salesforce research

CoDA: Coding LM via Diffusion Adaptation

Haolin Chen[*,†], **Shiyu Wang**[*,†], **Can Qin**[*,†], **Bo Pang**[†], **Zuxin Liu**[†], **Jielin Qiu**[†], **Jianguo Zhang**[†], **Yingbo Zhou**[†], **Zeyuan Chen**[†], **Ran Xu**[†], **Shelby Heinecke**[†], **Silvio Savarese**[†], **Caiming Xiong**[†], **Huan Wang**[†], **Weiran Yao**[†]

[†]Salesforce AI Research [*]Core Contributors

Abstract. Diffusion language models promise bidirectional context and infilling capabilities that autoregressive coders lack, yet practical systems remain heavyweight. We introduce CoDA, a 1.7B-parameter diffusion coder trained on TPU with a fully open-source training pipeline. CoDA pairs large-scale diffusion pre-training with code-centric mid-training and instruction tuning, enabling confidence-guided sampling that keeps inference latency competitive. On Humaneval, MBPP, and EvalPlus, CoDA-1.7B-Instruct matches or surpasses diffusion models up to 7B parameters. Our release includes model checkpoints, evaluation harnesses, and TPU training pipelines to accelerate research on lightweight diffusion-based coding assistants.

Code: https://github.com/SalesforceAIResearch/CoDA

Model: https://huggingface.co/Salesforce/CoDA-v0-Instruct

1 Introduction

Large language models (LLMs) have rapidly advanced automatic code generation, powering developer assistants that translate natural language intent into working programs. Open-source models such as StarCoder [Li et al., 2023] and Qwen3-Coder [Yang et al., 2025] exemplify the progress of the autoregressive (AR) paradigm, which predicts code token-by-token. Despite their success, AR decoders propagate errors sequentially, struggle to leverage context bidirectionally, and falter on tasks such as filling in missing code segments or editing large spans of text.

Diffusion language models (DLMs) offer an attractive alternative [Nie et al., 2025, Gong et al., 2025, Ye et al., 2025]. Instead of generating each token sequentially, DLMs generate sequences through an iterative denoising process: a forward stage corrupts text by adding noise (e.g., masking tokens), and a backward stage gradually reconstructs the original sequence. This two-phase approach enables parallel token generation and bidirectional context awareness—i.e. the model can utilize both left and right context when generating each part of the code—addressing limitations of purely left-to-right generation. Diffusion LLMs also offer enhanced controllability and naturally support text infilling, a desirable feature for code completion and editing tasks [Gong et al., 2025]. For instance, a diffusion model can fill in a missing block in the middle of a code file by considering the surrounding context, which poses a challenge for standard

AR models.

Prior work has validated the feasibility of scaling diffusion generation to natural language and code, yet current DLM coders typically target large model budgets (e.g., 7B–8B parameters). We introduce **CoDA**, a 1.7B-parameter diffusion language model for code that is trained on an efficient TPU pipeline. CoDA adapts the Qwen3 backbone [Yang et al., 2025] to a diffusion objective, combines ∼180B tokens of general pre-training with ∼20B tokens of curated code, and applies supervised fine-tuning for instruction following. We release the model weights, evaluation harness, and training recipes, enabling the community to reproduce diffusion-based coding models without proprietary tooling.

Our main contributions are as follows:

1. We develop a fast 1.7B diffusion coder, demonstrating that compact DLMs can deliver interactive latency while retaining the benefits of bidirectional decoding.
2. We show that CoDA achieves competitive pass@1 scores relative to diffusion and autoregressive models up to 7B parameters, narrowing the performance gap while using one quarter of the weights.
3. We open-source a high-performance TPU training pipeline that lowers the barrier to scaling future DLMs.

2 Related Works

AR code models remain the default choice for open-source assistants [Li et al., 2023, Yang et al., 2025], yet they decode tokens sequentially, one at a time. DLMs tackle these limitations by iteratively refining noisy token sequences, inheriting ideas from continuous generative diffusion while adapting them to discrete vocabularies [Lou et al., 2023, Shi et al., 2024]. The resulting masked-diffusion objectives and ratio-matching estimators provide the methodological footing needed for scaling text diffusion.

