dLLM: Simple Diffusion Language Modeling

Zhanhui Zhou[*] UC Berkeley
Lingjie Chen[*] UIUC
Hanghang Tong UIUC
Dawn Song UC Berkeley

dLLM: https://github.com/ZHZisZZ/dllm[†]
dllm-hub: https://huggingface.co/dllm-hub
Video: Watch on Google Drive

Abstract

Recent diffusion language models (DLMs) share core components but rely on fragmented, ad-hoc codebases, hindering reproducibility. To address this, we introduce **dLLM**, an open-source framework standardizing DLM training, inference, and evaluation. **dLLM** provides a unified, customizable pipeline to reproduce, finetune, and deploy large models like LLaDA and Dream. To further democratize research, we also release checkpoints and minimal recipes for building small DLMs from scratch or converting existing BERT-style and autoregressive models. The framework is designed for researchers and practitioners who wish to reproduce, compare, or extend DLM methods with minimal setup.

1 Introduction

Diffusion language models (DLMs) are a promising alternative to autoregressive models (Austin et al., 2021a; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024; Arriola et al., 2025), enabling iterative refinement (Wang et al., 2025; Havasi et al., 2025), flexible steering (Li et al., 2022; Schiff et al., 2025), and efficient decoding (Wu et al., 2026b,a; Ma et al., 2025; Ben-Hamu et al., 2025). Although many recent open-weight DLMs share common design choices (Nie et al., 2025a,b; Ye et al., 2025; Chandrasegaran et al., 2025; Bie et al., 2025), their implementations are frequently scattered across opaque, ad-hoc codebases, hindering reproduction, comparison, and extension.

To address this gap, we introduce **dLLM**, an open-source framework standardizing DLM development across three core components: (1) **Training**: Architecture-agnostic trainers support objectives like Masked (Sahoo et al., 2024) and Block Diffusion (Arriola et al., 2025), enabling users to reproduce, finetune (e.g., LLaDA (Nie et al., 2025b), Dream (Ye et al., 2025)), or build new models. (2) **Inference**: Lightweight abstractions allow plug-and-play decoding (Wu et al., 2026b) without modifying code. (3) **Evaluation**: A unified interface ensures reproducible results across models.

Beyond unifying DLM pipelines, **dLLM** provides minimal, reproducible recipes and checkpoints for building small DLMs from scratch with accessible compute. This includes end-to-end pipelines for converting existing LMs (e.g., BERT-style encoders (Devlin et al., 2019) and autoregressive models (Gong et al., 2025)) into DLMs.

The paper is organized as follows: Section 2 reviews discrete diffusion for language; Section 3 describes the three components of **dLLM** (trainer, sampler, evaluation); Section 4 presents open recipes and checkpoints; Section 5 discusses related work.

2 Preliminaries

Discrete diffusion for language treats text generation as denoising: the model refines a noisy or masked sequence to coherent text, unlike AR LMs. This section fixes notation and summarizes **dLLM**'s two dominant training objectives.

We denote a token sequence as $x = (x^1, \ldots, x^L) \in \mathcal{V}^L$ with continuous time $t \in [0, 1]$ and mask token $m \notin \mathcal{V}$. **Discrete diffusion** (Austin et al., 2021a; Sahoo et al., 2024) generates by reversing a forward process corrupting data (e.g., by masking) from x_0 to uninformative x_1; the model denoises x_t toward x_0. **Masked diffusion (MDLM)** (Sahoo et al., 2024; Shi et al., 2024) uses an absorbing-state mask: tokens are independently masked with probability t, and the model is trained to predict masked tokens with time-reweighted cross-entropy loss. It is simple and widely used (e.g., LLaDA, Dream) but decoding is iterative.

[*]Equal contribution.
Released under the MIT License.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 78–88

```
# MDLM PT
trainer = MDLMTrainer(
  model=model,
  tokenizer=tokenizer,
  train_dataset=dataset["train"],
  eval_dataset=dataset["test"],
  args=training_args,
  data_collator=(
    DataCollatorForSeq2Seq()
  )
)
trainer.train()
```

(a) MDLM (e.g., LLaDA, Dream) Pretraining.

```
# MDLM PT -> BD3LM PT
- trainer = MDLMTrainer(
+ trainer = BD3LMTrainer(
  model=model,
  tokenizer=tokenizer,
  train_dataset=dataset["train"],
  eval_dataset=dataset["test"],
  args=training_args,
  data_collator=(
    DataCollatorForSeq2Seq()
  )
)
trainer.train()
```

