models achieve impressive peak throughput under dynamic decoding, with maximum tokens generated per second (TGS) of 6600, 5510, 4236, 2520, and 1509 for block sizes 4, 8, 16, 32, and 64, respectively. Second, we observe a clear batch size dependency: SDAR models significantly outperform autoregressive decoding at smaller batch sizes, but this advantage diminishes as batch size increases. This phenomenon stems from SDAR's higher computational intensity per forward pass, which causes it to reach compute-bound operation earlier than standard autoregressive models.

This batch size sensitivity exhibits a clear relationship with block size—larger blocks provide greater acceleration advantages in low-batch regimes but encounter compute limitations more rapidly as batch size increases. This finding has profound implications for deployment strategies: **SDAR provides the greatest relative advantage in latency-sensitive, low-batch inference scenarios**, precisely the operating conditions most relevant for real-time applications and interactive services.

Examining Figure 8, we gain further insight into the underlying mechanics of this performance profile. The left panel demonstrates that forward pass time increases gradually with batch size until reaching a compute-bound threshold, after which the relationship becomes strictly linear with consistent slope across models. This inflection point represents the maximum effective parallelism the hardware can support for each model configuration.

The right panel of Figure 8 presents the realized TPF calculated using Equation 13, which closely aligns with our theoretical measurements in Figure 5. This correspondence validates that the algorithmic

speedup of SDAR successfully translates to practical deployment environments, though the absolute throughput advantage depends on the specific operating regime.

These findings suggest several optimization pathways for future inference engine implementations. First, eliminating the separate KV cache computation by integrating it with subsequent decoding steps could further improve throughput. Second, specialized kernel optimizations for the unique computation patterns of SDAR could extend the batch size range where SDAR maintains its advantage. Finally, the clear relationship between block size, batch size, and throughput enables deployment-time configuration optimization based on specific workload characteristics and latency requirements.

6 Reasoning potential of SDAR

Takeaway

(1) **Reasoning preserved.** SDAR adaptation retains and easily restores long CoT reasoning from strong AR reasoning models.
(2) **Domain shift for free.** SDAR enables effortless domain transfer during adaptation, excelling on tasks benefiting from local bidirectional context.
(3) **Scaling synergy.** Simple test-time scaling brings large gains, suggesting strong potential for further improvement via RL.

Over just the past year, the reasoning capabilities of large language models have emerged as a central focus of the field. Yet these advances typically come with a steep cost: test-time scaling methods place a heavy burden on inference. A core motivation of our study is the hypothesis that SDAR's parallel decoding capability can mitigate this challenge, offering a more efficient pathway to test-time scaling. To examine this potential, we adapt strong autoregressive (AR) baselines into the SDAR paradigm and evaluate whether the resulting models can preserve, and in some cases enhance, their reasoning ability.

To ensure fairness, we focus on the scientific domain, where benchmarks are less vulnerable to inflated results previously reported by base AR models. This allows us to better isolate the impact of the modeling paradigm itself. Our findings confirm that SDAR not only maintains the reasoning ability inherited from its AR counterpart but also demonstrates notable gains under test-time scaling strategies such as pass@k and majority voting. These improvements point to an important implication: reinforcement learning—a natural way to optimize reasoning directly—may unlock even greater benefits for SDAR models in the future. Furthermore, by incorporating local bidirectional attention, SDAR relaxes the strictly causal inductive bias of AR models, raising the question of whether this shift harms or benefits reasoning. Our experiments suggest the latter, highlighting SDAR's promise for complex scientific reasoning and beyond.

6.1 Experimental Setup

Training Setup Our methodology is designed to build a powerful science-reasoning model, SDAR-30B-A3B-Sci, and to rigorously evaluate its capabilities against a comparable autoregressive baseline. Both models originate from the Qwen3-30B-A3B checkpoint.

The development of our proposed SDAR-30B-A3B-Sci model follows a three-stage pipeline:

1. **Extensive Domain Pre-training (AR Objective):** To instill a deep foundation in scientific knowledge, we first continually pre-train the Qwen3 model using a standard autoregressive objective. This stage leverages a massive 1-trillion-token scientific corpus, which includes a 500B-token general science dataset followed by a 500B-token annealing dataset. This process yields a highly capable, science-aware intermediate AR checkpoint.

2. **Paradigm Conversion to SDAR:** Starting from this science-aware AR checkpoint, we perform a paradigm shift. We continue training the model using the blockwise diffusion objective on a curated 50B-token subset sampled from the annealing corpus. This step effectively converts the model's generative mechanism from autoregressive to SDAR, resulting in our base model, SDAR-30B-A3B-Sci-Base.
3. **Supervised Fine-tuning for Reasoning:** Finally, we fine-tune the SDAR-30B-A3B-Sci-Base on a collection of 0.5B high-quality, long CoT instruction datasets to produce the final reasoning model, SDAR-30B-A3B-Sci.

To ensure a fair and controlled comparison, we constructed the AR-30B-A3B-Sci baseline with a meticulously mirrored training process. It starts from the *exact same* 1T-token science-aware intermediate checkpoint. Then, to isolate the effect of the modeling objective, it is further trained on the *identical* 50B-token data subset using the standard autoregressive objective. This creates the AR-30B-A3B-Sci-Base. Subsequently, it undergoes the same SFT procedure on the identical instruction datasets. This rigorous parallel construction ensures that any observed performance differences are directly attributable to the core modeling paradigm (SDAR vs. AR).

Evaluation Setup We conduct a rigorous evaluation of our science-oriented models on a curated suite of frontier benchmarks designed to assess expert-level capabilities in complex reasoning, mathematics, specialized scientific domains, and advanced code generation. The evaluation datasets are organized as follows:

- **Expert Reasoning & Knowledge:** MMLU-Pro [54] and GPQA-diamond [45].
- **Competition-Level Mathematics:** AIME-2024, AIME-2025 [2], and LiveMathBench-hard (LMB-hard) [34].
- **Specialized Scientific Domains:** Chembench [39], PHYSICS [15], and ProteinLMBench [47].
- **Advanced Code Generation:** LiveCodeBench-v5 and LiveCodeBench-v6 (LCB) [25].

For our evaluation protocol, we define distinct decoding strategies for each model architecture to ensure a fair and comprehensive comparison. For our **SDAR-Sci** models, we use a fixed `block_length=4` and static decoding strategy. We report results under two sample strategies: **(G)** greedy decoding and **(S)** sampling-based decoding (with `temperature=1.0`, `top_p=1.0`, and `top_k=0`). To provide a strong baseline, the **AR-Sci** model is evaluated using a recommended sampling-based approach with `temperature=0.6`, `top_p=0.95`, and `top_k=20`. For all sampling-based evaluations, including AR-Sci and SDAR-Sci (S), we report the mean performance over multiple runs on key benchmarks: 8 runs for GPQA-diamond, and 32 runs for AIME-2024, AIME-2025, and LiveMathBench-hard.

Test-Time Scaling Setup To rigorously assess and augment the reasoning capabilities of our model on complex benchmarks, we employ inference-time strategies. Our standard evaluation protocol across all tasks utilizes Chain-of-Thought (CoT) prompting, compelling the model to externalize its deductive process. Specifically, the model generates its step-by-step reasoning within '<think>' tags before providing the final answer (e.g., '<think>... a detailed chain of thought ...</think>answer').

For a select subset of reasoning benchmarks, we augment this CoT approach with test-time scaling. We generate multiple diverse reasoning paths ($N = 8$ for GPQA-diamond and $N = 32$ for AIME-2024, AIME-2025, and LiveMathBench-hard) and report results using two evaluation schemes. The first is **majority vote**, where the most frequent final answer among the N samples is chosen. The second is the **pass@k** metric (where $k = N$), which credits a success if at least one of the k generated solutions is correct. It is crucial to note that, unless explicitly stated otherwise, all results presented in Section 6.2 are derived from the standard CoT protocol alone.