Two complementary development tracks have emerged for large DLMs. Training bespoke diffusion coders from scratch yields systems such as SMDM [Nie et al., 2024] and the LLaDA family [Nie et al., 2025]. In parallel, adaptation techniques repurpose pretrained AR checkpoints into diffusion decoders—DiffuGPT and DiffuLLaMA illustrate this recipe [Gong et al., 2024]—enabling rapid bootstrapping of strong models like Dream [Ye et al., 2025]. DiffuCoder extends the approach with GRPO-based reinforcement learning [Gong et al., 2025], while parallel works [Zhao et al., 2025, Wang et al., 2025] further advance the use of reinforcement learning in DLMs.

Inference optimization has followed suit. KV-cache-aware samplers shrink runtime for public DLMs [Wu et al., 2025], and proprietary offerings such as Mercury [Khanna et al., 2025], Gemini Diffusion [DeepMind], and Seed Diffusion [Song et al., 2025] report aggressive generation latencies. Collectively, these efforts establish diffusion-based coding as a credible alternative to purely autoregressive solutions.

3 Preliminaries

Denoising Diffusion Probabilistic Models (DDPMs, diffusion models) [Ho et al., 2020, Song et al., 2020] are a family of latent variable models that describe data generation as the reverse of a forward diffusion process. Given the data distribution $q(\boldsymbol{x}_0)$, the forward diffusion process is a Markov chain that gradually injects noise into data:

$$q(\boldsymbol{x}_T|\boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}), \tag{1}$$

where $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ is a noise distribution. The model p_θ then learns the reverse denoising process $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ by aligning the marginal distributions in the reverse process to the marginal distribution in the corresponding forward step.

Continuous Diffusion Models. For continuous data (e.g., images), $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ is typically a Gaussian distribution $\mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I})$, and the reverse process is modeled by $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t, t))$.

Discrete Diffusion Models. For discrete data (e.g., text tokens), suppose $\boldsymbol{x}_t \in \{0,1\}^V$ is a one-hot vector drawn from a vocabulary of size V. In this case $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ is a categorical distribution defined by a transition matrix $\boldsymbol{Q}_t$ whose (i,j)-entry $[\boldsymbol{Q}_t]_{ij}$ denotes the probability of token i transitioning to token j. Following Gong et al. [2024] and Shi et al. [2024], we define the transition such that with probability β_t a token is replaced by a special `mask` token; otherwise, it stays unchanged:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = Cat(\boldsymbol{x}_t; \boldsymbol{Q}_t^\top \boldsymbol{x}_{t-1}), \quad \boldsymbol{Q}_t = (1-\beta_t)\boldsymbol{I} + \beta_t \mathbf{1}\boldsymbol{e}_{\texttt{mask}}^\top, \tag{2}$$

where $\mathbf{1}$ is an all-one vector and $\boldsymbol{e}_{\texttt{mask}}$ is a one-hot vector where the index of `mask` token is 1. Let $\alpha_T := \prod_{t=1}^{T}(1-\beta_t)$ and $\bar{\boldsymbol{Q}}_T := \prod_{t=1}^{T} \boldsymbol{Q}_t = \alpha_T \boldsymbol{I} + (1-\alpha_T)\mathbf{1}\boldsymbol{e}_{\texttt{mask}}^\top$. Then it follows that

$$q(\boldsymbol{x}_T|\boldsymbol{x}_0) = Cat(\boldsymbol{x}_T; \bar{\boldsymbol{Q}}_T^\top \boldsymbol{x}_0) = \alpha_T \boldsymbol{x}_0 + (1-\alpha_T)\boldsymbol{e}_{\texttt{mask}}. \tag{3}$$