(b) Changes for BD3LM Pretraining.

```
# MDLM PT -> SFT
trainer = MDLMTrainer(
  model=model,
  tokenizer=tokenizer,
  train_dataset=dataset["train"],
  eval_dataset=dataset["test"],
  args=training_args,
  data_collator=(
+ NoAttentionMaskWrapper(
      DataCollatorForSeq2Seq(
+ label_pad_token_id=eos_token_id,
      ),
+ )
  ),
)
trainer.train()
```

(c) Changes for MDLM SFT.

```
# MDLM PT -> AR-to-MDLM Adaptation
trainer = MDLMTrainer(
  model=model,
  tokenizer=tokenizer,
  train_dataset=dataset["train"],
  eval_dataset=dataset["test"],
  args=training_args,
+ right_shift_logits=True,
  data_collator=(
+ PrependBOSWrapper(
    DataCollatorForSeq2Seq(),
+ )
  ),
)
trainer.train()
```

(d) Changes for AR-to-MDLM adaptation.

Figure 1: **A unified trainer interface supports diverse purposes via configuration changes.** Figure 1a shows MDLM pretraining. Figure 1b shows the single-line trainer swap from `MDLMTrainer` to `BD3LMTrainer`. Figure 1c shows the minimal changes to use `MDLMTrainer` for SFT: `NoAttentionMaskWrapper` keeps padding EOS visible, and `label_pad_token_id=eos_token_id` trains the model to generate EOS from mask tokens in inputs. Figure 1d shows the minimal changes to adapt an AR LM to MDLM: `right_shift_logits` reuses next-token prediction, and `PrependBOSWrapper` prepends BOS to provide the predictions for the first mask token.

Block diffusion (BD3LM) (Arriola et al., 2025) partitions into blocks and generates block-by-block via diffusion conditioned on previous blocks, allowing KV-cache reuse and tunable parallelism–coherence trade-off, speeding up inference at the cost of flexibility in decoding order.

3 dLLM Overview

We provide an overview of the three core components of **dLLM**: Trainer (Section 3.1), Sampler (Section 3.2) and Evaluation (Section 3.3).

3.1 Trainer

Unified training interface (Figure 1). Most open-weight DLMs to date are trained with Masked Diffusion (MDLM) (Sahoo et al., 2024; Nie et al., 2025b; Ye et al., 2025) or Block Diffusion (BD3LM) (Arriola et al., 2025). Accordingly, the current version of **dLLM** focuses on unified `MDLMTrainer` and `BD3LMTrainer` as core training modules (`dllm/core/trainers`) that support both pretraining and finetuning most DLMs. A typical workflow is to load a model and tokenizer, attach a dataset and data collator, instantiate the appropriate trainer with training arguments, and call `trainer.train()`; switching from MDLM to BD3LM pretraining, or from pretraining to SFT, amounts to changing the trainer class or a small set of options (Figure 1). At the same time, the framework's modular design can be naturally extended to new diffusion objectives. For example, **dLLM** also includes a reference implementation of an EditFlow (Havasi et al., 2025; Nguyen et al., 2025) trainer for text diffusion with parallel insertion, substitution, and deletion operations, enabling edit-style generation within the same framework.

```python
# Inference
sampler = MDLMSampler(model, tokenizer)
sampler = MDLMFastdLLMSampler(

terminal_visualizer =
↪  TerminalVisualizer(tokenizer)

messages = [[{"role": "user", "content":
↪  "Write a python function."}]]

inputs =
↪  tokenizer.apply_chat_template(messages,
↪  add_generation_prompt=True,
↪  tokenize=True)

outputs = sampler.sample(inputs,
↪  return_dict=True)

terminal_visualizer.visualize(
    outputs.histories, rich=True)
```

Figure 2: **Inference pipeline: sampler swap from vanilla to FastdLLM MDLM sampler.**

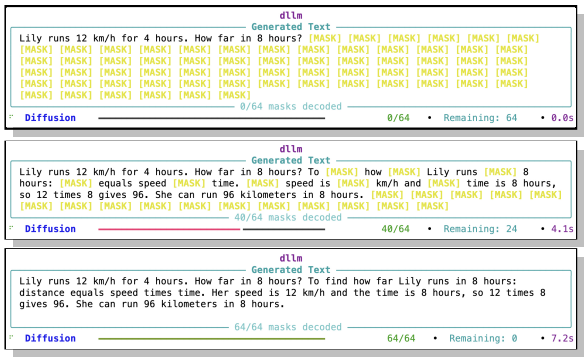


Figure 3: **Terminal Visualizer showing transition from masked to decoded tokens.**

Modular design enables easy customization (Figure 1). Our training pipeline follows a modular design that allows core components to be reused and extended with minimal changes, improving both flexibility and readability. Figure 1 illustrates this modularity in practice: switching between MDLM/BD3LM pretraining, MDLM SFT, and AR-to-MDLM adaptation requires only localized changes (e.g., swapping the trainer, toggling a small set of arguments, or wrapping the data collator), without altering the overall pipeline.