6.2 Summary of Evaluation Results

Our comprehensive evaluation shows that the SDAR paradigm aligns well with complex scientific and mathematical reasoning tasks. The clearest evidence comes from the controlled comparison against its autoregressive counterpart, AR-30B-A3B-Sci. Both models inherit reasoning ability from the Qwen3 AR base model and re-express it through supervised fine-tuning (SFT)—a less optimal path compared to reinforcement learning (RL), which can instill reasoning ability more directly. Nevertheless, as summarized in Tables 4–6, SDAR achieves comparable overall performance to AR, with a slight edge in scientific reasoning tasks such as chemistry.

SDAR Paradigm Strengthens Reasoning Capabilities On general reasoning and mathematics (Table 4), SDAR performs on par with the AR baseline for MMLU-Pro, AIME-2024, and AIME-2025, but stands out with a strong gain on LiveMathBench-Hard (+5.3%). In advanced code generation, results are mixed: SDAR achieves a notable improvement on LCB-v6 (+5.1%) while trailing on LCB-v5, yielding comparable performance on average. These findings suggest that in broad reasoning tasks, SDAR provides stability with selective advantages, signaling untapped potential when paired with more direct optimization methods such as RL.

Table 4: Strict comparison between AR-30B-A3B-Sci and SDAR-30B-A3B-Sci on non-science benchmarks. (G = greedy decoding, S = sampling). Bold numbers indicate improvements over the AR baseline, with Δ shown in a smaller font (green = improvement, red = drop).

Model	MMLU-pro	AIME2024	AIME2025	LMB-hard	LCB-v5	LCB-v6
AR-30B-A3B-Sci	78.3	74.9	60.7	55.4	51.5	46.3
SDAR-30B-A3B-Sci (G)	78.1 (-0.2)	76.7 (+1.8)	**60.0** (-0.7)	**60.7** (+5.3)	40.7 (-10.8)	42.3 (-4.0)
SDAR-30B-A3B-Sci (S)	77.9 (-0.4)	**73.4** (-1.5)	59.2 (-1.5)	**58.7** (+3.3)	49.1 (-2.4)	**51.4** (+5.1)

Table 5: Strict comparison between AR-30B-A3B-Sci and SDAR-30B-A3B-Sci on science benchmarks.

Model	**GPQA-diamond**	**ChemBench**	**PHYSICS**	**ProteinLMBench**
AR-30B-A3B-Sci	61.2	60.5	39.0	59.5
SDAR-30B-A3B-Sci (G)	**66.7** (+5.5)	**72.3** (+11.8)	**37.9** (-1.1)	**59.9** (+0.4)
SDAR-30B-A3B-Sci (S)	66.0 (+4.8)	**72.8** (+12.3)	**38.2** (-0.8)	**59.6** (+0.1)

Achieving State-of-the-Art Performance in Scientific Domains The science-focused benchmarks (Table 5) reveal SDAR's most dramatic advantages. It matches AR performance on PHYSICS and ProteinLMBench, but achieves substantial leaps on GPQA-diamond (+5.5%) and ChemBench (+12.3%). These results demonstrate that when downstream tasks demand reasoning over structured and associative knowledge—such as chemical formulas—SDAR's local bidirectional attention mechanism provides a decisive edge (see Appendix B). Thus, while SDAR proves competitive on general reasoning, it distinctly excels in scientific domains, establishing itself as a powerful alternative to AR for complex, domain-specific applications.

Beyond this AR comparison, SDAR-30B-A3B-Sci also proves competitive against much larger state-of-the-art open-source and proprietary models (Table 6). Despite its relatively modest 30B parameter count, its performance in several scientific domains approaches or even rivals that of models many times its size.

Enhancing Performance with Test-Time Scaling The application of test-time scaling techniques further amplifies SDAR's inherent reasoning advantages, revealing its exceptional potential when

Table 6: Performance comparison of our models (AR-30B-A3B-Sci, SDAR-30B-A3B-Sci), open-source foundation models, and closed-source commercial models.

Category	Model	MMLU-pro	AIME2025	GPQA-diamond	ChemBench	PHYSICS	ProteinLMBench
Ours	AR-30B-A3B-Sci	78.3	60.7	61.2	60.5	39.0	59.5
	SDAR-30B-A3B-Sci(S)	77.9 (-0.4)	59.2 (-1.5)	**66.7** (+5.5)	**72.8**(+12.3)	38.2 (-0.8)	**59.6** (+0.1)
Open-source	InternVL3-78B	73.0	10.7	49.9	61.3	23.1	61.6
	Qwen2.5-VL-72B	72.1	10.9	49.0	61.6	15.7	61.0
	DS-R1-0528	83.4	87.5	80.6	75.6	–	61.4
	Qwen3-235B-A22B	82.2	81.5	71.1	75.8	–	59.8
Closed-source	Gemini-2.5 Pro	86.0	83.0	83.8	82.8	40.0	62.9
	o3	85.0	88.9	83.3	81.6	47.9	67.7
	Grok-4	85.9	91.7	87.5	83.3	42.8	66.2

Table 7: Strict comparison between AR-30B-A3B-Sci and SDAR-30B-A3B-Sci. (G = greedy decoding, S = sampling). Bold numbers indicate best performance in each column among the main models.

Model	GPQA-diamond	AIME2024	AIME2025	LMB-hard
AR-30B-A3B-Sci	61.2	74.9	60.7	55.4
SDAR-30B-A3B-Sci (G)	66.7 (+5.9)	76.7 (+1.8)	60.0 (-0.7)	60.7 (+5.3)
SDAR-30B-A3B-Sci (S)	66.0 (+4.8)	73.4 (-1.5)	59.2 (-1.5)	58.7 (+3.3)
w/ Majority	68.2 (+7.0)	86.7(+11.8)	80.0(+19.3)	71.1(+15.7)
w/ pass@k	84.3(+23.1)	93.3(+18.4)	86.7(+26.0)	87.5(+32.1)

combined with advanced inference strategies. As demonstrated in Table 7, our SDAR model exhibits remarkable performance gains when augmented with ensemble methods. The majority voting approach yields substantial improvements across all benchmarks, with particularly striking gains on mathematical reasoning tasks: **+11.8%** on AIME-2024, **+19.3%** on AIME-2025, and **+15.7%** on LiveMathBench-hard compared to the AR baseline. Even more impressive are the results achieved with the pass@k methodology, which pushes performance to unprecedented levels—reaching **84.3%** on GPQA-diamond (**+23.1%** over baseline), **93.3%** on AIME-2024 (**+18.4%**), **86.7%** on AIME-2025 (**+26.0%**), and **87.5%** on LiveMathBench-hard (**+32.1%**). These improvements highlight an important observation: SDAR's parallel generation paradigm tends to produce more diverse and complementary reasoning trajectories when sampling multiple candidates, suggesting stronger synergy with test-time scaling strategies such as majority voting or pass@k evaluation. This interplay between SDAR's architectural design and test-time scaling methods points to a promising new direction for enhancing machine reasoning efficiency—indicating that the combined effect of these techniques can extend beyond what either alone achieves in isolation.

7 Related Work

Our work, SDAR, resides at the intersection of autoregressive and diffusion-based language modeling, aiming to synthesize their respective strengths while mitigating their inherent weaknesses. We structure our review by first examining these two dominant paradigms, then analyzing prior attempts to unify them, and finally contrasting our approach with orthogonal methods for inference acceleration.

7.1 The Autoregressive Paradigm: Dominance and Inherent Constraints

Autoregressive (AR) models, which factorize the joint probability of a sequence into a left-to-right product of conditionals, represent the de facto standard in large-scale language modeling [44, 1, 18, 55].

