PLANNED DIFFUSION

Daniel Israel^{1*} Tian Jin^{2*} Ellie Cheng² Guy Van den Broeck¹ Aditya Grover¹ Suvinay Subramanian³ Michael Carbin²

¹University of California, Los Angeles ²MIT CSAIL ³Google

ABSTRACT

A central challenge in large language model inference is the trade-off between generation speed and output quality. Autoregressive models produce high-quality text but generate tokens sequentially. Diffusion models can generate tokens in parallel but often need many iterations to match the same quality. We propose *planned diffusion*, a hybrid method that combines the strengths of both paradigms. Planned diffusion works in two stages: first, the model creates a short autoregressive plan that breaks the output into smaller, independent spans. Second, the model generates these spans simultaneously using diffusion. This approach expands the speed—quality Pareto frontier and provides a practical path to faster, high-quality text generation. On AlpacaEval, a suite of 805 instruction-following prompts, planned diffusion achieves Pareto-optimal trade-off between quality and latency, achieving 1.27x to 1.81x speedup over autoregressive generation with only 0.87% to 5.4% drop in win rate, respectively. Our sensitivity analysis shows that the planning mechanism of planned diffusion is minimal and reliable, and simple runtime knobs exist to provide flexible control of the quality-latency trade-off.

1 Introduction

Language model text generation is subject to a fundamental tradeoff between modeling textual dependencies and leveraging the parallel computation of modern hardware. Autoregressive (AR) models have defined the state-of-the-art in quality by excelling at the former (Radford et al., 2019); generating tokens sequentially allows them to capture dependencies with high fidelity, but this process inherently creates a latency bottleneck. Conversely, diffusion language models (Sahoo et al., 2024) are designed for parallelism but can struggle with coherence when generating with a low number of iterative steps needed for low latency (Wu et al., 2025). This establishes a performance tradeoff for LLMs, in which there exists a difficult compromise between inference speed and output quality.

We propose *planned diffusion*, a framework that challenges this compromise by treating text generation as a dynamic parallel scheduling problem. The motivation behind this technique is that the dependency structure among tokens is context-dependent; typical language model responses include semantically independent spans of tokens that can generate concurrently. For example, an answer that contains a bulleted list can generate each bullet point in parallel (Ning et al., 2024).

Planned diffusion realizes this concept through a novel hybrid architecture where a single, unified model transitions between autoregressive and diffusion-based generation. First, in a sequential planning stage, the model operates autoregressively to generate a high-level *execution plan* composed of structural control tags. This plan partitions the task into a set of conditionally independent sub-tasks. Second, in a parallel diffusion stage, the model executes this plan, simultaneously generating the text for all planned segments. By

^{*}Equal contribution.



Figure 1: A real example of planned diffusion. (1) **Autoregressive Plan**: The model first generates a sequential plan using control tags to define the structure and length of independent text spans. (2) **Programmatic Scaffold**: This plan is then translated into a scaffold where each span is initialized with a corresponding number of mask tokens. (3) **Diffusion Denoise**: Finally, the model denoises all spans in parallel with diffusion, generating the text for each section simultaneously to produce the complete response.

integrating these two modes, our method optimizes for parallel generation while preserving coherence and quality. This single-model, hybrid approach offers a distinct architectural advantage over other acceleration techniques, such as speculative decoding (Leviathan et al., 2023), which require multiple, separate models. To the best of our knowledge, this is the first text-only model that is trained with both discrete diffusion and autoregressive objectives. Figure 1 presents a sample generation produced by our planned diffusion model.

We make the following contributions:

- 1. We introduce *planned diffusion*, a new parallel generation technique that decomposes text generation into a sequential planning stage and a parallel diffusion stage.
- 2. We designed the control tag language, model training methodology, and inference algorithm that enable a single model to perform this hybrid generation process.
- 3. We demonstrate that planned diffusion achieves a state-of-the-art trade-off between speed and quality. On AlpacaEval, a suite of 805 instruction-following prompts, planned diffusion achieves Pareto-optimal trade-off between quality and latency; it achieves a 1.27x to 1.81x speedup over autoregressive generation while incurring only a 0.87% to 5.4% drop in win rate, respectively.
- 4. We present sensitivity analysis which validates that the planning mechanism design of our model is minimal and reliable; we show that simple runtime knobs exist to offer tunable control over the trade-off between generation speed and quality.

2 RELATED WORK

Our work builds upon recent developments in non-autoregressive and parallel decoding strategies. We position our contributions in the context of two primary research areas: diffusion-based language models and methods for achieving semantic parallelism.

