DiffusionGemma Technical Report

DiffusionGemma Team, Google DeepMind[1]

We introduce DiffusionGemma, an experimental open-weight language model that uses discrete diffusion to generate text at exceptionally high speed. Rather than decoding one token at a time, DiffusionGemma iteratively refines blocks of 256 tokens in parallel, avoiding the sequential decoding bottleneck of conventional autoregressive (AR) large language models. Instead of training from scratch, we obtain DiffusionGemma by fine-tuning the mixture-of-experts Gemma 4 model with 3.8B activated and 25.2B total parameters. Our compute-efficient two-stage training pipeline uses fewer than 10% of the starting AR model's total training token budget. The first stage uses supervised fine-tuning to teach bidirectional denoising, while the second stage combines reinforcement learning with sampler distillation to jointly improve generation quality and inference efficiency. DiffusionGemma establishes a new Pareto frontier for the trade-off between generation speed and model capability. Averaged across our full evaluation suite, it generates around 20 tokens per forward pass and achieves roughly 1,500 output tokens per second on a single NVIDIA H100 GPU, which is substantially faster than AR models even with state-of-the-art speculative decoding. DiffusionGemma also retains the starting model's support for thinking mode, multimodal inputs, and long contexts. Despite diffusion fine-tuning, it remains capable of AR generation with only minor performance degradation, suggesting a path toward hybrid diffusion-AR decoding.

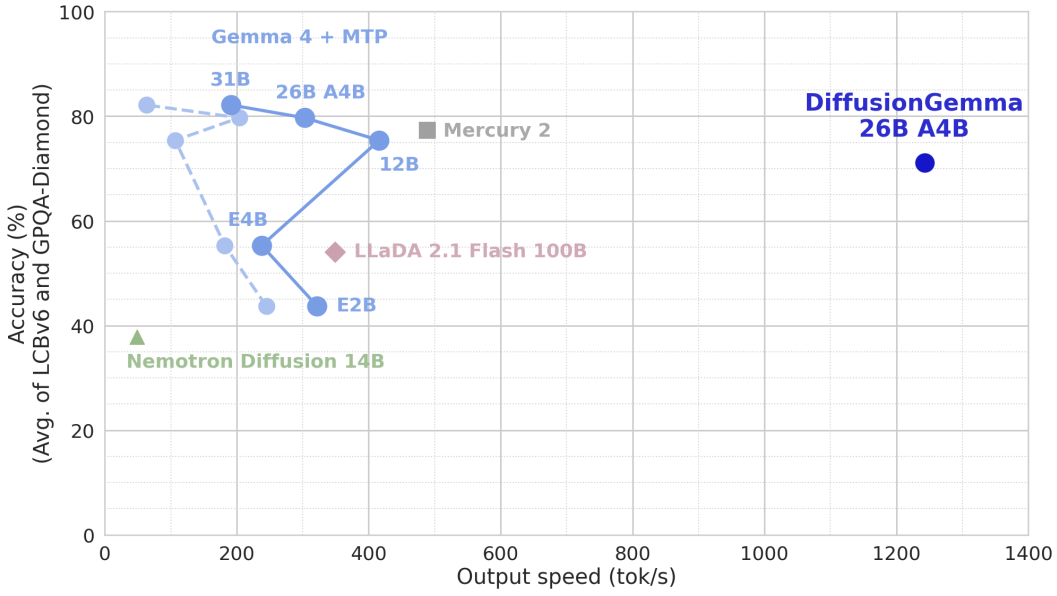

Figure 1 | **Pareto plot of quality versus output decoding speed comparing DiffusionGemma to the Gemma 4 model family and other diffusion models.** Quality and output speed are calculated as the average across GPQA-Diamond and LiveCodeBench-v6 for all models. Gemma 4 and DiffusionGemma output speeds are measured on a single H100 (FP8, batch size 1).[2] Nemotron 14B is measured on a single H100 (bfloat16, batch size 1). LLaDA 2.1 Flash 100B is measured on 8× NVIDIA B200 GPUs (bfloat16, batch size 1).[3] Mercury 2 was measured via the public OpenRouter API with `high` reasoning effort (see Appendix E for speed estimation). The light blue dots and dashed line represent the Gemma 4 family of models *without* MTP.

[1]Contributions, Acknowledgements and full author list: Page 38. Correspondence: `diffusiongemma-external@google.com`.

[2] The Gemma 4 E4B and E2B models are designed primarily for mobile applications and consequently are slower than expected on GPU.

[3] FP8 performance numbers are not available for Nemotron and LLaDA. FP8 quantization provides an upper-bound speedup of 2×, with typical gains closer to 1.5× depending on model architecture.

Figure 2 | **Overview of our two-stage training pipeline that converts an autoregressive model (Gemma 4 26B A4B) into a text diffusion model (DiffusionGemma).** Initialized from the AR model weights, the model first undergoes SFT that adapts it to discrete text diffusion and bidirectional attention across 256-token canvases. This is followed by an online sampler distillation and reinforcement learning phase, which jointly maximizes reward-driven generation quality and compresses the denoising steps to unlock ultra-low latency.

1. Introduction

Autoregressive (AR) models dominate the current Large Language Model (LLM) landscape, but their strict left-to-right, token-by-token generation creates a problematic memory bottleneck. When serving many requests simultaneously, one can achieve acceptable throughput by batching, but serving single or low-concurrency requests is fundamentally memory-bound: time spent transferring model weights and context KV cache from memory to the accelerator far exceeds time spent on actual computation. This leaves the accelerator's compute units under-utilized and limits per-user generation speed. Speculative decoding can improve utilization by drafting a candidate sequence, typically 8 tokens, from a small "drafter" model and feeding the draft to the LLM for verification (Chen et al., 2023a; Leviathan et al., 2023; Xia et al., 2024). With a draft length of 8 this can produce 3-6 tokens per forward pass (TPF) (Li et al., 2026). However, the draft-then-verify paradigm is still limited: on one hand, AR drafters are bottlenecked by sequential generation; on the other hand, parallel drafters—building on early blockwise decoding (Stern et al., 2018) and lookahead strategies (Zhao et al., 2024)—exhibit declining acceptance rates at later draft positions (Cheng et al., 2026).

Text diffusion circumvents this bottleneck by predicting entire blocks of tokens simultaneously (Figure 22), effectively shifting execution from a memory-bound regime toward a compute-bound one. Recently, there has been a surge of interest in text diffusion (Deschenaux and Gulcehre, 2024), with models like Gemini Diffusion (Google DeepMind, 2025), Mercury (Inception Labs et al., 2025), LLaDA (Bie et al., 2025; Nie et al., 2026), Seed Diffusion (Song et al., 2025), and Nemotron-Labs-Diffusion (Fu et al., 2026). However, the current landscape forces a stark compromise between speed, intelligence, and accessibility. Some models, e.g., Gemini Diffusion and Mercury, are locked behind proprietary APIs. On the other hand, existing open-weights alternatives either exhibit limited reasoning capabilities and multimodal understanding, or fail to deliver on the extreme latency benefits promised by the diffusion technology, or both. Until now, there has been no text diffusion model that is highly intelligent, exceptionally fast, and openly accessible.

We introduce DiffusionGemma to bridge this gap. A finetuned text diffusion variant of the Gemma 4 26B A4B mixture-of-experts (MoE) model (Gemma Team et al., 2026), the model establishes a new Pareto frontier in the intelligence-to-speed trade-off for generative text modeling (Figure 1). Notably, it extends the speed frontier beyond both prior text diffusion models and the Gemma 4 AR family across all parameter scales (from E2B up to 31B), even when the AR models are equipped with multi-token prediction (MTP, DeepSeek-AI, 2024; Google DeepMind, 2026), a state-of-the-art speculative decoding technique. To unlock this new speed-to-intelligence frontier, DiffusionGemma maximizes algorithmic efficiency by outputting blocks of 256 tokens simultaneously in around 12 forward passes.

Total	25.2B
Activated	3.85B
Vision Encoder	550M
Embedder	740M
Self-Conditioning	7.8M
Active / Total Experts	8 / 128 + 1 shared

Table 1 | **Parameter counts.** DiffusionGemma's architecture is a mixture-of-experts transformer with a vocabulary of 262k tokens. The total number of activated parameters does not include the vision encoder. It includes an additional MLP block for the purpose of self-conditioning.

Canvas Length	256
Sampler Maximum Denoising Steps	48
Adaptive Stopping Entropy Threshold	0.005
Token Selection Entropy Threshold	0.1
Temperature Schedule (Linear)	0.8 → 0.4

Table 2 | **Diffusion denoising and recommended sampler hyperparameters.** In practice, the average sampler denoising steps is 12 with adaptive stopping, depending on the task.

Put differently, DiffusionGemma generates on average 20 TPF—a step-change improvement over the ~3-6 TPF that can be achieved by state-of-the-art speculative decoding methods. This massive reduction in total forward passes directly offsets the computational cost of the diffusion model's heavier individual forward passes and as a result, DiffusionGemma yields generation speeds of around 1,500 tokens per second (TPS) on a single NVIDIA H100 GPU. Furthermore, early benchmarking by third-party inference providers such as Unsloth (2026) demonstrates these speeds can scale up to 2,000 TPS on the NVIDIA RTX 6000.

To avoid the prohibitive computational cost of pretraining it is common practice to initialize text diffusion models from pretrained AR models (Gong et al., 2025; Han et al., 2024). We warm-start DiffusionGemma from the final post-trained, publicly released weights of the Gemma 4 26B A4B MoE model (Gemma Team et al., 2026). By adopting the same transformer backbone, we efficiently repurpose the AR weights to support both causal attention for context encoding and bidirectional attention for the diffusion process while inheriting their full capabilities. We employ a two-stage training pipeline (see Figure 2) that utilizes less than 10% of the AR model's total training tokens:

1. **Supervised fine-tuning (SFT)**: We first run an SFT phase to adapt the model to attend to a context of clean tokens as well as denoising a block of 256 noisy tokens (as dictated by the diffusion process) with bidirectional attention across the block.
2. **Sampler distillation and reinforcement learning (SD·RL)**: We then apply an online learning phase that simultaneously improves generation quality (by maximizing rewards) and unlocks ultra-low latency (by substantially reducing the number of forward passes).

By preserving features of its AR starting point, DiffusionGemma exhibits strong multimodal understanding, long-context capabilities, and thinking mode, enabling it to generate reasoning traces prior to responding. These are hallmarks of current frontier systems (Gemini Team, 2025). Although adapting the model to the diffusion regime introduces a performance penalty compared to the original AR baseline, the massive gains in inference speed offer an entirely new operating point for latency-critical applications. DiffusionGemma retains the ability to generate text autoregressively. This *dual mode* opens up the possibility to route requests dynamically based on latency constraints and task complexity, as well as to employ hybrid decoding approaches.

Open-weights release. We release DiffusionGemma with open weights under a permissive Apache 2.0 license, aiming to democratize access to state-of-the-art text diffusion technology. By providing full, unrestricted access to the model parameters, we hope to empower a diverse ecosystem of researchers, developers, and practitioners. For the research community, this transparent release provides a robust, white-box baseline to deeply probe the underlying mechanics of discrete diffusion, accelerating foundational research and pushing the theoretical frontier of text generation. For developers, the liberal licensing removes friction, ensuring seamless integration into both experimental prototypes and commercial applications without restrictive usage barriers.

Crucially, DiffusionGemma's highly efficient architecture makes it an ideal foundation for rapid, domain-specific adaptation. By dramatically lowering the computational barrier to entry, the community can easily finetune ultra-fast, task-specific models tailored to their own unique use cases, even in environments with constrained compute budgets. This capacity for lightweight, rapid iteration has catalyzed immediate community adoption. Despite the model being available for only a few weeks at the time of writing, it is already serving as the engine for specialized downstream applications. These rapidly emerging use cases showcase the model's versatility across highly diverse and demanding domains, spanning from multilingual automatic speech recognition (Interfaze, 2026) to interactive radiology report drafting in healthcare (Van Puyvelde et al., 2026).

Outline. The remainder of this technical report is organized as follows. Section 2 formalizes the discrete diffusion framework and the theoretical foundations of our approach. Section 3 details the DiffusionGemma architecture, including the bidirectional decoding mechanism and sampling algorithm. We then describe the two stages of our training pipeline: the SFT phase in Section 4, followed by our online learning phase combining sampler distillation and reinforcement learning in Section 5. Section 6 breaks down our low-level inference optimizations. Section 7 presents experimental results. Section 8 provides instructions and a practical example of finetuning DiffusionGemma for downstream applications. In Section 9, we showcase some practical advantages of text diffusion. Finally, we discuss limitations and known issues in Section 10, before concluding in Section 11.

2. Generative Text Modeling with Discrete Diffusion

Historically, LLMs have treated text generation as a strictly sequential process (Bengio et al., 2003; Graves, 2013; Gu et al., 2018; Mikolov et al., 2010; Sutskever et al., 2014; Vaswani et al., 2017). Classical AR language modeling factorizes the joint probability of a sequence x of length L into an exact product of conditional probabilities: $p(x) = \prod_{i=1}^{L} p(x^i \mid x^{<i})$, where i indicates the token position in the sequence. While theoretically rigorous, this single-token factorization fundamentally limits generation speed on modern hardware accelerators due to severe memory-bandwidth constraints as well as strictly preventing the model from bidirectionally revising tokens based on future tokens (Leviathan et al., 2023; Savinov et al., 2022).

Diffusion models approximate the joint distribution $p(x)$ by factorizing the generative process over a sequence of noise levels (a Markov chain) rather than strictly left-to-right spatial positions (Ho et al., 2020; Holderrieth and Erives, 2025; Sohl-Dickstein et al., 2015; Song et al., 2021c), building upon foundational work in score-matching and continuous-time flow models (De Bortoli et al., 2021; Lipman et al., 2023; Meng et al., 2022; Song et al., 2021a; Song and Ermon, 2019; Song et al., 2021b). During training, a forward process gradually corrupts clean data into random noise; a neural network is trained to learn the reverse denoising process. During inference, the model generates full sequences of data by iteratively refining the output sequence in parallel, starting from random noise.

Adapting diffusion to text requires handling the discrete nature of language (Yi et al., 2024). Early

approaches adapt standard Gaussian diffusion to text by mapping discrete tokens into a continuous embedding space. These models differ primarily in how they map back to text: architectures like Diffusion-LM (Li et al., 2022) and SED (Strudel et al., 2022) produce continuous vectors that have to be forcibly projected or "rounded" back to discrete tokens, whereas methods like CDCD (Dieleman et al., 2022) maintain continuous processes but predict categorical logits directly to sample tokens. Regardless of the decoding mechanism, these formulations face theoretical and geometric challenges that ultimately limit their effectiveness compared to native discrete diffusion models. In particular, hard rounding operations fundamentally break exactness of diffusion likelihood bounds established by continuous-time and variational formulations (Kingma et al., 2021) and the valid vocabulary tokens occupy an infinitesimally small fraction of a high-dimensional latent space. As a result, the reverse generative process often drifts into empty, "meaningless" regions of the space; when rounded to nearest token, these degenerate embeddings map to arbitrary, unrelated tokens and produce incoherent text (Meshchaninov et al., 2025; Shabalin et al., 2025). To combat this spatial drift, methods have been developed to rely on per-step discretization mechanisms, but these generally underutilize the continuous space and restrict generative flexibility (Hu et al., 2026).

Discrete diffusion has emerged as a highly effective alternative (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; Zhao et al., 2025). This paradigm generalizes the single-step corruption heuristics of early bidirectional models into formal, multi-step Markov processes. The probabilistic transitions defined between categorical states in modern diffusion, such as absorbing mask states and multinomial distributions across the vocabulary, are direct descendants of masking and random token swapping introduced by BERT (Devlin et al., 2019; Liu et al., 2019; Sanh et al., 2019), as well as the plausible token replacements of ELECTRA (Clark et al., 2020). By operating directly on discrete states rather than continuous vectors, discrete diffusion avoids embedding projection mismatches entirely and offers better theoretical grounding for categorical transitions (Gat et al., 2024). DiffusionGemma builds upon this lineage of discrete diffusion to ensure high-fidelity token generation.

2.1. Probability Paths and Denoising

To formalize our discrete diffusion framework, we adopt the continuous-time Markov chain (CTMC) approach established in recent literature (Campbell et al., 2022, 2024; Gat et al., 2024). Let $\mathcal{V}$ define a categorical token vocabulary of size V. We refer to the sequence of tokens undergoing iterative refinement as the *canvas*, with a length of C. A realization of this canvas at time $t \in [0, 1]$ is denoted by the vector $x_t \in \mathcal{V}^C$. We construct a marginal probability path to smoothly interpolate between a clean data distribution p at time $t = 0$ and a fully corrupted source canvas at time $t = 1$. At the start of this path, the clean tokens are defined as $x_0 \sim p(\cdot)$, which ultimately transition into the fully corrupted state $x_1 \sim \text{Unif}(\mathcal{V}^C)$. Here, $\text{Unif}(\mathcal{V}^C)$ denotes the uniform prior over the joint state space, where each of the C tokens is independently uniformly distributed over $\mathcal{V}$. This corruption process parallels the explicit transition matrices—such as uniform noise or absorbing masking—commonly utilized in order-agnostic discrete diffusion models (Hoogeboom et al., 2022). By operating in continuous time, this framework bypasses the sequential, left-to-right decoding bottleneck of traditional AR models, allowing tokens across the entire canvas to be processed and refined in parallel.

2.1.1. The Forward Process

The forward process dictates the transition from clean text tokens to uniformly distributed tokens. Conditioned on a fixed clean starting canvas x_0, the forward transition probability path factorizes

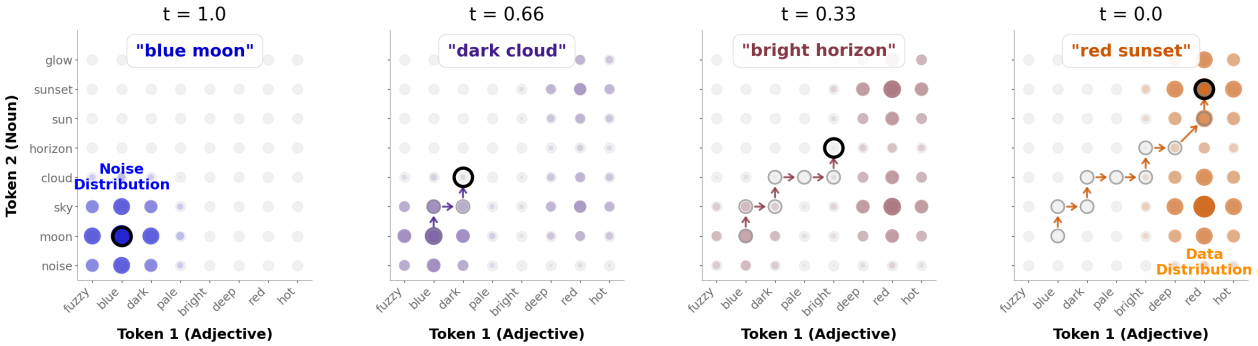

Figure 3 | **Stylized example of discrete diffusion probability paths and parallel sampling trajectory.** For illustrative purposes, the state space uses distinct, token-specific vocabularies: adjectives on the horizontal axis and nouns on the vertical axis. As time moves backward from $t = 1.0$ (noise) to $t = 0.0$ (data), the marginal distribution smoothly interpolates, concentrating mass away from the uniform noise distribution and onto valid data modes. Black circles track the discrete jump transitions of an individual sequence realization (from "blue moon" at $t = 1.0$ to "red sunset" at $t = 0.0$), demonstrating how parallel canvas dimensions coordinate non-autoregressively over time.

independently over each token coordinate i:

$$\mathbb{P}(X_t = x_t \mid X_0 = x_0) = \prod_{i=1}^{C} \left[\kappa_t \delta(x_t^i, x_0^i) + (1 - \kappa_t) \frac{1}{V} \right], \tag{1}$$

where $\kappa_t \in [0, 1]$ is a smoothly varying, monotonically decreasing noise schedule from $\kappa_0 = 1$ to $\kappa_1 = 0$, and δ represents the Kronecker delta. In practical terms, as time t progresses toward 1, each token is increasingly likely to be replaced by a token sampled uniformly at random from the vocabulary (Austin et al., 2021; Hoogeboom et al., 2021). We use this closed-form forward process to generate training data from clean text.

2.1.2. The Backward Denoising Process

To reconstruct data from noise, we learn a reverse process to undo this categorical corruption. Discrete flow matching theory shows that to perfectly reverse the forward trajectory, we need to infer the conditional distribution of the original uncorrupted tokens given a corrupted state (Campbell et al., 2024; Gat et al., 2024; Ou et al., 2025). For a given realization $X_t = x_t$, stepping backward in time by a small increment Δt is governed by some transition mapping, which we denote Step, which outputs the probability distribution for the next intermediate step[1]:

$$\mathbb{P}(X_{t-\Delta t} = \cdot \mid X_t = x_t) \approx \text{Step}\left(x_t, \mathbb{P}(X_0 = \cdot \mid X_t = x_t)\right). \tag{2}$$

To compute this update, we need the true posterior distribution of clean tokens given a noisy state, $\mathbb{P}(X_0^i = v \mid X_t = x_t)$. We approximate this posterior with a neural network $p_\theta(v \mid x_t)$. During generation, we use this approximation to sample the next token state: $x_{t-\Delta t} \sim \text{Step}(x_t, p_\theta(\cdot \mid x_t))$.