The success of this paradigm is rooted in its strict causal inductive bias, which aligns naturally with the sequential structure of human language [7, 52, 43]. This alignment, combined with the stable and efficient cross-entropy training objective, has enabled unprecedented scaling and performance on a wide array of general-purpose tasks [27, 9].

However, the very causal bias that underpins the AR model's success becomes its **Achilles' heel** for tasks demanding non-local or holistic reasoning. This fundamental mismatch has been shown to impede performance on problems where solutions depend on global constraints, such as mathematical puzzles (e.g., Sudoku, 24-point game), Boolean satisfiability, and long-term planning [24, 6, 56]. The model's rigid, token-by-token generation process struggles to reason about future dependencies or revise earlier decisions in light of global information. For instance, recent studies demonstrate that masked generative diffusion models can dramatically outperform strong AR baselines on combinatorial problems like Sudoku (e.g., 91.5% vs. 45.8% accuracy [56]), underscoring a clear paradigm-level advantage.

This limitation also manifests in scientific domains. Modeling chemical formulas (e.g., SMILES), for instance, requires bidirectional context. Similarly, modeling protein sequences to resolve structural motifs and functional domains is inherently at odds with the unidirectional autoregressive (AR) process, a foundational challenge for generative models like the ProGen series [37, 42, 8]. This has motivated a shift towards architectures that can leverage global, bidirectional context. Examples include powerful representation learners like the ESM series [38, 23, 32] and non-autoregressive generators such as discrete diffusion models dplm2 [53]. As argued by Liu et al. [35], the standard sequential paradigm is fundamentally suboptimal for biological sequences due to their critical long-range dependencies. Our own experiments affirm this hypothesis: by locally relaxing this causal constraint, SDAR achieves a 72.8% accuracy on the challenging Chembench benchmark [39], significantly outperforming its 60.5% pure AR counterpart.

7.2 Diffusion Models: Holistic Modeling at a Prohibitive Cost

As an alternative, discrete diffusion models circumvent the strict causal constraints of AR models by treating sequence generation as a holistic denoising process, which learns to reverse a gradual corruption of the data [50, 21, 19, 48, 36]. This architectural freedom offers two profound advantages: (1) a natural capacity for parallel decoding, directly addressing the latency bottleneck of AR inference, and (2) a flexible, bidirectional inductive bias that is better suited for the aforementioned non-local reasoning tasks.

Despite this promise, the practical application of diffusion language models has been severely hampered by their prohibitive training cost [40, 19]. This inefficiency stems from two primary sources. First, the training objectives, such as the Evidence Lower Bound (ELBO), are often more difficult to optimize and converge slower than the standard cross-entropy loss used in AR models [40, 3]. Second, the learning task of reconstructing a fully structured sequence from varying levels of noise is information-theoretically more challenging than next-token prediction. Empirical studies quantify this gap, showing that masked discrete diffusion models can require up to 16$\times$ more FLOPs to match the validation NLL of an AR equivalent [40], a disparity that widens to as much as 64$\times$ for continuous diffusion models [19].

7.3 Hybrid and Conversion Models: Bridging the Gap

Recognizing the complementary nature of AR and diffusion models, several lines of research have explored hybrid architectures.

Block-wise Hybrid Models. A prominent approach fuses the paradigms by employing an autoregressive structure at a global, inter-block level while using a parallel, diffusion-based mechanism for

intra-block generation [3, 21, 13]. This strategy elegantly preserves macroscopic causal flow, enabling variable-length generation and KV-caching. However, these models have traditionally adopted a monolithic training regime, where the AR and diffusion components are trained jointly from scratch. This subjects them to the full training inefficiency of the diffusion objective, compounded by the complexity of a hybrid loss function, presenting a significant barrier to scaling.

AR-to-Diffusion Conversion Models. Another strategy attempts to leverage the efficiency of AR pre-training by first training a standard LLM and then adapting it to a non-autoregressive or diffusion-based objective. Works like DiffuLLaMA [16] and Dream 7B [57] follow this path. While conceptually appealing, this conversion has proven to be computationally expensive (e.g., requiring 580B tokens for adaptation in Dream 7B) and can result in notable performance degradation compared to the original AR model. In stark contrast, SDAR's **decoupled training and inference paradigm** and block-level adaptation requires a minimal fine-tuning budget (e.g., 50B tokens) to unlock parallel decoding, while fully preserving, and on specialized tasks enhancing, the performance of the foundational AR model.

8 Conclusion

This work presents SDAR, a Synergistic Diffusion–AutoRegression paradigm that redefines the design space of large language models by decoupling training efficiency from inference parallelism. Through a unified framework integrating autoregressive pretraining with blockwise diffusion inference, SDAR provides a scalable and practical pathway beyond the limitations of purely autoregressive modeling.

Our analyses confirm that autoregressive training remains the most compute-efficient formulation for large language model. Building upon this foundation, SDAR performs a lightweight adaptation that converts a well-trained autoregressive model into a blockwise diffusion model, thereby preserving AR-level performance while introducing parallel intra-block decoding. This adaptation retains critical AR functionalities—such as variable-length generation and KV caching—while alleviating the sequential bottleneck and restrictive causal inductive bias of token-by-token decoding.

Through extensive controlled experiments, we demonstrate that SDAR models maintain the performance of their autoregressive counterparts under identical computational budgets, validating the feasibility and robustness of the adaptation process. Scaling experiments across both dense and MoE architectures further show that the SDAR paradigm scales without compromise: as model capacity increases, SDAR models preserve accuracy and exhibit increasingly higher parallel decoding efficiency. Larger models tolerate greater block sizes and more aggressive decoding thresholds, revealing a virtuous cycle of scaling in which enhanced model capability directly translates into improved computational throughput.

Beyond efficiency, SDAR demonstrates strong reasoning and domain-specific capabilities, particularly in scientific and mathematical contexts that benefit from localized bidirectional context. Our findings show that SDAR not only preserves the reasoning strength of its AR foundation but also achieves substantial gains under test-time scaling strategies such as majority voting and pass@k, highlighting its compatibility with reinforcement learning and other post-training optimization techniques.

In summary, SDAR is not merely a hybrid architecture, but a new language modeling paradigm that unifies the strengths of autoregression and diffusion while mitigating their respective limitations. It retains the optimization efficiency and controllability of AR models, inherits the holistic representation capacity of diffusion, and achieves scalable, parallelizable generation without loss of quality. We believe SDAR establishes a principled and extensible foundation for the next generation of language models—one that moves beyond the autoregressive frontier toward a broader spectrum of efficient and synergistic modeling paradigms.

9 Acknowledgements

We thank Shen Nie for insightful discussions on the training and evaluation details of LLaDA, and for suggesting that the training transition could proceed without logits shift or attention mask annealing. We thank Yu Zhang for valuable guidance in optimizing the pretraining pipeline and training framework, as well as for providing infrastructure support in the early stages of the project.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] AIME. AIME problems and solutions, 2025. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

[3] 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. *arXiv preprint arXiv:2503.09573*, 2025.

[4] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

[5] 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.

[6] Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In *International Conference on Machine Learning*, pages 2296–2318. PMLR, 2024.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[8] Aadyot Bhatnagar, Sarthak Jain, Joel Beazer, Samuel C Curran, Alexander M Hoffnagle, Kyle Ching, Michael Martyn, Stephen Nayfach, Jeffrey A Ruffolo, and Ali Madani. Scaling unlocks broader generation and deeper functional understanding of proteins. *bioRxiv*, pages 2025–04, 2025.

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

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[12] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

[13] Nima Fathi, Torsten Scholak, and Pierre-André Noël. Unifying autoregressive and diffusion-based sequence generation. *arXiv preprint arXiv:2504.06416*, 2025.

[14] 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.

[15] Kaiyue Feng, Yilun Zhao, Yixin Liu, Tianyu Yang, Chen Zhao, John Sous, and Arman Cohan. Physics: Benchmarking foundation models on university-level physics problem solving. *arXiv preprint arXiv:2503.21821*, 2025.