Simple yet scalable training powered by 🤗 HF infrastructure. Our training pipeline builds on the HuggingFace ecosystem. We use `accelerate` to support diverse training configurations (e.g., FSDP (Zhao et al., 2023) and DeepSpeed (Rajbhandari et al., 2020) for distributed training), and `peft` for parameter-efficient finetuning. Our custom trainers (e.g., `MDLMTrainer`) are wrappers around the `transformers` Trainer (Wolf et al., 2020). By using these components as building blocks, the framework stays easy to learn, which allows users to focus on DLM-specific logic, and remains scalable enough to support large-model pretraining and research experimentation.

3.2 Sampler

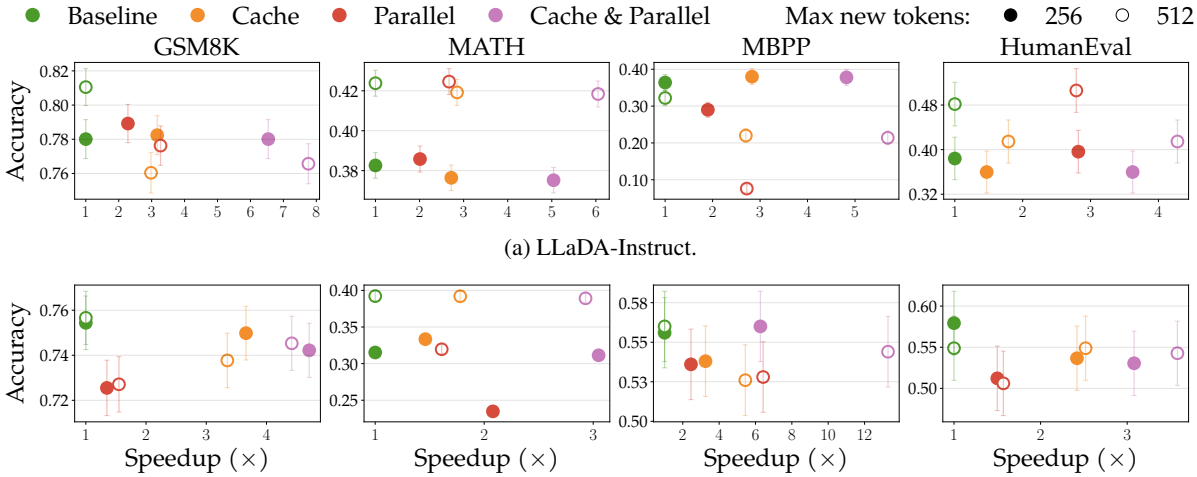
Unified inference interface (Figure 2). Different DLMs and inference algorithms expose inconsistent inference APIs, making it hard to reuse and compare inference algorithms across models. To address this issue without modifying existing model implementations, we introduce a lightweight inference abstraction: `Sampler(model).sample()`. This wrapper decouples models from inference algorithms, allowing different samplers to be swapped in a plug-and-play manner while keeping the underlying model unchanged. Figure 2 illustrates the unified inference pipeline enabled by this interface.

Terminal visualizer (Figure 3). Unlike AR LMs, which decode tokens strictly left-to-right, DLMs decode tokens in any order. As a result, the decoding order, beyond the final decoded output, is an important feature of DLMs and is valuable for analysis. To support debugging and interpretability, we provide a terminal visualizer that reveals the token decoding order and the evolution of the sample over decoding steps (Figure 3).

Efficient DLM inference (Figures 2 & 4). DLM inference speed is a practical bottleneck (Wu et al., 2026b,a; Ma et al., 2025; Ben-Hamu et al., 2025). Building on the unified inference interface, **dLLM** includes an implementation of Fast-dLLM (Wu et al., 2026b) for accelerated MDLM decoding: `MDLMFastdLLMSampler`, which can be used as a drop-in replacement for the standard `MDLMSampler` (Figure 2). We report benchmarking results consistent with official Fast-dLLM implementations, demonstrating substantial inference speedups (see Figure 4 for visualization and Github for detailed results).

