dLLM



A Hands-on Tutorial for Diffusion Language Models with dLLM

KDD 2026

Presenter: Lingjie Chen

Contributors: Zhanhui Zhou, Hanghang Tong, and Dawn Song

What you will be able to do

- Run and configure an open DLM and its inference controls.
- Train it with the Hugging Face Trainer workflow.
- Convert an AR checkpoint into a diffusion language model.
- Evaluate it with a reproducible configuration.
- Prototype a custom DLM research idea.

Scope of this tutorial

Focus

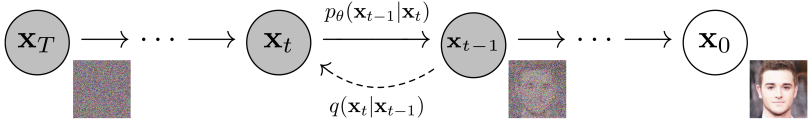
- Masked DLM inference, training, and evaluation.
- OpenRecipes for A2D conversion and research extensions.

Not in depth

- Full derivations or continuous/latent implementations.
- RL post-training or large-scale distributed training.

MODULE 1

From image diffusion to first inference



- **Forward:** progressively corrupt a clean image.
- **Reverse:** repeatedly denoise a random image.
- Text diffusion follows this structure with discrete corruption.

AR generation versus diffusion generation

Autoregressive

- Commit one token at a time, left to right.
- Each token becomes context for the next step.

Masked diffusion

- Start from a masked response canvas.
- Commit different positions over multiple calls.

Where masked diffusion fits

- **Continuous diffusion** acts on embeddings or latent states.
- **Discrete diffusion** acts on token states—our tutorial scope.
- **Masked diffusion** maps tokens to `[MASK]`—our main case.

Text diffusion inference: masked canvas to response

Masked canvas `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]`

↓ *predict every masked position*

Candidate `Diffusion language models can generate many tokens in parallel.`

↓ *keep confident candidates; remask the rest*

Next canvas `Diffusion language models [MASK] generate [MASK] tokens in parallel.`

↓ ··· *repeat*

Final generation `Diffusion language models can generate multiple tokens in parallel.`

Inside one masked-diffusion step

`masked tokens`

↓

`model forward`

↓

`select and commit part of the candidates`

Detailed scoring and transfer policies come in Module 2.

dLLM sampler: implementation code

```
sampler = dllm.core.samplers.MDLMSampler(
    model=model, tokenizer=tokenizer
)
config = dllm.core.samplers.MDLMSamplerConfig()

inputs = tokenizer.apply_chat_template(
    messages, add_generation_prompt=True, tokenize=True
)
output = sampler.sample(inputs, config, return_dict=True)
```

Inside `sample()`: initialize → forward → schedule/select → update → repeat

Continue in Notebook 01A

Open the hands-on notebook

`notebooks/kdd2026-dllm-hands-on.ipynb`

Section 01 · First diffusion generation

- Open cell `01A`: Load Tiny-A2D and generate one answer.

Module 1 takeaway

- **Generation paradigm:** DLMs refine multiple token positions over several iterations instead of committing strictly left to right.
- **Tutorial scope:** we study discrete token diffusion, with masked diffusion as the main hands-on setting.
- **Basic intuition:** begin with a masked response canvas and progressively turn it into a complete generation.

MODULE 2

Inside the sampler loop

Masked canvas `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]` `[MASK]`

↓ **Model forward:** canvas → token logits; propose one candidate per position

Candidate `Diffusion language models can generate many tokens in parallel.`

↓ **Scheduler:** set the commit budget; **selection:** rank masked positions

Next canvas `Diffusion language models [MASK] generate [MASK] tokens in parallel.`

↓ **Commit:** write selected tokens into the canvas; the sampler repeats

Final generation `Diffusion language models can generate multiple tokens in parallel.`

Core inference hyperparameters

- `max_new_tokens` — width of the generated response canvas.
- `steps` — budget for iterative denoising updates.
- **scheduler** — progress and token-transfer budget across updates.
- `temperature` — noise applied when proposing candidate tokens.
- `remasking` — ranking policy among positions that remain masked.

Change one thing at a time

Start from one baseline; change only one control per comparison.

Keep the baseline fixed	Run one comparison at a time
Model and checkpoint revision	Baseline vs. fewer denoising steps
Prompt, seed, and output length	Baseline vs. nonzero temperature
Hardware and precision	Baseline vs. another ranking policy

Mask-to-token transfer vs. editable decoding

Mask-to-token transfer

`[MASK]` ⟶ `token`

Only currently **masked positions** can be updated.

Examples

LLaDA · Dream

Nemotron-Labs-Diffusion

Editable decoding

`token` ⟶ `token'`

Existing canvas tokens can be **revised directly**.