Figure 3 presents a highly stylized example to illustrate this reverse sampling trajectory within a simplified two-token canvas ($C = 2$). As time runs backward from $t = 1.0$ to $t = 0.0$, the model smoothly shifts probability mass away from a uniform noise distribution (e.g., around "blue moon")

[1]As a typical example, the function Step can be instantiated as a discrete Euler update step introduced in Gat et al. (2024), in which $\mathbb{P}(X_{t-\Delta t}^i = v \mid X_t = x_t) \approx \delta(v, x_t^i) - \Delta t \frac{\dot{\kappa}_t}{1-\kappa_t} \left[\mathbb{P}(X_0^i = v \mid X_t = x_t) - \delta(v, x_t^i)\right]$ with $\kappa_t \in [0, 1]$.

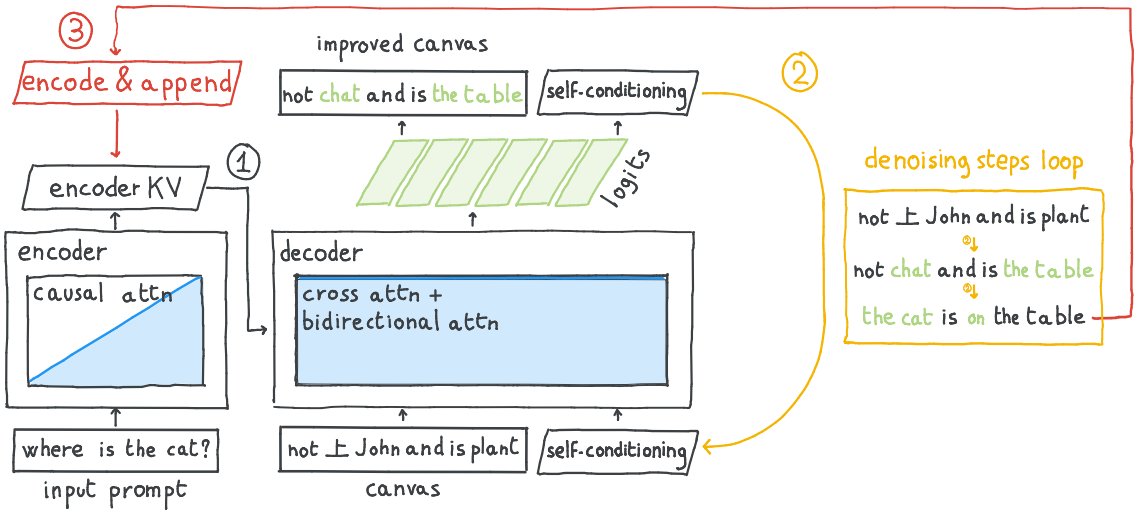

Figure 4 | **The DiffusionGemma generation pipeline.** The process consists of three main stages: 1) **Context encoding:** The input prompt is processed by the causal encoder to initialize the Key-Value (KV) cache. 2) **Denoising loop:** A noisy canvas is iteratively refined by the decoder, using bidirectional attention across the canvas and cross-attention to the KV cache, until the text is fully denoised. 3) **Encode & append:** The finalized clean canvas is passed back through the causal encoder and appended to the KV cache, setting the context for the next block of tokens.

toward valid data modes. Concurrently, individual sequence coordinates undergo continuous-time jump transitions—visualized by the step-by-step path from "blue moon" at $t = 1.0$, to "dark cloud" at $t = 0.66$, and finally landing on a high-probability mode like "red sunset" at $t = 0.0$. This highlights how parallel dimensions coordinate over time without requiring sequential left-to-right generation.

Because this denoising process is factorized, it assumes conditional independence during individual reverse steps. A newly updated token at position i is conditioned globally on the current noisy canvas x_t, but it cannot see the simultaneous sampling choices made at other positions ($j \neq i$). This structural tradeoff can occasionally introduce local inconsistencies or conflicting grammatical predictions. In the next section, we counteract these uncoordinated mechanics directly through our *self-conditioning architecture* and *entropy-bounded sampling* strategies.

3. The DiffusionGemma Architecture

The DiffusionGemma architecture functions as an encoder-decoder transformer (Vaswani et al., 2017) with *shared weights* θ. Rather than pretraining a diffusion model from scratch, we initialize our model with the publicly released Gemma 4 26B A4B MoE checkpoint (Gemma Team et al., 2026). Tying our architecture to an existing AR backbone is a pragmatic choice that trades off absolute generation quality for a substantial reduction in the computational cost of training. This initialization also allows us to inherit the base model's advanced features—such as its extended context window and native multimodal understanding. Moreover, our experiments demonstrate that our final model weights retain the ability to be sampled from autoregressively (see Table 3).

The remainder of this section details how DiffusionGemma uses this architecture to generate text at inference time. We break this process down into three components: the block-autoregressive decoding strategy for handling long sequences (Section 3.1), the general framework for denoising individual canvases (Section 3.2), and a specific entropy-bounded sampler (Section 3.3).

3.1. Block-Autoregressive Generation

The discrete flow matching formulation introduced in Section 2 operates on fixed-length sequences. To generate open-ended text, we use a block-AR generation strategy. The model denoises a canvas of 256 tokens at a time, and once a canvas is fully denoised, it is committed to the sequence history, and the model begins denoising the next canvas.

KV cache initialization. As illustrated in Figure 4, the first step in block-AR generation is to encode the context, $h \in \mathcal{V}^L$, into a KV cache H. The context includes system instructions and user prompts of maximum token count L.[2] We write the encoding as

$$H = \text{Encoder}_\theta(h), \tag{3}$$

where Encoder_θ is the transformer forward pass with weights θ and causal attention masking.

Canvas denoising conditioned on KV cache. By cross-attending to the KV cache H, canvas generation can be conditioned on system instruction, user inputs and past responses. Canvas generation starts from a canvas of uniformly random tokens and is iteratively denoised using the process described in Sections 3.2 and 3.3. Once a canvas is fully denoised,[3] denoted by $\hat{x}_0$, its keys and values are appended to the KV cache (depicted in red in Figure 4):

$$H \leftarrow H \oplus \text{Encoder}_\theta(\hat{x}_0). \tag{4}$$

We repeat this denoising process for subsequent canvases until the model generates a special token marking the end of the model's turn, after which all canvases $\hat{x}_0$ are concatenated into a final response. Since we use causal attention in the encoder, it lets the encoder update the KV cache by appending only the newest canvas. This constitutes an architectural inversion of standard encoder-decoder models such as BART (Lewis et al., 2020) or T5 (Raffel et al., 2020). Whereas those models utilize a bidirectional encoder for context and a causal decoder for generation, our approach utilizes a causal encoder for the sequence history and a bidirectional decoder for the diffusion-based canvas generation. This causal encoding prevents early context tokens from attending to the latest canvas, but it eliminates the need to re-encode the growing context from scratch, making our approach scalable to long reasoning generations and yielding a structural blend of diffusion and AR generation. This block-AR strategy was developed by our group in an unpublished June 2023 manuscript; analogous approaches to blockwise sequence modeling and KV caching were independently developed by Wu et al. (2025), Arriola et al. (2025), and Deschenaux and Gulcehre (2026a).

3.2. The Denoising Framework at Inference

Canvas initialization. To generate a canvas, we iteratively refine it over a maximum of N denoising steps, with step size $\Delta t = 1/N$. Let $x_t \in \mathcal{V}^C$ denote the corrupted canvas at step t. We initialize the process at $t = 1$ with a canvas x_1 of uniformly random tokens from the vocabulary $\mathcal{V}$.

The decoder forward pass. At each step t, we apply the transformer with the shared weights θ as a decoder, written as Decoder_θ, to predict the probability distribution of the clean tokens. The decoder takes three inputs: the current noisy canvas x_t, the context KV cache H, and a continuous *self-conditioning* signal $z_t \in \mathbb{R}^{C \times d}$ that feeds the model's previous predictions back into itself (Chen

[2] In practice, h also includes interleaved multimedia embeddings which are not part of the token vocabulary.
[3] A canvas is fully denoised either by reaching $t = 0$ or by the adaptive stopping condition described in Section 3.3.

et al., 2023c; Jo et al., 2026; Strudel et al., 2022). Using bidirectional attention across the canvas tokens and cross-attention to the KV cache, the decoder outputs the unnormalized logits L_t:

$$L_t = \text{Decoder}_\theta(x_t, z_t, H) \in \mathbb{R}^{C \times V}. \tag{5}$$

The denoising iteration. At each iteration, we compute the logits L_t and evaluate the clean token probabilities $\hat{p}_0$, update the self-conditioning signal for the next step, and sample the refined canvas:

$$\begin{aligned} \hat{p}_0 &= \text{Softmax}\left(L_t/\tau_t\right) \in [0,1]^{C \times V}, \\ z_{t-\Delta t} &= \text{FFW}(\hat{p}_0 E) \in \mathbb{R}^{C \times d}, \\ x_{t-\Delta t} &\sim \text{Step}(x_t, \hat{p}_0). \end{aligned} \tag{6}$$

Here, $E \in \mathbb{R}^{V \times d}$ is the token embedding matrix and FFW is a standard feedforward network. The time-dependent temperature $\tau_t > 0$ can sharpen the model's predictions before we calculate the final transition probabilities. A transition mapping Step, as described in Equation (2), computes the categorical distribution used to sample the updated canvas $x_{t-\Delta t} \in \mathcal{V}^C$, which, along with the new self-conditioning signal $z_{t-\Delta t}$, is then fed into the next denoising step. Section 3.3 details our specific choices for transition mapping Step and the temperature τ_t.

Multinomial diffusion. Our denoising iteration (Equation 6) uses multinomial (or uniform) diffusion rather than masked diffusion (Austin et al., 2021; Hoogeboom et al., 2021). Because all tokens can transition between one another, the model can continuously correct its own errors: tokens accepted during earlier denoising steps ($t' > t$) within the current canvas can still be revised. However, we note that tokens from previously generated canvases are permanently frozen.

Remark on interpretability. Recent interpretability analyses demonstrate that while the self-conditioning signal z_t introduces a continuous latent space injection into the sequence, these intermediate self-conditioning vectors map robustly to an interpretable token bottleneck, preserving the model's algorithmic transparency (Asaria et al., 2026; Engels et al., 2026).

3.3. The DiffusionGemma Sampler

In contrast to AR generation, which relies on rigid heuristics like temperature, top-p, and top-k, diffusion modeling introduces a vastly richer sampling design space that includes discrete predictor-corrector mechanisms and multi-step solvers (Deschenaux et al., 2026; Lezama et al., 2023; Liu et al., 2026a; Ren et al., 2025; Yao et al., 2026). Framing text generation as an iterative temporal process allows us to decouple the inference-time algorithm from the underlying architecture, granting fine-grained control over the trade-off between computational cost and generation quality.

While we described the overall approach of canvas denoising in Section 3.2, this subsection details a specific instantiation: the entropy-bounded sampler (Ben-Hamu et al., 2026) with temperature annealing and adaptive stopping (Algorithm 1). While this is our default and recommended sampler, DiffusionGemma is modular and is not strictly bound to this exact sampling configuration for high-quality inference.

Entropy-bounded token refinement. After producing the marginal distributions $p_\theta(x_0 \mid x_t, z_t, H)$ with the denoiser and applying temperature scaling (detailed below), tokens are sampled from the resulting probability distribution. We employ an entropy-bounded sampler (Ben-Hamu et al., 2026),

Algorithm 1 DiffusionGemma Sampling with Entropy-Bounded Denoising and Adaptive Stopping

Require: Context h; maximum number of denoising steps $N = 48$
Require: Entropy budget $b = 0.1$; stopping threshold $e_{\text{stop}} = 0.005$
Require: Temperature-schedule endpoints $\tau_{\max} = 0.8$ and $\tau_{\min} = 0.4$

Initialization

$H \leftarrow \text{Encoder}_\theta(h)$ ▷ Causal-attention KV cache
$\Delta t \leftarrow 1/N$
$x_1 \sim \text{Unif}(\mathcal{V}^C), \quad z_1 \leftarrow \mathbf{0}$
$\hat{x}_0^{\text{prev}} \leftarrow \perp$

Denoising

for $n = 1, \ldots, N$ **do**
  $t \leftarrow 1 - (n-1)\Delta t$
  $L_t \leftarrow \text{Decoder}_\theta(x_t, z_t, H), \quad L_t \in \mathbb{R}^{C \times V}$
  $\hat{x}_{0|t} \leftarrow \arg\max(L_t)$ ▷ Current deterministic prediction
  $\tau_t \leftarrow (\tau_{\max} - \tau_{\min})t + \tau_{\min}$
  $\hat{p}_0 \leftarrow \text{Softmax}(L_t/\tau_t)$
  $e^i \leftarrow \text{Entropy}(\hat{p}_0^i)$ for all $i \in \{1, \ldots, C\}$
  $\bar{e} \leftarrow \dfrac{1}{C}\sum_{i=1}^{C} e^i$
  if $n > 1 \wedge \bar{e} \le e_{\text{stop}} \wedge \hat{x}_{0|t} = \hat{x}_0^{\text{prev}}$ **then**
    return $\hat{x}_{0|t}$ ▷ Adaptive stopping
  end if
  $\hat{x}_0^{\text{prev}} \leftarrow \hat{x}_{0|t}$
  if $n < N$ **then**
    $z_{t-\Delta t} \leftarrow \text{FFW}(\hat{p}_0 E)$ ▷ E is the embedding matrix
    $x_D \sim \hat{p}_0$
    Let $(\pi_1, \ldots, \pi_C)$ order positions by non-decreasing entropy ▷ Break ties by position index
    $e^{\pi_1} \le e^{\pi_2} \le \cdots \le e^{\pi_C}$
    $k \leftarrow \max\left\{ m \in \{1, \ldots, C\} : \sum_{j=1}^{m-1} e^{\pi_j} \le b \right\}$
    $U \leftarrow \{\pi_1, \ldots, \pi_k\}$
    For each $i \in \{1, \ldots, C\}$, set
    $$x_{t-\Delta t}^i \leftarrow \begin{cases} x_D^i, & i \in U, \\ \tilde{x}^i, \quad \tilde{x}^i \sim \text{Unif}(\mathcal{V}), & i \notin U. \end{cases}$$
  end if
end for
return $\hat{x}_{0|t}$

whereby tokens are accepted in rank order from lowest to highest entropy (similar to the MaskGIT decoding scheme by Chang et al. (2022)), ensuring that their mutual information bound remains strictly below a predefined error tolerance threshold ($b = 0.1$). Once the threshold is attained, all other tokens are renoised uniformly at random, maintaining these uncommitted positions as a uniform prior to force local exploration during the next forward pass.

Temperature annealing. To balance rate of convergence and linguistic diversity, token probabilities are artificially sharpened via tempering. A temperature $\tau_t < 1$ is annealed linearly from an initial value of $\tau_{\max} = 0.8$ down to $\tau_{\min} = 0.4$ across the fractional denoising timescale $t \in [0, 1]$. While static temperature adjustments are traditionally used to truncate the unreliable tail of token distributions and prevent text degeneration, this dynamic annealing ensures that the model explores diverse token possibilities in early, highly-noised states and aggressively commits to high-confidence sequences as the semantic structure crystallizes (Zhang et al., 2024).

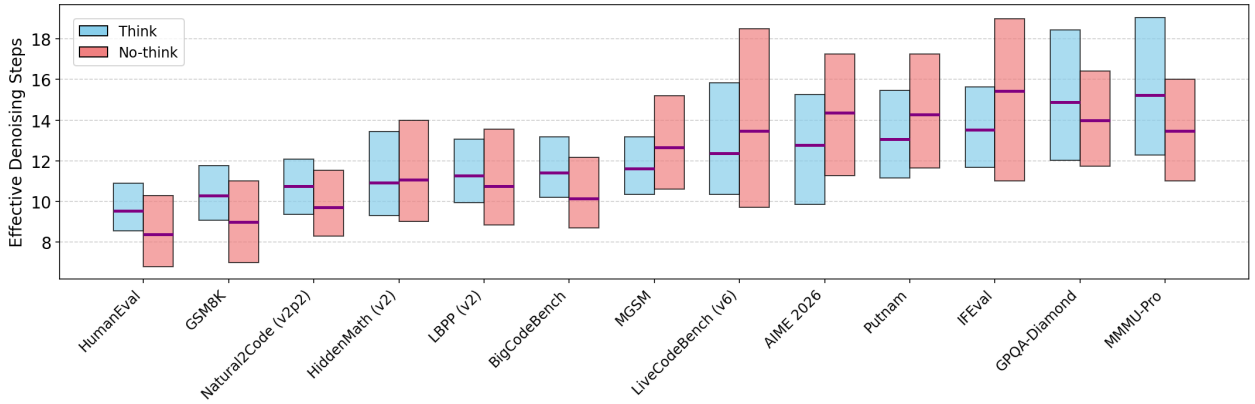

Figure 5 | **Adaptive stopping enables DiffusionGemma to dynamically adjust its number of denoising steps to task complexity and domain.** We report median, first and third quartiles of the effective denoising steps (Equation 9). See Table 4 for full benchmarks and latency metrics.

Adaptive stopping heuristic. To optimize inference efficiency, the sampler dynamically halts the denoising process based on the model's step-wise uncertainty, strictly capping iterations at N. This early termination is triggered when two conditions are simultaneously satisfied:

- *Confident predictions:* The mean predictive entropy across the entire canvas falls below a predefined threshold ($e_{\text{stop}} = 0.005$).
- *Stable predictions:* The deterministic sequence predictions (i.e., the most likely tokens) from two consecutive denoising steps are identical.

By successfully bypassing redundant refinement steps, this mechanism allows the model to dynamically scale its inference-time compute to the complexity of the prompt. As illustrated in Figure 5, instead of exhausting the maximum $N = 48$ budget, the model averages approximately 12 effective denoising steps (defined in Section 3.4) across the downstream evaluations shown in the figure. This yields a 4× reduction in overall latency without sacrificing generation quality. Furthermore, the boxplots reveal distinct convergence profiles across domains: the model tends to deploy fewer steps for structured tasks like code and more for natural language. Moreover, it is not only the domain, but also the task complexity that dictates this behavior; for instance, harder code problems (e.g., LiveCodeBench) naturally require more steps to converge than easier ones (e.g., HumanEval).

3.4. Inference Efficiency Metrics

Because adaptive stopping turns the number of denoising steps into a random variable, we introduce a set of metrics to quantify denoising inference efficiency. Let K be the total number of canvases generated, $N_k \leq N$ be the number of denoising steps executed for the k-th canvas, and $C_k \leq C$ be the number of valid tokens produced in that canvas (i.e., all C tokens, or up to the first end-of-sequence token if one is generated). First, we define the Total Tokens as the sum of all valid tokens generated across the entire sequence:

$$\text{Total Tokens} \triangleq \sum_{k=1}^{K} C_k. \tag{7}$$

We define the **Total Denoising Steps** as the absolute number of denoising steps executed across the entire generation:

$$\text{Total Denoising Steps} \triangleq \sum_{k=1}^{K} N_k. \tag{8}$$

Since each denoising step incurs a fixed computational cost, the Total Denoising Steps is the primary driver of end-to-end generation latency. However, because this absolute metric inherently scales with the total sequence length, we also evaluate a normalized measure of model efficiency. We define the **Effective Denoising Steps** as the token-weighted average of steps across all canvases:

$$\text{Effective Denoising Steps} \triangleq \frac{1}{\text{Total Tokens}} \sum_{k=1}^{K} N_k C_k. \tag{9}$$

We weight by the number of valid tokens to prevent the metric from being downwardly biased by the final canvas, which tends to require fewer denoising steps when it is only partially filled. Note that without adaptive stopping, the Effective Denoising Steps metric always reduces to the denoising budget N. Finally, we calculate the **Tokens Per Forward (TPF)**:

$$\text{TPF} \triangleq \frac{\text{Total Tokens}}{\text{Total Denoising Steps} + K - 1}, \tag{10}$$

where the $K - 1$ term accounts for the single additional forward pass required between canvases to encode the newly generated clean tokens and append their key-value pairs to the KV cache.

3.5. Retained Autoregressive Capability

Because DiffusionGemma shares the exact same transformer architecture as Gemma 4, the final DiffusionGemma weights can be seamlessly loaded back into the original architecture to perform standard AR generation using causal attention, exactly as the base model does. As demonstrated in Section 7, Table 3, the model maintains robust capabilities in this setting; its performance scores in AR mode land squarely between those of DiffusionGemma's primary text diffusion mode and the baseline Gemma 4 checkpoint from which DiffusionGemma was initialized.