3.3 Evaluation

Open-weight DLMs (Nie et al., 2025b; Ye et al., 2025) rely on different evaluation tools, making unified evaluation difficult. This is further complicated because DLMs are especially sensitive to inference hyperparameters, as prior work often re-

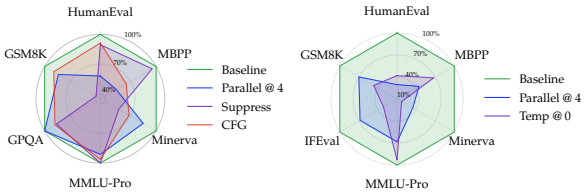


(a) LLaDA-Instruct.

(b) Dream-Base.

Figure 4: **Fast-dLLM evaluation results with max new tokens @ 256 and 512.** Model selection follows the original Fast-dLLM evaluation for fair comparison. CACHE uses block-wise approximate KV caching within each block; PARALLEL uses confidence-based parallel updates; CACHE & PARALLEL combines both.

lies on task-specific hyperparameter tuning and postprocessing for best performance. For example, even a single change in an inference parameter can significantly alter performance (Figure 5).



(a) LLaDA-Instruct (b) Dream-Instruct

Figure 5: **Sensitivity to decoding hyperparameters.** We vary individual sampling hyperparameters at inference time and observe that performance can degrade sharply from the optimal configuration. BASELINE denotes the best-performing setting; SUPPRESS does not suppress `<eos>` from the beginning of generation; CFG sets `cfg_scale`=0.5; PARALLEL @ 4 generates four tokens per step; and TEMP @ 0 sets `temperature`=0.0.

A unified evaluation pipeline must therefore be flexible enough to support customization while faithfully reproducing the evaluation configurations used in prior work. To achieve this, we extend the `lm-evaluation-harness` (Gao et al., 2024) framework and carefully match the preprocessing, decoding settings, and post-processing used for each model–task pair with its corresponding official pipeline. These details vary across models and tasks and require manual verification, but this enables our framework to reproduce the reported, model-specific scores while supporting consistent comparisons across models. Tables 4 and 5 (Appendix C) compare our reproduced results against the originally reported results, showing that our evaluation framework closely matches the official results. Our evaluation is based entirely on standard benchmarks and reproduction of official results; we did not conduct user studies or human evaluations.

Takeaway

DLM evaluation is highly sensitive to inference hyperparameters (e.g., max new tokens in Figure 4 and other hyperparameters in Figure 5), a fact that is rarely explicitly stated in prior work, but is evident from our reproduction experiments.

4 Open DLMs with Open Recipes

Beyond the core framework, we provide reproducible recipes that show how to use **dLLM** end-to-end and lower the barrier to trying DLMs. These recipes serve both as documentation (minimal code changes for different use cases) and as baselines: all checkpoints are released in 🤗 dllm-hub with evaluation results. We cover two themes: (1) finetuning open-weight large DLMs (e.g., LLaDA, Dream) for reasoning, and (2) building small DLMs from scratch by converting BERT-style encoders or autoregressive LMs into DLMs with minimal compute.

4.1 Finetuning Open-Weight Large DLMs

Reasoning before final answer has proven effective for complex tasks; recent work (e.g., d1 (Zhao

Model	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-Instruct	79.91	34.80	19.92	11.62	35.37	42.02
+ SFT (MDLM)	80.59	35.40	26.95	14.36	36.59	43.97
LLaDA-Base	64.67	10.20	10.16	0.34	25.00	40.08
+ SFT (MDLM)	73.62	17.40	8.98	0.00	18.90	28.79
Dream-Instruct	64.44	28.20	22.27	6.93	35.98	44.36
+ SFT (MDLM)	70.43	32.80	20.31	19.68	37.20	45.53
Dream-Base	49.05	20.40	10.55	1.07	15.85	22.96
+ SFT (MDLM)	63.00	23.40	8.59	1.03	29.88	23.35

Table 1: **MDLM SFT evaluation results**. Instruct models show consistent gains, Base models gain on in-distribution math but may regress on out-of-distribution planning and coding.

et al., 2025)) explores similar capabilities in DLMs. Using **dLLM**'s unified trainer, we show that MDLM-style SFT elicits reasoning in open-weight DLMs and improves downstream performance.