Thus, at timestep T, with probability $(1-\alpha_T)$ the input $\boldsymbol{x}_0$ enters the absorbing state associated with the `mask` token. This discrete-time Markov chain generalizes to a continuous Markov process. For every $0 \le s < t \le 1$:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_s) = Cat(\boldsymbol{x}_t; \bar{\boldsymbol{Q}}(s,t)^\top \boldsymbol{x}_s) = \frac{\alpha_t}{\alpha_s}\boldsymbol{I} + (1-\frac{\alpha_t}{\alpha_s})\mathbf{1}\boldsymbol{e}_{\texttt{mask}}^\top. \tag{4}$$

The corresponding reverse process is given by:

$$q(\boldsymbol{x}_s|\boldsymbol{x}_t, \boldsymbol{x}_0) = Cat(\boldsymbol{x}_s; \bar{\boldsymbol{R}}^{\boldsymbol{x}_0}(t,s)^\top \boldsymbol{x}_t), \quad \bar{\boldsymbol{R}}^{\boldsymbol{x}_0}(t,s) = I + \frac{\alpha_s - \alpha_t}{1-\alpha_t}\boldsymbol{e}_{\texttt{mask}}(\boldsymbol{x}_0 - \boldsymbol{e}_{\texttt{mask}})^\top. \tag{5}$$

We then use a model f_θ (typically a transformer) to approximate $f_\theta(\boldsymbol{x}_t) \approx \boldsymbol{x}_0$ and compute $p_\theta(\boldsymbol{x}_s|\boldsymbol{x}_t, f_\theta(\boldsymbol{x}_t)) \approx q(\boldsymbol{x}_s|\boldsymbol{x}_t, \boldsymbol{x}_0)$.

Loss Function Following previous works [Shi et al., 2024, Gong et al., 2024], we pick $\alpha_t := 1-t$ with $t \in (0, 1)$. For a sequence of N tokens $\boldsymbol{x} := \boldsymbol{x}_0 = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}]$, the loss function is given by:

$$\mathcal{L}(f_\theta, \boldsymbol{x}_0) = \mathbb{E}_{t\sim U[0,1]} - \frac{1}{t}\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} \sum_{n=1}^{N} 1_{\boldsymbol{x}_t^n = \boldsymbol{e}_{\texttt{mask}}} (\boldsymbol{x}_0^{(n)})^\top (\log f_\theta(\boldsymbol{x}_t))^{(n)}, \tag{6}$$

where (n) indicates the n-th element in the corresponding vector. Equation (6) implies that the forward process effectively performs three steps: (1) sample a noise level t; (2) mask the input $\boldsymbol{x}_0$ to obtain $\boldsymbol{x}_t$; and (3) compute the cross-entropy loss of f_θ only on the masked tokens. When t is fixed, the loss reduces to the masked-language-model objective used to train BERT [Devlin et al., 2019].

4 Training

4.1 Overview

CoDA builds on the Qwen3-1.7B model. We divide training into three stages: pre-training, mid-training, and post-training. The first two stages run on TPUs with a custom trainer for optimized performance, and the final post-training stage runs on GPUs using the DiffuLLaMA framework [Gong et al., 2024].

A key challenge in training a DLM is the discrepancy in noise distribution between the training and inference phases. As illustrated in Figure 1, token masking is random during pre-training and mid-training. In post-training, masking is applied randomly to the assistant's response, conditioned on a user query. During inference, however, the entire response is initialized as a contiguous block of `mask` tokens, and the model learns to denoise it conditioned on the user's query. To bridge the gap between these varying noise distributions and ensure a smooth transition across stages, we introduce a progressive masking schedule.

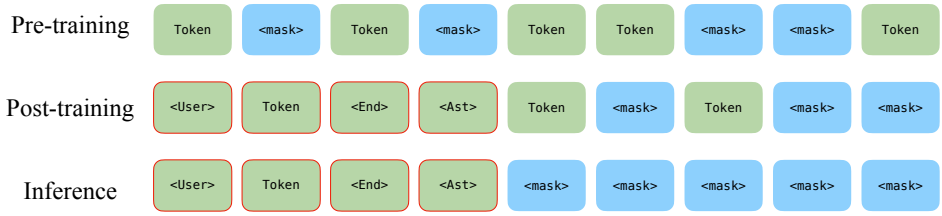

Figure 1: The masking distribution during different stages. Green tiles represent text tokens, blue tiles are mask tokens, and tiles with red border lines indicate tokens that are conditioned not to be masked. During pre-training or mid-training, masking is random. In the post-training stage, a structured masking strategy is applied. For inference, the model is conditioned on a prefix to perform infilling.