4. Supervised Finetuning

We start from the publicly released Gemma 4 26B A4B (Gemma Team et al., 2026) checkpoint and run an extended finetuning phase where the model adapts to predicting blocks of 256 tokens from noisy inputs. We use a block-diagonal attention mask, enabling bidirectional attention within each block without allowing the model to condition on other denoising blocks. For a given canvas, the model conditions on the prompt and previous (uncorrupted) tokens via the encoder KV cache. We use discrete *multinomial diffusion* as our corruption process and uniformly sample noisy tokens from the vocabulary (Austin et al., 2021; Hoogeboom et al., 2021). For a given canvas, we sample a noise level $t \sim \text{Unif}[0, 1]$ and noise each token in the canvas with probability t. Given a clean context of prompt and previous canvases H (encoded through the KV cache), a self-conditioning signal z_t, and a noisy canvas x_t, the model is trained to minimize the cross-entropy loss between its predictions and the ground-truth canvas extracted from the training data (Gat et al., 2024):

$$L(\theta) = -\sum_{i=1}^{C} \log p_\theta(x_0^i \mid x_t, z_t, H), \tag{11}$$

where superscript i denotes indexing along the canvas dimension; p_θ is parameterized via a softmax transformation over the neural network's output logits. As shown in Figures 6 and 7, denoising performance improves rapidly within the initial steps of training, after which it settles into a log-linear performance improvement trend. Thinking performance benefits from extended SFT, as the model initially struggles with maintaining coherent reasoning traces, often collapsing into stuttering or cycles—a common challenge for internalizing reasoning in language models (Zelikman et al., 2024).

Figure 6 | **Evolution of downstream performance during SFT.** Prior to SFT, the model is incapable of denoising text. Only a moderate amount of SFT is needed to achieve good performance in non-thinking mode, however extended SFT is crucial for the model to learn thinking behaviour. Results use entropy-bounded sampling with adaptive stopping and a maximum of $N = 192$ denoising steps.

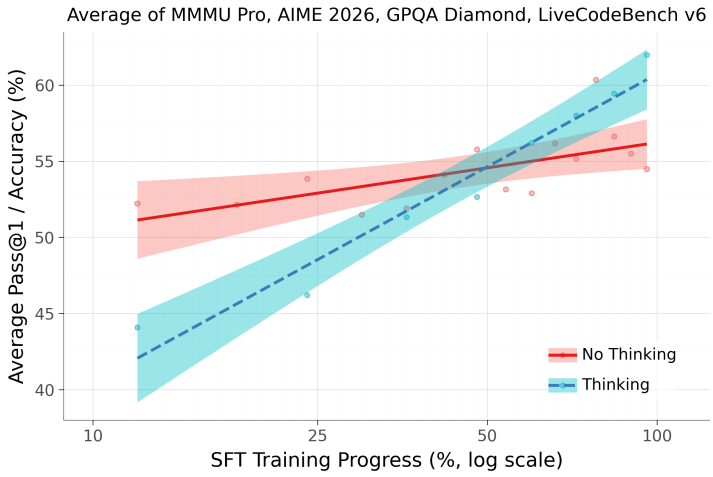

Figure 7 | **Downstream performance during SFT improves log-linearly with training progress.** The log-linear trend for thinking performance starts off at a lower point yet exhibits a steeper slope than non-thinking mode. Results use the EntropyBounded sampler with adaptive stopping and a maximum of $N = 192$ denoising steps.

5. Sampler Distillation & Reinforcement Learning

Following the SFT stage, the model achieves strong generation quality when using a high number of denoising steps. However, its performance on advanced reasoning and coding tasks is somewhat poorer than the baseline AR model. More critically, when operating in the few-step regime required for ultra-low latency inference, generation quality collapses. To address this, we target a dual improvement: pushing the model's intelligence while simultaneously compressing its denoising trajectory.

Traditionally, achieving these two goals requires a decoupled, multi-stage pipeline that treats reward-driven alignment and sampler distillation as distinct phases. We bypass this with a unified *online learning* stage, coined **sampler distillation & reinforcement learning** (SD·RL), which optimizes both axes concurrently. Relying on a joint objective, a single gradient update drives:

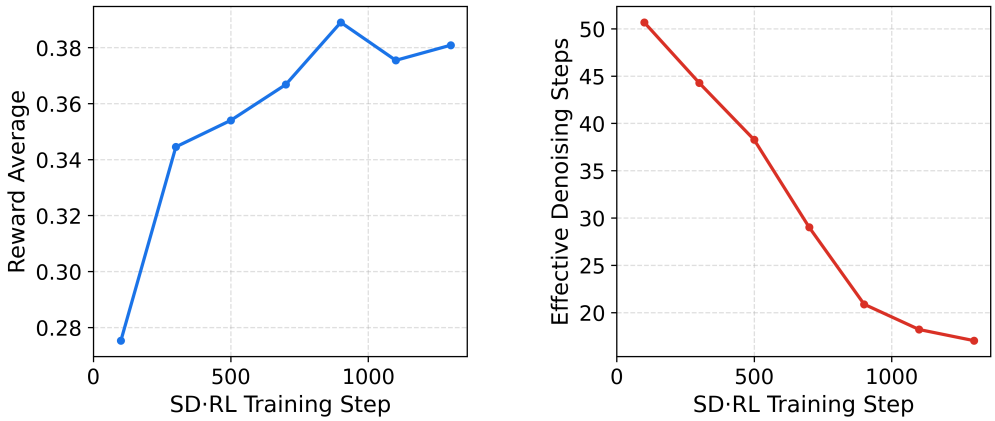

Figure 8 | SD·RL **training simultaneously increases average reward and reduces the effective denoising steps of the online teacher** (training metrics shown in buckets of 200 steps). Both effects together push the quality-speed Pareto frontier of the model.

1. **Reward maximization**: Elevating absolute generation quality and alignment, analogous to standard RL for AR and diffusion models (Black et al., 2024; Clark et al., 2024; Dong et al., 2023; Fan et al., 2023; Lee et al., 2023; Liu et al., 2025; Ma et al., 2026; Wallace et al., 2024; Zhao et al., 2026; Zheng et al., 2026).
2. **Sampler distillation**: Mapping this high-quality generation to the few-step regime, addressing a compression challenge unique to iterative diffusion frameworks (Deschenaux and Gulcehre, 2025; Fu et al., 2025; Hoogeboom et al., 2026; Liu et al., 2024; Luo et al., 2023; Salimans and Ho, 2022; Sauer et al., 2024; Song et al., 2023; Yin et al., 2024).

Training setup. We adapt the data distribution used by the Gemma 4 RL recipe (Gemma Team et al., 2026) for our use case. Encompassing both thinking and non-thinking modes, our setup targets improvements across a variety of capabilities such as helpfulness, mathematical reasoning, coding and instruction-following. Initialized from the SFT weights, the model acts as an online teacher that generates denoising trajectories (using a sampler configured with high maximum denoising steps and mild temperature annealing) to establish a high-quality reference. The SD·RL joint objective uses these trajectories to simultaneously maximize reward and drive sampler distillation in order to compress the model's highest-quality output into the few-step regime.

Training dynamics & implicit curriculum effect. Over the course of SD·RL training, two synergistic dynamics emerge, as shown in Figure 8. First, the online teacher's average reward steadily increases, reflecting improved fundamental capabilities. Second, facilitated by the adaptive stopping mechanism, the online teacher progressively requires fewer effective denoising steps to achieve these high rewards. This acceleration occurs because the SD·RL objective systematically reduces the predictive entropy of the model. Crucially, the interplay between the reward objective and adaptive stopping induces a curriculum learning effect. Early in training, high predictive entropy delays the adaptive stopping trigger. As the model's confidence improves and entropy drops, adaptive stopping triggers earlier. This seamlessly shifts the training distribution toward ever shorter denoising trajectories, allowing the algorithm to dynamically pace its own sampler distillation. Consequently, and in contrast to RL for AR models, prolonging the SD·RL phase remains highly beneficial even after the reward metric plateaus; continued entropy reduction translates directly into further inference speedups.

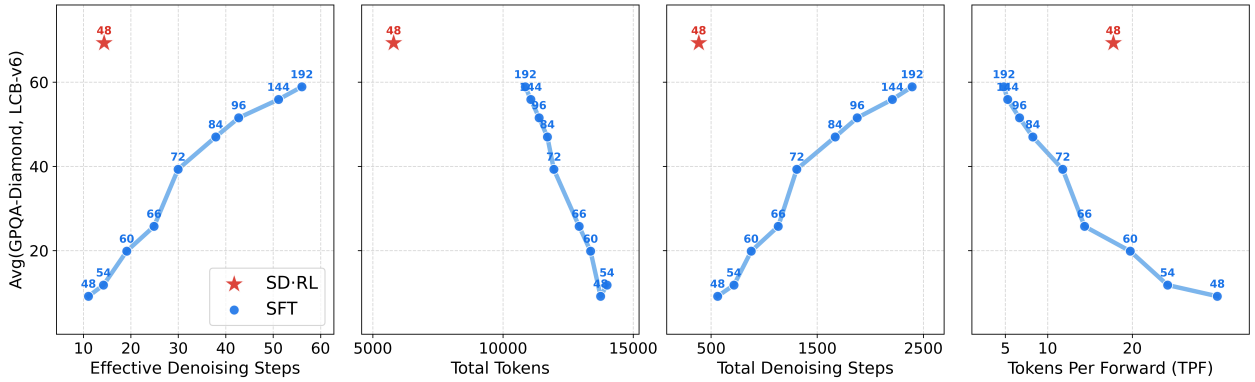

Figure 9 | **SD·RL significantly advances the quality-speed Pareto frontier.** The SD·RL configuration uses the DiffusionGemma sampler with a maximum of $N = 48$ denoising steps. The SFT frontier is derived by sweeping N from 48 to 192 (note that higher N yields milder temperature annealing). Both quality (y-axis) and inference efficiency (x-axes, see Section 3.4) are calculated as the average between GPQA-Diamond and LiveCodeBench-v6 in thinking mode, averaged over 3 seeds.

Improved speed-to-intelligence Pareto frontier. As illustrated in Figure 9, SD·RL training significantly expands the Pareto frontier established by the SFT checkpoint. On the quality axis, it yields a 10-point improvement on the combined GPQA-Diamond and LiveCodeBench-v6 score. On the efficiency axis, it quadruples the TPF from 5 to nearly 20, unlocking ultra-low latency inference.

As shown in Figure 9, evaluating the SFT checkpoint with the default DiffusionGemma sampler (maximum of $N = 48$ denoising steps) results in poor downstream accuracies and artificially low effective denoising steps. This counterintuitive behavior stems from the SFT model frequently degenerating into repetitive token loops in this restricted-step regime. Once it falls into a loop, its predictive entropy collapses, which prematurely triggers the adaptive stopping mechanism. Figures 16 and 17 (Appendix B) provide generated samples on the GPQA-Diamond benchmark that illustrate how these degeneracies manifest: the SFT model begins with a valid, logical reasoning trace but suddenly collapses into a token repetition loop. In contrast, the samples post-SD·RL avoid these degenerative traps to sustain coherent reasoning at minimal latency.

SD·RL specialization in the few-step regime. For the SFT checkpoint, downstream performance scales consistently with the maximum number of denoising steps N up to 192, at which point denoising is roughly any-order autoregressive. Following SD·RL, the scaling behavior improves faster in N and plateaus earlier: performance improves steadily up to $N = 48$, but exhibits diminishing returns thereafter (Figure 10). This early saturation emerges because the SD·RL objective explicitly specializes the model for the few-step regime by aggressively minimizing predictive entropy.

Emergent conciseness. An emergent property of our SD·RL optimization is that it encourages the model to produce concise, token-efficient outputs. This contrasts with RL for AR models, which often maximizes reward by inducing longer reasoning traces. Our final checkpoint produces generations nearly 2× shorter than the SFT checkpoint (Figure 9). While this means the model forgoes some capability gains typically associated with extended reasoning, it acts as a multiplier on inference speed gains. Compounding fewer total tokens with fewer effective denoising steps per canvas directly drives exceptionally low end-to-end latency: DiffusionGemma requires less than 5% of the total forward passes used by the Gemma 4 AR baseline across our eval suite.

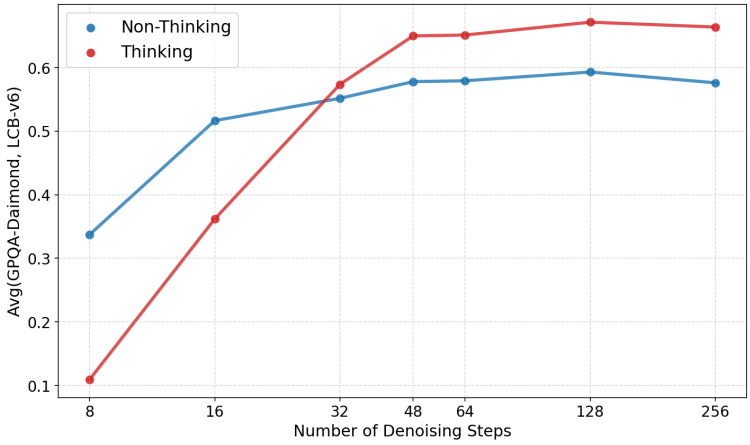

Figure 10 | **Performance vs. number of denoising steps N, without adaptive stopping or temperature annealing.** Performance is measured as the average score over GPQA-Diamond, LiveCodeBench-v6 (3 seeds). To remove confounding effects of early stopping and temperature annealing, we disabled adaptive stopping and use temperature $\tau_t = 1$.

6. Inference Optimizations

The preceding sections focused on maximizing the number of tokens produced per forward pass through our SFT and SD-RL pipeline. We now turn to the complementary axis: minimizing the wall-clock cost of each forward pass through targeted GPU-level optimizations. Fundamentally, the throughput advantage of text diffusion over AR decoding is dictated by two competing quantities: on one hand, each denoising step decodes TPF tokens, and thus requires TPF times fewer forward passes than an AR baseline. On the other hand, because each forward pass processes a canvas of 256 tokens, each such step is r times slower, and so the overall throughput relative to the AR baseline becomes TPF/r. The forward pass overhead can be substantially reduced by hardware optimizations.

Low batch size serving. While text diffusion inference requires more floating-point operations (FLOPs) per generated token than AR models, it relies on significantly fewer forward passes. Because LLM serving on modern hardware accelerators is typically memory-bound—largely dictated by KV cache capacity and memory bandwidth (Kwon et al., 2023; Pope et al., 2023)—this reduction in memory transfers creates a latency advantage that outweighs the higher computational cost. As a result, text diffusion is highly effective in low-batch-size scenarios, leveraging available compute capacity to minimize per-request latency. For simplicity, our analysis focuses on the single-request inference throughput (a batch size of 1) of DiffusionGemma, comparing it directly to its AR counterpart, Gemma 4 26B A4B. Reference inference implementations are available in HuggingFace Transformers (Google DeepMind Team, 2026) and vLLM (The vLLM Team and Google DeepMind Team, 2026).

GPU time breakdown. Figure 11 presents the per-step GPU kernel time breakdown for both models; we focus on GPU kernel time rather than end-to-end latency to isolate model-specific bottlenecks. For each model individually, end-to-end latency exceeds GPU kernel time by approximately 1 ms due to detokenization and other CPU-side serving overhead, with this gap being similar for both the Gemma 4 AR and DiffusionGemma models. Each DiffusionGemma step processes 256× more tokens, yet is only 3.2× slower than the single-token AR step. The time difference can be mainly attributed to three operations: mixture-of-experts (MoE), sampling, and attention. Other operations (e.g., shared expert, attention output projection, etc.) are at most 2× slower.

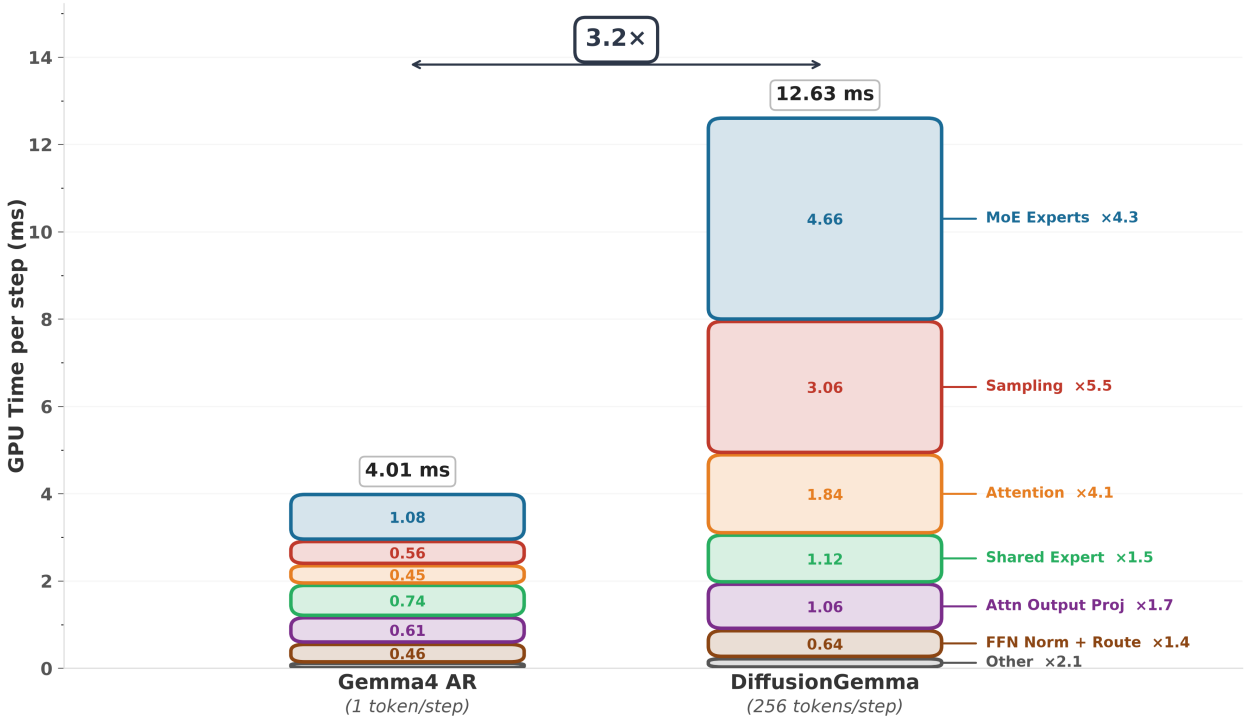

Figure 11 | **Per-step GPU time breakdown: DiffusionGemma processes 256 tokens per step with only a 3.2× increase in per-step latency compared to single-token AR generation.** Serving a single request (batch size 1) on a H100, FP8 precision, with 4096 input tokens, 1024 output tokens.

- **MoE.** When serving a single request, the MoE layer computation is memory-bound, with its running time dominated by the transfer of expert weights from high-bandwidth memory, a well-documented bottleneck when serving sparse MoE models (Huang et al., 2024; Rajbhandari et al., 2022). For the Gemma 4 AR model, only 8 unique experts are activated per token per MoE layer. For the DiffusionGemma model, on average, approximately 84 unique experts are activated per canvas of 256 tokens per MoE layer, as measured on the PG-19 benchmark (Rae et al., 2020).[4] Activating more experts per forward pass results in a 4.3× slower MoE kernel. It is the same type of penalty incurred by verification in speculative decoding, but scaled to larger token parallelism. For a dense architecture, this overhead would be eliminated, reducing the per-step feed-forward network slowdown to less than 2×.
- **Sampling.** Beyond AR's single-token softmax-and-sample, text diffusion sampling requires additional operations, most notably a self-conditioning embedding matmul and softmax over the full canvas of 256 tokens (see Algorithm 1). These operations are performed in full precision with a vocabulary dimension of 262k. Consequently, text diffusion sampling takes 3.06ms while AR sampling takes only 0.56ms. We implement sampling using standard PyTorch primitives optimized with `torch.compile`, rather than hand-written GPU kernels, to facilitate easier extensibility by the community.
- **Attention.** Unlike AR, DiffusionGemma uses bidirectional attention over a canvas of 256 tokens. As such, we cannot utilize the fast single-token decoding attention available to AR models, but we can take advantage of the highly-optimized FlashAttention-4 kernel (Dao et al., 2022; Zadouri et al., 2026). Under these optimizations, the attention operation is 4× slower for the DiffusionGemma model than the Gemma 4 AR model.

[4]Dataset available at https://github.com/google-deepmind/pg19.