Training. We finetune LLaDA and Dream Base/Instruct with MDLM SFT and LoRA on s1K (Muennighoff et al., 2025); see Table 1 and Figure 6 for results and training curves.

Evaluation results. We evaluate models SFTed on reasoning data by prepending `<reasoning>` at inference to force reasoning (Table 1) using pipelines from (Zhao et al., 2025). For Instruct models, reasoning SFT yields consistent gains across math, planning, and coding benchmarks. Base models show improvements on in-distribution math tasks (e.g., GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021b)) but regress on out-of-distribution benchmarks. Overall, SFT is an effective start for reasoning in DLMs, via the unified trainer (Figure 1) with minimal changes.

4.2 Training Small DLMs from Scratch

We also provide recipes and checkpoints for small DLMs from scratch: (1) BERT-style encoders (Devlin et al., 2019) to DLMs and (2) autoregressive LMs to DLMs (Gong et al., 2025).

4.2.1 BERT-Chat: Converting BERTs to DLMs

BERT-style models (Devlin et al., 2019; Sahoo et al., 2024) offer bidirectional representations for diffusive generation. We turn off-the-shelf BERT into a diffusion chatbot with instruction-following SFT only (no architectural change), using ModernBERT (Warner et al., 2025) as backbone, and release two 🤗 checkpoints, ModernBERT-base-chat-v0.1 and ModernBERT-large-chat-v0.1.

Model	GSM8K	BBH	MATH	MMLU	Hella.	LAMB.	WinoG.
BERT-base	3.6	21.1	3.1	26.2	34.5	49.3	48.8
BERT-large	9.3	25.6	3.6	29.6	40.9	46.3	49.0
Qwen-0.5B	22.0	18.3	3.1	39.2	48.2	48.6	55.0
Qwen-0.5B-Chat	11.3	18.2	2.1	35.0	36.9	41.2	52.0
GPT-2	0.7	6.9	1.8	22.9	31.1	46.0	51.6
GPT-2-med	2.1	17.8	1.4	22.9	39.4	55.5	53.1

Table 2: **ModernBERT-Chat evaluation results.** ModernBERT-Chat (Warner et al., 2025) and GPT-2 (Radford et al., 2019) models are evaluated with **dLLM**'s pipeline; Qwen1.5 (Bai et al., 2023) numbers are reported by original sources. "Hella." stands for HellaSwag, "LAMB." for LAMBADA, "WinoG." for WinoGrande; BERT-base/large denote ModernBERT-base/large-chat, Qwen-0.5B denotes Qwen1.5-0.5B, GPT-2-med denotes GPT-2-medium.

Training. We finetune ModernBERT-base and ModernBERT-large via MDLM SFT on instruction-tuning data (no continual pretraining). See Table 2 and Figure 7 for results and training curves; reproduction scripts are at dllm/examples/bert.

Evaluation results. We evaluate BERT-Chats using **dLLM**'s unified evaluation pipeline (Table 2). A gap remains compared to decoder-only ARLMs of similar size (e.g., Qwen1.5-0.5B and Qwen1.5-0.5B-Chat (Bai et al., 2023) on MMLU (Hendrycks et al., 2021a) and HellaSwag (Zellers et al., 2019)), yet the results are still noteworthy: ModernBERT-large-chat surpasses both GPT-2 (Radford et al., 2019) variants by a wide margin and outperforms Qwen1.5-0.5B-Chat on BBH (Suzgun et al., 2023) and MATH (Hendrycks et al., 2021b), despite being encoder-only with no modification for generation. Thus, BERT-style backbones are a viable, if under-explored, starting point for DLMs.

4.2.2 Tiny-A2D: Converting ARLMs to DLMs

Prior work explored AR-to-diffusion conversion (e.g., RND1 (Chandrasegaran et al., 2025), DiffuLLaMA (Gong et al., 2025)). We convert Qwen3-0.6B (Yang et al., 2025) into a diffu-

Model	GSM.	BBH	MATH	MMLU	M-Pro	Hella.	H.E.	MBPP
MDLM	29.3	26.7	8.7	40.0	17.3	42.1	30.5	29.2
BD3LM	46.3	26.6	12.9	39.1	13.8	39.3	46.3	38.2
Qwen2.5-0.5B	41.6	20.3	19.5	47.5	15.7	52.1	30.5	39.3
Qwen3-Base	59.6	41.5	32.4	52.8	24.7	47.4	32.3	36.6