[16] Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. In *The Thirteenth International Conference on Learning Representations*.

[17] Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

[18] 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.

[19] Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.

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

[21] Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. *arXiv preprint arXiv:2210.17432*, 2022.

[22] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[23] Chloe Hsu, Robert Verkuil, Jason Liu, Zeming Lin, Brian Hie, Tom Sercu, Adam Lerer, and Alexander Rives. Learning inverse folding from millions of predicted structures. *ICML*, 2022. doi: 10.1101/2022.04.10.487779. URL https://www.biorxiv.org/content/early/2022/04/10/2022.04.10.487779.

[24] Edward J Hu, Moksh Jain, Eric Elmoznino, Younesse Kaddar, Guillaume Lajoie, Yoshua Bengio, and Nikolay Malkin. Amortizing intractable inference in large language models. *arXiv preprint arXiv:2310.04363*, 2023.

[25] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint*, 2024.

[26] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

[27] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[28] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[29] Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese, 2023.

[30] Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*, 2025.

[31] 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.

[32] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Sal Candido, et al. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv*, 2022.

[33] Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 6884–6915, 2024.

[34] Junnan Liu, Hongwei Liu, Linchen Xiao, Ziyi Wang, Kuikun Liu, Songyang Gao, Wenwei Zhang, Songyang Zhang, and Kai Chen. Are your llms capable of stable reasoning? *arXiv preprint arXiv:2412.13147*, 2024.

[35] Ke Liu, Shuaike Shen, and Hao Chen. From sentences to sequences: Rethinking languages in biological system. *arXiv preprint arXiv:2507.00953*, 2025.

[36] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32819–32848, 2024.

[37] Ali Madani, Ben Krause, Eric R Greene, Subu Subramanian, Benjamin P Mohr, James M Holton, Jose Luis Olmos Jr, Caiming Xiong, Zachary Z Sun, Richard Socher, et al. Large language models generate functional protein sequences across diverse families. *Nature biotechnology*, 41(8):1099–1106, 2023.

[38] Joshua Meier, Roshan Rao, Robert Verkuil, Jason Liu, Tom Sercu, and Alexander Rives. Language models enable zero-shot prediction of the effects of mutations on protein function. *bioRxiv*, 2021. doi: 10.1101/2021.07.09.450648. URL https://www.biorxiv.org/content/10.1101/2021.07.09.450648v1.

[39] Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, Amir Mohammad Elahi, Mehrdad Asgari, Juliane Eberhardt, Hani M. Elbeheiry, María Victoria Gil, Maximilian Greiner, Caroline T. Holick, Christina Glaubitz, Tim Hoffmann, Abdelrahman Ibrahim, Lea C. Klepsch, Yannik Köster, Fabian Alexander Kreth, Jakob Meyer, Santiago Miret, Jan Matthias Peschel, Michael Ringleb, Nicole Roesner, Johanna Schreiber, Ulrich S. Schubert, Leanne M. Stafast, Dinga Wonanke, Michael Pieler, Philippe Schwaller, and Kevin Maik Jablonka. Are large language models superhuman chemists? *arXiv preprint arXiv: 2404.01475*, 2024.

[40] Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

[41] 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.

[42] Erik Nijkamp, Jeffrey A Ruffolo, Eli N Weinstein, Nikhil Naik, and Ali Madani. Progen2: exploring the boundaries of protein language models. *Cell systems*, 14(11):968–978, 2023.

[43] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[44] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[45] 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.

[46] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[47] Yiqing Shen, Zan Chen, Michail Mamalakis, Luhan He, Haiyang Xia, Tianbin Li, Yanzhou Su, Junjun He, and Yu Guang Wang. A fine-tuning dataset and benchmark for large language models for protein understanding. In *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2390–2395. IEEE, 2024.

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

[49] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

[50] Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, et al. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*, 2022.

[51] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

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

[53] Xinyou Wang, Zaixiang Zheng, Fei Ye, Dongyu Xue, Shujian Huang, and Quanquan Gu. Dplm-2: A multimodal diffusion protein language model. *arXiv preprint arXiv:2410.13782*, 2024.

[54] 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. *arXiv preprint arXiv:2406.01574*, 2024.

[55] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[56] Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint arXiv:2410.14157*, 2024.

[57] 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.

[58] Runpeng Yu, Qi Li, and Xinchao Wang. Discrete diffusion in large language and multimodal models: A survey. *arXiv preprint arXiv:2506.13759*, 2025.

[59] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5673–5684, 2023.

[60] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Appendix

A Infrastructure

A.1 Training Infrastructure

The training of the SDAR model requires specialized attention masks and training objectives that differ from standard autoregressive language models. As a result, widely adopted kernels such as FlashAttention for attention computation and fused cross-entropy kernels from projects like Liger-Kernel cannot be directly applied. To address this, we adopt FlexAttention[1], which achieves significant speedups compared to the scaled dot-product attention provided in PyTorch.

For the loss computation, we implement a custom Triton kernel that performs a fused operation to calculate the training loss. This design avoids fully materializing the logits tensor—a highly memory-intensive step—thereby reducing GPU memory usage and improving overall efficiency.

While the full-scale, industry-grade training framework we use is not entirely open source, we plan to release a research-oriented open-source implementation based on existing widely used frameworks. In terms of efficiency, the raw throughput, measured in tokens processed per GPU per second, is comparable to conventional autoregressive model training. However, due to SDAR's training paradigm,which requires processing both clean and noised versions of each sample, the effective training speed is approximately halved relative to standard autoregressive models.

A.2 Inference Infrastructure

Unlike many diffusion-based models that rely on approximate or specialized cache mechanisms (e.g., dLLM-cache), the SDAR model maintains a mathematically identical key-value (KV) cache to that of standard autoregressive models. This compatibility allows us to leverage established cache management techniques, including PagedAttention as implemented in vLLM.

We design a custom attention kernel to handle the specialized attention mask required during the prefill stage. For the decoding stage, we integrate PagedAttention [28], ensuring inference efficiency that is competitive with state-of-the-art autoregressive inference engines. Our implementation extends autoregressive infrastructure with block-oriented optimizations, employing block-aligned memory allocation that provisions $(n/n+1)$ blocks to support n forward passes. The inference pipeline operates through iterative resource allocation, batched block-level inference, and state updates. Key optimizations include unified treatment of denoising and cache-filling operations during forward passes, fully batched unmasking operations to avoid device synchronization, and deferred stop condition evaluation to handle mid-block termination sequences. Beyond efficiency, the JetEngine[2] is intentionally developed to be lightweight and modular, making it well-suited for future research directions such as integrating novel sampling strategies and reinforcement learning–based algorithms.

We thank the lmdeploy team for integrating SDAR into their industrial-grade inference engine,[3] enabling prefix-based KV-cache sharing across requests and thereby lowering computational and memory overheads for interactive generation.

[1] https://pytorch.org/blog/flexattention/
[2] https://github.com/Labman42/JetEngine
[3] https://github.com/InternLM/lmdeploy

B Example Outputs

SDAR-30B-A3B-Sci Example Output for chemistry problem

User:
There is a single choice question about chemistry. Answer the question by replying A, B, C or D.
Question: Generate a molecule based on this description.
The molecule is an oligosaccharide.
A. I believe the answer is c1([nH]c2c(c3c1c(c1c(c4c(cc(C)cc4)nc1c3OC)OC)O)cccc2)=O
B. I believe the answer is CC(=O)OC1C23C(C(OC(C)=O)C(OC(=O)c4ccccc4)CC2C)
(COC(=O)c2ccccc2)C(OC(c2ccccc2)=O)C(OC(C)=O)C1C(O3)(C)C
C. As for my answer, it is C(CCCCC(OCC(OC(=O)CCC=CCC=CCC=CCC=CCC=CCC=CCC)
COP([O-])([O-])=O)=O)CCC=CCCCCCCCC
D. In my opinion, the answer would be
C1(CC(OC2C(C(OC(C2)OC2CC(OC34C(c5c(O)c6c(O)c(C)c(O)cc6cc5CC4C
(C(=O)C(C(C)=O)=C3O)OC)=O)OC(C2O)C)C)O)OC(C)C1O)(C)O
Answer:

Assistant:
<think>
Okay, so I need to figure out which of these four options is an oligosaccharide. Let me start by recalling what an oligosaccharide is. Oligosaccharides are carbohydrates composed of three to ten monosaccharide units linked together by glycosidic bonds. Monosaccharides are simple sugars like glucose, fructose, galose, etc. So the molecule should be a short chain of these sugar units.