Examples

LLaDA 2.1 · LLaDA 2.2

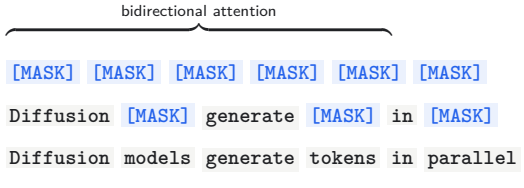
DiffusionGemma

Why full-sequence diffusion limits KV reuse

- Each iteration processes the active full output canvas.
- The canvas is not a stable left-to-right prefix.
- Block diffusion freezes completed blocks.

Visualizing generation: MDLM vs. BD3LM

Full-sequence MDLM



BD3LM



Sahoo et al., MDLM (2024); Arriola et al., Block Diffusion (2025).

Full-sequence vs. block diffusion

	Full-sequence MDLM	BD3LM
Active region	Full output canvas	Current block
Previous output	Reprocessed canvas	Frozen prefix blocks
Prefix KV cache	Not reused	Reused within a block
Length	Fixed canvas per call	Extended block by block

Notebook 02B–02C: MDLM vs. BD3LM

Structure	Tiny-A2D checkpoint	Sampler
MDLM	`Qwen3-0.6B-diffusion-mdlm-v0.1`	`MDLMSampler`
BD3LM	`Qwen3-0.6B-diffusion-bd3lm-v0.1`	`BD3LMSampler`

Compare execution structure, not checkpoint quality.

Run 00A–01A, then 02B before 02C.

MDLM vs. BD3LM: inference comparison

MDLM

Full 32-token canvas

`max_new_tokens=32`, `steps=32`

No prefix KV reuse

BD3LM

Four 8-token blocks

`max_new_tokens=32`, `block_size=8`, `steps=32`

Reuse completed-prefix KV

INFERENCE TIME: MDLM > BD3LM

Same prompt, seed, output length, hardware, and precision; CUDA-synchronized timing.

MODULE 3

Forward corruption and denoising prediction

Forward corruption q_t

$$z_t^i \sim q_t(\cdot \mid x_0^i) = \begin{cases} x_0^i, & \text{with probability } \alpha(t), \\ [\mathrm{MASK}], & \text{with probability } 1 - \alpha(t). \end{cases}$$

clean tokens x_0 $\rightarrow$ partially masked tokens z_t

Denoising prediction p_θ

$$p_\theta(x_0^i \mid z_t)$$

partially masked z_t $\rightarrow$ predict the original token x_0

Masked-token training objective

$$\mathcal{L}(\theta) = -\mathbb{E}\left[\frac{1}{t}\sum_{i}\mathbf{1}[z_t^i = \mathrm{MASK}]\log p_\theta(x_0^i \mid z_t)\right]$$

- The cross-entropy is computed only for tokens corrupted to `[MASK]`.
- Random t trains one predictor at many corruption levels.

The DLM loss: recover clean masked tokens

Original target	`"language"`
Corrupted input	`[MASK]`
Model prediction	$P(\texttt{"language"} \mid z_t)$
Training penalty	$w(t)\,[-\log P(\texttt{"language"} \mid z_t)]$

- Compute cross-entropy only at eligible masked positions.
- Weight and normalize with the configured diffusion loss settings.
- Default SFT prompt masking excludes prompt positions.

AR and DLM use the same training infrastructure

AR `data` → `tokenizer` → `AR model` → `Trainer` → `resulting AR model`

AR and DLM use the same training infrastructure

AR `data` → `tokenizer` → `AR model` → `Trainer` → `resulting AR model`

DLM `data` → `tokenizer` → `DLM model` → `Trainer` → `resulting DLM model`

AR and DLM use the same training infrastructure

AR `data → tokenizer → AR model → Trainer → resulting AR model`

DLM `data → tokenizer → DLM model → Trainer → resulting DLM model`

THE OUTER TRAINER WORKFLOW IS THE SAME!

- Both use the Hugging Face Trainer infrastructure.
- The DLM-specific difference is the corruption and loss path.

Notebook 03B: two-step training smoke test

```
RUN_TRAINING = False  # Set True to launch the smoke test.
training_args = dllm.core.trainers.MDLMConfig(
    output_dir=OUTPUT_DIR, max_steps=2,
    eval_strategy="no", report_to="none"
)
if RUN_TRAINING:
    trainer = dllm.core.trainers.MDLMTrainer(
        model=model, tokenizer=tokenizer,
        train_dataset=train_dataset,
        args=training_args, data_collator=data_collator,
    )
    trainer.train()
```

Continue in Notebook 03A–03B

Open the hands-on notebook

`notebooks/kdd2026-dllm-hands-on.ipynb`

Section 03 · Training with an AR-style recipe

- Run 03A to stream two OpenThoughts-114k training examples.
- In 03B, set `RUN_TRAINING=True` to run two training steps.