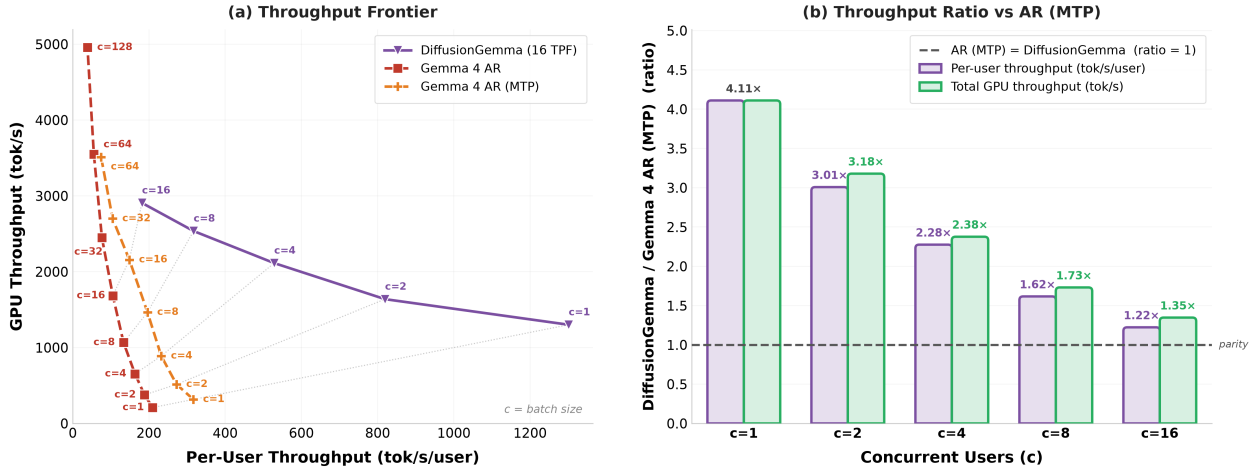

Figure 12 | **Trade-off between total and per-user throughput of the Gemma 4 AR model (with and without MTP) and DiffusionGemma.** In the low batch size regime, DiffusionGemma offers substantially higher TPS per user *and* higher total throughput. It is only at moderate batch sizes (around 32 concurrent requests) that AR models begin to have a throughput advantage. All models are run on an H100 with FP8 precision using the PG-19 benchmark (4096 input tokens, 1024 output tokens); the Gemma 4 AR (MTP) model uses a draft length of 4.

Eliminating CPU-GPU synchronization. There is additional complexity when serving DiffusionGemma: denoising steps depend on adaptive stopping and there are two types of forward passes (the denoising step and the KV cache update) that require different attention masks and that can occur within the same batch. To minimize request latency, it is important that this complexity is handled solely by GPU operations without triggering any additional CPU-GPU synchronization. This is achieved by extending asynchronous scheduling to the text diffusion models and introducing a per-sequence causal attention flag (see The vLLM Team and Google DeepMind Team, 2026 for more details).

Throughput for batch size of 1 serving. The decoding throughput for text diffusion models is calculated by the following formula:

$$\text{TPS} \triangleq \frac{\text{TPF}}{t_{\text{fwd}}}, \tag{12}$$

where TPS is Tokens Per Second, TPF is Tokens Per Forward (see Equation 10), and t_{fwd} is the time needed for a single denoising step, which varies with context length (due to attention). On an H100 GPU (FP8 precision), a single denoising step of DiffusionGemma takes $t_{\text{fwd}} = 13.56$ms on average (end-to-end; the per-step GPU time is 12.63ms, see Figure 11) when serving a single request with 4096 input tokens and 1024 output tokens. TPF varies depending on the task. Assuming a TPF of 19.74 (the average TPF across the 7 benchmarks reported in Table 3), the average decoding throughput of the model is 1456 TPS—a 7.1× improvement over the Gemma 4 AR model (204 TPS) and a 4.8× improvement over the AR model with MTP (303 TPS), on the same device setup.

Multi-user throughput. Figure 12 shows the trade-off between total and per-user throughput of the Gemma 4 AR model (with MTP) and DiffusionGemma depending on the number of concurrent users (i.e., batch size). Importantly, these results are without targeted optimization for batch sizes larger than 1: current kernel selection is suboptimal, and sampling has not been tuned to scale with batch size; for example, applying top-k truncation to the sampling step is expected to yield significant throughput gains at higher batch sizes with negligible impact on output quality. Despite

this, DiffusionGemma offers substantially higher TPS per user *and* higher total throughput than the Gemma 4 AR (MTP) model in the low batch size regime, with AR models beginning to gain a throughput advantage only at moderate batch sizes (around 32 concurrent requests).

Toward real traffic throughput. Compared to the AR equivalent, DiffusionGemma changes the compute characteristics of two transformer layers: for the attention layer, it performs TPF× fewer transfers of KV cache; for the MoE layer, it performs proportionally more FLOPs (scaling with the number of effective denoising steps). This has the potential to address one of the challenges in modern LLM serving: memory-bound attention limiting compute utilization in FFW/MoE layers (Tang et al., 2024; Zhu et al., 2025a,b), which is particularly relevant for agentic workflows with long context. DiffusionGemma effectively trades data movement for compute, which is favorable on modern GPU hardware where compute-to-bandwidth ratios continue to grow; a thorough empirical analysis under realistic traffic conditions is beyond the scope of this work.

7. Experimental Results

We evaluate DiffusionGemma's capabilities and inference efficiency across four operational modes: text diffusion (TD) versus autoregressive (AR) generation, each evaluated with and without thinking enabled; unless otherwise specified, TD with thinking is enabled. We benchmark the model against its AR initialization (Gemma 4 26B A4B), contemporary open-weight text diffusion models (LLaDA 2.1 Flash 100B and Nemotron Diffusion 14B), and the proprietary Mercury 2 API.

We consider a diverse suite of benchmarks spanning core domains such as mathematical reasoning, code generation, general knowledge, multimodal understanding, and instruction following alongside agentic capabilities. Specifically, for mathematical reasoning, we utilize AIME (Dekoninck et al., 2026), GSM8K (Cobbe et al., 2021), MGSM (Shi et al., 2023), Putnam (Tsoukalas et al., 2024), and HiddenMath (internal). Coding performance is measured against LiveCodeBench-v6 (Jain et al., 2025), Codeforces (Quan et al., 2025), HumanEval (Chen et al., 2021), BigCodeBench (Zhuo et al., 2025), LBPP (v2) (Matton et al., 2024), and Natural2Code (internal). Broad and expert-level general knowledge is measured against GPQA-Diamond (Rein et al., 2024), BIG-Bench (Suzgun et al., 2023), MMMLU (OpenAI, 2024), and MMLU-Pro (Wang et al., 2024), while multimodal reasoning is evaluated on MMMU-Pro (Yue et al., 2025). Finally, strict instruction following is assessed via IFEval (Zhou et al., 2023), and agentic task completion is evaluated through the Tau-bench suite, encompassing the Retail, Airline, and Telecom environments (Yao et al., 2024).

The complete per-benchmark results for all models and inference configurations are reported in Table 3. Table 4 provides a complementary analysis of DiffusionGemma's decoding efficiency, reporting Tokens Per Forward (TPF; Equation 10), Tokens Per Second (TPS; Equation 12), effective denoising steps (Equation 9), total generated tokens (Equation 7), and end-to-end generation latency, excluding prefill time. For a higher-level comparison, Figure 13 groups the benchmarks into three capability areas—reasoning and knowledge, coding, and instruction following and agentic behaviour—and reports the unweighted mean of the 0–100 benchmark scores within each area, together with output throughput. A model is shown for a capability area only if it completed every constituent benchmark; missing bars therefore indicate incomplete benchmark coverage rather than a score of zero. Hatched bars denote the no-think variants.

DiffusionGemma sets a new performance frontier for text diffusion models, substantially outperforming existing open-weight diffusion baselines while increasing TPF by approximately an order of magnitude. In terms of quality it is highly competitive with Mercury 2, a closed-weight text-diffusion model, while reaching roughly 1,500 output tokens per second on a single previous-generation H100

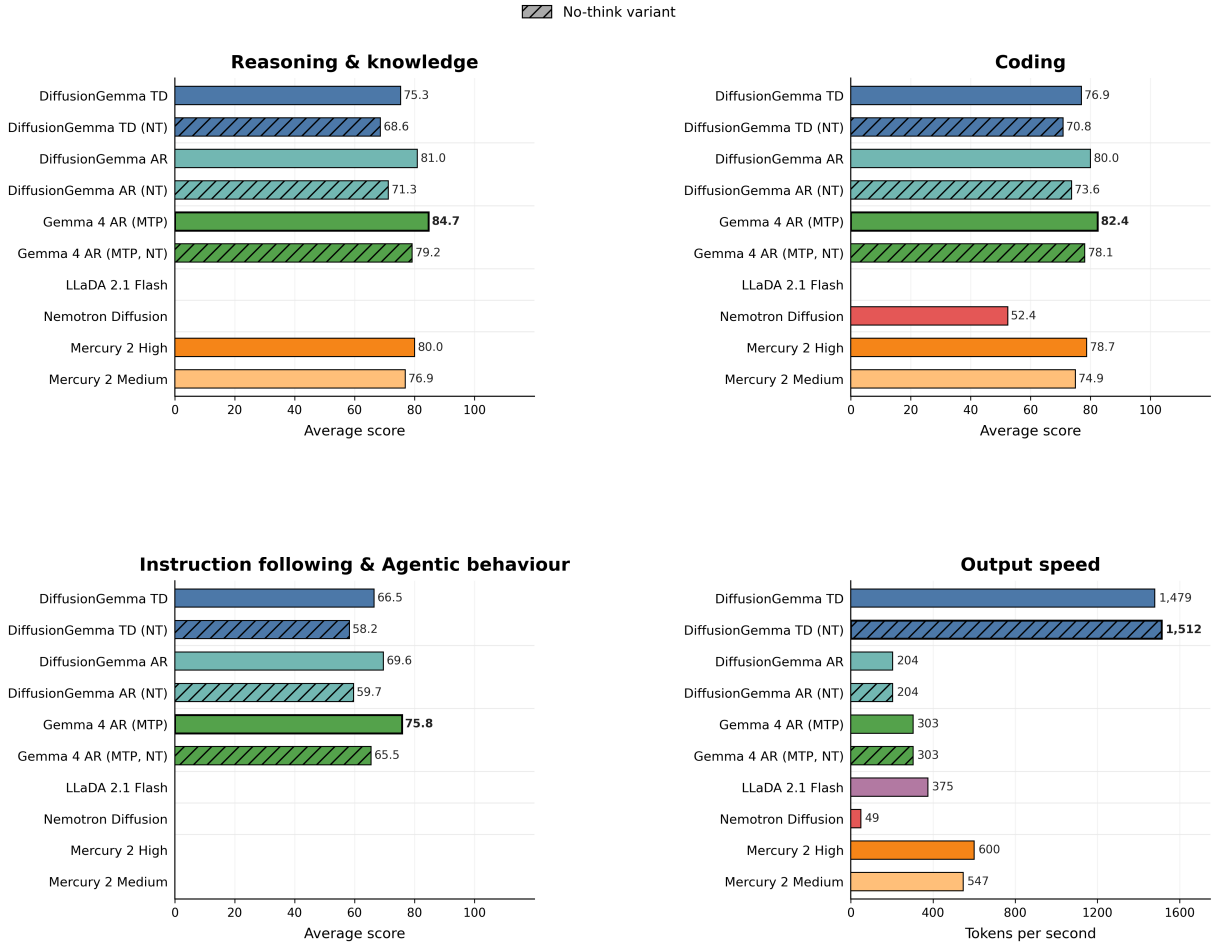

Figure 13 | **Performance by capability area[*] (unweighted mean of 0–100 scores across the area) and output speed.** We omit models for which we do not have complete coverage. Non-thinking variants are indicated using striped bars. Output speed is averaged over the 7 benchmarks for which we have full coverage, see Table 3.

[*] Benchmarks are grouped into: reasoning and knowledge (*AIME 2026*, *GPQA Diamond*, *BigBench EH*, *GSM8K*, *MGSM*, *MMMLU*, *Putnam*, *MMLU-Pro*, and *HiddenMath*); coding (*LiveCodeBench V6*, *HumanEval*, *BigCodeBench*, *LBPP*, and *Natural2Code*); and instruction following and agentic behaviour (*IFEval*, *Tau2 Retail*, *Tau2 Airline*, and *Tau2 Telecom*).

GPU, a roughly 2.5× speedup over Mercury 2.

Relative to the AR model used for initialization, DiffusionGemma in text-diffusion (TD) mode trades some absolute benchmark performance for substantially greater decoding speed. Although the conversion reduces performance across the three capability areas, TD mode delivers nearly 5× the output throughput of the original Gemma 4 AR baseline under heavily optimized MTP serving: 1,479 tokens per second compared with 303 tokens per second. The two-stage training pipeline nevertheless retains support for autoregressive decoding. When run in standard left-to-right AR mode, DiffusionGemma recovers part of the performance gap observed in TD mode and narrows the capability gap to the original baseline, albeit at lower throughput. This dual-mode capability could enable requests to be routed dynamically according to latency requirements and task complexity.

	Open-weight Models								Closed-weight Model	
	DiffusionGemma 26B A4B				Gemma 4 26B A4B		LLaDA 2.1 Flash 100B	Nemotron Diffusion 14B	Mercury 2 Unknown	
Mode	TD	TD (No-think)	AR	AR (No-think)	AR (MTP)	AR (MTP, No-think)	TD (S Mode)	TD (Diffusion Mode)	High	Medium
AIME 2026	69.1	50.8	84.2	57.5	88.3	80.0	80.0	40.0	91.7	82.5
GPQA Diamond	73.2	64.6	79.8	67.2	82.3	73.7	68.7	47.0	75.2	66.7
LiveCodeBench-V6	69.1	60.6	71.4	58.3	77.1	72.6	39.4	28.6	79.4	74.9
Codeforces ELO	1429	959	1569	1059	1718	1529	718	-	1986	1629
BigBench EH	47.6	40.0	59.1	42.2	64.8	56.2	-	-	48.9	43.8
GSM8K	96.3	95.8	96.6	96.1	96.7	96.4	45.0	-	96.5	95.8
MGSM	84.8	80.7	87.9	84.3	92.9	91.5	6.8	69.3	91.9	91.2
MMMLU	81.5	76.3	82.2	78.0	86.3	78.0	-	-	81.9	80.6
MMMU Pro	54.3	66.0	63.3	66.7	73.8	72.5	-	-	-	-
Putnam	67.4	57.1	74.7	59.7	81.0	72.9	-	45.8	73.6	73.6
HumanEval	94.5	92.7	98.2	97.6	98.8	97.6	90.2	86.0	98.2	98.2
BigCodeBench	46.0	41.9	47.7	45.9	50.2	48.1	-	33.5	47.6	45.3
LBPP	81.0	68.9	86.3	74.1	89.5	77.3	45.7	40.7	89.2	85.0
IFEval	97.4	94.5	97.2	95.7	98.7	97.8	-	72.1	97.0	94.5
Tau2 Retail	71.5	57.5	75.4	61.0	85.5	79.0	-	-	-	-
Tau2 Airline	69.0	49.0	72.0	50.0	76.0	51.0	-	-	-	-
Tau2 Telecom	28.1	32.0	33.8	32.0	43.0	34.2	-	-	-	-
MMLU-Pro	77.6	77.9	78.8	79.1	82.6	82.6	-	-	77.6	75.5
Natural2Code	94.0	90.1	96.2	92.3	96.3	94.7	86.9	73.3	79.1	71.3
HiddenMath	80.6	74.3	85.4	77.5	87.2	81.6	-	44.3	82.7	82.3
Output Speed (TPS)	1479	1512	204	204	303	303	375	49	600	547
Tokens Per Forward (TPF)	19.74	18.76	1.00	1.00	1.40	1.40	4.63	1.79	-	-
Average Total Tokens	4,001	829	5,184	1,025	7,207	1,816	4,371	941	3,882	1,222

Table 3 | **Comparison of model performance, TPS and TPF across various benchmarks.** TPS excludes prefill time; "–" denotes missing data. For speed measurements: DiffusionGemma and Gemma 4 are measured on 1× H100 (FP8, batch size 1); Nemotron 14B on 1× H100 (bfloat16, batch size 1); LLaDA 2.1 Flash 100B on 8× B200 (bfloat16, batch size 1); Mercury 2 via its public API (see Appendix E for speed estimation). TPS, TPF and total tokens are averaged over the 7 benchmarks for which we have full coverage: AIME 2026, GPQA Diamond, LiveCodeBench-v6, MGSM, HumanEval, LBPP, and Natural2Code. TPS and TPF for Gemma 4 (MTP) are measured using SPEED-Bench (Abramovich et al., 2026). The Natural2Code and HiddenMath rows are highlighted as they are proprietary, unleaked evals.

	Score (↑)		TPF (↑)		TPS (↑)		Effective DNS (↓)		Total Forwards (↓)		Total Tokens		E2E Time (s) (↓)	
Benchmark	**Think**	**No-Think**	**Think**	**No-Think**	**Think**	**No-Think**	**Think**	**No-Think**	**Think**	**No-Think**	**Think**	**No-Think**	**Think**	**No-Think**
AIME 2026	69.1	50.8	19.3	16.7	1365.4	1333.0	12.6	14.1	390.6	91.1	6,445	1,309	4.72	0.98
GPQA Diamond	73.2	64.6	16.7	16.5	1207.8	1330.2	15.1	13.4	443.4	48.1	5,647	726	4.68	0.55
LiveCodeBench-V6	69.1	60.6	18.5	16.9	1278.3	1333.4	13.8	14.0	581.8	195.7	7,534	1,847	5.89	1.39
Codeforces ELO	1429	959	15.1	14.0	950.5	1040.3	17.1	18.5	959.6	521.1	11,622	4,279	12.23	4.11
BigBench EH	47.6	40.0	20.8	17.7	1390.2	1415.1	11.9	12.8	434.9	68.1	9,062	1,233	6.52	0.87
GSM8K	96.3	95.8	23.2	24.1	1866.2	1966.4	9.1	7.5	43.4	13.2	883	298	0.47	0.15
MGSM	84.8	80.7	19.0	16.8	1526.7	1367.5	11.5	11.9	63.6	19.5	1,085	291	0.71	0.21
MMMU Pro	54.3	66.0	17.7	15.4	1351.3	1255.1	13.8	15.3	191.6	31.6	3,178	472	2.35	0.38
Putnam	67.4	57.1	18.1	16.0	1330.8	1282.4	13.4	14.4	303.8	77.4	4,725	1,103	3.55	0.86
HumanEval	94.5	92.7	23.0	24.3	1838.2	1981.2	9.4	8.0	55.6	14.3	1,174	305	0.64	0.15
BigCodeBench	46.0	41.9	19.6	19.4	1560.1	1579.0	11.3	10.1	77.3	21.8	1,410	394	0.90	0.25
LBPP	81.0	68.9	20.5	19.2	1509.8	1545.4	11.6	10.9	264.3	79.4	4,730	859	3.13	0.56
IFEval	97.4	94.5	17.2	9.0	1368.3	732.1	13.0	14.4	100.8	24.0	1,464	239	1.07	0.33
Natural2Code	94.0	90.1	21.1	21.0	1682.7	1706.5	10.5	9.6	70.5	24.7	1,391	465	0.83	0.27
HiddenMath	80.6	74.3	21.0	19.4	1591.2	1564.7	11.3	11.5	206.4	49.7	3,501	844	2.20	0.54

Table 4 | **Performance and latency metrics of DiffusionGemma TD in Thinking vs. No-Thinking mode.** We report accuracy/score, Tokens Per Forward (TPF, higher is faster), Tokens Per Second (TPS), Effective Denoising Steps (DNS, lower is faster), Total Forwards, Total Tokens, and End-to-End Latency per sample in seconds (excluding prefill time). The latency metrics are averaged across all samples within each benchmark.

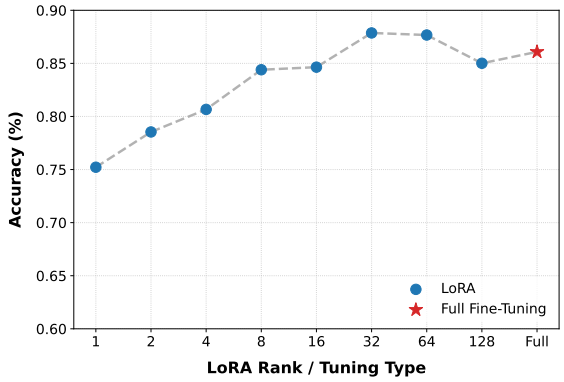

Figure 14 | **Sudoku Downstream SFT performance.** The x-axis is the LoRA rank and the y-axis is the accuracy of the model.

Model	Denoising Steps	Accuracy (%)
DiffusionGemma	40.65	0.00
+ LoRA finetuning	**10.72**	**84.40**

Table 5 | **Sudoku accuracy of the finetuned model with LoRA rank 8.** The base model fails to produce a correct Sudoku grid before finetuning. Finetuning the model leads to accuracy above 80%, while using significantly less effective denoising steps due to lower predictive entropy.

8. Open-Source Downstream SFT

Alongside DiffusionGemma, we release an open-source finetuning toolkit that allows practitioners to adapt the model to their own domain-specific datasets. We build on top of Hackable Diffusion (Crepy et al., 2026), a modular open-sourced research toolbox for generative modeling. We provide Low-Rank Adaptation (LoRA) recipes (Hu et al., 2022) to enable finetuning on consumer hardware.