Diffusion Language Models. Diffusion models have recently emerged as a new paradigm for generative language tasks (Austin et al., 2021; Sahoo et al., 2024). A significant body of research is focused on accelerating the inference process, which traditionally involves many iterative denoising steps (Liu et al., 2025a). These acceleration techniques include KV caching for diffusion models (Ma et al., 2025; Liu et al., 2025c), the use of autoregressive verification (Hu et al., 2025; Israel et al., 2025a), and the development of fast sampling strategies that reduce the number of required steps (Wu et al., 2025; Li et al., 2025; Hong et al., 2025). Our proposed planned diffusion framework is orthogonal and complementary to these methods; any diffusion sampling strategy can be integrated into the diffusion component of our algorithm to further increase performance. Other related work include block diffusion (Arriola et al., 2025) which enforces an autoregressive structure over blocks and planned denoising (Liu et al., 2025b) which learns an adaptive denoising schedule. While relevant, neither targets semantic parallelism, which is the purpose of planned diffusion.

Semantic Parallelism. While diffusion models achieve token-wise parallelism, they are not trained to achieve parallelism across larger portions of text at the semantic level. We define *semantic parallelism* as a broad class of techniques that produce models capable of parallelizing over semantically independent chunks of tokens. Many recent works explore semantic parallelism (Ning et al., 2024; Liu et al., 2024; Jin et al., 2025; Rodionov et al., 2025; Pan et al., 2025; Wen et al., 2025; Yang et al., 2025). While existing works operate within a purely autoregressive framework, to the best of our knowledge, our work is the first to propose a hybrid autoregressive-diffusion model for text. By combining an autoregressive planning phase with a parallel diffusion phase, our method utilizes diffusion parallelism in a structured manner, presenting a novel approach for efficient text generation.

Other Parallel Generation Techniques. There is a rich line of work on parallel generation. Insertion-based models parallelize by inserting tokens into partial positions, reducing decoding depth via simultaneous span updates (Stern et al., 2019). Speculative decoding accelerates autoregressive models by drafting multiple tokens and verifying them in parallel (Leviathan et al., 2023; Chen et al., 2023; Zhang et al., 2024). Planned diffusion differs conceptually as we use a single hybrid model for both autoregressive and parallel generation.

3 PRELIMINARIES

Generative language models learn a probability distribution over sequences of discrete tokens. In this work, our focus is on two main paradigms: autoregressive and discrete diffusion.

Autoregression. Autoregressive models are the standard for sequential text generation. They factorize the joint probability of a token sequence $x=(x_1,x_2,\cdots)$. The probability distribution under this model, which we denote as p_{AR} , is given by:

$$p_{AR}(x) = \prod_{i=1}^{|x|} p_{\theta}(x_i | x_{< i})$$
 (1)

where p_{θ} is a parameterized conditional distribution over tokens. In autoregression, each token is sequentially sampled conditioned on all previously generated tokens.

Discrete Diffusion. Discrete diffusion models learn to reverse a fixed data corruption process that gradually introduces noise into a clean sequence. For text, this involves incrementally replacing tokens with a special *mask* token (Austin et al., 2021). Let x^0 be a clean sequence of tokens. The forward corruption process q

produces a noisy version x^t at a timestep $t \in [0,1]$. The distribution of a corrupted sequence x^t follows:

$$q_{t|0}(x_i^t \mid x_i^0) = \begin{cases} t, & \text{if } x_i^t = [\text{MASK}] \\ 1 - t, & \text{if } x_i^t = x_i^0 \\ 0 & \text{otherwise} \end{cases} q_{t|0}(x^t \mid x^0) = \prod_i q_{t|0}(x_i^t \mid x_i^0)$$
 (2)

Because the true posterior over a forward noising process is intractable to compute exactly (Lou et al., 2024), diffusion models learn an approximation by maximizing a variational lower bound on the log-likelihood (Sahoo et al., 2024).

$$\log p_{\mathsf{D}}(x^0) \ge \mathbb{E}_{t \sim U(0,1), x^t \sim q(x^t | x^0)} \sum_{i} \mathbb{1}(x_i^t = [\mathsf{MASK}]) \log p_{\theta}(x_i^0 | x^t)$$
 (3)

Unlike autoregressive sampling, the above diffusion distribution emits a sampling algorithm capable of sampling multiple tokens in parallel. For brevity, when referring to diffusion we will let $p_D(x)$ denote the clean distribution obtained through the above process.

In planned diffusion, we will assume p_{AR} and p_D can be conditioned on prior context tokens $c=(c_1,c_2,\cdots)$. Thus, we shall utilize the distributions $p_{AR}(\cdot \mid c)$ and $p_D(\cdot \mid c)$ respectively.