Looking at the options given: A, B, C, D. Each seems like a complex string of molecular symbols. Let's parse each one.

Option A is a complicated structure with multiple rings. The formula starts with c1([h2]c2...). It looks like there's a benzene ring involved, maybe some heterocyclic rings. Hmm, there's nitrogen here. Wait, adenine or something? No, in the context of a sugar, maybe not. Wait, the description says it's an oligosaccharide. If this molecule has a nitrogen-containing sugar, like N-acetylglucosamine, that could be part of a glycosaminoglycan, but is this option an oligosaccharide? Not sure yet. Let me check the other options first.

Option B has a lot of acetyl groups (O2CH3), benzoyl groups (OC(=O)C6H5). It seems like a complex ring structure with many substituents. There's a lot of rings and a phosphate group (O-P([-)(O)=O) in the structure. Wait, phosphate might indicate part of a nucleotide or something else, not an oligosaccharide. Maybe, but not sure. The presence of a phosphate would make it a phosphoryated sugar derivative, which could be part of a larger molecule but not alone oligosaccharide.

Option C is a very long chain with multiple double bonds, probably a fatty acid or a glycerophospholipid. Wait, the formula starts with C(C...C(OCC(...COP([O-])([O-]))=O). There's a phosphate group again, and a long hydrocarbon tail. This looks more like a phospholipid, which is part of cell membranes. Definitely not an oligosaccharide.

Option D: a molecule with multiple sugar rings. Let's parse it. The structure starts with

C1(C(C)OC2...). There's a lot of oxygen rings, possibly pyranose rings (six-membered sugar rings). There are a lot of oxygen atoms connected to carbon rings, which is typical of sugars. The molecule has multiple sugar units connected by oxygen bridges. For example, the presence of -OC(C2O)C... suggests glycosidic linkages. Also, there are several O-C attached to other sugar rings. Let me count: oligosaccharides have multiple monosaccharide units. If this molecule has two such rings connected via a glycosidic bond, then it would be an oligosaccharide. The structure in option D seems complex and has multiple sugar rings connected by oxygen atoms. The presence of multiple oxygen bridges and six-membered rings (like pyranose) and five-membered rings (furanose) would be consistent with an oligosaccharide.

Wait, but let's check each option again. Option D's structure starts with a central carbon (C1) with several substituents, including a ring (OC2). The second ring (OC2) is also connected to other rings (OC34C...). The "OC34" suggests a bridge, maybe a disaccharide or more. The structure seems to have sugar units connected via glycosidic bonds. For instance, the presence of "OC(C2O)C" might indicate a connection between two sugar molecules. Also, the multiple hydrox groups (-OH) are common in sugars. The structure in D seems to have several monosaccharide units linked together.

Option A's structure looks like a complex polycyclic aromatic system with nitrogen. Maybe a purinergic alkaloid, but not a sugar. Option B's structure has a lot of acetyl and benzoyl groups, which are typical in sugar derivatives (like N-acetylneurosamine?), but the structure seems to be a complex ring with more than one sugar. Wait, option B might be a glycoside, but how many sugars? Let me think: the structure in B has a hexose ring with multiple ester groups (O2(=O) and benzoyl groups. But is that a single sugar or multiple oligosaccharide? If one sugar with substituents, then it's a monosaccharide derivative, not an oligosaccharide. Option D has more evidence of multiple linked sugar units.

Option D is a bit hard to parse, but let me look again. The formula starts with "C1(CC(OC2C(C(O(C2O)C...))..." which seems like a first sugar ring (C1), connected via an oxygen to another ring (OC2). That's a glycosidic linkage. If there are two rings connected like this, then it's a disaccharide. If there are more connections, more units. The presence of multiple such connections would indicate oligosaccharide. Also, in option D, there's "OC(C2O)C" which might be a linkage between two sugars. Also, there are multiple O's in the structure. So this seems more likely to be an oligosaccharide.

Options A and B don't seem to have multiple rings. Option C is a lipid. So the answer is D.
</think>

The correct answer is D because the molecule described in this option contains multiple sugar units (pyranose and furanose rings) connected by glycosidic bonds, which is the defining feature of an oligosaccharide. The presence of oxygen bridge linkages between these monosaccharide units confirms its classification as an oligosaccharide. Other options either represent single sugar derivatives (B), acyclic or aromatic structures (C, A), or are not composed of monosaccharide units entirely.

Answer: D

SDAR-30B-A3B-Sci Example Output for math problem

User:
The 9 members of a baseball team went to an ice cream parlor after their game. Each player had a singlescoop cone of chocolate, vanilla, or strawberry ice cream. At least one player chose each flavor, and the number of players who chose chocolate was greater than the number of players who chose vanilla, which was greater than the number of players who chose strawberry. Let N be the number of different assignments of flavors to players that meet these conditions. Find the remainder when N is divided by 1000.
Remember to put your final answer within
boxed.

Assistant:
<think>
Okay, so there's this problem about a baseball team with 9 members. They all went to an ice cream parlor and each had one scoop of chocolate, vanilla, or strawberry. The conditions are that each flavor was chosen by at least one person, and the number of chocolate lovers is greater than vanilla, which is greater than strawberry. We need to find the number of possible assignments (N) that meet these conditions and then find N mod 1000.
Hmm, let's break this down. First, we have 9 players, each choosing one of three flavors. The constraints are:
1. Each flavor is chosen by at least one person.
2. Chocolate > Vanilla > Strawberry.
We need to count all possible triples (C, V, S) where C + V + S = 9, C, V, S >= 1, and C > V > S >= 1. Then, for each valid triple, calculate the number of ways to assign the flavors to the players, which is a multinomial coefficient.
Wait, actually, first, we have to find all possible triples (C, V, S) of positive integers such that C > V > S and C + V + S = 9. Then, for each such triple, compute the number of assignments, which is 9! / (C! V! S!), and sum all those up. Then take N mod 1000.
Okay, so step 1: Find all triples (C, V, S) with C > V > S >= 1 and C + V + S = 9.
First, let's find all integer partitions of 9 into three distinct parts, each part at least 1, in descending order.
Let me think. Let's list all possible combinations. Since C > V > S >= 1, and S must be at least 1, then S >= 1, V >= S + 1, C >= V + 1. Also, C + V + S = 9.
Let's denote S >= 1, V >= S + 1, C >= V + 1. So substituting:
C + V + S >= (V + 1) + V + S = 2V + S + 1. But since C + V + S = 9, then 2V + S + 1 <= 9.
But maybe a better way is to iterate over possible values of S.
Since S must be at least 1, and V and C must be smaller and larger than S respectively. Let's consider S starting at 1.
Case S = 1:
Then V must be at least S + 1 = 2. Then C must be at least V + 1 = V + 1. And C + V = 8.
So C + V = 8, with C > V >= 2.
Let's list possible V:
If V = 2, C = 6. Since 6 > 2 > 1, this works.
V = 3, C = 5. 5 > 3 > 1.
V = 4, C = 4. But then C = V, which doesn't satisfy C > V. So stop here.
So for S = 1, two triples: (6,2,1) and (5,3,1).
Case S = 2:
Then V >= 3, C >= V + 1. C + V = 7.
Possible V:
V = 3: C = 4. 4 > 3 > 2. Okay.