Finetuning procedure. Our open-source SFT toolkit includes both a causal encoder and a diffusion decoder objective. Training sequences are of length $P + KC$ where P is the number of prompt tokens, K is the number of canvases and C is the canvas size. To compute the total loss, the encoder first processes all tokens in the entire sequence, populates a KV cache H, and provides next token predictions that are fed into a standard cross-entropy loss for the encoder. The decoder loss is computed by uniformly sampling a canvas k from the K canvases available. For canvas k, we evaluate a denoising cross-entropy loss using the decoder's predicted logits given the current noisy state, x_t, the KV cache H for the prompt and any prior canvases in the sequence (i.e., canvas 1, 2, ..., $k-1$); for 50% of the data-points in a batch, the decoder is also conditioned on a self-conditioning state z_t computed from a previous forward pass. The other 50% have $z_t = \mathbf{0}$. Formally, we write the losses as

$$L_{\text{encoder}}(\theta) = -\frac{1}{P+KC}\sum_{j=1}^{P+KC}\log p_\theta(x^j \mid x^{1:j-1}), \qquad L_{\text{decoder}}(\theta) = -\frac{1}{C}\sum_{i=1}^{C}\log p_\theta(x_0^i \mid x_t, z_t, H). \tag{13}$$

For the encoder loss, j indexes along all tokens in the sequence (prompt and canvases); for the decoder loss, i indexes along tokens in the given canvas. The final loss is the sum of the two losses.

LoRA for parameter-efficient finetuning. LoRA is applied to all linear operations (attention projections, MLP gates, MoE routers, and the self-conditioning feedforward block). This allows us to achieve strong downstream performance while training only a small fraction of the model's parameters, using 2× A100 80GB GPUs. All of our training details are reported in Appendix C.

Case study: Sudoku puzzle solving. Solving Sudoku puzzles is a compelling testbed for discrete diffusion due to the non-autoregressive nature of the task. We finetune DiffusionGemma on an

open-source dataset of Sudoku puzzles.[5] With full finetuning, the sampler (Algorithm 1) achieves >85% puzzle-level accuracy evaluated on a held out set of 4096 puzzles (Figure 14). Decreasing the LoRA rank trades-off accuracy and compute. In Table 5, we report performance of the original model as well as the finetuned model. See Appendix C for additional results on PubMedQA (Jin et al., 2019).

9. Practical Advantages of Text Diffusion

In this manuscript, we have primarily emphasized low latency as the principal advantage of text diffusion over standard AR language modeling (Dieleman et al., 2022; Ghazvininejad et al., 2019; Gong et al., 2023; Li et al., 2022). However, the architectural paradigm of text diffusion offers several benefits that extend beyond computational efficiency. Here, we demonstrate the practical advantages through concrete examples and qualitative analysis of samples generated by the model. To isolate the intrinsic capabilities of the architecture, we disable explicit thinking modes in both DiffusionGemma and the baseline Gemma AR model. This allows us to observe and evaluate the underlying generative process.

9.1. Bidirectional Reasoning and Self-Correction

AR next-token prediction is inherently causal; during the generation of a given token, the model can only attend to the preceding context. It fundamentally lacks the capacity to condition upon tokens that have yet to be generated. By contrast, text diffusion operates with full bidirectional attention across the canvas. This non-causal property allows tokens at arbitrary positions across the canvas to attend simultaneously to both past and future representations—a critical capability for complex planning and reasoning tasks where early decisions depend on eventual outcomes (e.g., Kitouni et al., 2024; Nagarajan et al., 2025; Papadopoulos et al., 2024; Zhang-Li et al., 2024). Consequently, locally within a canvas, future tokens can directly influence the formation of earlier tokens.

Because text diffusion employs an iterative refinement process, it possesses a built-in mechanism for self-correction. Any premature commitments generated in earlier denoising steps can be revised during subsequent denoising iterations (Han et al., 2023; Reid et al., 2022). While we have previously demonstrated the efficacy of bidirectional reasoning and self-correction in structured logic puzzles like Sudoku, we demonstrate these mechanics in two reasoning examples.

Consider the multi-step arithmetic problem detailed in Figure 15. Constrained by causal generation, the AR Gemma model must commit to the first token of the final answer before articulating the intermediate calculations. This forces an incorrect initial prediction (−1) that the model must later correct. DiffusionGemma, conversely, leverages its bidirectional attention canvas to simultaneously evolve the final answer and its underlying logic. As visualized in the accompanying denoising trace, the parallel diffusion process explores incorrect intermediate states but uses the emerging reasoning tokens to course-correct, seamlessly converging on the correct answer (−25). See Appendix G.1 for the second example of logical reasoning and correction via bidirectional attention and denoising.

9.2. Dynamic and Adaptive Computation

Unlike AR models, which inherently allocate a fixed amount of computation per generated token, text diffusion facilitates dynamic test-time compute. Text diffusion models can autonomously calibrate the computational effort expended on a given prompt, trading increased inference time for enhanced generative performance based on the inherent difficulty of the task (Ye et al., 2024). DiffusionGemma

[5] https://www.kaggle.com/datasets/rohanrao/sudoku

Prompt

"Solve $7 \times (\sqrt{111 + 10} - 3) - 9^2$. Give the answer first then provide the reasoning, be concise."

Gemma 4 26B A4B: −1
Reasoning:
1. Inside parentheses: 111 + 10 = 121.
2. Square root: $\sqrt{121} = 11$.
3. Inside parentheses continued: 11 − 3 = 8.
4. Multiplication: $7 \times 8 = 56$.
5. Exponent: $9^2 = 81$.
6. Subtraction: 56 − 81 = −25.

Correction: Wait, recalculating: 56 − 81 = −25.
Correct Answer: −25

DiffusionGemma: −25
Reasoning:
1. Simplify inside the square root: $\sqrt{111 + 10} = \sqrt{121} = 11$.
2. Solve inside the parentheses: (11 − 3) = 8.
3. Multiply: $7 \times 8 = 56$.
4. Calculate the exponent: $9^2 = 81$.
5. Final subtraction: 56 − 81 = −25.

Analysis: Constrained by causal generation, Gemma 4 commits to the incorrect answer "−1" in its very first tokens. As it autoregressively generates its reasoning, it discovers its own error and is forced to append a correction. Conversely, DiffusionGemma produces the correct answer "−25" immediately, preventing logical contradictions. Note that both models are operating in non-thinking mode.

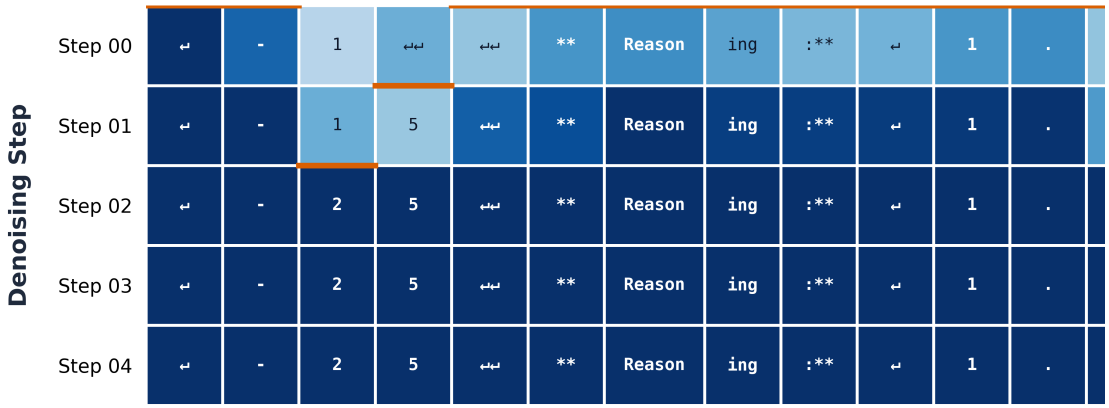

Figure 15 | **Denoising trace for the math reasoning problem (zoomed in to the first 12 tokens), where color intensity indicates model confidence**. DiffusionGemma initially makes the same causal estimation mistake as the AR model, assigning high probability to −1. However, through iterative bidirectional refinement, it suggests −15 in the next step before fully converging to the correct answer −25. The parallel denoising process allows the final answer and reasoning steps to inform one another. Denoising is completed in just 5 steps.

automatically adapts to task difficult via adaptive stopping. As illustrated in Figure 5, tasks of varying complexity organically elicit different numbers of effective denoising steps. This allows the model to conserve compute on easier queries while expending more compute on complex reasoning. In Appendix G.2 we concretely illustrate the adaptive behavior of text diffusion by contrasting a structurally "hard" generation task against a structurally "easy" task and show that the easy task requires fewer denoising steps, as well as showing how the information propagates differently across the canvas of tokens in these two cases.

Furthermore, the maximum number of denoising steps serves as an explicit configuration parameter to manage the latency-quality tradeoff. A lower step count forces the model to traverse the reverse process more coarsely, yielding faster results with a corresponding reduction in output precision. Expanding the step count allows for a finer resolution during generation, maximizing quality while extending the required compute time. Figure 10 plots this continuous relationship, demonstrating how the model can be dynamically calibrated to suit varied operational constraints.

9.3. Structured and Constrained Outputs

In many practical applications, the desired output adheres to a rigid, highly structured format (e.g., JSON schemas) or exhibits strong lexical dependence on the input prompt, such as in optical character recognition (OCR), code editing tasks, or fine-grained syntactic control (Chen et al., 2023b; Li et al., 2022). Because text diffusion generates and refines all tokens in parallel, it seamlessly exploits these structural priors to accelerate convergence. AR models lack this capability; constrained by strict sequential decoding, they must incur the same $O(N)$ computational cost to generate N tokens, even when those tokens consist of rigid boilerplate or verbatim copies of the input context. By contrast, DiffusionGemma can identify and lock in predictable syntactic structures across the entire sequence simultaneously. We highlight this efficiency through two real-world examples: strict JSON extraction (Appendix G.3, Figures 26 and 27) and Python code debugging (Appendix G.3, Figures 28 and 29). Both cases clearly illustrate that the highly constrained output allows the diffusion process to converge in merely two to three steps, drastically reducing latency compared to sequential decoding.

10. Limitations & Known Issues

In the previous section, we discussed new emergent properties of text diffusion and advantages of our approach relative to AR language modeling. While DiffusionGemma establishes a new Pareto frontier for generative text modeling and inference efficiency, the current experimental release has these known limitations:

- **Performance gap relative to the AR baseline:** Lower absolute performance than its AR initialization (Gemma 4 26B A4B) stems from several practical constraints: bypassing native diffusion pretraining to warm-start from AR weights; relying on a comparatively short SFT phase due to compute budget constraints; using an online learning algorithm (SD·RL) that explicitly targets ultra-low latency, intrinsically trading off asymptotic performance; and inheriting architectural, optimization, and data-mixture decisions from the AR baseline that may be suboptimal for the discrete diffusion paradigm.
- **Generation length and conciseness:** As mentioned in Section 5, our final checkpoint produces highly concise outputs. While this emergent brevity acts as a multiplier for inference speed, it precludes the model from leveraging the quality improvements typically unlocked by longer, more elaborate reasoning traces.
- **Occasional token stuttering:** In rare instances, the model's output degenerates into repetitive loops or localized stuttering (e.g., endlessly repeating a common token like "the the the"). While

our SD·RL training successfully mitigates the vast majority of such instances, this uncommon artifact remains a direct consequence of operating in an ultra-low latency regime, where the aggressively reduced number of denoising steps can occasionally compromise the robustness of the generation process.

- **Occasional omission of closing thought tags in multimodal tasks:** When processing multimodal prompts, the model does not always reliably generate a closing thought tag (even when the reasoning is correct). This can artificially drag down performance in thinking mode on specific benchmarks; for example, on MMMU-Pro, the thinking score drops below the non-thinking score (54.3 vs. 66.0). This was identified too late in the pipeline to apply a fix for this release.
- **Throughput limits at high batch sizes:** DiffusionGemma excels at low batch sizes by trading memory-bandwidth costs for compute, outperforming Gemma 4 AR (with MTP) in both per-user and total throughput for up to ~32 concurrent users (Figure 12). Beyond this point, the higher per-token compute cost causes AR models to gain a throughput advantage. As noted in Section 6, these results are obtained without targeted batch-size optimization; a thorough empirical analysis under realistic traffic conditions remains future work.

11. Conclusion

DiffusionGemma demonstrates a practical, compute-efficient path to ultra-fast text generation. By finetuning the existing Gemma 4 26B A4B AR model to perform text diffusion through our two-stage training pipeline (SFT and SD·RL), it establishes a new Pareto frontier for the speed-to-intelligence tradeoff—achieving around 1,500 tokens per second on a single H100 while retaining highly competitive reasoning and multimodal capabilities. By releasing DiffusionGemma as an experimental open-weight model—alongside reference implementations in HuggingFace Transformers and vLLM—we aim to empower the open-source community to push the boundaries of text diffusion. We hope researchers and practitioners will build upon this approach, whether by finetuning for specialized tasks, exploring novel sampling algorithms, or further optimizing inference efficiency.

References

T. Abramovich, M. Ashkenazi, I. Putterman, B. Chislett, T. Mitra, B. D. Rouhani, R. Zilberstein, and Y. Geifman. SPEED-Bench: A unified and diverse benchmark for speculative decoding. In *Proceedings of the International Conference on Machine Learning*, 2026.

M. Arriola, A. K. Gokaslan, J. T. Chiu, Z. Yang, Z. Qi, J. Han, S. S. Sahoo, and V. Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *Proceedings of the International Conference on Learning Representations*, 2025.

A. Asaria, T. Salomone, and D. Gandhi. Neither parallel nor sequential: How DiffusionGemma actually commits tokens. *arXiv preprint arXiv:2606.14620*, 2026.

J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, 2021.

H. Ben-Hamu, I. Gat, D. Severo, N. S. Nolte, and B. Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *Advances in Neural Information Processing Systems*, 2026.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

T. Bie, M. Cao, K. Chen, L. Du, M. Gong, Z. Gong, Y. Gu, J. Hu, Z. Huang, Z. Lan, et al. LLaDA 2.0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

K. Black, M. Janner, Y. Du, I. Kostrikov, and S. Levine. Training diffusion models with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2024.

T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. MEDUSA: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the International Conference on Machine Learning*, 2024.

A. Campbell, J. Benton, V. De Bortoli, T. Rainforth, G. Deligiannidis, and A. Doucet. A continuous time framework for discrete denoising models. In *Advances in Neural Information Processing Systems*, 2022.

A. Campbell, J. Yim, R. Barzilay, T. Rainforth, and T. Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. In *Proceedings of the International Conference on Machine Learning*, 2024.

H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. MaskGIT: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023a.

J. Chen, Y. Huang, T. Lv, L. Cui, Q. Chen, and F. Wei. TextDiffuser: Diffusion models as text painters. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

J. Chen, Y. Liang, and Z. Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the International Conference on Machine Learning*, 2026.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

T. Chen, R. Zhang, and G. E. Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. In *Proceedings of the International Conference on Learning Representations*, 2023c.

X. Cheng, X. Yu, C. Shao, J. Li, Y. Xiong, Y. Qian, J. Zhu, S. Ma, X. Zhang, J. Ye, Q. Chen, C. Deng, J. Yu, D. Dai, Z. Zhang, Y. Wei, Y. Tan, W. Yang, R. Xu, Y. Wu, Z. Xu, X. Wang, M. Chen, R. Tian, X. Bi, Z. Hao, S. Chen, H. Cao, W. Zhang, A. Xu, H. Zhang, D. Zhao, and W. Liang. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation, 2026.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *Proceedings of the International Conference on Learning Representations*, 2020.

K. Clark, P. Vicol, K. Swersky, and D. J. Fleet. Directly fine-tuning diffusion models on differentiable rewards. In *Proceedings of the International Conference on Learning Representations*, 2024.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

C. Crepy, V. De Bortoli, A. Galashov, K. Greff, and I. Korshunova. Hackable Diffusion: A modular toolbox written in jax to experiment and educate around diffusion modeling, 2026. URL https://github.com/google/hackable_diffusion. Sponsors: A. Doucet and R. Elie.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

V. De Bortoli, J. Thornton, J. Heng, and A. Doucet. Diffusion Schrödinger bridge with applications to score-based generative modeling. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

J. Dekoninck, N. Jovanović, T. Gehrunger, K. Rögnvaldsson, I. Petrov, C. Sun, and M. Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with LLMs. In *3rd AI for Math Workshop at the International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=DmPE4byHuN.

J. Deschenaux and C. Gulcehre. Promises, outlooks and challenges of diffusion language modeling. *arXiv preprint arXiv:2406.11473*, 2024.

J. Deschenaux and C. Gulcehre. Beyond autoregression: Fast LLMs via self-distillation through time. In *Proceedings of the International Conference on Learning Representations*, 2025.

J. Deschenaux and C. Gulcehre. BlockGen: Flexible blockwise sequence modeling with hybrid samplers. In *Proceedings of the International Conference on Learning Representations*, 2026a.

J. Deschenaux and C. Gulcehre. Language modeling with hyperspherical flows. *arXiv preprint arXiv:2605.11125*, 2026b.

J. Deschenaux, C. Gulcehre, and S. S. Sahoo. The diffusion duality, chapter II: ψ-samplers. In *Proceedings of the International Conference on Learning Representations*, 2026.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

S. Dieleman, L. Sartran, A. Roshannai, N. Savinov, Y. Ganin, P. H. Richemond, A. Doucet, R. Strudel, C. Dyer, C. Durkan, C. Hawthorne, R. Leblond, W. Grathwohl, and J. Adler. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.

H. Dong, W. Xiong, D. Goyal, R. Pan, S. Diao, J. Zhang, K. Shum, and T. Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.

J. Engels, C. McDougall, B. Chughtai, J. Kramar, S. Rajamanoharan, C. Wu, A. Conmy, A. Q. Chen, J. Tarbouriech, M. Ma, B. O'Donoghue, J. G. L. de Oliveira, R. Shah, and N. Nanda. How transparent is DiffusionGemma? *arXiv preprint arXiv:2606.20560*, 2026.

Y. Fan, O. Watkins, Y. Du, H. Liu, K.-H. Moon, C. Boutilier, P. Abbeel, B. Lin, and H. Lee. DPOK: Reinforcement learning for fine-tuning text-to-image diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

F. Fu, T. Guo, and Z. Liu. Learnable sampler distillation for discrete diffusion models. In *Advances in Neural Information Processing Systems*, 2025.

Y. Fu, L. Whalen, A. Garg, C. Wu, M. Khadkevich, N. Oswald, E. Xie, D. Egert, S. T. Sreenivas, S. Diao, C. Yu, Y. Yu, W. Chen, S. Norouzi, J. Liu, S. Lan, L. Zhu, J. Wang, J. Jiang, M. Mardani, M. Maghoumi, S. Han, A. Jukic, N. Tajbakhsh, J. Kautz, and P. Molchanov. Nemotron-Labs-Diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. Technical report, NVIDIA, 2026. Technical report.

I. Gat, T. Remez, N. Shaul, F. Kreuk, R. T. Chen, G. Synnaeve, Y. Adi, and Y. Lipman. Discrete flow matching. In *Advances in Neural Information Processing Systems*, 2024.

Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Gemma Team, S. E. Abd, V. Aggarwal, R. Algayres, A. Andreev, O. Bachem, I. Ballantyne, C. Brick, V. Cărbune, M. Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

M. Ghazvininejad, O. Levy, Y. Liu, and L. Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2019.

S. Gong, M. Li, J. Feng, Z. Wu, and L. Kong. Diffuseq: Sequence to sequence text generation with diffusion models. In *Proceedings of the International Conference on Learning Representations*, 2023.

S. Gong, S. Agarwal, Y. Zhang, J. Ye, L. Zheng, M. Li, C. An, P. Zhao, W. Bi, J. Han, H. Peng, and L. Kong. Scaling diffusion language models via adaptation from autoregressive models. In *Proceedings of the International Conference on Learning Representations*, 2025.

Google DeepMind. Gemini diffusion, 2025. URL https://deepmind.google/models/gemini-diffusion/.

Google DeepMind. Accelerating gemma 4: faster inference with multi-token prediction drafters, 2026. URL https://blog.google/innovation-and-ai/technology/developers-tools/multi-token-prediction-gemma-4/.

Google DeepMind Team. DiffusionGemma implementation in HuggingFace transformers. https://github.com/huggingface/transformers/pull/46540, 2026. Pull Request.

A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

J. Gu, J. Bradbury, C. Xiong, V. O. Li, and R. Socher. Non-autoregressive neural machine translation. In *Proceedings of the International Conference on Learning Representations*, 2018.

K. Han, K. Kenealy, A. Barua, N. Fiedel, and N. Constant. Transfer learning for text diffusion models. *arXiv preprint arXiv:2401.17181*, 2024.