4 PLANNED DIFFUSION

We first formalize planned diffusion as a probabilistic model that factorizes the text generation process into distinct planning and diffusion components. We then describe concrete implementation details towards realizing planned diffusion. The details of the implementation comprise of three key contributions: (i) a synthetic data curation pipeline to produce text annotated with *planning control tags*, (ii) a tailored training objective with a custom attention masking to enforce the specified conditional independencies, and (iii) an inference procedure that utilizes KV caching to facilitate efficient parallel decoding.

4.1 FORMAL DESCRIPTION

Ultimately, planned diffusion will be comprised of multiple iterations of planning and diffusion stages. We shall formalize the generative process for a single iteration of planned diffusion. An iteration begins with an initial context c which may contain the prompt or any previously generated tokens. It then produces a plan $z=(z_1,z_2,\ldots)$ followed by its corresponding asynchronous execution x, where both are sequences of tokens. The plan z defines the structure for the subsequent parallel generation, specifying a set of b(z) asynchronous spans. For each span $k \in \{1,\ldots,b(z)\}$, the plan also specifies its length $l_k(z)$.

Let x(k) be the sequence of tokens corresponding to the k-th span, where $|x(k)| = l_k(z)$. The complete sequence generated in the diffusion phase is the concatenation of these spans, $x = \bigoplus_{k=1}^{b(z)} x(k)$. The joint probability of generating the plan z and the content x within a single iteration of planning and diffusion, conditioned on the context c, is given by the following factorization:

$$p_{\text{PD}}(z, x|c) = \underbrace{p_{\text{AR}}(z|c)}_{\text{Planning}} \underbrace{p_{\text{D}}(x|z, c)}_{\text{Diffusion}} \tag{4}$$

where p_{PD} denotes the distribution of planned diffusion. In this definition, the diffusion distribution p_D is conditioned on the plan z. Planned diffusion entails a denoising distribution that samples each span x(k) independently given z. Thus, the probability of a whole span x factorizes over individual spans x(k).

$$p_{D}(x|z,c) = \prod_{k=1}^{b(z)} p_{D}(x(k)|z,c)$$
 (5)

Thus, planned diffusion is able to exploit conditional independence between spans to achieve greater parallelism. In this formulation, we describe one stage of planned diffusion composed of autoregressive planning and conditional diffusion. In general, this framework can be extended to multiple iterations of planning and diffusion chained together by setting \boldsymbol{c} to the output of the previous iteration. This process can be seen in Algorithm 1.

4.2 DATA

To train a model that adheres to the factorization in Equation 5, we require data that distinguishes between the autoregressive planning and parallel diffusion phases. Below, we de-

```
Algorithm 1 Planned Diffusion
```

```
1: function PLANNED_DIFFUSION(c)
        loop
2:
3:
             Sample plan z \sim p_{AR}(\cdot|c)
             Parse z to get spans b(z) and their lengths \{l_k(z)\}_{k=1}^{b(z)}
4:
5:
             for k = 1, ..., b(z) do in parallel
6:
                  Sample tokens x(k) \sim p_D(\cdot|z,c)
7:
             end for
             x \leftarrow \bigoplus_{k=1}^{b(z)} x(k)
8:
9:
             c \leftarrow c \oplus z \oplus x
10:
             if z[end] = \langle eos/ \rangle then
11:
                  break
12:
              end if
         end loop
13:
14:
         c \leftarrow RemoveControlTags(c)
15:
         return c
16: end function
```

scribe the data curation pipeline we introduce to delineate these distinct generation stages.

Control Tags. A span intended for parallel generation is first defined during a sequential planning stage. This is done using a paired <topic>...</topic> tag structure. Within this structure, the model generates a concise description of the span's content (e.g., "definition" in Figure 1) and its predicted length (e.g., "30" in Figure 1). During the parallel diffusion stage, the model generates the tokens for each planned span within a corresponding <async>...</async> tag pair. Finally, the <sync/> tag conveys generation dependency: tokens that follow <sync/> may require details produced inside the preceding <async> spans, so the sampling algorithm continues sequential planning only after those spans are filled and available. We add all control tags to the model's vocabulary for training and inference and strip them during post-processing of final outputs.