V = 4: C = 3. But C < V here. Doesn't work. So only (4,3,2).
Case S = 3:
Then V >= 4, C >= V + 1. C + V = 6. But V >= 4, so V = 4, C = 2. But C < V. Not possible. So no solutions here.
Wait, S = 3, must have V > S, so V >= 4, and C >= V + 1 >= 5. Then C + V >= 5 + 4 = 9. But C + V = 9 - S = 6. So 6 >= 9? Not possible. Therefore, S cannot be 3 or more.
Therefore, all possible triples are:
(6,2,1), (5,3,1), (4,3,2).
Wait, hold on. Let's check again.
Wait, S = 2:
C + V = 9 - 3 = 6? Wait, wait, S = 2, so C + V = 9 - S = 7.
Wait, S = 2: C + V = 9 - 2 = 7. Yes, that's right. So V = 3, C = 4. 4 + 3 = 7, correct.
Wait, S = 1: C + V = 8, so V = 2, C = 6; V = 3, C = 5.
S = 2: C + V = 7, V = 3, C = 4.
S = 3: C + V = 6. But V must be 4, so C = 2, which is invalid.
So only three triples:
(6,2,1), (5,3,1), (4,3,2). Wait, wait, wait. Wait, (4,3,2): C=4 > V=3 > S=2. Correct.
Wait, so 6 + 2 + 1 = 9; 5 + 3 + 1 = 9; 4 + 3 + 2 = 9. Correct.
So only three triples.
Wait, but wait, is there another case when S=1, V=4? Wait, if V=4, then C=4, but C must be greater than V, so no. So yes, only three triples.
Wait, but hold on, let me check again. Let's list all possible triples with S >= 1, C > V > S.
Alternatively, since all flavors are distinct, and with order C > V > S, so S must be at least 1, V at least S+1, C at least V +1.
But another way: let's think of the numbers as three distinct positive integers in descending order adding to 9.
So partitions of all permutations of 9 into 3 parts, each part >=1, decreasing.
Wait, how would that go?
Let me list all partitions of 9 into 3 distinct parts.
Wait, partitions with distinct parts:
Start with the largest possible part.
First partition: 6,2,1. 6+2+1=9.
Next: 5,3,1. 5+3+1=9.
Next: 4,3,2. 4+3,2=9.
Is there another?
Wait, 4,4,1? Not distinct. 5,2,2? Not distinct. 3,3,3? Not distinct. 7,1,1? Not. So yes, only three triples.
Therefore, only three valid triples.
Therefore, the total number of assignments is the sum over these three triples of the multinomial coefficients.
So for each triple (C, V, S), the number of assignments is 9! / (C! V! S!)!.
Therefore:
First triple: (6,2,1). Assign: 9! / (6! 2! 1!) = (362880)/(720 * 2 * 1) = 362880 / 1440 = 252.
Wait, 6! = 720, 2! = 2, 1! = 1. So denominator 720*2=1440. 9! = 362880. 362880 / 1440: Let's compute.
Divide numerator and denominator by 100: 362880 / 1440 = 3628. / 14.4? Wait, maybe better: Divide numerator and denominator by 10: 36288 / 144.
Then divide numerator and denominator by 12: 3024 / 12.
Wait, 144 * 200 = 28800. 36288 - 28800 = 7488. 144 * 50 = 7200. 7488 - 7200 = 288. 144*2=288. So total 200 + 50 + 2 = 252. Correct.

Second triple: (5,3,1). 9! / (5! 3! 1!) = 362880 / (120 * 6 * 1) = 362880 / 720 = 504.
Wait, 5! = 120, 3! = 6, so denominator 120*6=720. 362880 / 720 = 504. Correct.
Third triple: (4,3,2). 9! / (4! 3! 2!) = 362880 / (24 * 6 * 2) = 362880 / 288.
Let me compute 362880 / 288.
Divide numerator and denominator by 10: 36288 / 28.8.
Wait, better divide numerator and denominator by 24: 362880 /24 = 15120; 288 /24 = 12. So 15120 / 12 = 1260.
Wait, 24*6 = 144. Wait, no:
Wait, 4! = 24, 3! = 6, 2! = 2. So denominator: 24*6*2 = 288.
362880 / 288: Let's do 362880 divided by 288.
Divide numerator and denominator by 10: 36288 / 28.8. Not helpful.
Alternatively, 288 * 1000 = 288,000. 362,880 - 288,000 = 74,880.
288 * 200 = 57,600. 74,880 - 57,600 = 17,280.
288 * 60 = 17,280. So total: 1000 + 200 + 60 = 1260. Correct.
Therefore, the three multinomial coefficients: 252, 504, and 1260. Summing them: 252 + 504 = 756; 756 + 1260 = 2016.
Wait, so the total number N = 2016. Then the problem asks for N mod 1000, so 2016 divided by 1000 is 2 remainder 16. So 2016 mod 1000 = 16.
Wait, but hold on, that seems low. Wait, 2016 mod 1000 is 16. Hmm.
But wait, let me check again. Did I miss all possible triples?
Wait, but according to my earlier analysis, there are only three triples. Let me just make sure that's correct.
If S=1, then V can be 2 or 3.
If V=2, C=6; V=3, C=5.
If S=2, then V=3, C=4.
If S=3, impossible.
But wait, could there be triples where S=1, V=4, but C=4? But then C would equal V, which is not allowed. Similarly, V=4, S=1: V < C.
Wait, another thought: what if S=1, V=1? But no, V must be greater than S, so V >= 2.
Wait, okay, seems only three triples.
Wait, but wait, another way to think: think of the number of integer solutions to C + V + S = 9 with C > V > S >= 1.
This is equivalent to the number of integer solutions with C > V > S >= 1.
Let me think of substitution variables.
Let's set:
Let a = S >= 1
Let b = V - S >=1 (since V > S, so V - S >=1)
Let c = C - V >=1 (since C > V, so C - V >=1)
Then C = V + c, V = S + b, so C = S + b + c.
So C + V + S = (S + b + c) + (S + b) + S = 3S + 2b + c = 9.
With a >= 1, b >=1, c >=1, and S = a.
Wait, so substituting variables:
S = a >=1
b = S + b' something? Wait, maybe my initial substitution approach is complicating.
Alternatively, since C > V > S >=1, we can let:
Let's define S = S
1 <= S < V < C
And C + V + S = 9.
Let's shift variables as:
Let S' = S - 1 >=0

V′ = V - 2 >=0 (since V >= S + 1 >= 2)
C′ = C - 3 >=0 (since C >= V +1 >= 3)
Wait, but this might not work because V and C depend on S.
Alternatively, use substituting variables for the differences:
Let's define:
Let's let x = S >=1
Let y = V - S >=1 (since V > S)
Let z = C - V >=1 (since C > V)
Therefore:
C = S + y + z
V = S + y
So C + V + S = (S + y + z) + (S + y) + S = 3S + 2y + z = 9
With x >=1, y >=1, z >=1.
But x = S >=1, so we can write substituting variables:
Let S′ = S - 1 >=0
y′ = y -1 >=0
z′ = z -1 >=0
Then:
3(S′ +1) + 2(y′ +1) + (z′ +1) = 9
=> 3S′ + 3 + 2y′ + 2 + z′ +1 = 9
=> 3S′ + 2y′ + z′ + 6 = 9
=> 3S′ + 2y′ + z′ = 3
Now we need to find all non-negative integers S′, y′, z′ such that 3S′ + 2y′ + z′ = 3.
This equation can be solved by enumerating possible values of S′.
Case S′ =0:
Then 2y′ + z′ =3
Possible y′ =0: z′=3
y′=1: z′=1
y′=2: 2*2=4 >3, no.
So two solutions.
Case S′ =1:
3*1 + 2y′ + z′ =3 => 2y′ + z′=0 => y′=0, z′=0.
One solution.
S′ >=2: 3S′ >=6 >3, no.
So total three solutions:
1. S′=0, y′=0, z′=3 => S=1, y=1, z=4. Then:
S = S + y + z = 1 + 1 +4 =6? Wait, wait:
Wait, original variables:
Wait, S = S′ +1, right?
Wait, S′ = S -1, so S = S′ +1.
Similarly, y = y′ +1, z = z′ +1.
So:
Case 1: S′=0, y′=0, z′=3:
S =1, y=1, z=4
Then C = S + y + z =1 +1 +4 =6
V = S + y =1 +1=2
So (6,2,1). Correct.
Case 2: S′=0, y′=1, z′=1:
S=1, y=2, z=2
C =1 +2 +2=5. V =1 +2=3. So (5,3,1). Correct.