Table 3: **Qwen-A2D evaluation results.** MDLM and BD3LM models are evaluated with **dLLM**'s pipeline; AR Qwen2.5/Qwen3 baselines are from original sources (Qwen Team et al., 2024; Yang et al., 2025). "GSM." stands for GSM8K, "M-Pro" for MMLU-Pro, "Hella." for HellaSwag, "H.E." for HumanEval; MDLM and BD3LM denote the Qwen3-0.6B diffusion variants. See Figure 8 for training curves.

sion chatbot with minimal changes, tuning under MDLM (Sahoo et al., 2024) and BD3LM (Arriola et al., 2025) on instruction data, and release two 🤗 checkpoints, Qwen3-0.6B-diffusion-mdlm-v0.1 and Qwen3-0.6B-diffusion-bd3lm-v0.1.

Training. Both variants are trained with SFT only (no continual pretraining) on a standard instruction mix; we omit logits right-shifting (Gong et al., 2025; Chandrasegaran et al., 2025) as it hurt performance in our experiments. See Table 3 and Figure 8 for results and training curves; scripts are at dllm/examples/a2d.

Evaluation results. We evaluate both converted models using **dLLM**'s unified evaluation pipeline (Table 3). The BD3LM variant shows particular strength on code generation, with HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b) scores surpassing Qwen3-0.6B-Base (Yang et al., 2025) despite being trained with SFT alone. Overall, both DLM variants still trail their AR counterparts on most knowledge and reasoning benchmarks (e.g., MMLU (Hendrycks et al., 2021a), BBH (Suzgun et al., 2023)), reflecting the expected gap at this scale. Nonetheless, obtaining a competitive DLM from an off-the-shelf ARLM with SFT alone shows AR-to-diffusion conversion is practical and compute-efficient.

Takeaway

Existing pretrained models, both BERT-style encoders and autoregressive LMs, can be converted into functional DLMs with minimal compute (e.g., via SFT)

5 Related Work

Open-weight DLMs. Scaling DLMs progressed rapidly. Nie et al. first scaled masked diffusion to 1.1B parameters. Converting pretrained autoregressive models into DLMs proved effective: DiffuGPT/DiffuLLaMA adapt GPT-2 and LLaMA (127M–7B) (Gong et al., 2025); RND1 (Chandrasegaran et al., 2025) extends this to 30B; and LLaDA2.0 (Bie et al., 2025) scales to 100B with a 3-phase block scheme. At the 7–8B scale, Dream (Ye et al., 2025) adapts Qwen-2.5 with context-adaptive noise rescheduling, and LLaDA (Nie et al., 2025b) trains an 8B MDLM from scratch, achieving performance competitive with LLaMA3-8B (Grattafiori et al., 2024). Commercial systems such as Mercury (Khanna et al., 2025) demonstrate DLM viability in production.

Open tools for DLMs. Prior open-weight DLMs (Nie et al., 2025b; Ye et al., 2025; Gong et al., 2025; Chandrasegaran et al., 2025; Bie et al., 2025) often lack unified development pipelines, making reproduction and comparison difficult. Open efficient inference tools for DLMs such as Fast-dLLM (Wu et al., 2026b) and Fast-dLLM v2 (Wu et al., 2026a) accelerate decoding but are developed independently of training and evaluation. Evaluation pipelines also vary across papers (e.g., task sets, inference hyperparameters), and interfaces remain inconsistent. A framework unifying training, inference, and evaluation has been lacking. **dLLM** fills this gap with modular trainers, a plug-and-play sampler abstraction, and a reproducible evaluation pipeline aligned with official benchmarks (Section 3, Section 4).

6 Conclusion

We present **dLLM**, an open-source framework unifying DLM training, inference, and evaluation in a modular pipeline. By standardizing components shared across recent DLMs, **dLLM** lowers the barrier to reproducing, finetuning, and comparing existing models while making it straightforward to integrate new designs. We also provide minimal recipes and checkpoints showing that BERT-style and autoregressive LMs can be converted into competitive DLMs with lightweight finetuning, making DLM development accessible with minimal compute. We hope **dLLM** accelerates research and lowers the entry barrier for the broader community.

Future work. Future **dLLM** updates will integrate new open-weight models and emerging techniques, such as standard RL algorithms.

Acknowledgements

Zhanhui Zhou gratefully acknowledges support from the Berkeley Fellowship. This work is supported by NSF (2324770).

References

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021b. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Lewis Tunstall, Colin Raffel, Leandro von Werra, Thomas Wolf, and 1 others. 2025. SmolLM2: When Smol goes big – data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. 2025. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *Advances in Neural Information Processing Systems*.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, and 12 others. 2025. LLaDA2.0: Scaling up diffusion language models to 100B. *arXiv preprint arXiv:2512.15745*.