Progressive Masking Schedule We adopt a progressive masking schedule that gradually increases the difficulty of the training objective across epochs. In addition to standard random masking, we introduce three structured masking strategies designed to better capture prompt

alignment, variable-length completion, and infilling capabilities:

- **S1 Unmaskable Prefix**: A randomly selected prefix is left unmasked, forcing the model to consistently condition on the initial segment of the sequence. This strategy encourages stronger reliance on prompts and improves prefix-based generation.
- **S2 Truncated Suffix**: A randomly chosen suffix is truncated and replaced with unmaskable `pad` tokens. This setting exposes the model to incomplete contexts and prevents overfitting to full-length sequences, thereby enhancing robustness to limited or noisy inputs.
- **S3 Block Masking**: Rather than masking isolated tokens, we mask contiguous spans of length k (with $k \in \{2, 4, 8\}$ for example) while keeping the sequence-level masking probability at $1 - \alpha_t$. This block-level corruption better simulates real-world scenarios such as text infilling, style rewriting, and document editing.

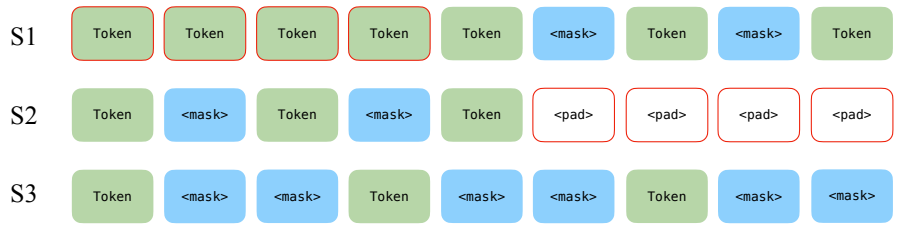

Figure 2: A visualization of the masking schedule. S1: a randomly chosen prefix is conditioned and unmaskable; S2: a randomly chosen suffix is replaced with the `pad` token and made unmaskable; S3: A block masking of size $k = 2$.

This progressive masking schedule both gradually increases task difficulty to move from pretraining to fine-tuning and, aligns training with downstream practical applications. The unmaskable prefix strategy (S1) encourages the model to rely on prompts, stabilizing optimization and improving prefix-conditioned generation during supervised fine-tuning. The truncated suffix strategy (S2) teaches the model to handle sequences of varying length and prevents overfitting to full-length examples, which benefits tasks involving truncated queries or constrained context windows. The block masking strategy (S3) exposes the model to contiguous spans of missing tokens, closely matching infilling, editing, and style-transfer scenarios. Together these strategies enhance robustness to prompts, adaptability to variable-length inputs, and performance on editing or completion tasks. Increasing the masking difficulty across epochs yields a smooth progression from easier to harder training conditions.

4.2 Infrastructure

Our training infrastructure runs on Google's TPU VMs. We enable PyTorch-based training with the PyTorch/XLA library and build on `torchprime` [Hypercomputer, 2025] to develop a custom trainer tailored to our workflow. The trainer provides robust distributed checkpointing for efficient saving and loading of model states across TPU cores, integrates the progressive

masking schedule directly into the training loop, and drives high-throughput data pipelines capable of processing up to one billion tokens per update.

4.3 Pre-training

During pre-training, transitioning a causal LM to a DLM requires exposing the model to massive data so it shifts from causal attention and autoregressive decoding to bidirectional attention and unmasking. We follow the adaptation strategy of Gong et al. [2024] but omit attention-mask annealing because FlashAttention does not support customized masking.

4.3.1 Pre-training Data Corpora

Our pre-training corpus is a comprehensive collection of text and code totaling 179.27 billion tokens, designed to provide broad knowledge and strong reasoning capabilities. After data cleaning, deduplication, and pre-tokenization, the corpus composition appears in Table 1.