Case3: S'=0, y'=0, z'=3?
Wait, wait: Wait, for S'=0:
First solution was y'=0, z'=3.
Second solution y'=1, z'=1.
Third, y'=2 gives no solution.
Wait, for S'=1:
y'=0, z'=0:
S=1 +1=2, y=0 +1=1, z=0 +1=1.
Wait, S=2, V = S + y =2 +1=3, C = S + y + z =2 +1 +1=4. So (4,3,2). Correct.
Therefore, exactly three triples. So my analysis of triples was correct.
Thus, the total number of assignments is indeed 252 + 504 + 1260 = 2016.
So 2016 mod 1000 is 16.
Wait, but wait, hold on. Wait, is this correct? Because the problem says "the number of different assignments of flavors to players".
But in multinomial coefficients, each permutation is counted differently. Wait, but in our case, we are assigning different counts to different flavors. So the multinomial coefficient counts the number of ways to divide the 9 players into groups of size C, V, S. But since the flavors are distinguishable (chocolate, vanilla, strawberry), each group's assignment to a flavor is fixed.
Wait, actually, in the multinomial coefficient, 9! / (C! V! S!) counts the number of ways to assign each player a flavor of type C, V, S. Wait, actually, no. Wait, multiple players can have the same flavor.
Wait, actually, the multinomial coefficient formula counts the number of ways to partition the 9 players into a group of size C (chocolate), a group of size V (vanilla), and S (strawberry). So since the flavors are fixed, that's exactly the number of assignments. So yes, that calculation is correct.
But wait, but in the problem statement, the constraints are on the counts: C > V > S. So we already considered that when finding the triples, so the multinomial coefficients are correct.
Therefore, total number N = 2016. 2000 / 1000 = 2, remainder 16. So answer is 16.
Wait, but wait, I feel a bit uncertain because 2016 seems a bit low, but given that the multinomial coefficients are not that big. Let me check:
Wait, 252 + 504 = 756, plus 1260 = 2016. Hmm, okay.
Wait, but wait, the total number of assignments without any constraints (except at least one of each flavor) would be $3^9 - 3 * 2^9 + 3 * 1^9$ = using inclusion-exclusion. Let's compute that.
$3^9 = 19683$
$2^9 = 512$, so $3 * 512 = 1536$
$1^9 = 1$, so $3 * 1 = 3$
Thus total is 19683 - 1536 + 3 = 19683 - 1533 = 18150.
But our N is 2016, which is less than 18150. So that seems possible, since we have a lot more constraints.
Alternatively, if we consider the number of solutions to C + V + S =9 with C, V, S >=1, that's C(8,2)=28. Then, with C > V > S, we have 3 triples, which is 3/28. Then 18150 total assignments with at least one of each, times 3/28 is roughly (18150 / 28) *3 $\approx$ 648 *3 = 1944, which is close to 2016. So that seems plausible.
But wait, actually, the assignments with all three flavors present and C > V > S would be equal to the number of ordered triples times the multinomial coefficients, but actually, no. Wait, no, each ordered triple (C, V, S) with C > V > S corresponds to exactly one multinomial coefficient. But in reality, in the multiset, the assignments are counted based on the number of people per each flavor. So if you have different counts of C, V, S, then each permutation of the counts would correspond to different assignments. Wait, but in our problem, the flavors are fixed. That is, the assignment must have a certain number of chocolate, vanilla, strawberry, with the counts

C > V > S. Therefore, the multinomial coefficient correctly counts the number of assignments for each triple. So yes, 2016 seems okay.
Wait, but wait, in the problem statement, it says ̈the number of different assignments of flavors to players.̈ So each assignment is a function from the 9 players to the 3 flavors, with the constraints that each flavor is used at least once, and the number of chocolate > vanilla > strawberry.
Therefore, our approach is correct. So N is 2016, so mod 1000 is 16. Hmm.
Wait, but just to make sure, let's see if there's another way this problem could be interpreted.
Wait, for example, if we thought that assignments are different if the counts are different, but considering the permutations. Wait, but no, in combinatorics, assignments are usually different if a player has a different flavor. So each assignment is a mapping of each player to one of the three flavors, so the entire distribution matters. So the multinomial coefficient approach is correct.
Wait, but let me think again. Suppose we have three people: Player A, B, C. If all three choose different flavors, then the number of assignments is 3! = 6. Similarly, in our case, the multinomial coefficient counts all possible distributions where the counts are fixed. So in our problem, for each triple (C, V, S), the number of assignments is indeed 9! / (C! V! S!). So the calculation seems correct.
Wait, so the answer should be 2016 mod 1000 = 16. Hmm, but I have a doubt because sb qbq maybe when we consider that the counts must be C > V > S, but the problem says ässignments,̈ so maybe if two assignments have the same counts but different flavor assignments? Wait, no, because the flavors are fixed. Wait, actually, no. Wait, actually, chocolate > vanilla > strawberry. Wait, wait, hold on.
Wait a second. Wait, in the problem statement, the number of players who chose chocolate was greater than vanilla, which was greater than strawberry. So the counts must satisfy C > V > S, but the flavors are assigned to these counts. Wait, but in our analysis, we assumed that the counts C, V, S correspond to the flavors chocolate, vanilla, strawberry. Wait, but does the problem specify that the flavors are fixed? That is, if a player has a chocolate cone, that's fixed. But in our counts, we have considered the number of assignments to the counts C, V, S where C > vanilla, which is greater than strawberry. Wait, no, actually, in our problem, the counts C, V, S correspond to the actual counts of chocolate, vanilla, strawberry. Wait, actually:
Wait, no. Wait, hold on. Wait, confusion here.
Wait, let's re-express:
Let me clarify. Let me denote:
Let C = number of players who chose chocolate
V = number of players who chose vanilla
S = number of players who chose strawberry
Given that C > V > S >=1, and C + V + S =9.
Therefore, when we compute the multinomial coefficient, we are calculating the number of ways to assign C players to chocolate, V to vanilla, and S to strawberry. Since each assignment is a mapping from players to flavors with exactly those counts. Therefore, our calculations are correct.
Therefore, the total N = 252 + 504 + 1260 = 2016, so answer 16.
Wait, but in the problem statement, it says ̈the number of different assignments of flavors to players.̈ So ässignments̈here are the different functions, where each player is assigned one flavor. Then, if two assignments have the same counts but different shuffles, they are different. So yes, the multinomial coefficient counts all such assignments.
Therefore, I think my answer is correct.
Wait, but just to confirm, let's check an example. Suppose we have C=4, V=3, S=2.
Then the number of assignments is 9! / (4! 3! 2!) = 1260. Which is correct, because you can