Keshigeyan Chandrasegaran, Armin W. Thomas, Jerome Ku, Federico Berto, Jae Myung Kim, Garyk Brixi, Eric Nguyen, Stefano Massaroli, and Michael Poli. 2025. RND1: Simple, scalable AR-to-Diffusion conversion. Radical Numerics Technical Report.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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), NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, and 1 others. 2024. EleutherAI/lm-evaluation-harness: v0.4.3. Zenodo.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. 2025. Edit flows: Variable length discrete flow matching with sequence-level edit operations. In *Advances in Neural Information Processing Systems*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*.

Siming Huang, Tianhao Cheng, J. K. Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, Jiaheng Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu.

2024. OpenCoder: The open cookbook for top-tier code large language models. *arXiv preprint arXiv:2411.04905*.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. 2025. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. 2022. Diffusion-LM improves controllable text generation. In *Advances in Neural Information Processing Systems*.

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

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. dKV-Cache: The cache for diffusion language models. In *Advances in Neural Information Processing Systems*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

John Nguyen, Marton Havasi, Tariq Berrada, Luke Zettlemoyer, and Ricky T. Q. Chen. 2025. OneFlow: Concurrent mixed-modal and interleaved generation with edit flows. *arXiv preprint arXiv:2510.03506*.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. 2025a. Scaling up masked diffusion models on text. In *International Conference on Learning Representations*.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025b. Large language diffusion models. In *Advances in Neural Information Processing Systems*.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*.

Yair Schiff, Subham Sekhar Sahoo, Hao Phung, Guanghan Wang, Sam Boshar, Hugo Dalla-torre, Bernardo P. de Almeida, Alexander M. Rush, Thomas Pierrot, and Volodymyr Kuleshov. 2025. Simple guidance mechanisms for discrete diffusion models. In *International Conference on Learning Representations*.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. 2024. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Remasking discrete diffusion models with inference-time scaling. In *Advances in Neural Information Processing Systems*.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. 2025. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical*

Methods in Natural Language Processing: System Demonstrations.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. 2026a. Fast-dLLM v2: Efficient block-diffusion LLM. In *International Conference on Learning Representations*.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2026b. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *International Conference on Learning Representations*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. 2025. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *Advances in Neural Information Processing Systems*.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860.

A Detailed Training Setup

A.1 Finetuning Open-Weight Large DLMs

We finetune both the Base and Instruct variants of LLaDA (Nie et al., 2025b) and Dream (Ye et al., 2025) using MDLM SFT with LoRA on the s1K dataset (Muennighoff et al., 2025). Loss is computed only on response tokens. We use a maximum sequence length of 4096, 20 epochs, a learning rate 10^{-5}, a global batch size 32 with gradient accumulation steps of 4. We apply LoRA adaptation with $r = 128$, $\alpha = 256$, and weight decay 0.1. We adopt a cosine learning-rate schedule with 10% warmup. Training is conducted on 8× A100 GPUs using DeepSpeed ZeRO-2. Scripts to reproduce are at dllm/examples/llada and dllm/examples/-dream.

A.2 BERT-Chat

We finetune ModernBERT-base and ModernBERT-large via MDLM SFT (no continual pretraining) on a mixture of instruction-tuning datasets: Tulu 3 SFT (Lambert et al., 2024) and SmolTalk (Ben Allal et al., 2025). The loss is computed only on response tokens. We use maximum sequence length 1024, 10 epochs, learning rate 10^{-4}, global batch size 384, bf16 precision, and a cosine learning-rate schedule with 10% warmup. Training runs on 8× A100 GPUs with DeepSpeed ZeRO-2 (Rajbhandari et al., 2020). Scripts to reproduce the models are at dllm/examples/bert.

A.3 Tiny-A2D (ARLM to DLM)

We train both MDLM and BD3LM variants with only SFT (no continual pretraining) on the mixture of Tulu 3 SFT (Lambert et al., 2024), SmolTalk (Ben Allal et al., 2025), and opc-sft-stage1&2 (Huang et al., 2024). Loss is computed only on response tokens and we do not apply the logits right shifting tricks as in prior work (Gong et al., 2025; Chandrasegaran et al., 2025), because in our experiments this leads to performance degradation. For the MDLM variant we use maximum sequence length 1024; for the BD3LM variant we use length 512 and block size 32. Both use 10 epochs, learning rate 10^{-4}, global batch size 2048, bf16 precision, and a cosine learning-rate schedule. Training is run on 64× A100 GPUs with DeepSpeed ZeRO-2 (Rajbhandari et al., 2020). Scripts to reproduce are at dllm/examples/a2d.