A significant portion of the data comes from web text and source code. For web data, we use `dclm-baseline-1.0` [Li et al., 2024], a large-scale web crawl that contributes 60.17 billion tokens. For source code, we include two subsets from The Stack v2 dataset [Lozhkov et al., 2024]: 1. `the-stack-v2 (Python)` with 50.75 billion tokens of Python code, 2. `the-stack-v2-train-smol` with 38.22 billion tokens spanning multiple programming languages.

To bolster mathematical and scientific reasoning, we include several curated datasets. OpenWebMath [Paster et al., 2023] offers 12.98 billion tokens of mathematical text sourced from the web. Additionally, we use the arXiv subset of RedPajama [Weber et al., 2024], which provides 9.18 billion tokens from academic papers available on arXiv. For mathematical problem solving, we incorporate the DeepMind Mathematics dataset [Saxton et al., 2019], contributing 3.24 billion tokens. To ensure a solid foundation of general knowledge, we add the English Wikipedia dump [Foundation], which supplies 5.41 billion tokens of high-quality encyclopedic text.

Table 1: Pre-training data corpus with categories and token counts (in billions)

Dataset	Category	Token (Billion)
dclm-baseline-1.0	Text (Web)	60.17
The Stack v2 (Python)	Code	50.75
The Stack v2 (train-smol)	Code	38.22
OpenWebMath	Text (Mathematics)	12.98
Redpajama Arxiv	Text (Academic)	9.18
Wikipedia (English)	Text (Encyclopedia)	5.41
DeepMind Mathematics	Text (Mathematics)	3.24

4.3.2 Pre-training Recipe

We run pre-training on a TPU v4-1024 VM with Torch XLA and FSDP. The global batch size is 512 with a sequence length of 8192, totaling around four million tokens per update. We use the Adafactor optimizer [Shazeer and Stern, 2018] with linear learning-rate scheduling and set the peak learning rate to 3×10^{-4}. With probability 0.01, each batch applies masking strategy

S1 or S2; independently, with the same probability, S3 applies to the batch. Training spans 50,000 steps, just over one epoch.

4.4 Mid-training

To strengthen the model's capacity to handle complex masking patterns and to bridge pre-training with downstream fine-tuning, we extend training with high-quality corpora comprising both coding and textual datasets.

4.4.1 Mid-training Data Corpora

During mid-training, we incorporate two high-quality textual corpora totaling 12.42 billion tokens: RedPajama Arxiv [Weber et al., 2024] and Gutenberg [Stroube, 2003]. We also include three coding corpora amounting to 8.55 billion tokens: the `python-edu` subset of SmolLM-Corpus [Ben Allal et al., 2024], the OpenCoder Annealing Corpus [Huang et al., 2024], and Python code extracted from The Stack v2 [Lozhkov et al., 2024]. We reuse a small slice of The Stack v2 data from pre-training to ensure a smooth transition between stages. Table 2 provides a detailed breakdown of the mid-training datasets.

Table 2: Mid-training data corpora with categories and token counts (in billions).

Dataset	Category	Tokens (B)
Redpajama Arxiv	Text (Academic)	9.18
Gutenberg	Text (ebooks)	3.24
SmolLM-Corpus (python-edu)	Code	1.88
OpenCoder (opc-annealing-corpus)	Code	6.47
The Stack v2 (Python)	Code	0.20

4.4.2 Mid-training Recipe

We run mid-training on a TPU v4-512 with Torch XLA and FSDP. The global batch size is 256 with a sequence length of 8192, yielding around two million tokens per step. We use the Adafactor optimizer with linear learning-rate scheduling and set the peak learning rate to 2×10^{-4}. Training spans 50,000 steps (roughly five epochs). Throughout mid-training we evenly shuffle the datasets, maintain five full passes, and gradually increase the probabilities of applying S1, S2, and S3 according to a fixed curriculum (see table 3).

Table 3: Progressive masking schedule over mid-training epochs. Probabilities indicate the proportion of sequences affected by each strategy (S1, S2, S3).