Finetuning Dataset. We annotate the SlimOrca instruction—finetuning dataset (Lian et al., 2023) for parallel generation, following Jin et al. (2025). We prompt a GEMINI model with the syntax and semantics of the control tags. See Appendix A for more details. GEMINI inserts the control tags into the response from each instruction-response pair in the dataset. The opening <async> carries two attributes: topic (a concise label, ≤ 3 words) and tokens (a coarse length estimate, e.g., multiples of 10 tokens). We impose that every non-control-tag token lies inside exactly one <async>...</async> span (no nesting, no overlap). During preprocessing for training, we insert 0–10 tokens of stochastic padding inside each <async> span, allowing the diffusion model to generate with variable length shorter than the masked input. We validate well-formedness (balanced tags, coverage, non-overlap, attribute types/ranges) and discard malformed cases. Figure 2 contains a concrete example of the tagging language used by planned diffusion.

4.3 Training

Training Loss. The goal of training is to maximize the joint probability over plan tokens and their content. Given an annotated dataset \mathcal{D} in which a single clean example is given by $Y \in \mathcal{D}$, we decompose Y into sets of planning tokens Z and content tokens X, such that $Y = Z \cup X$. X^t is a noised sequence of tokens under noise distribution $q_{t|0}$ as previously defined. Thus, X^t contains masked tokens with probability t. Let f_{θ} be the function to instantiate planned diffusion, parameterized by θ . We use the notation $f_{\theta}(x,i)$ to signify that the model takes input x and makes a prediction at index i of the sequence. Finally, let $M_i(X)$ be an attention masking function that takes as input a sequence X and outputs a subset $M_i(X) \subseteq X$ of the sequence that f_{θ} has access to at a particular index i. We describe the attention masking in more detail in the following

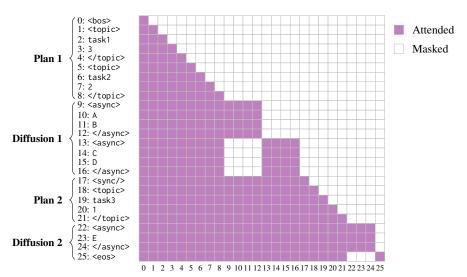


Figure 2: The attention mask for planned diffusion combines causal and bidirectional attention. Causal attention is used for sequential planning stages. Bidirectional attention is used within <async> spans for parallel denoising, with concurrent spans isolated from each other. After a <sync/> token, subsequent tokens can to attend to all prior tokens. For illustrative purposes, this example shortens the diffusion spans.

paragraph. With CE as the cross-entropy loss, our overall training objective is given by

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{\substack{Y \sim \mathcal{D} \\ t \sim U(0,1)}} \frac{1}{|Y|} \sum_{y_i \in Y} \underbrace{\mathbb{1}(y_i \in Z) \text{CE}(f_{\theta}(y_{< i}, i), y_i)}_{\text{Autoregressive}} + \underbrace{\frac{1}{t} \mathbb{1}(y_i \in X) \text{CE}(f_{\theta}(M_i(X^t \cup Z), i), y_i)}_{\text{Diffusion}}$$
(6)

Note that in the training objective, the same noise parameterized by t is applied to diffusion spans across multiple iterations of planning and diffusion, where future diffusion spans will be conditioned on previously sampled diffusion spans. In this decision, we are applying the interpretation of a diffusion model as an any-order autoregressive model capable of supporting arbitrary conditional queries at inference time (Shi et al., 2024). The same technique is used by Llada 7B (Nie et al., 2025), which is trained with a diffusion objective, but during inference used for semi-autoregressive block sampling.

Attention Mask. We implement the following rules via the attention mask M_i . First, planning tokens, which are composed of control tags and their attributes, are given causal attention (Plan 1 and 2 in Figure 2). Second, tokens inside the same <async>...</async> span use bidirectional attention, as required for diffusion-based parallel denoising (Diffusion 1 and 2 in Figure 2). In Section 5, we also show that allowing fully bidirectional dense attention over diffusion spans is another valid and promising way to train planned diffusion denoising. Third, spans remain isolated from other spans until synchronization: before <sync/>, we enforce no cross-span attention; after <sync/>, subsequent tokens may attend to all previously completed <async> spans.

4.4 INFERENCE

Variable Length Denoising. Typically, diffusion models are configured to generate given a fixed number of denoising steps. Generation quality increases and speed decreases with the number of steps. Unlike vanilla diffusion, planned diffusion does not generate a predetermined number of tokens, so the number of denoising steps cannot be fixed. We define a parameter r called the *steps ratio*. Given a generation length

|x|, the number of denoising steps will be s=r*|x|. Note that for a plan z and multiple diffusion spans x(k) for $k\in\{1,...,b(z)\}$, the step ratio is defined as $s=r*\max_k l_k(z)$. The number of denoising steps depends only on the length of the longest span, because spans are denoised in parallel. Higher step ratio corresponds to higher quality and slower generation, while lower step ratio leads to faster generation but worse quality.