B Training Curves

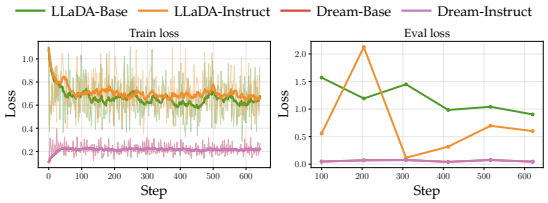


Figure 6: Training loss for finetuning open-weight DLMs to reason (Section 4.1).

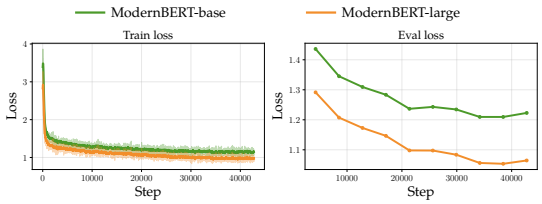


Figure 7: Training loss for finetuning BERT to chat (Section 4.2.1).

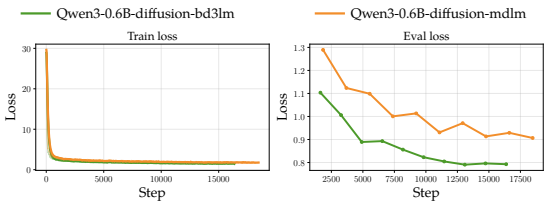


Figure 8: Training loss for finetuning autoregressive LMs to be DLMs (Section 4.2.2).

C Evaluation Reproduction

In this section, we report evaluation results comparing the official implementation (as reported in the original paper) with our unified **dLLM** reimplementation under the same configurations. Overall, our framework reproduces the official results closely across benchmarks, indicating that our evaluation pipeline and implementation are consistent with the official setup (with only minor necessary adjustments).

Tables 4 and 5 report our reproduced evaluation results for LLaDA (Nie et al., 2025b) and Dream (Ye et al., 2025) with **dLLM**. Our Fast-dLLM reimplementation achieves similar accuracy to the official numbers while substantially improving generation throughput; see Figure 4 in the main paper for visualization and Github for detailed results.

Table 4: **LLaDA evaluation results.** "Official" denotes results from the original paper; "**dLLM**" denotes results from our **dLLM** reimplementation. "Hella." stands for HellaSwag, "HEval" stands for HumanEval and "WinoG." stands for WinoGrande.

(a) LLaDA-Base

	MMLU	BBH	ARC-C	Hella.	WinoG.	PIQA	GSM8K	MATH	GPQA	HEval	MBPP
Official	65.9	49.7	45.9	70.5	74.8	73.6	70.3	31.4	25.2	35.4	40.0
dLLM	65.9	47.2	44.1	69.2	70.4	70.7	70.7	32.4	31.9	32.9	38.8

(b) LLaDA-Instruct

	MMLU	MMLU-Pro	ARC-C	Hella.	GSM8K	Math	GPQA	HEval	MBPP
Official	65.5	37.0	88.5	74.6	69.4	31.9	33.3	49.4	41.0
dLLM	69.8	36.2	86.4	76.7	74.7	31.9	30.6	47.0	40.0

Table 5: **Dream evaluation results.** "Official" denotes results from the original paper; "**dLLM**" denotes results from our **dLLM** reimplementation. "Hella." stands for HellaSwag, "HEval" stands for HumanEval and "WinoG." stands for WinoGrande.

(a) Dream-Base

	MMLU	BBH	ARC-C	Hella.	WinoG.	PIQA	GSM8K	Math	GPQA	HEval	MBPP
Official	69.5	57.9	59.9	73.3	74.8	75.8	77.2	39.6	36.6	57.9	56.2
dLLM	70.0	63.7	59.0	73.5	72.5	76.4	77.0	42.4	34.6	56.7	56.0

(b) Dream-Instruct

	MMLU	MMLU-Pro	ARC-C	Hella.	GSM8K	Math	GPQA	HEval	MBPP
Official	67.0	43.3	—	—	81.0	39.2	33.0	55.5	58.8
dLLM	69.8	45.5	61.4	71.8	82.0	48.6	31.5	57.9	58.2