Epoch	S1 : S2 : S3
1	1% : 1% : 5%
2	5% : 5% : 10%
3	10% : 10% : 15%
4	15% : 15% : 20%
5	20% : 20% : 25%

4.5 Post-training

After mid-training, we conduct supervised fine-tuning as the post-training stage so the model can address real-world coding tasks. Post-training runs on a single H200 cluster. We use the stage 1 and stage 2 SFT subsets of the OpenCoder dataset [Huang et al., 2024] and truncate sequences to 1024 tokens. For stage 1, we train for 10 epochs (with early stopping) using the AdamW optimizer and a cosine scheduler, using a peak learning rate of 2×10^{-5} with a 10% warm-up. During the first epoch, we gradually transition from unconditional training to conditioning on the user prompt by linearly increasing the conditioned span. The global batch size is 176, and we reuse the same hyperparameters for stage 2 SFT.

5 Experiments

5.1 Model Evaluation

We evaluate CoDA against other DLMs and similarly sized AR models on two coding benchmarks: Humaneval [Chen et al., 2021] and MBPP [Austin et al., 2021]. For both datasets, we also report the EvalPlus-enhanced variants provided by Gao et al. [2024]. All evaluations use pass@1 as the primary metric. Our evaluation harness builds on Dream [Ye et al., 2025] and Qwen2.5-Coder [Hui et al., 2024], with adaptations to support diffusion decoding and data parallelism.

For CoDA, we cap the maximum number of generated tokens at 768 and align the diffusion schedule to the sequence length, i.e., a single token is produced at each diffusion step. We adopt the confidence-based sampling strategy described in Ye et al. [2025], which reweights denoising updates using per-token posterior entropy. Self-reported AR baselines use the default configuration provided by their released inference environments, including nucleus sampling and temperature values recommended by their maintainers.

Table 4 summarizes the performance of all systems on the evaluation suite. Across diffusion models, CoDA-1.7B-Instruct narrows much of the gap to larger Dream models while outperforming other DLMs on most reported metrics. The 25-point improvement of CoDA-1.7B-Instruct over CoDA-1.7B-Base on Humaneval (pass@1) illustrates the potential of instruction tuning. Although Dream-7B-Instruct remains the strongest diffusion baseline on MBPP-Instruct, CoDA-1.7B-Instruct delivers comparable EvalPlus scores within a significantly smaller parameter count. When compared with AR models of similar scale, CoDA-1.7B-Instruct trails the Qwen model series but surpasses the Gemma or LLaMA models. These results highlight that diffusion decoding can remain competitive at small model sizes.

5.2 Inference Scaling Dynamics

We investigate how the number of diffusion steps affects the inference time of CoDA-1.7B-Instruct and evaluate the model's performance across diffusion steps ranging from 32 to 1024. We conducted experiments on a single NVIDIA A100 40GB. We applied a KV cache variant to accelerate inference and adopted confidence-aware parallel decoding to balance the inference efficiency and quality following Wu et al. [2025]. The model inferred on the Humaneval dataset

Table 4: Comparison of code generation performance on Humaneval and MBPP. Evalplus scores are calculated as an average of pass@1 scores on plus-enhanced variants. Bold numbers indicate metrics where CoDA models achieve the strongest diffusion-model result. $*$ represents self-reported scores.

Model	Humaneval		MBPP		Evalplus
	-	Plus	-	Plus	
Diffusion Models					
CoDA-1.7B-Base	29.3	23.8	35.2	46.0	34.9
CoDA-1.7B-Instruct	54.3	47.6	47.2	**63.2**	**55.4**
Dream-7B-Base	56.7	50.0	68.7	57.4	53.7
Dream-7B-Instruct	57.9	53.7	68.3	56.1	54.9
LLaDA-8B-Instruct	35.4	31.7	31.5	28.6	30.2
AR Models of similar size					
Qwen3-1.7B*	66.5	61.6	46.2	65.9	63.8
Qwen2.5-Coder-1.5B	43.9	36.6	69.2	58.6	47.6
Qwen2.5-Coder-1.5B-Instruct	70.7	66.5	69.2	59.4	62.3
Gemma-3-1B-it*	39.6	35.4	39.4	63.5	49.5
LLaMA-3.2-1B-Instruct*	35.4	31.1	24.4	53.7	42.4

with a 768-token budget. We run each sample for two trials when measuring the inference time to reduce variance.