KV Caching. KV caching plays a substantial role in the efficiency of planned diffusion. The KV cache of planned diffusion is derived from the model's hybrid attention mask. The general principle is that a token can be cached if its key and value embeddings are unaffected by future tokens. This is determined by whether a token attends to future positions in the sequence. The autoregressive planning stage uses causal attention and a conventional application of KV caching (Pope et al., 2023). In contrast, the diffusion stage employs a bidirectional mask, in which tokens inside an <async> span attend to each other. Because bidirectional attention does not support KV caching (Israel et al., 2025b), tokens inside an <async> span cannot be cached until their respective denoising process is complete. However, subsequent tokens, such as those in a new planning stage following a <sync/> tag, can efficiently attend to the KV cache of the preceding planning and diffusion stages. This caching mechanism is essential for combining the speed of diffusion with the computational savings of autoregressive KV caching.

Diffusion Inference. While diffusion models are trained on an objective that implies a random unmasking order, in practice diffusion does not achieve the best results in this setting. Planned diffusion can integrate any inference algorithm that determines diffusion unmask order. We apply entropy-ordered unmasking, which is a default inference algorithm of Dream 7B (Ye et al., 2025).

5 EXPERIMENTAL EVALUATION

We experimentally assess the performance of planned diffusion, focusing on its trade-off between generation quality and latency. Our results show that planned diffusion expands the latency-quality Pareto frontier for text generation when compared to autoregressive and other diffusion-based approaches. Furthermore, we demonstrate that our method scales better with additional compute; planned diffusion continues to improve with more training, whereas the performance of the autoregressive baseline plateaus.

Training setup. We fine-tune Dream-7B-Base (Ye et al., 2025; Qwen et al., 2025); the base model is first pre-trained autoregressively and then further pre-trained with a diffusion objective. We train with AdamW (Kingma & Ba, 2017; Loshchilov & Hutter, 2019), peak learning rate 5×10^{-5} with linear decay, and bfloat16 precision. We use per-GPU batch size 1 and global batch size 4. Because autoregressive and diffusion language models have different optimal epoch counts (Prabhudesai et al., 2025), we sweep epochs over $\{2,4,8,16\}$. We fine-tune on Gemini-annotated SlimOrca instruction-following data (Section 4.2); we keep the control tags for planned diffusion and strip them for autoregressive and diffusion baselines. Training runs on $4 \times \text{H}200$ (141 GB) with PyTorch (Imambi et al., 2021) and HuggingFace (Wolf et al., 2020).

Decoding strategies. We compare four decoding strategies. (i) *Autoregressive* samples tokens sequentially from the autoregressive model. (ii) *Diffusion* samples masked tokens in parallel from the diffusion model; we configure the number of denoising steps to equal the number of new tokens as this produces the highest quality generation (Lou et al., 2024; Shi et al., 2025; Sahoo et al., 2024). (iii) *Fast-dLLM* (Wu et al., 2025) samples from the same diffusion model but with an inference-time only optimization. We configure denoising steps to be half the number of new tokens and use a confidence threshold of 0.9, which Wu et al., 2025 use as the default value. (iv) *Planned diffusion* (ours) samples a plan autoregressively from the planned diffusion model, then samples masked tokens within each span in parallel from the same model; for each span, we configure denoising steps to equal its predicted token count. (v) *Planned diffusion with dense attention* (ours) is a hardware-friendly variant of planned diffusion. Vanilla planned diffusion treats concurrently generated spans as conditionally independent and denoises such spans using block-sparse at-

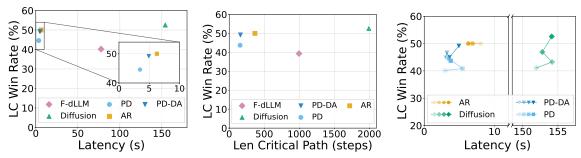


Figure 3: Evaluation of planned diffusion on the AlpacaEval benchmark. **Left:** A comparison of latency versus length-controlled win rate shows planned diffusion establishing a new Pareto frontier, offering a better trade-off between speed and quality. **Middle:** An analysis of the average critical path length reveals that planned diffusion requires substantially fewer sequential forward passes than autoregressive generation. **Right:** A scaling analysis shows that planned diffusion's win rate continues to improve with more training, while the autoregressive baseline's performance flatlines. Within each method, color brightness encodes training epochs: lightest to darkest corresponds to 2, 4, 8, and 16 epochs. **AR**: Autoregressive; **PD**: Planned Diffusion; **PD-DA** Planned Diffusion with Dense Attention; **F-dLLM**: Fast-dLLM.

tention (c.f., Figure 2). Planned diffusion with dense attention removes this independence assumption and uses dense attention during training and inference, so concurrently generated spans can fully cross-attend; this eliminates block-sparse attention and improves GPU utilization.