X. Han, S. Finkelstein, S. Sharma, Y. Chen, and H. He. SSD-LM: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2023.

Y. He, N. Murata, C.-H. Lai, Y. Takida, T. Uesaka, D. Kim, W.-H. Liao, Y. Mitsufuji, J. Z. Kolter, R. Salakhutdinov, and S. Ermon. Manifold preserving guided diffusion. In *Proceedings of the International Conference on Learning Representations*, 2024.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

P. Holderrieth and E. Erives. An introduction to flow matching and diffusion models. *arXiv preprint arXiv:2506.02070*, 2025.

E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

E. Hoogeboom, A. A. Gritsenko, J. Bastings, B. Poole, R. v. d. Berg, and T. Salimans. Autoregressive diffusion models. In *Proceedings of the International Conference on Learning Representations*, 2022.

E. Hoogeboom, D. Ruhe, J. Heek, T. Mensink, and T. Salimans. Beyond single tokens: Distilling discrete diffusion models via discrete MMD. *arXiv preprint arXiv:2603.20155*, 2026. doi: 10.48550/arXiv.2603.20155.

E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations*, 2022.

K. Hu, L. Qiu, Y. Lu, H. Zhao, T. Li, Y. Kim, J. Andreas, and K. He. ELF: Embedded language flows. *arXiv preprint arXiv:2605.10938*, 2026.

H. Huang, N. Ardalani, A. Sun, L. Ke, S. Bhosale, H.-H. S. Lee, C.-J. Wu, and B. Lee. Toward efficient inference for mixture of experts. In *Advances in Neural Information Processing Systems*, 2024.

Inception Labs. Introducing Mercury 2. https://www.inceptionlabs.ai/blog/introducing-mercury-2, February 2026. Accessed: 2026-07-22.

Inception Labs, S. Khanna, S. Kharbanda, S. Li, H. Varma, E. Wang, S. Birnbaum, Z. Luo, Y. Miraoui, A. Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Interfaze. The first open source diffusion audio asr model. https://interfaze.ai/blog/the-first-open-source-diffusion-audio-asr-model, 2026. Blog post.

N. Jain, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *Proceedings of the International Conference on Learning Representations*, 2025.

Q. Jin, B. Dhingra, Z. Liu, W. Cohen, and X. Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2019.

M. Jo, J. Yoon, J. Deschenaux, C. Gulcehre, and S. Ahn. Loopholing discrete diffusion: Deterministic bypass of the sampling wall. In *Proceedings of the International Conference on Learning Representations*, 2026.

D. P. Kingma, T. Salimans, B. Poole, and J. Ho. Variational diffusion models. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

O. Kitouni, N. Nolte, A. Williams, M. Rabbat, D. Bouchacourt, and M. Ibrahim. The factorization curse: Which tokens you predict underlie the reversal curse and more. In *Advances in Neural Information Processing Systems*, 2024.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

C. Lee, J. Yoo, M. Agarwal, S. Shah, J. Huang, A. Raghunathan, S. Hong, N. M. Boffi, and J. Kim. Flow map language models: One-step language modeling via continuous denoising. *arXiv preprint arXiv:2602.16813*, 2026.

K. Lee, H. Liu, M. Ryu, O. Watkins, Y. Du, C. Boutilier, and P. Abbeel. Aligning text-to-image models using human feedback. *arXiv preprint arXiv:2302.12192*, 2023.

Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the International Conference on Machine Learning*, 2023.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020.

J. Lezama, T. Salimans, L. Jiang, H. Chang, J. Ho, and I. Essa. Predictor-corrector sampling for discrete diffusion models. In *Proceedings of the International Conference on Learning Representations*, 2023.

X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto. Diffusion-LM improves controllable text generation. In *Advances in Neural Information Processing Systems*, 2022.

Y. Li, F. Wei, C. Zhang, and H. Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, 2026.

Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nicklas, and M. Le. Flow matching for generative modeling. In *Proceedings of the International Conference on Learning Representations*, 2023.

E. Liu, X. Ning, Y. Wang, and Z. Lin. NI sampling: Accelerating discrete diffusion sampling by token order optimization. In *Proceedings of the International Conference on Learning Representations*, 2026a.

J. Liu, G. Liu, J. Liang, Y. Li, J. Liu, X. Wang, P. Wan, D. Zhang, and W. Ouyang. Flow-GRPO: Training flow matching models via online RL. In *Advances in Neural Information Processing Systems*, 2025.

J. Liu, X. Dong, Z. Ye, R. Mehta, Y. Fu, V. Singh, C. Zhang, and P. Molchanov. TiDAR: Think in diffusion, talk in autoregression. In *Proceedings of Machine Learning and Systems*, 2026b.

X. Liu, X. Zhang, J. Ma, J. Peng, and Q. Liu. InstaFlow: One step is enough for high-quality diffusion-based text-to-image generation. In *Proceedings of the International Conference on Learning Representations*, 2024.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

A. Lou, C. Meng, and S. Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the International Conference on Machine Learning*, 2024.

S. Luo, Y. Tan, L. Huang, J. Li, and H. Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

H. Ma, O. Nabati, A. Rosenberg, B. Dai, O. Lang, C. Boutilier, N. Li, S. Mannor, L. Shani, and G. Tenneholtz. Reinforcement learning with discrete diffusion policies for combinatorial action spaces. In *Proceedings of the International Conference on Machine Learning*, 2026.

A. Matton, T. Sherborne, D. Aumiller, E. Tommasone, M. Alizadeh, J. He, R. Ma, M. Voisin, E. Gilsenan-McMahon, and M. Gallé. On leakage of code generation evaluation datasets. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024.

C. Meng, Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *Proceedings of the International Conference on Learning Representations*, 2022.

V. Meshchaninov, E. Chimbulatov, A. Shabalin, A. Abramov, and D. Vetrov. Cosmos: Compressed and smooth latent space for text diffusion modeling. In *Advances in Neural Information Processing Systems*, 2025.

T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, 2010.

V. Nagarajan, C. H. Wu, C. Ding, and A. Raghunathan. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. In *Proceedings of the International Conference on Machine Learning*, 2025.

S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li. Large language diffusion models. In *Advances in Neural Information Processing Systems*, 2026.

OpenAI. Multilingual massive multitask language understanding (MMMLU). https://huggingface.co/datasets/openai/MMMLU, 2024.

J. Ou, S. Nie, K. Xue, F. Zhu, J. Sun, Z. Li, and C. Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *Proceedings of the International Conference on Learning Representations*, 2025.

V. Papadopoulos, J. Wenger, and C. Hongler. Arrows of time for large language models. In *Proceedings of the International Conference on Machine Learning*, 2024.

R. Pope, S. Douglas, A. Chowdhery, J. Devlin, S. Ghemawat, J. He, D. Maher, S. Narang, S. Mishra, A. Ni, et al. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 5, 2023.

P. Potaptchik, J. Yim, A. Saravanan, P. Holderrieth, E. Vanden-Eijnden, and M. S. Albergo. Discrete flow maps. *arXiv preprint arXiv:2604.09784*, 2026.

P. Pynadath, J. Shi, and R. Zhang. CANDI: Hybrid discrete-continuous diffusion models. In *Proceedings of the International Conference on Machine Learning*, 2026.

S. Quan, J. Yang, B. Yu, B. Zheng, D. Liu, A. Yang, X. Ren, B. Gao, Y. Miao, Y. Feng, et al. CodeElo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *arXiv preprint arXiv:2501.01257*, 2025.

J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. In *Proceedings of the International Conference on Learning Representations*, 2020.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of the International Conference on Machine Learning*, 2022.

M. Reid, E. Marrese-Taylor, and Y. Matsuo. DiffusER: Discrete diffusion via edit-based reconstruction. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (AACL-IJCNLP)*, 2022.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *Proceedings of the Conference on Language Modeling*, 2024.

Y. Ren, H. Chen, Y. Zhu, W. Guo, Y. Chen, G. M. Rotskoff, M. Tao, and L. Ying. Fast solvers for discrete diffusion models: Theory and applications of high-order algorithms. In *Advances in Neural Information Processing Systems*, 2025.

S. S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. T. Chiu, A. Rush, and V. Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, 2024.

T. Salimans and J. Ho. Progressive distillation for fast sampling of diffusion models. In *Proceedings of the International Conference on Learning Representations*, 2022.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

A. Sauer, D. Lorenz, A. Blattmann, and R. Rombach. Adversarial diffusion distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

N. Savinov, J. Chung, M. Binkowski, E. Elsen, and A. van den Oord. Step-unrolled denoising autoencoders for text generation. In *Proceedings of the International Conference on Learning Representations*, 2022.

A. Shabalin, V. Meshchaninov, E. Chimbulatov, V. Lapikov, R. Kim, G. Bartosh, and D. Vetrov. TEncDM: Understanding the properties of the diffusion model in the space of language model encodings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

F. Shi, M. Suzgun, M. Freitag, X. Wang, S. Srivats, S. Vosoughi, H. W. Chung, Y. Tay, S. Ruder, D. Zhou, D. Das, and J. Wei. Language models are multilingual chain-of-thought reasoners. In *Proceedings of the International Conference on Learning Representations*, 2023.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the International Conference on Machine Learning*, 2015.

J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *Proceedings of the International Conference on Learning Representations*, 2021a.

Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Y. Song, C. Durkan, I. Murray, and S. Ermon. Maximum likelihood training of score-based diffusion models. In *Advances in Neural Information Processing Systems*, volume 34, 2021b.

Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *Proceedings of the International Conference on Learning Representations*, 2021c.

Y. Song, P. Dhariwal, M. Chen, and I. Sutskever. Consistency models. In *Proceedings of the International Conference on Machine Learning*, 2023.

Y. Song, Z. Zhang, C. Luo, P. Gao, F. Xia, H. Luo, Z. Li, Y. Yang, H. Yu, X. Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

M. Stern, N. Shazeer, and J. Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

R. Strudel, C. Tallec, F. Altché, Y. Du, Y. Ganin, A. Mensch, W. Grathwohl, N. Savinov, S. Dieleman, L. Sifre, and R. Leblond. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*, 2022.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. Le, E. Chi, D. Zhou, and J. Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023.

J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the International Conference on Machine Learning*, 2024.

The vLLM Team and Google DeepMind Team. DiffusionGemma: The first diffusion LLM (dLLM) natively supported in vLLM. https://vllm.ai/blog/2026-06-10-diffusion-gemma, June 2026. vLLM Blog. Implementation: https://github.com/vllm-project/vllm/pull/45163.

G. Tsoukalas, J. Lee, J. Jennings, J. Xin, M. Ding, M. Jennings, A. Thakur, and S. Chaudhuri. PutnamBench: Evaluating neural theorem-provers on the Putnam mathematical competition. In *Advances in Neural Information Processing Systems*, 2024.

Unsloth. DiffusionGemma. https://unsloth.ai/docs/models/diffusiongemma, 2026. Accessed: 2026-06-11.

M. Van Puyvelde, H. I. Gulluk, W. Van Criekinge, and O. Gevaert. Discrete diffusion language models for interactive radiology report drafting. *arXiv preprint arXiv:2607.01436*, 2026.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

B. Wallace, M. Dang, R. Rafailov, L. Zhou, A. Lou, S. Purushwalkam, S. Ermon, C. Xiong, S. Joty, and N. Naik. Diffusion model alignment using direct preference optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024.

C. Wu, H. Zhang, S. Xue, S. Diao, Y. Fu, Z. Liu, P. Molchanov, P. Luo, S. Han, and E. Xie. FastDLLM v2: Efficient block-diffusion LLM. *arXiv preprint arXiv:2509.26328*, 2025.

H. Xia, Z. Yang, Q. Dong, P. Wang, Y. Li, T. Ge, T. Liu, W. Li, and Z. Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Y. Yao, H. Zhou, A. Han, W. Huang, and M. Sugiyama. Accelerating discrete diffusion models with parallel-in-time sampling. *arXiv preprint arXiv:2607.00773*, 2026.

J. Ye, S. Gong, L. Chen, L. Zheng, J. Gao, H. Shi, C. Wu, X. Jiang, Z. Li, W. Bi, and L. Kong. Diffusion of thought: Chain-of-thought reasoning in diffusion language models. In *Advances in Neural Information Processing Systems*, 2024.

Q. Yi, X. Chen, C. Zhang, Z. Zhou, L. Zhu, and X. Kong. Diffusion models in text generation: a survey. *PeerJ Computer Science*, 2024.

T. Yin, M. Gharbi, R. Zhang, E. Shechtman, F. Durand, and T. Park. One-step image translation with text-to-image models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

X. Yue, T. Zheng, Y. Ni, Y. Wang, K. Zhang, S. Tong, Y. Sun, B. Yu, G. Zhang, H. Sun, Y. Su, W. Chen, and G. Neubig. MMMU-Pro: A more robust multi-discipline multimodal understanding benchmark. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2025.

T. Zadouri, M. Hoehnerbach, J. Shah, V. Thakkar, and T. Dao. FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. In *Proceedings of Machine Learning and Systems*, 2026.

E. Zelikman, G. Harik, Y. Shao, V. Jayasiri, N. Haber, and N. D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. In *Proceedings of the Conference on Language Modeling*, 2024.

S. Zhang, Y. Bao, and S. Huang. EDT: Improving large language models' generation by entropy-based dynamic temperature sampling. *arXiv preprint arXiv:2403.14541*, 2024.

D. Zhang-Li, N. Lin, J. Yu, Z. Zhang, Z. Yao, X. Zhang, L. Hou, J. Zhang, and J. Li. Reverse that number! Decoding order matters in arithmetic learning. *arXiv preprint arXiv:2403.05845*, 2024.

L. Zhao, X. Ding, L. Yu, and L. Akoglu. Unified discrete diffusion for categorical data. *Journal of Machine Learning Research*, 26(215):1–49, 2025.

S. Zhao, D. Gupta, Q. Zheng, and A. Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *Advances in Neural Information Processing Systems*, 2026.

Y. Zhao, Z. Xie, C. Liang, C. Zhuang, and J. Gu. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2024.

K. Zheng, H. Chen, H. Ye, H. Wang, Q. Zhang, K. Jiang, H. Su, S. Ermon, J. Zhu, and M.-Y. Liu. DiffusionNFT: Online diffusion reinforcement with forward process. In *Proceedings of the International Conference on Machine Learning*, 2026.

J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

K. Zhu, Y. Gao, Y. Zhao, L. Zhao, G. Zuo, Y. Gu, D. Xie, T. Tang, Q. Xu, Z. Ye, K. Kamahori, C.-Y. Lin, Z. Wang, S. Wang, A. Krishnamurthy, and B. Kasikci. NanoFlow: towards optimal large language model serving throughput. In *Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation*, 2025a.

R. Zhu, Z. Jiang, C. Jin, P. Wu, C. A. Stuardo, D. Wang, X. Zhang, H. Zhou, H. Wei, Y. Cheng, J. Xiao, X. Zhang, L. Liu, H. Lin, L.-W. Chang, J. Ye, X. Yu, X. Liu, X. Jin, and X. Liu. MegaScale-Infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In *Proceedings of the ACM SIGCOMM 2025 Conference*, 2025b.

T. Y. Zhuo, V. M. Chien, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, S. Brunner, C. GONG, J. Hoang, A. R. Zebaze, X. Hong, W.-D. Li, J. Kaddour, M. Xu, Z. Zhang, P. Yadav, N. Jain, A. Gu, Z. Cheng, J. Liu, Q. Liu, Z. Wang, D. Lo, B. Hui, N. Muennighoff, D. Fried, X. Du, H. de Vries, and L. V. Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *Proceedings of the International Conference on Learning Representations*, 2025.

Contributions and Acknowledgments (listed alphabetically)

Core contributors (workstream leads marked with '*')

Adrien Ali Taïga
James Assiene
Daniele Calandriello*
Rahma Chaabouni
João Gante*
Tamara von Glehn*
Nate Keating
Chris Knutsen
Martin Kukla*
Tianlin Liu
Ivan Lobov*
Ofir Nabati
João Gabriel Oliveira
Nicolas Perez-Nieves*
Nastasia Prutianova
Bobak Shahriari*
Jean Tarbouriech*
Pavel Tyletski
Çağlar Ünlü
Cindy Wu

Contributors

Glenn Cameron
Jerome Connor
Sertan Girgin
Maarten Grootendorst
Alon Levkovitch
Eliya Nachmani
Omar Sanseviero
Piotr Stanczyk

Finetuning framework

Quentin Berthet
Andrew Campbell
Clément Crepy
Valentin De Bortoli
Arnaud Doucet
Romuald Elie
Alexandre Galashov
Klaus Greff
Alexis Jacq
David Ruhe
Yu-Han Wu

Leads

Sebastian Flennerhag
Brendan O'Donoghue
George Scrivener
Shantanu Thakoor

Acknowledgements

Sander Dieleman
Lucas Dixon
Johan Ferret
Parnian Kassraie
Preethi Lahoti
Gaël Liu
Sarah Perrin
Angéline Pouget
Louis Rouillard
Pier Giuseppe Sessa
Danilla Sinopalnikov
Gemma Team

Sponsors

Olivier Bachem
Jeff Dean
Zoubin Ghahramani
Raia Hadsell
Demis Hassabis
Prateek Jain
Armand Joulin
Koray Kavukcuoglu
Marc'Aurelio Ranzato
Oriol Vinyals

A. Related work

A.1. Continuous Diffusion for Text

While discrete diffusion remains highly effective, recent advances indicate a resurgence of continuous and hybrid approaches. Flow-matching frameworks like Embedded Language Flows have demonstrated that by abandoning per-step token supervision and remaining in an unrestricted continuous embedding space until a final discretization step, continuous models can substantially outperform discrete baselines (Hu et al., 2026). Building on this continuous paradigm, recent advances in flow maps for discrete data have shown that by aligning training dynamics with the geometry of the probability simplex, these generative trajectories can be compressed into single-step mappings, achieving high-quality parallel language generation in one or a few steps (Lee et al., 2026; Potaptchik et al., 2026). Similarly, models utilizing pretrained contextual autoencoders (e.g., Cosmos, TEncDM) map text into compressed, smooth latent spaces where Gaussian diffusion operates efficiently without rounding errors, providing a continuous manifold that lends itself well to advanced guidance techniques (He et al., 2024; Meshchaninov et al., 2025; Shabalin et al., 2025), while recent approaches like hyperspherical flows avoid Gaussian corruption by rotating token embeddings on a hypersphere to better match the geometric structure of language (Deschenaux and Gulcehre, 2026b). Other hybrid frameworks, such as CANDI (Pynadath et al., 2026), address the "temporal dissonance" of applying Gaussian noise to discrete data by decoupling discrete and continuous corruption, allowing the model to simultaneously learn conditional structure and continuous geometry. Concurrently, Score Entropy Discrete Diffusion has bridged these domains by applying continuous-time Markov chains to learn the probability ratios of discrete data distributions (Lou et al., 2024). These breakthroughs suggest that the initial failures of continuous text diffusion may have been artifacts of sub-optimal spatial geometries and restrictive training objectives rather than inherent modality limitations.

A.2. Speculative Decoding

Speculative decoding (Chen et al., 2023a; Leviathan et al., 2023; Xia et al., 2024) reduces serving latency by utilizing a smaller draft model to propose token sequences that are verified in parallel by a larger target model. The overall latency depends on the generation time of both the draft and target models. AR drafters (Li et al., 2026) are fundamentally constrained by sequential generation: increasing draft quality requires more parameters, which increases per-token-latency and diminishes end-to-end gains. Parallel drafters such as Medusa (Cai et al., 2024) circumvent this by generating many tokens at once, but do so independently, yielding suboptimal acceptance rates. Diffusion-based drafters recover inter-token dependencies by instead modeling the joint distribution over draft tokens. TiDAR (Liu et al., 2026b) uses a single model for diffusion-based drafting, and AR verification—a configuration that DiffusionGemma also supports—while DFlash (Chen et al., 2026) employs a separate model as the drafter. However, it exhibits suffix decay, with acceptance rates declining at later draft positions (Cheng et al., 2026). DSpark (Cheng et al., 2026) addresses this by adding a lightweight AR module on top of the parallel backbone. In contrast, DiffusionGemma operates as a standalone diffusion model, eliminating the verification bottleneck and scaling to longer generated canvases than draft-then-verify approaches.

B. Samples Before and After SD·RL Training

To illustrate the impact of our SD·RL phase, Figures 16, 17 contrast representative generations from the SFT model and the final checkpoint. They show how SD·RL training resolves the severe repetitive looping observed in the SFT baseline, allowing the model to complete complex reasoning traces.