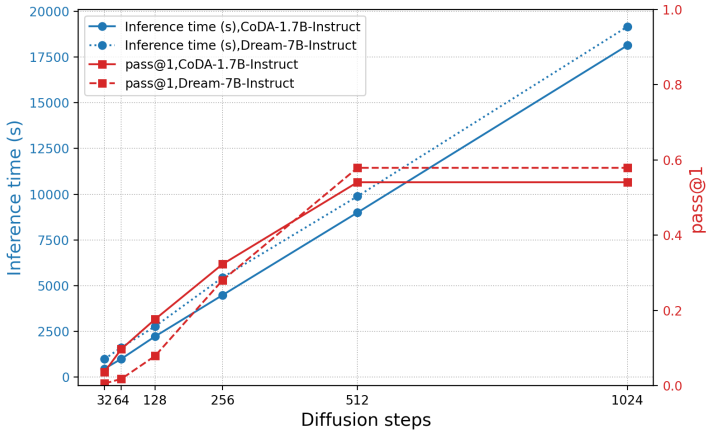

Figure 3: Relationship between diffusion steps, inference time, and CoDA-1.7B-Instruct performance. Inference time is measured by the total inference time on the Humaneval dataset. Model performance is measured by pass@1 on the same dataset with a 768-token budget.

Figure 3 illustrates the trade-off between inference time and model performance for CoDA-1.7B-Instruct across varying diffusion steps. As shown by the blue curve, inference time for both CoDA-1.7B-Instruct and Dream-7b-Instruct increases almost linearly with diffusion steps, highlighting the computational cost of lengthening the diffusion trajectory. CoDA-1.7B-Instruct achieves 39.64% lower latency than Dream-7B-Instruct across the same range of diffusion steps, due to its smaller parameter footprint. The red curves track task performance (pass@1 on Humaneval with a 768-token budget). Both CoDA-1.7B-Instruct and Dream-7B-Instruct show

steady improvements as steps increase, but gains saturate around 512 steps. Notably, CoDA-1.7B-Instruct surpasses Dream-7B-Instruct at smaller step counts despite its more compact size, we attribute this behavior to our block masking strategies, since when the diffusion step is much smaller than the total tokens to generate, we are essentially unmasking multiple tokens at a time.

5.3 Discussion

Table 4 shows that CoDA-1.7B-Instruct is competitive with other diffusion models of larger size while staying within a 1.7B-parameter budget. Instruction tuning paired with confidence-guided sampling closes most of the gap to Dream-7B-Instruct on Humaneval and EvalPlus, validating that diffusion adaptation can unlock strong coding accuracy without resorting to heavyweight backbones. Against autoregressive (AR) coders, CoDA retains a balanced profile: it trails the strongest AR baselines on Humaneval but delivers competitive MBPP performance, suggesting that diffusion decoding provides complementary inductive biases even when model scale is modest.

The diffusion schedule analysis in Figure 3 highlights a second advantage of compact DLMs. Because the inference latency scales roughly linearly with the number of steps, yet accuracy saturates beyond 512 steps. This observation indicates that practical deployments can adopt adaptive schedules that cap the number of steps when confidence is high, yielding faster responses than larger autoregressive models that require long decoding traces. Our open-source TPU pipeline further demonstrates that diffusion pre-training and instruction tuning remain tractable when optimized end-to-end on modern pods, enabling rapid iteration on architectural variations and sampling strategies.