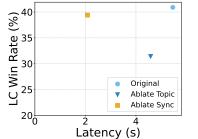
Inference setup. We sample with temperature 0.2 and top-p 0.95 following (Ye et al., 2025), and cap the total sequence length at 2048 tokens. Inference runs on the same H200 hardware configuration.

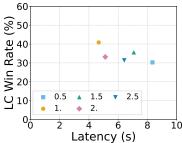
Benchmark and metrics. We evaluate on AlpacaEval (805 instruction-following prompts) (Li et al., 2023; Dubois et al., 2024). For each method we report: (i) *average latency*—the mean wall-clock time per response; and (ii) *quality*—length-controlled win rate (LCWR) with an LLM-as-judge. We use the recommended default configuration from (Dubois et al., 2024) due to its high correlation with human preference. We fix the LCWR reference to the best autoregressive baseline. We identify this reference by evaluating the autoregressive variants (2, 4, 8, and 16 epochs) against the 2-epoch variant and choosing the model with the highest quality ¹. The 16-epoch model wins and serves as the fixed reference for all LCWR scores.

Quality and Latency. We plot the latency–quality trade-off in the left figure of Figure 3. Planned diffusion (PD) achieves a 22.4× speedup over Fast-dLLM (F-dLLM) and higher quality (44.6% vs. 40.2% length-controlled win rate). Relative to autoregressive decoding (AR), planned diffusion achieves a 1.81× speedup while achieving a 44.6% length-controlled win rate against the autoregressive reference (50.0%). Planned diffusion with dense attention (PD-DA) variant further improves quality to 49.2% at 1.27× speedup relative to autoregressive decoding. Diffusion attains the highest quality (52.6%) but requires substantially more inference time, requiring 25× the latency of autoregressive decoding.

Speedup Analysis. We attribute much of planned diffusion's speedup over autoregressive decoding to its shorter *critical path* of generation. We define critical path length as the number of forward passes required to produce the final response. The middle panel of Figure 3 shows that, on AlpacaEval, the average critical path of autoregressive decoding is 2.8× as long as that of planned diffusion (PD, 367.3 vs. 155.2 steps) and 2.3× as long as the dense attention variant (PD-DA, 367.3 vs. 160.0 steps). This is expected, as planned diffusion enables multiple spans to denoise simultaneously. The realized speedup (1.85×) is smaller than the critical path reduction (2.8× for PD, 2.3× for PD-DA) because each planned diffusion step does more

¹They tie on length controlled win rate, so we break ties using raw win rates.





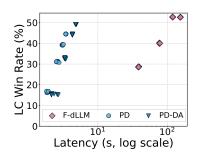


Figure 4: Additional analysis. **Left (Plan ablation):** Removing topic from autoregressive plan harms quality yet removing <sync/> tokens boosts speedup without significantly hurting quality. **Middle (Span lengths):** LCWR peaks at a length-scaling factor of 1.0; length prediction in autoregressive plan does not exhibit systemic error. **Right (Quality versus Latency Sweep):** Varying the step ratio $(r = \{0.25, 0.5, 0.75, 1\})$ and the confidence threshold $(\tau = \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\})$ hyperparameters produces a smooth quality–latency trade-off.

work: KV-cache reuse is lower and per-step compute is heavier than an autoregressive token step. We also observe a difference in the total number of tokens generated (this includes control tokens and pad tokens) between planned diffusion and autoregressive decoding. Planned diffusion and planned diffusion with dense attention produce approximately $9.1\%^2$ and 3.4% fewer tokens, respectively.

Scaling. The right panel of Figure 3 shows how the latency-quality tradeoff evolves with training epochs for the four training configurations we examine. Autoregressive training shows no benefit from additional training compute: the length-controlled win rate remains flat at 50.0% across 2, 4, 8, and 16 epochs. Both variants of planned diffusion benefit moderately from additional training compute. Planned diffusion improves from 40.2% (2 epochs) to 43.7% (16 epochs), a gain of 3.5 percentage points, while planned diffusion with dense attention improves from 44.9% (2 epochs) to 49.2% (16 epochs), a gain of 4.3 percentage points. Diffusion benefits significantly from additional training compute, rising from 41.1% (2 epochs) to 52.6% (16 epochs), a gain of 11.5 percentage points that ultimately surpasses the autoregressive baseline.

Takeaway. Planned diffusion sets a new latency–quality Pareto frontier and continuously improves with more training, while the autoregressive baseline plateaus.