Prompt You perform a high-throughput experiment on white lupine to find genes contributing to resistance to the fungal disease anthracnose. You receive three candidate genes of unknown function – G1, G2, and G3. You create three knock-out mutants (g1, g2, g3) and double-mutants (g1g2, g1g3, g2g3). You know that at least one of these genes is a transcription factor acting upstream of other gene(s). After testing with the pathogen, you receive the following results (100% = wild-type resistance):

Genotype	Resistance	Genotype	Resistance	Genotype	Resistance
g1	75%	g1g3	10%	g1g2	0%
g2	0%	g2g3	0%		
g3	50%				

Which conclusion can you draw?
(A) G2 is a TF, G1&G3 show pleiotropy, G1 epistatic towards G3
(B) G2 is a TF, G1&G3 show gene redundancy, G1 epistatic towards G3
(C) G2 is a TF, G1&G3 share a promoter, G3 epistatic towards G1
(D) G1 is a TF, G2&G3 show pleiotropy, G2 epistatic towards G1

***Correct answer:* (B)**

Before SD·RL × Incorrect (empty)

Thinking (truncated):

- Wild-type (control): 100% resistance.
  - g1 mutant: 75% resistance (slight decrease).
  - g2 mutant: 0% resistance (complete loss).
  - g3 mutant: 50% resistance (moderate decrease).
  - g1g3 double mutant: 10% resistance (significant decrease, more than g1 or g3 alone).
  - g2g3 double mutant: 0% resistance (same as g2).
  - g1g2 double mutant: 0% resistance (same as g2). [...]
  - If G2 is a transcription factor (TF) that activates G1 and G3:
    - If G2 is G, G1 and G3 are targets.
    - If G2 is knocked out, G1 and G3 are not expressed.
    - So, g2 would have the same phenotype as g2g1 and g2g3.
    - This matches the data: g2 (0%), g2g1 (0%), g2g3 (0%).
  - Now, let's look at G1 and G3 again:
    - g1: 75%
    - g3: 50%
    - g1g3: 10%
    - If G1 and G3 act in parallel pathways (pathways„ loss1111111111111111111111 G11 G G11 G G G1 G G1 G G G G G G G1 G1 G G1 G G G G G G G1 G G G G G G111 G111 G G G1111 G111111111 G1111 G G1111111111111111111

[. . . repeated 1 tokens until the max tokens budget.]

Response:

[empty — no response produced]

After SD·RL ✓ Correct

Thinking (truncated):

- Wild-type (control): 100% resistance.
  - g1: 75% resistance (slightly reduced).
  - g2: 0% resistance (completely sensitive).
  - g3: 50% resistance (moderately reduced).
  - g1g3: 10% resistance (highly sensitive).
  - g2g3: 0% resistance (completely sensitive).
  - g1g2: 0% resistance (completely sensitive).

g2 has 0% resistance. This means G2 is essential for the resistance. g1g3 has 10%, much lower than both g1 (75%) and g3 (50%). This suggests that G1 and G3 work in parallel pathways (redundancy).
If G2 is a TF upstream of G1 and G3, then knocking out G2 (g2) would prevent the expression of both G1 and G3. This would explain why g2, g2g3, and g1g2 all have 0% resistance.

[...re-examines epistasis definitions, checks all options, confirms (B)...]

Given the options, (B) is the only one that correctly identifies G2 as a transcription factor and G1/G3 as redundant.

Response:

To determine the correct conclusion regarding the genes G1, G2, and G3, we analyze the experimental resistance data provided:

1. **Wild-type (control):** 100% resistance.
2. **g1 mutant:** 75% resistance (G1 contributes slightly to resistance). *[...]*

Analysis of the interactions:

- **Transcription Factor:** If G2 is a transcription factor upstream of both G1 and G3, knocking out G2 would prevent the expression of both G1 and G3, *[...]*

Final Answer: (B)

Figure 16 | **Example from GPQA-Diamond (Biology) showing model behavior with the DiffusionGemma sampler before and after SD·RL training.** The SFT model begins with *correct* reasoning—identifying G2 as an essential transcription factor and recognizing G1/G3 redundancy—but degenerates into a repetitive token loop (the digit '1'), producing an empty final response. After SD·RL, the model completes the analysis cleanly and outputs the correct answer.

Prompt A quantum mechanical particle of mass m moves in two dimensions in the following potential, as a function of (r, θ):

$$V(r,\theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta)$$

Find the energy spectrum.

(A) $E = (n_x + 3n_y + \frac{3}{2})\,\hbar\sqrt{k/m}$

(B) $E = (2n_x + n_y + \frac{3}{2})\,\hbar\sqrt{k/m}$

(C) $E = (3n_x + 2n_y + \frac{1}{2})\,\hbar\sqrt{k/m}$

(D) $E = (2n_x + 3n_y + \frac{1}{2})\,\hbar\sqrt{k/m}$

Correct answer: **(B)**

Before SD·RL × **Incorrect (empty)**

Thinking (truncated):

The potential is given in polar coordinates (r, θ) as: $V(r,\theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta)$. We know:

$x = r\cos(\theta)$,
$y = r\sin(\theta)$,
$r^2 = x^2 + y^2$,
$\cos^2(\theta) = x^2/r^2$.

Substitute these into the potential expression:

$V(x,y) = \frac{1}{2}k(x^2+y^2) + \frac{3}{2}kx^2 = 2kx^2 + \frac{1}{2}ky^2$

[...]

Plugging in the values for ω_x and ω_y:

[...]

The calculated energy spectrum $E = (2n_x + n_y + \frac{3}{2})\,\hbar\sqrt{k/m}$, which matches option (B).

[... re-verifies answer three additional times, all correctly arriving at (B) ...]

Let's re-read the options and . = (= =xx_nn__33/))))))))))222____y +//))))))))))))))))))2 2 2222)))

[... repeated closing-parenthesis until max token budget.]

Response:

[empty — no response produced]

After SD·RL ✓ **Correct**

Thinking (truncated):

The potential is $V(r,\theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta)$.

The coordinates are polar (r, θ).

$x = r\cos\theta$
$y = r\sin\theta$
$r^2 = x^2 + y^2$
$\cos\theta = \frac{x}{r}$
$V(r,\theta) = \frac{1}{2}k(x^2+y^2) + \frac{3}{2}k(x^2+y^2)(\frac{x}{r})^2$

[...]

The Hamiltonian is separable into two one-dimensional harmonic oscillators:

$H = H_x + H_y$

[...]

For a 1D harmonic oscillator $H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2x^2$, the energy eigenvalues are: $E = (n+\frac{1}{2})\hbar\omega$, where $n = 0, 1, 2, \ldots$

[...]

The calculated $E = (2n_x + n_y + 3/2)\hbar\sqrt{k/m}$ matches option (B).

[... verifies calculation twice, confirms match with option (B).]

Response (excerpt):

To find the energy spectrum of the quantum mechanical particle of mass m in the given potential $V(r,\theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta)$, we first convert the potential into Cartesian coordinates (x, y).

1. **Convert the potential to Cartesian coordinates:** The relationship between polar and Cartesian coordinates is:
[...]
2. **Identify the Hamiltonian:** The Hamiltonian for a particle in two dimensions is: *[...]*
3. **Calculate the energy eigenvalues for each oscillator:** The energy levels of a 1D harmonic oscillator $\frac{1}{2}m\omega^2x^2$ are $E = (n+\frac{1}{2})\hbar\omega$.
[...]
4. **Find the total energy:**
[...]

$E = (2n_x + n_y + \frac{3}{2})\hbar\sqrt{\frac{k}{m}}$

Final Answer: (B)

Figure 17 | **Example from GPQA-Diamond (2D harmonic oscillator energy spectrum) showing model behavior with the DiffusionGemma sampler before and after SD·RL training.** The SFT model begins with a *correct* derivation—converting to Cartesian coordinates, identifying the separable Hamiltonian, and computing the correct energy spectrum—but degenerates into a repetitive token loop (closing parentheses), producing an empty final response. After SD·RL, the model completes the same derivation cleanly and outputs the correct answer.

C. Additional Open-Source Downstream Finetuning Results

In Section 8, we describe our finetuning strategy as well as our main results on Sudoku puzzle solving. We now provide full training recipes as well as additional results for PubMedQA (Jin et al., 2019).

Sudoku solving trace summary. In Figure 18, we present one trace summary for a successful solving of a Sudoku puzzle.

Practical recipe summary. Table 7 summarizes the key hyperparameters used in Sudoku and PubMedQA. Full training and evaluation code is available open-source via the Hackable Diffusion adapter. For our LoRA strategy to update all the linear layers, using rank 8 for Sudoku solving problem, we only finetune 8M parameters.

Case study: PubMedQA. To demonstrate applicability beyond structured reasoning, we finetune DiffusionGemma on PubMedQA (Jin et al., 2019), a biomedical question-answering benchmark where the model must read a medical research abstract and produce both a categorical answer (*yes*, *no*, or *maybe*) and a detailed explanatory paragraph. The model is evaluated only on the long task using BLEU score against reference explanations, showing that DiffusionGemma can be effectively adapted to domain-specific natural language generation tasks with minimal data and compute.

Model	Effective Denoising Steps	Accuracy (%)	BLEU
DiffusionGemma	18.09	75.6	10.76
+ LoRA finetuning	31.57	76.62	20.67

Table 6 | **PubMedQA performance of the finetuned model with LoRA rank 4.** Finetuning leads to a slight increase in accuracy on a model with a good base performance.

Hyperparameter	**Sudoku (LoRA)**	**Sudoku (Full)**	**PubMedQA**
LoRA rank	8	—	4
Canvas size	256	256	128
Number of canvases	1	1	2
Prompt length	256	256	1024
Batch size	2	8	2
Peak learning rate	3×10^{-4}	1.125×10^{-4}	1.0×10^{-4}
End learning rate	3×10^{-5}	1.125×10^{-5}	1.0×10^{-5}
Training steps	8,000	2,000	2,000
Optimizer	Adam	Adafactor	Adam
LR schedule	Cosine with warmup	Cosine with warmup	Cosine with warmup
Warmup iterations	400	100	100
Weight decay	10^{-4}	10^{-4}	10^{-4}
Min. hardware	2× A100 80GB	8× A100 80GB	2× A100 80GB

Table 7 | **Summary of downstream finetuning hyperparameters for the Sudoku and PubMedQA recipes.** Both LoRA recipes use adapters applied to all linear layers and train for 2,000 steps. A full-weight alternative is also provided for Sudoku, using Adafactor to manage memory on 8 GPUs.

Prompt

System: Solve the following Sudoku puzzle. Empty cells are represented by 0. Output ONLY the solved puzzle immediately as a 9x9 grid of numbers separated by spaces. Do not include ####, explanations, or any other text.

User: 4 0 0 0 0 1 0 0 6 0 2 1 8 0 9 0 0 0 0 0 9 0 4 0 2 0 0 0 0 0 0 0 0 4 8 6 0 0 0 0 5 0 2 7 4 1 9 9 0 7 5 1 0 3 0 0 6 1 3 2 9 0 5 0 0 5 0 9 0 8 0 0 6 4 2 8 4 6 7 0 1 9 0

Before Downstream Finetuning × Incorrect

Thinking (truncated):

```
<|turn>model
<|channel>thought
<channel|>4 8 3 7 5 1 9 2 6
7 2 1 8 6 9 5 3 4
6 9 5 4 3 2 8 1 7
1 4 8 9 9 6 3 7 5
3 5 6 2 7 4 1 9 8
9 7 2 5 1 3 4 6 1
2 3 9 1 4 5 7 8 6
5 6 7 1 2 8 4 4 3
8 1 4 6 7 6 2 9 5

Wait, let me re-solve that carefully as the input string was
a bit fragmented.
Input:
4 0 0 | 0 0 1 | 0 0 6
0 2 1 | 8 0 9 | 0 0 0
0 9 0 | 4 0 2 | 0 0 0
0 0 0 | 0
[... truncated]
```

Response:

[empty — answer cut]

After Downstream Finetuning ✓ Correct

Thinking:

[empty — successfully suppressed thought channel]

Response:

```
4 5 8 7 3 1 9 2 6
3 2 1 8 6 9 7 4 5
7 9 6 4 5 2 8 3 1
1 3 2 9 4 8 6 5 7
8 6 5 3 2 7 4 1 9
9 4 7 5 1 6 3 8 2
6 1 3 2 9 4 5 7 8
5 7 9 1 8 3 2 6 4
2 8 4 6 7 5 1 9 3
```

Figure 18 | **Example evaluating formatting rule-following on a Sudoku puzzle before and after Open-Source Downstream Finetuning.** Before Open-Source Downstream Finetuning, the model fails to follow the strict negative constraints (“Output ONLY the solved puzzle immediately”), improperly attempting to build an erroneous grid in its thought channel and failing to provide a final response. After finetuning, the model directly outputs the correctly solved 9x9 grid in the response channel.

D. Prompt Formatting

To facilitate complex, multi-turn interactions, multimodal inputs, and agentic workflows, the model employs a structured Jinja2-based chat template. This template serializes the conversation history, system instructions, and tool schemas into a standardized string format using specialized control tokens. Explicit formatting ensures the model can accurately distinguish between user inputs, internal reasoning, tool invocations, and system-level contexts. What follows is a summary of the main features, but please refer to the implementation[6] for full details.

D.1. BOS and EOS Special Tokens

As in other Gemma models, every conversation must start with a special BOS token, which has no text rendering but corresponds to token integer 2. The BOS special token must either be manually prepended to the tokenized input or usually by the tokenizer via an `add_bos=True` keyword argument. At the other end, when a block-AR generation completes, it ends in the usual special token `<turn|>` followed by padding with the EOS token. Similarly, the EOS token corresponds to the integer 1 but has no text rendering when detokenized.

D.2. Conversational Structuring

The template strictly delineates conversation turns using opening and closing tags. `<|turn>` and `<turn|>` tokens encapsulate individual messages, and the opening token is appended with the specific role (e.g., `<|turn>system\n`). The model expects four roles: `system`, `user`, `model`, and `tool` for tool call responses.

The template natively supports multimodal routing by parsing content arrays for specific media types, injecting `<|image|>` tokens into the context stream where appropriate. In the forward pass of the model, these tokens are replaced by the input image features, as opposed to the corresponding token embedding.

D.3. Thinking Channels

As with the Gemma 4 models, DiffusionGemma supports thinking mode, which can be enabled by adding the thinking token `<|think|>` to the system instruction. When thinking is enabled, the model will emit an internal reasoning channel (which could technically still be empty) followed by the final answer:

```
<|channel>thought
# DiffusionGemma's internal reasoning goes here.
<channel|>
# DiffusionGemma's final answer goes here
```

Importantly, even when the `<|think|>` token is not present in the system instruction, the model will still emit an empty thought channel as follows:

```
<|channel>thought
<channel|>
```

[6] https://huggingface.co/google/diffusiongemma-26B-A4B-it/blob/main/chat_template.jinja

```
[final answer]
```

For multi-turn conversations, do **not** include previous hidden thoughts in the conversation history. Only include the final assistant response before the next user turn.

D.4. Tool and Function Calling Serialization

A significant portion of the template is dedicated to parsing and serializing JSON-like tool schemas into a compact, token-efficient format. All JSON-like schemas, e.g. tool definitions or tool responses, rely on the custom delimiter `<|"|>` (as opposed to a plain `"`) for clarity.

- `<|tool>` and `<tool|>`: Used within the **system prompt** to define available function schemas, including their descriptions, parameters, and required arguments.
- `<|tool_call>` and `<tool_call|>`: Model requests to invoke tools are formatted as `<|tool_call>call:function_name{arguments}<tool_call|>`.
- `<|tool_response>` and `<tool_response|>`: Results returned from external tools are appended to the context window wrapped by these tokens, allowing the model to seamlessly integrate external data into its subsequent turns.

E. Mercury 2 Speed Estimation

Estimation data. As Mercury 2 (Inception Labs, 2026) is a closed source model, we do not have direct access to generate speed metrics such as TPF and TPS. We take a black-box approach and estimate TPS by querying OpenRouter's API [7] with the default maximum sequence (input+output) length of 50,000. We ran two independent rounds of queries — the first on July 9–10, 2026 (with a small number of retries on July 11), the second round between July 25–26, 2026 — and obtained consistent results across both measurements. Each API response includes the following metadata:

- the number of input tokens (prompt tokenization);
- how many prompt tokens were cached (KV-cache hits vs. fresh prefill);
- total output tokens (split into thinking tokens and answer tokens);
- the wall-clock time for the request (including network latencies and other overheads).

Response metadata does not include a breakdown of generation speed. We estimate generate TPS via a least-squares model, detailed below.

Estimation model. We estimate per-token generation speed by fitting the following model via non-negative least-squares (NNLS):

$$\text{Total Time} = \alpha \cdot \text{effective\_prefill\_tokens} + \beta \cdot \text{total\_output\_tokens} + \gamma, \tag{14}$$

where effective_prefill_tokens = prompt_tokens – cached_tokens is the number of tokens requiring fresh computation, α is the per-token prefill time, β is the per-token generation time, and γ captures constant per-request overhead (network latency, etc.). From the fitted β, we obtain the generation speed as $1/\beta$ TPS. The non-negativity constraint reflects that processing tokens, generating tokens, and per-request overhead can only add time. In practice, we disable caching across queries.

[7] https://openrouter.ai

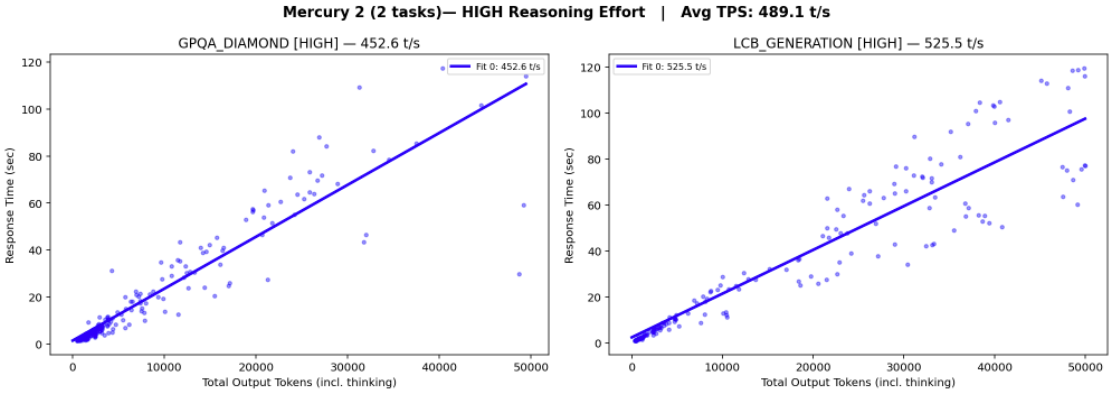

Figure 19 | **Mercury 2 token throughput (two benchmarks, `high` reasoning effort).** Including outliers where the model produces no answer tokens (i.e., outputting thinking tokens only). Evaluated on GPQA Diamond and LiveCodeBench-v6; used to compute the speed reported in Figure 1. The average of per-task TPS estimates yields 489 TPS.

In order to capture any potential per-task variation in generation speed (e.g. from adaptive computation methods), we fit Equation (14) independently per benchmark, before taking an arithmetic average. Similarly, for open weight models we compute aggregate TPS measurements by first measuring per-task generative speed and then taking the arithmetic average.

Speed estimates. Our speed measurements for Figure 1 are based on the GPQA-Diamond and LiveCodeBench-v6 datasets. For Mercury 2, we estimate the generative speed per task using the above methodology. Figure 19 shows the corresponding analysis; we obtain a TPS of 452.6 for GPA Diamond and 525.5 TPS for LiveCodeBench-v6, which translates into an average speed of 489 TPS.

Our speed measurements for Table 3 are based on seven benchmarks where we have measurements for all models: AIME 2026, GPQA Diamond, HumanEval, LBPP, LiveCodeBench-v6, MGSM, and Natural2Code. Again, for Mercury 2 we fit our speed estimation model on each task separately first before taking an arithmetic average. Figures 20 and 21 report the corresponding analysis; we obtain average TPS estimates of 600 TPS for `high` reasoning effort and 547 TPS for `medium` reasoning effort. We note that some evals (notably HumanEval and Natural2Code) have outliers that generate 50,000 tokens at extremely high speeds. These give a slightly favorable bias to our generation speed estimates and also explain why `high` reasoning effort have a higher TPS than `medium` reasoning effort. These outliers represent corrupted outputs where the model gets stuck in a thinking loop and exhaust the maximum generation length without returning a valid response.

Inception reports ~1000 TPS on NVIDIA Blackwell GPUs[8] while Artificial Analysis[9] reports a median of ~987 TPS on undisclosed hardware. The difference to our estimate can be due to a number of reasons—in particular different hardware, serving optimizations, and/or different datasets used for measurement (this may have a large impact due to adaptive computation). It is worth noting that Artificial Analysis also reports substantial variance. Our estimate reflects the average speed a user would experience via the OpenRouter API.

[8] https://www.inceptionlabs.ai/blog/introducing-mercury-2, retrieved July 28, 2026.
[9] https://artificialanalysis.ai/models/mercury-2, retrieved July 28, 2026.