6 Conclusion

We present CoDA, a fast 1.7B diffusion language model for code that demonstrates competitive performance with several 7B models while maintaining efficiency suitable for lightweight hardware budgets. By releasing model checkpoints, training pipelines, and evaluation harnesses, we aim to ease the challenge for the community to explore diffusion-based coding assistants and accelerate progress in this emerging paradigm. Looking forward, we plan to explore hybrid diffusion/AR decoding, reinforcement learning–based fine-tuning, and more refined pre-training strategies to push the frontier of diffusion coders toward higher accuracy, faster inference, and broader applicability.

Acknowledgement

We would like to thank Lingpeng Kong for insightful discussions and Jialei Chen for technical support in TPU.

References

J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

L. Ben Allal, A. Lozhkov, G. Penedo, T. Wolf, and L. von Werra. Smollm-corpus, 2024. URL https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. 2021.

G. DeepMind. Gemini Diffusion — deepmind.google. https://deepmind.google/models/gemini-diffusion/. [Accessed 20-09-2025].

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

W. Foundation. Wikimedia downloads. URL https://dumps.wikimedia.org.

L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

S. Gong, S. Agarwal, Y. Zhang, J. Ye, L. Zheng, M. Li, C. An, P. Zhao, W. Bi, J. Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

S. Gong, R. Zhang, H. Zheng, J. Gu, N. Jaitly, L. Kong, and Y. Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

S. Huang, T. Cheng, J. K. Liu, J. Hao, L. Song, Y. Xu, J. Yang, J. H. Liu, C. Zhang, L. Chai, R. Yuan, Z. Zhang, J. Fu, Q. Liu, G. Zhang, Z. Wang, Y. Qi, Y. Xu, and W. Chu. Opencoder:

The open cookbook for top-tier code large language models. 2024. URL https://arxiv.org/pdf/2411.04905.

B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

G. C. A. Hypercomputer. torchprime, 2025. URL https://github.com/AI-Hypercomputer/torchprime.

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

J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Y. Gadre, H. Bansal, E. Guha, S. S. Keh, K. Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

A. Lou, C. Meng, and S. Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W.-D. Li, M. Risdal, J. Li, J. Zhu, T. Y. Zhuo, E. Zheltonozhskii, N. O. O. Dade, W. Yu, L. Krauß, N. Jain, Y. Su, X. He, M. Dey, E. Abati, Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. McAuley, H. Hu, T. Scholak, S. Paquet, J. Robinson, C. J. Anderson, N. Chapados, M. Patwary, N. Tajbakhsh, Y. Jernite, C. M. Ferrandis, L. Zhang, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. Starcoder 2 and the stack v2: The next generation, 2024.

S. Nie, F. Zhu, C. Du, T. Pang, Q. Liu, G. Zeng, M. Lin, and C. Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

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. *arXiv preprint arXiv:2502.09992*, 2025.

K. Paster, M. D. Santos, Z. Azerbayev, and J. Ba. Openwebmath: An open dataset of high-quality mathematical web text. *arXiv preprint arXiv:2310.06786*, 2023.

D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models. *ArXiv*, abs/1904.01557, 2019. URL https://api.semanticscholar.org/CorpusID:85504763.

N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

J. Shi, K. Han, Z. Wang, A. Doucet, and M. Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

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.

B. Stroube. Literary freedom: Project gutenberg. *XRDS: Crossroads, The ACM Magazine for Students*, 10(1):3–3, 2003.

Y. Wang, L. Yang, B. Li, Y. Tian, K. Shen, and M. Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*, 2025.

M. Weber, D. Y. Fu, Q. Anthony, Y. Oren, S. Adams, A. Alexandrov, X. Lyu, H. Nguyen, X. Yao, V. Adams, B. Athiwaratkun, R. Chalamala, K. Chen, M. Ryabinin, T. Dao, P. Liang, C. Ré, I. Rish, and C. Zhang. Redpajama: an open dataset for training large language models. *NeurIPS Datasets and Benchmarks Track*, 2024.

C. Wu, H. Zhang, S. Xue, Z. Liu, S. Diao, L. Zhu, P. Luo, S. Han, and E. Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

J. Ye, Z. Xie, L. Zheng, J. Gao, Z. Wu, X. Jiang, Z. Li, and L. Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

S. Zhao, D. Gupta, Q. Zheng, and A. Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.