6 Additional Analysis

We present additional analysis on several design decisions key to planned diffusion.

Plan Ablation. We remove two components of the autoregressive plan and evaluate their impact on inference quality and latency. We train all model variants for 4 epochs in this ablation study. The original 4-epoch planned diffusion achieves an LC win rate of 40.9% at 5.46 s latency. Removing the topic attribute from the training data and re-training the planned diffusion model significantly harms inference quality. Specifically, it reduces the length controlled win rate from 40.9% to 31.4% and the latency from 5.46 seconds to 4.58 seconds. We conclude that topic attributes are critical for maintaining inference quality.

Recall that <sync/> marks a synchronization barrier: decoding beyond it begins only after all prior content is finalized. Deleting all <sync/> tokens from the training data and re-training the planned diffusion model

²The gap in output length is in part due to difference in optimal training epochs for planned diffusion (8ep) and autoregressive (16ep). The output length difference between 16ep planned diffusion and autoregressive reduces to 5.6%.

significantly reduces latency while modestly reducing quality. Specifically, it reduces latency from 5.46 s to 2.08 s while the LC win rate falls only from 40.9% to 39.4%. We conclude that omitting <sync/> can offer a favorable quality—latency trade-off in latency-sensitive settings.

Span Lengths. We test whether the model predicts span lengths accurately. Accurate length prediction is key to achieving good generation quality for planned diffusion, as the diffusion denoising phase of generation cannot alter the span length. Systematic over-prediction wastes time by adding masks and denoising steps, while systematic under-prediction harms quality by forcing content truncation.

To test for potential systematic deviation from the optimal generation length, we multiply the model's predicted span length by a length scaling factor to set the number of masks, sweeping the factor over $\{0.5, 1.0, 1.5, 2.0, 2.5\}$. We then measure length-controlled win rate (LCWR) and latency under identical inference settings. Latency rises with factors above 1.0 as expected because larger spans require more mask tokens and denoising steps. Quality peaks at the model's originally predicted length (i.e scaling factor of 1.0). Deviating by $\pm 50\%$ of the predicted length reduces LCWR. Interestingly, we do not observe additional denoising steps leading to accuracy improvements. The model's length predictions are accurate; we do not observe systematic over/under-prediction.

Quality versus Latency Sweep. By default, to maximize generation quality, we set the number of denoising steps for a span equal to that span's length, and we unmask a position only when the model is highly confidence of its prediction. In this analysis, we measure the quality–latency trade-off by varying the number of denoising steps as well as confidence threshold for decoding.

Planned diffusion employs a hyperparameter called step ratio (see Section 4.4), which determines the number of denoising steps relative to the span length. The step ratio r scales steps relative to span length. When r equals 1 the number of steps equals the number of tokens; smaller r uses fewer steps per token. The confidence threshold τ used by Wu et al. (2025) selects which positions to unmask at each denoising step. For each masked position, if the model's top-token probability at that position is at least τ , we decode that token and unmask the position; otherwise we keep it masked for decoding at a later time step.

Sweeping r over $\{0.25, 0.5, 0.75, 1\}$ and τ over $\{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ yields a smooth trade-off between generation quality and inference latency; for example, at r=0.75 and $\tau=0.9$, planned diffusion yields a new Pareto-optimal operating point, achieving LCWR of 39.5% and latency of 3.20 seconds.

Across the step ratio and the confidence threshold, planned diffusion yields higher quality at equal or lower latency for most operating points, while planned diffusion with dense attention and fast-dLLM can sometimes reach higher quality at the cost of increased latency.

Takeaway. Our analysis validates that our planning mechanism is minimal and reliable. At inference time, the step ratio and confidence threshold provide tunable control of the speed–quality trade-off.

7 CONCLUSION

In this work, we introduce planned diffusion, a novel hybrid architecture combining autoregressive planning with parallel diffusion-based execution to improve the trade-off between latency and quality in text generation. Our model exploits opportunities for parallelism within the semantic structure of text. Experimental evaluation shows that planned diffusion expands the latency—quality Pareto frontier, achieving a significant speedup over pure autoregressive and diffusion baselines with a minimal drop in quality. This efficiency gain is driven by a shorter critical path and our approach's scalability with respect to training. We also show planned diffusion offers fine-grained control over the speed-quality trade-off at inference time. Planned diffusion provides a promising framework for developing faster and more efficient large language models.