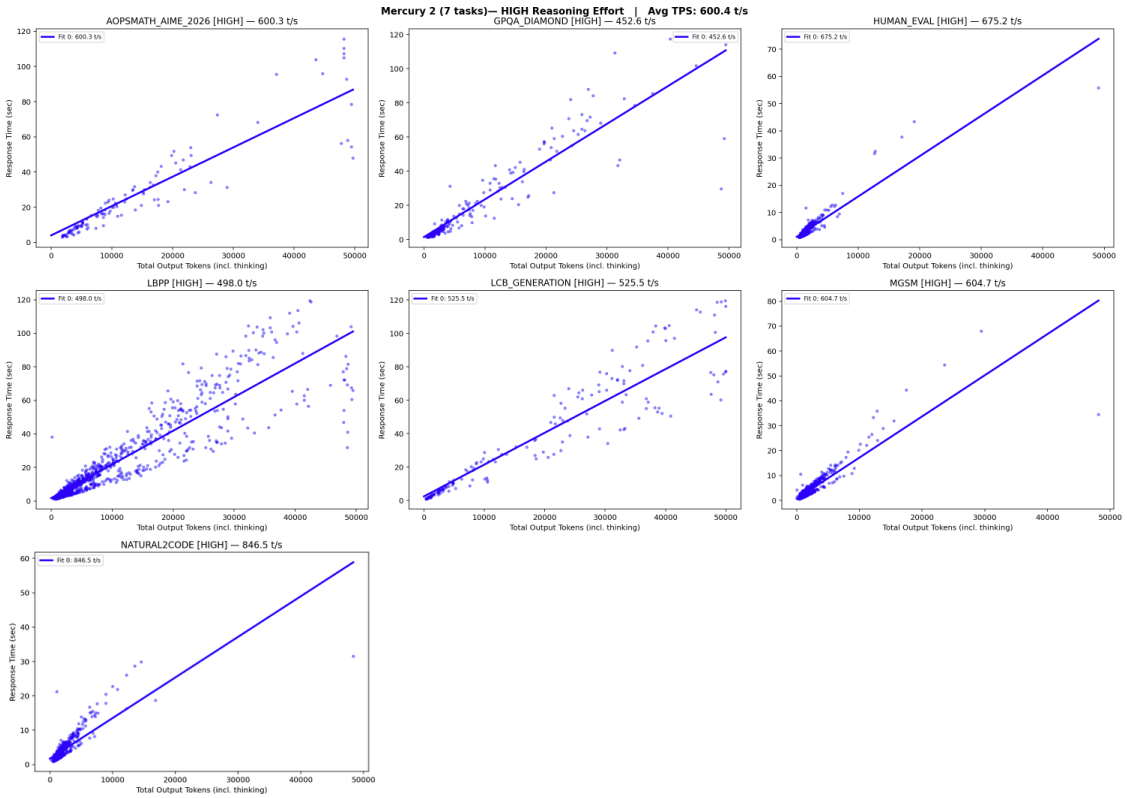

Figure 20 | **Mercury 2 token throughput (seven benchmarks, `high` reasoning effort).** Including outliers where the model produces no answer tokens (i.e., outputting thinking tokens only). Used to compute the `high` effort speed reported in Table 3. Average TPS estimates yields 600 TPS.

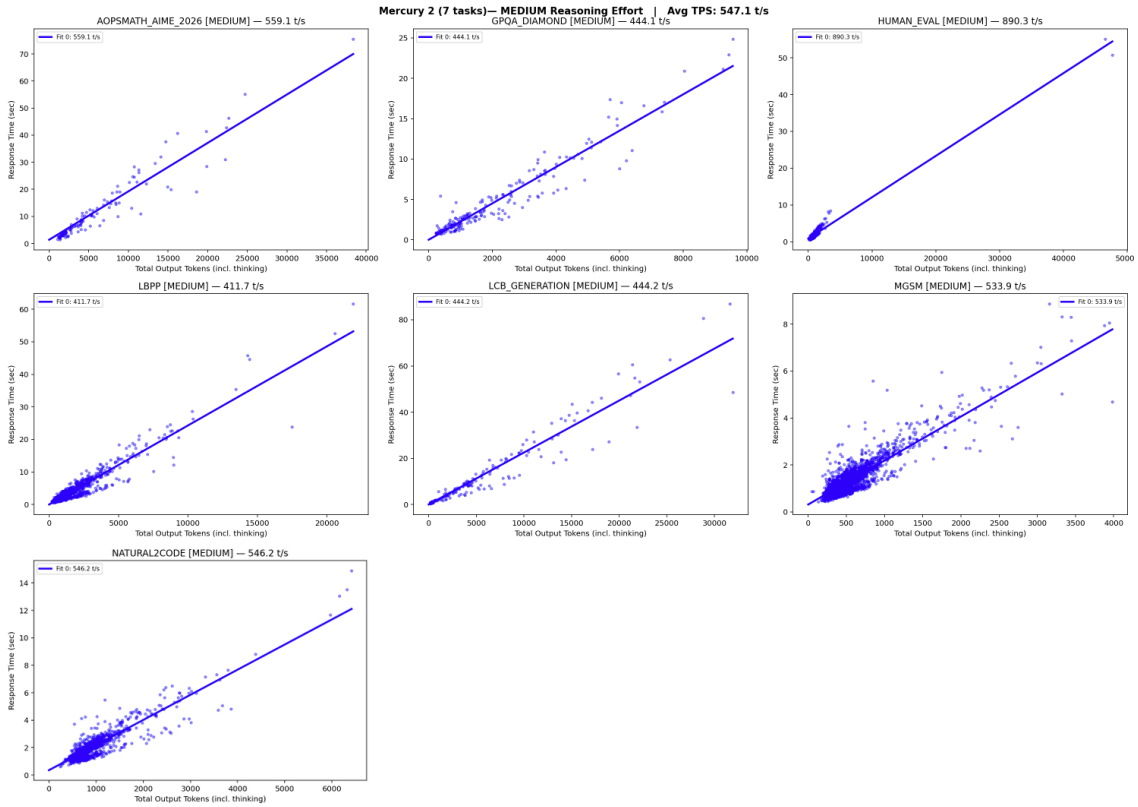

Figure 21 | **Mercury 2 token throughput (seven benchmarks, `medium` reasoning effort).** Including outliers where the model produces no answer tokens (i.e., outputting thinking tokens only). Used to compute the `medium` effort speed reported in Table 3. Average TPS estimates yields 547 TPS.

F. Denoising trajectory sampled from DiffusionGemma

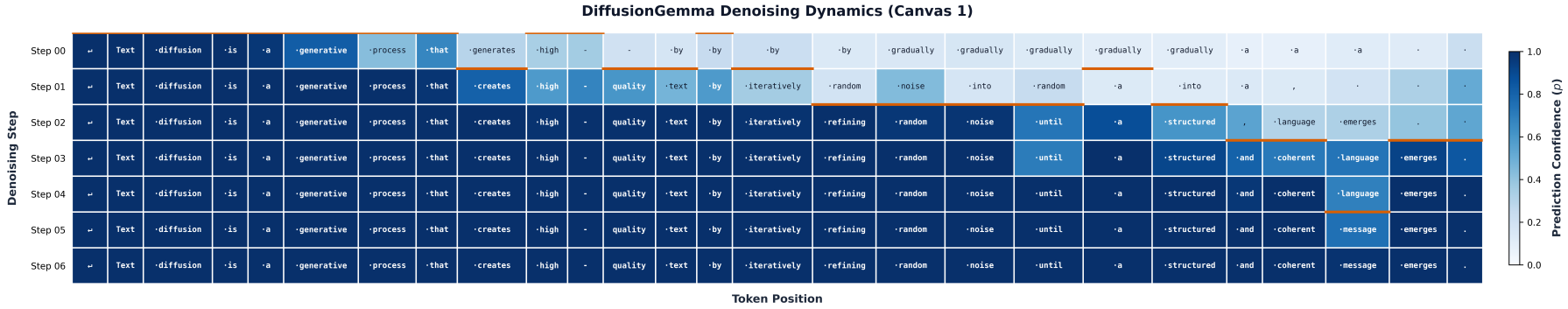

Figure 22 | **Example denoising trajectory sampled from DiffusionGemma**. Prompt: “Please write a short sentence describing text diffusion.” The color intensity denotes model confidence. The final response is “Text diffusion is a generative process that creates high-quality text by iteratively refining random noise until a structured and coherent message emerges.” which requires 7 denoising steps (forward passes) to generate.

G. More on Practical Advantages of Text Diffusion

G.1. Bidirectional Reasoning and Self-Correction

Consider the frog crossing puzzle in Figure 23. This example highlights the necessity of self-correction in logical deduction. The prompt introduces a deceptive mathematical trap—an infinite loop—that the AR baseline blindly falls into, generating a contradictory “Yes”. DiffusionGemma’s parallel generation exhibits a distinct self-correction trajectory. While it initially assigns high probability to the intuitive but incorrect “Yes”, the bidirectional propagation of the logical constraints across the sequence enables the model to revise its stance and cleanly output “No” before the final tokens are sampled.

Prompt

"A frog is trying to cross a 25-foot pond. Each hop covers exactly 5 feet and takes 1 minute. However, the lily pad located at exactly the 20-foot mark is magical: if the frog lands on it, it instantly teleports the frog backward to the 5-foot mark. Is it possible to reach the 25-foot pad?"

Gemma 4 26B A4B: Yes, it is possible. The frog can reach the 25-foot mark by avoiding the 20-foot mark. Since the frog's hops are exactly 5 feet, it would land on the 5, 10, 15, 20, and 25-foot marks. To avoid the 20-foot mark, the frog would need a different jump distance, but based on the strict rule of "exactly 5 feet" per hop, it is impossible because the frog is forced to land on the 20-foot mark to reach the 25-foot mark.

DiffusionGemma: No. To reach 25 feet, the frog must land on the 20-foot mark, which teleports it back to 5 feet, creating an infinite loop.

Analysis: Gemma 4 commits to "Yes" without foreseeing the mathematical trap, resulting in a response that contradicts its own opening statement. DiffusionGemma converges cleanly to the correct "No" since the response and the reasoning can co-evolve over denoising. Note that both models are operating in non-thinking mode.

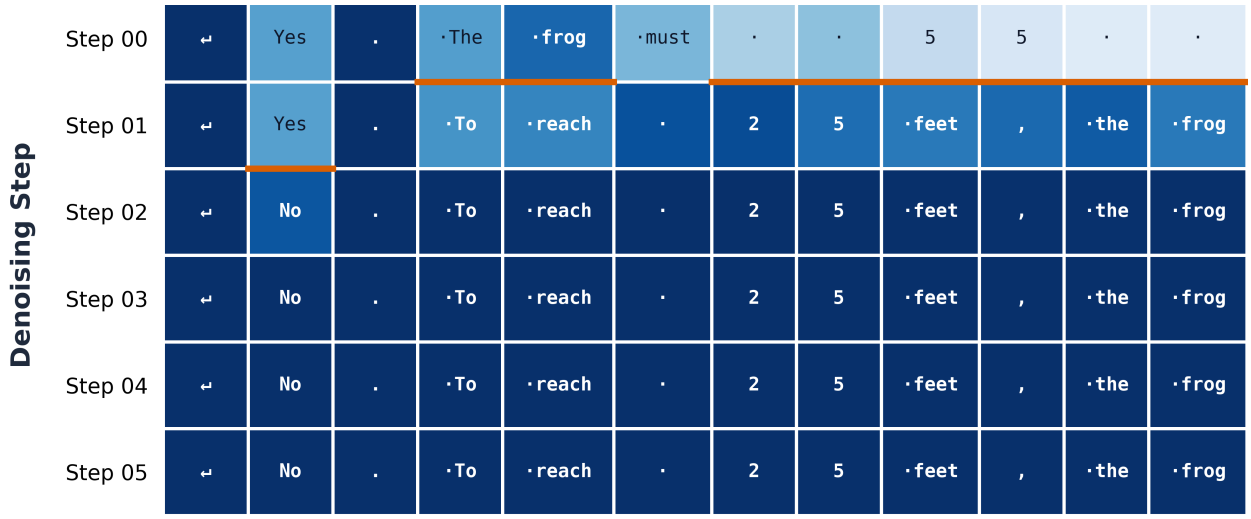

Figure 23 | **Denoising trace for the frog puzzle (zoomed to first 12 tokens), where color intensity indicates model confidence**. DiffusionGemma initially assigns high probability to the intuitive but incorrect "Yes." After several denoising steps, bidirectional attention allows the model to realize the task is impossible and revises its initial prediction to "No". Denoising is completed in 6 steps.

G.2. More on Dynamic and Adaptive Computation

To concretely demonstrate the adaptive behavior of text diffusion, we contrast a structurally "hard" generation task against an "easy" task in Figures 24 and 25. Both tasks involve generating a sequence of binary digits governed by identical local logical rules. However, in the structurally hard variant (Figure 24), each new token strictly depends on the two immediately preceding *generated* tokens. This causal dependency requires sequential reasoning, forcing the model to resolve the logic in a predominantly left-to-right manner. Consequently, the diffusion process adaptively expends more computational effort, requiring 7 denoising steps to fully resolve the sequence. It is crucial to note, however, that even on this structurally difficult, sequential problem, DiffusionGemma successfully decodes the entire sequence of 40 tokens (20 binary digits and 20 delimiting spaces) in only 7 denoising steps—a substantial acceleration compared to the 40 discrete sequential steps an AR model would require.

Conversely, the "easy" convolutional variant (Figure 25) asks the model to apply the exact same logic rules over a statically provided *input* string. Because the causal dependency on the model's own dynamic output is removed, the task lacks sequential structure. DiffusionGemma immediately leverages its bidirectional attention to independently resolve all local rules in parallel, allowing the entire output sequence to converge simultaneously in merely 4 denoising steps.

Prompt

"Generate a sequence of 20 tokens (0s and 1s) based on the following strict rules. To generate the next token, look at the two immediately preceding tokens: If the previous two tokens are 1, 1, the next token is 0. If the previous two tokens are 1, 0, the next token is 1. If the previous two tokens are 0, 1, the next token is 1. If the previous two tokens are 0, 0, the next token is 0. The starting sequence is: 0 1 Continue the sequence step-by-step until you reach exactly 20 tokens. Only emit the tokens, do not emit any other text."

DiffusionGemma: 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1

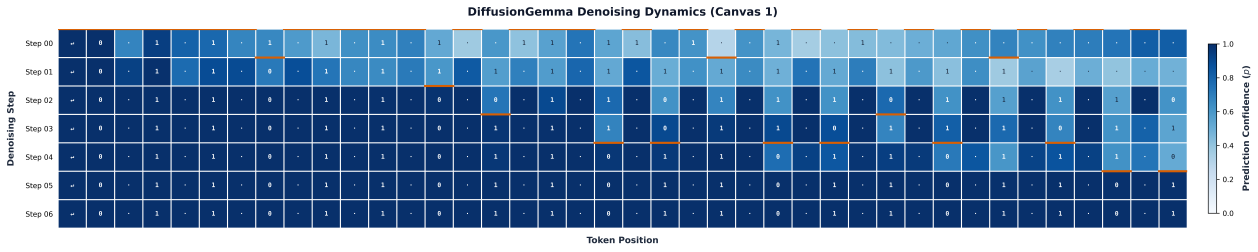

Figure 24 | **Denoising trace for the sequential dependency task**. Because there is a strict sequential dependency in the target output, the tokens are organically resolved in roughly a left-to-right ordering. The model adaptively allocates more compute to this "hard" task, taking 7 denoising steps to resolve the entire sequence of 40 tokens.

Prompt

"I will provide an input sequence of 20 binary digits. I want you to output a new sequence of the exact same length based on the following strict rules. To generate the output token at position n, look at the input sequence at position n and the input sequence at position n−1. If the two input tokens are 1, 1, the output token is 0. If the two input tokens are 1, 0, the output token is 1. If the two input tokens are 0, 1, the output token is 1. If the two input tokens are 0, 0, the output token is 0. For the very first token, assume the "previous" input token was a 0. Input sequence: 0 1 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0. Only emit the tokens, do not emit any other text."

DiffusionGemma: 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 1

Analysis: Unlike the sequential task, this prompt asks the model to apply rules over a static input string. Because the causal dependency on prior *outputs* is removed, DiffusionGemma can successfully resolve all local rules in parallel.

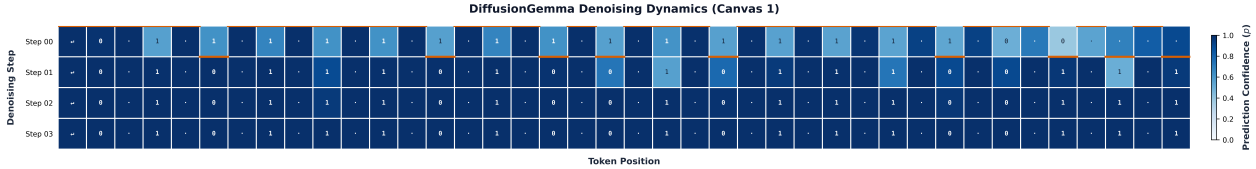

Figure 25 | **Denoising trace for the convolutional dependency task**. Because the rules depend entirely on the statically provided input sequence rather than the model's own dynamic output, bidirectional attention allows DiffusionGemma to evaluate the "easy" task in parallel, only requiring 4 denoising steps to converge.

G.3. Structured and Constrained Outputs

Prompt

```
You are a precise data extraction assistant.  Your task is to extract
information from the user's text and return it strictly as a JSON object
matching the schema below.  Do not include any conversational text,
explanations, or markdown formatting.  Output only the raw JSON object.
```

Schema:

```
  {
  "type":  "object",
  "properties":  {
    "first_name":  { "type":  "string" },
    "last_name":  { "type":  "string" },
    "age":        { "type":  "integer" },
    "hobbies":    { "type":  "array",
           "items":  { "type":  "string" } },
    "is_student":  { "type":  "boolean" }
  },
  "required":  ["first_name", "last_name",
           "age", "hobbies", "is_student"]
}
```

Input text: "Hi there, my name is Arthur Pendelton. I just turned 34 last week. In my free time, I really enjoy woodworking, baking sourdough bread, and cycling. I currently work full-time as an accountant, having finished my master's degree five years ago."

Figure 26 | Prompt used for the structured JSON extraction example detailed below.

Figure 27 | **DiffusionGemma structured generation.** Per-token prediction confidence across denoising steps for a JSON output task. Color encodes model confidence p. Because the target output is heavily dictated by the provided JSON schema, text diffusion leverages parallel generation to converge almost instantaneously. The initial canvas (left) is largely correct and exhibits an average confidence exceeding 83% per token. By the second denoising step (right), the output has fully converged to 100% confidence, allowing the final response to be returned in just two steps.

Prompt

```
Please fix this buggy python code.  Output only the fixed code, nothing
else:

def get_even_metrics(numbers):
    """Returns the total count of even numbers and a list of them."""
    # Find all even numbers
    evens = (n for n in numbers if n % 2 == 0)

    # Calculate metrics
    count = sum(1 for _ in evens)
    even_list = list(evens)

    return count, even_list
```

Figure 28 | Prompt used for the buggy Python code editing example.

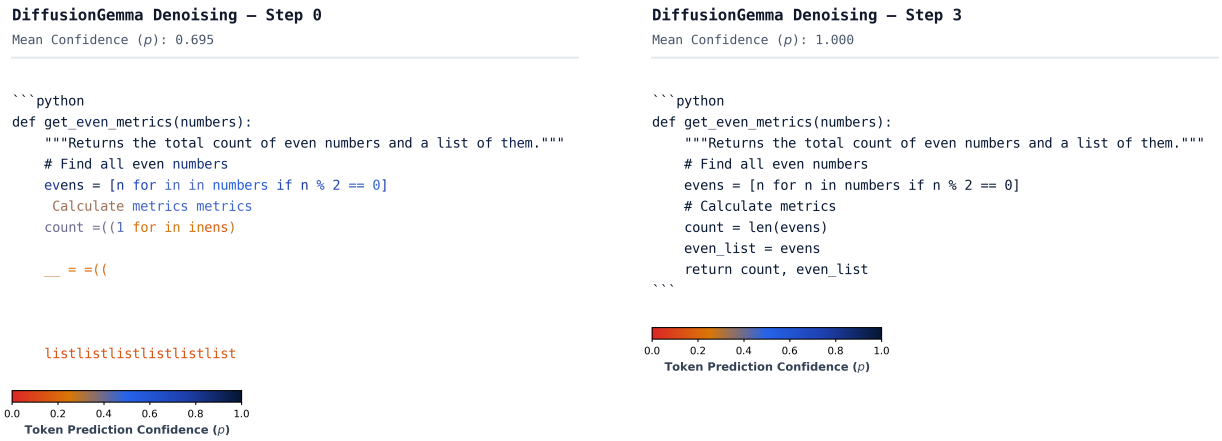

Figure 29 | **DiffusionGemma code editing.** Per-token prediction confidence across denoising steps for a code debugging task. Color encodes model confidence p. Because the output space is strictly constrained by the input—the vast majority of tokens are copied verbatim, and only a localized semantic fix is required—the diffusion process converges rapidly. The corrected, bug-free implementation is generated in just three denoising steps.