choose 4 people for chocolate, then 3 out of remaining 5 for vanilla, and the last 2 are strawberry. Which is C(9,4)*C(5,3)*C(2,2) = 126 * 10 *1= 1260. Correct.
Similarly, for (6,2,1), C(9,6)*C(3,2)*C(1,1) = 84 * 3 *1 = 252. Correct.
Same with (5,3,1): C(9,5)*C(4,3)*C(1,1) = 126 *4 *1 = 504. Correct.
So all counts are correct.
Therefore, the total N = 2016, 2016 mod 1000 is 16.
Wait, but wait, I have a doubt because in some problems, similar ones have different answers.
Wait, let me check with another approach.
Alternatively, we can use generating functions.
But not sure. Alternatively, think of the number of ordered triples with C > V > S >=1, and compute the sum of multinomial coefficients.
But since we already did the case analysis and confirmed it, and the multinomial coefficients seem correct, maybe 16 is the right answer.
Wait, but let me recall that in the past, similar AIME problems have answers larger than 1000, but since this problem asks for N mod 1000, even if N is bigger than 1000, the answer can be anything. However, in our case, 2016 is a four-digit number, mod 2016, 2016 - 1000*2 = 16. So 16.
Wait, but wait, hold on. Wait, let me check again the sum:
252 + 504 is 756; 756 + 1260 is 2016. Yes.
But let me check the multinomial coefficients again:
First triple (6,2,1):
9! / (6! 2! 1!) = 362880 / (720 * 2) = 362880 / 1440 = 252. Correct.
Second (5,3,1): 362880 / (120 * 6) = 362880 / 720 = 504. Correct.
Third (4,3,2): 362880 / (24 *6 *2) = 362880 / 288 = 1260. Correct.
Yes, so total is 2016.
Wait, but now I just thought: Wait, is there an error in interpreting C > V > S? Because the problem says "the number of players who chose chocolate was greater than the number who chose vanilla, which was greater than strawberry." So C > V > S. So we have considered that. So if C, V, S are the counts for chocolate, vanilla, strawberry, then yes, C > V > S.
But in the problem statement, does it specify that the counts correspond to the flavors in that order? That is, is chocolate the largest, then vanilla, then strawberry? Yes, exactly. So our approach is correct.
Therefore, the answer should be 16.
Wait, but just to be 100% sure, maybe I made a mistake in missing the triples?
Wait, let's list all possible ordered triples (C, V, S) where C, V, S >=1 and C + V + S =9, and C > V > S.
We already found:
(6,2,1), (5,3,1), (4,3,2). Is there another?
Wait, let's suppose:
Suppose (5,4,0), but S must be >=1. Not okay.
(4,2,3): but S must be the smallest, so if order matters C > V > S, so (4,3,2) is the only.
Wait, another thought:
Wait, (7,1,1). But C > V > S requires all three distinct, so no.
Wait, (3,2,4). Again, ordering would be (4,2,2)? No.
Wait, if we list all possible partitions with distinct parts:
Wait, partitions of 9 into 3 distinct parts >=1:
The largest part is 6: 6,2,1
Largest part 5: 5,3,1
Largest part 4: 4,3,2.
That's it. So three partitions.

Therefore, no more triples.
Thus, the answer is indeed 16.
Wait, but wait. I found another resource online. Wait, no, I shouldn't do that. But according to my calculations, 16. Alternatively, maybe I made a mistake in the multinomial coefficients.
Wait, wait another check:
Wait, (6,2,1): The number of assignments is 9! / (6! 2! 1!) = (9*8*7*6!) / (6! 2*1)) = (9*8*7)/2 = 504 / 2 = 252. Correct.
(5,3,1): (9! / (5! 3! 1!)) = (362880) / (120 * 6) = 362880 / 720 = 504. Correct.
(4,3,2): (9! / (4! 3! 2!)) = (362880) / (24*6*2) = 362880 / 288 = 1260. Correct.
Sum: 2016. Yes.
Thus, my conclusion is that the answer is 16.
Wait, but hold on! Wait, hold on. Wait, hold on. Wait, wait a second. Wait, the problem says "the number of different assignments of flavors to players." Wait, but in our problem, we have labeled players, right? So each assignment is unique. So multinomial coefficients are correct.
Alternatively, if the players were unlabeled, but no, in AIME problems, people are considered distinguishable. So multinomial coefficients are correct.
Wait, but let me check a smaller case. Suppose there are 3 players, flavors chocolate, vanilla, strawberry. Each has flavor, with C > V > S >=1. So C=2, V=1, S=0. But S must be >=1. Wait, impossible. Wait, so another case.
Wait, maybe the calculation is correct.
Alternatively, if in the original problem, the total was 6 players, with C > V > S >=1. Then triples would be (3,2,1). Assignments: 6! / (3! 2! 1!) = 720 / 12 = 60. Or another triple (4,1,1), but not descending. So only (3,2,1). So 60 assignments. Makes sense.
Therefore, confident that our approach is correct.
Thus, the answer is 16.
Final Answer

$$\boxed{16}$$

</think>

To solve the problem, we are given a team of 9 players, each of whom chooses one scoop of ice cream of either chocolate, vanilla, or strawberry. The goal is to count the number of valid assignments, N, such that:
1. Each flavor is chosen by at least one person.
2. The number of people who chose chocolate is greater than the number who chose vanilla, which is greater than the number who chose strawberry.
We are then to compute the remainder when N is divided by 1000.
—
Step 1: Understand the Problem
We are to find the number of ways to assign 9 players to 3 flavors (chocolate, vanilla, and strawberry), such that:
Each flavor is chosen by at least one player.
The count of players choosing chocolate is greater than the count choosing vanilla, which is greater than the count choosing strawberry.
Let's denote the number of players who chose chocolate, vanilla, and strawberry as C, V, S respectively. Then the constraints become:
$C + V + S = 9$
$C > V > S \geq 1$
—
Step 2: Find All Valid Triples (C, V, S)
We seek all integer triples (C, V, S) that satisfy the constraints above.

Let's iterate through possible values of S (the smallest) and find corresponding V and C:
If $S = 1$:
$V \geq 2, C \geq V + 1$, and $C + V = 8$
Try $V = 2 \to C = 6 \to (6, 2, 1)$
Try $V = 3 \to C = 5 \to (5, 3, 1)$
$V = 4 \to C = 4 \to C = V$ invalid
If $S = 2$:
$V \geq 3, C \geq V + 1$, and $C + V = 7$
Try $V = 3 \to C = 4 \to (4, 3, 2)$
$V = 4 \to C = 3 \to C < V$ invalid

If $S = 3$:
$V \geq 4, C \geq 5$, and $C + V = 6$
$C + V \geq 9$ which contradicts $C + V = 6 \to$ no solutions

So the only valid triples are:
$(6, 2, 1)$
$(5, 3, 1)$
$(4, 3, 2)$

—
Step 3: Compute the Number of Valid Assignments
Each triple (C, V, S) corresponds to a multiset of assignments of the 9 players to the 3 flavors, with exactly C players choosing chocolate, V choosing vanilla, and S choosing strawberry.
The number of such assignments is given by the multinomial coefficient:

$$\frac{9!}{C!V!S!}$$

Let's compute this for each triple:
1. $(C, V, S) = (6, 2, 1)$:

$$\frac{9!}{6! \cdot 2! \cdot 1!} = \frac{362880}{720 \cdot 2} = \frac{362880}{1440} = 252$$

2. $(C, V, S) = (5, 3, 1)$:

$$\frac{9!}{5! \cdot 3! \cdot 1!} = \frac{362880}{120 \cdot 6} = \frac{362880}{720} = 504$$

3. $(C, V, S) = (4, 3, 2)$:

$$\frac{9!}{4! \cdot 3! \cdot 2!} = \frac{362880}{24 \cdot 6 \cdot 2} = \frac{362880}{288} = 1260$$

Now, sum the number of assignments:

$$N = 252 + 504 + 1260 = 2016$$

—
Step 4: Compute the Remainder Mod 1000

$$N \mod 1000 = 2016 \mod 1000 = \boxed{16}$$

—
Final Answer

$$\boxed{16}$$