ACKNOWLEDGMENTS

This work was funded in part by the DARPA ANSR, CODORD, and SAFRON programs under awards FA8750-23-2-0004, HR00112590089, and HR00112530141, NSF grant IIS1943641, and gifts from Adobe Research, Cisco Research, Qualcomm, and Amazon. This work was also supported by NSF CAREER Grant #2341040, Schmidt Sciences Early Career Fellowship, Okawa Foundation Research Award. Approved for public release; distribution is unlimited. This material is based upon work supported in-part by the National Science Foundation Graduate Research Fellowship under Grant No. 2141064. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This work was supported in part by CoCoSys, one of the seven centers sponsored by the Semiconductor Research Corporation (SRC) and DARPA under the Joint University Microelectronics Program 2.0 (JUMP 2.0). We acknowledge the MIT Office of Research Computing and Data for providing high performance computing resources that have contributed to the research results reported within this paper.

REPRODUCIBILITY STATEMENT

We specify the starting checkpoint, fine-tuning hyperparameters, inference settings, and hardware/software setup in Section 5. Section A includes a shortened version of the data-annotation prompt; we omit in-context examples for brevity.

LLM USE DISCLOSURE

We used LLMs to improve sentence-level writing and to search for related work.

REFERENCES

- 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.
- 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.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.
- Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.
- Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv preprint arXiv:2507.18578*, 2025.
- Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv* preprint arXiv:2505.21467, 2025.
- Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. In *Programming with TensorFlow: solution for edge computing applications*, pp. 87–104. Springer, 2021.

- Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025a.
- Daniel Israel, Aditya Grover, and Guy Van den Broeck. Enabling autoregressive models to fill in masked tokens. *arXiv preprint arXiv:2502.06901*, 2025b.
- Tian Jin, Ellie Y Cheng, Zachary Ankner, Nikunj Saunshi, Blake M Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ZfX43ZZRZR.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.
- Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language models know the answer before decoding. *arXiv* preprint arXiv:2508.19982, 2025.
- Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.
- Wing Lian, Guan Wang, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". Slimorca: An open dataset of gpt-4 augmented flan reasoning traces, with verification, 2023. URL https://https://huggingface.co/Open-Orca/SlimOrca.
- Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion, 2025a. URL https://arxiv.org/abs/2410.01949.
- Mingdao Liu, Aohan Zeng, Bowen Wang, Peng Zhang, Jie Tang, and Yuxiao Dong. Apar: Llms can do auto-parallel auto-regressive decoding, 2024. URL https://arxiv.org/abs/2401.06761.
- Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising, 2025b. URL https://arxiv.org/abs/2410.06264.
- Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. arXiv preprint arXiv:2506.06295, 2025c.
- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.
- Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL https://arxiv.org/abs/2310.16834.
- Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.
- 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.

- Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting llms for efficient parallel generation, 2024. URL https://arxiv.org/abs/2307.15337.
- Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models, 2025. URL https://arxiv.org/abs/2504.15466.
- Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.
- Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion beats autoregressive in data-constrained settings, 2025. URL https://arxiv.org/abs/2507.15857.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.
- 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.
- Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel Ilm generation via concurrent attention, 2025. URL https://arxiv.org/abs/2504.06261.
- 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.
- 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.
- Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data, 2025. URL https://arxiv.org/abs/2406.04329.
- Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations, 2019. URL https://arxiv.org/abs/1902.03249.
- Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute, 2025. URL https://arxiv.org/abs/2509.04475.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation, 2025. URL https://arxiv.org/abs/2506.09991.

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.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft&; verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.607. URL http://dx.doi.org/10.18653/v1/2024.acl-long.607.

A DATA ANNOTATION PROMPT

We present a shortened version of our data-annotation prompt below. We use it to instruct Gemini Flash 2.0 (temperature = 1.0, top-p = 0.95) to annotate our training data.

You will first identify whether the given chatbot response may be generated in parallel. You are to then annotate the chatbot response using specific tags that highlight segments suitable for parallel generation.

Use <async> tags to denote segments of text that may be generated asynchronously in parallel with respect to the text that follows. Thus apply <async> tags only to sentences that do not serve as necessary context for subsequent sentences. Sentences that are crucial for understanding or generating following text are not suitable for parallel asynchronous generation. For each <async> tag, include a very concise topic description of the text surrounded within the <async> tags. The topic description will be accessible to text generation after the closing async tag to ensure continuity and coherence.

Use the singleton <sync/> tag for synchronization. All content generated before <sync/>, including text marked by <async> is accessible to subsequent text generation after the <sync/> tag, ensuring continuity and coherence.

Detailed Instructions:

- Tagging Rules:
- Use <async> tag in pairs.
- Ensure all text content is encapsulated within <async> tags.
- Ensure that each <async> tag encompasses at least five words.
- Refrain from altering the content of the response during annotation.
- Use a maximum of 3 words in the topic description.
- Use <sync/> sparingly as it introduces significant slowdown.