OpenRecipes: A2D conversion

`AR checkpoint → A2D config + attention → load compatible weights`

dLLM Hub collection	Representative checkpoints
Tiny-A2D	Qwen3-0.6B MDLM / BD3LM
BERT-Chat	ModernBERT-base / ModernBERT-large

`https://huggingface.co/dllm-hub`

Why start from a strong AR checkpoint?

- Reuse language knowledge, tokenizer, and existing instruction behavior.
- Avoid the cost of competitive DLM training from scratch.
- Common route: build on a pretrained AR foundation (DiffusionGemma; Nemotron-Labs-Diffusion).

Continue in the Jupyter Notebook

Open the hands-on notebook

`notebooks/kdd2026-dllm-hands-on.ipynb`

Section 03 · Training with an AR-style recipe

- Section 03 trains the already-converted Tiny-A2D checkpoint; it does not convert Qwen3.
- Follow the linked official A2D recipe for full AR-to-diffusion conversion and warmup.

What makes an evaluation reproducible

Record explicitly:

- **Model:** checkpoint and tokenizer revision.
- **Task:** task, split, and data revision.
- **Prompt:** chat template, few-shot count, and stop strings.
- **Decoding:** length, steps, temperature, and transfer policy.
- **Runtime:** seed, precision, process count, and package versions.
- **Remember:** a score alone is not a full experiment.

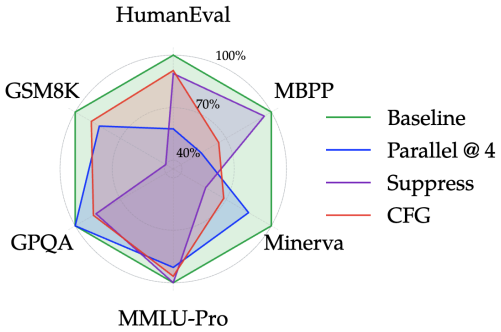
Why existing DLM baselines are difficult to reproduce

Baseline	Official route	Settings disclosed?
LLaDA-8B-Base	custom lm-eval + OpenCompass	largely documented
Dream-v0-Base-7B	repository-local lm-eval adapter	task scripts
DiffuCoder	lm-harness + qwencoder-eval	partial

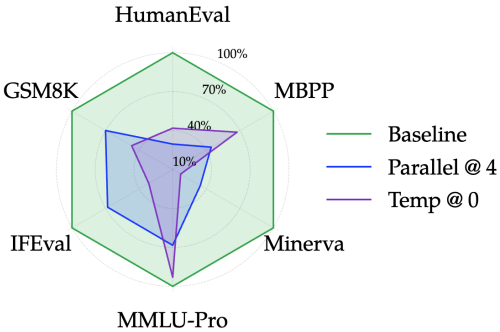
Exact evaluator revisions must be pinned.

Official LLaDA EVAL, Dream evaluation scripts, and DiffuCoder Appendix B.2.

DLM results depend on the evaluation configuration



(a) LLaDA-Instruct



(b) Dream-Instruct

The dLLM evaluation structure

`task YAML + CLI / optional eval.sh`

↓

`eval.py` → `registered A2D harness`

- The evaluation framework is an orchestration wrapper; it does not prescribe a sampler.
- **For readability,** keep all settings in one explicit script per model–dataset pair.

Notebook 04A: launch official GSM8K evaluation

```
RUN_EVALUATION = False
EVALUATION_MODEL = "mdlm"  # The notebook derives this MDLM command from the selection.

accelerate launch --num_processes 1 \
  dllm/pipelines/a2d/eval.py \
  --tasks gsm8k_cot --model a2d_mdlm \
  --apply_chat_template --num_fewshot 0 \
  --model_args \
  "pretrained=dllm-hub/Qwen3-0.6B-diffusion-mdlm-v0.1,\
steps=256,block_size=256,cfg_scale=0.0,temperature=0.0"
```

Tutorial notebook 04A; first run prints the command, then enable `RUN_EVALUATION` to launch it.

Launch, inspect, and compare

- Leave `RUN_EVALUATION=False` first and inspect the printed command.
- Select `mdlm` or `bd3lm`; then enable the flag only when ready for full GSM8K.
- Read the aggregate metric from the evaluation command output.
- Compare with a stated tolerance; one match does not validate all settings.

MODULE 5

From framework user to DLM researcher

Research question	Primary extension surface
Alpha schedule / corruption	scheduler + trainer + sampler/eval
Corrective remasking	sampler policy
New block structure	model attention + trainer + sampler
New learning objective	Trainer subclass
Transfer across tasks	evaluation adapter

Define a custom DLM as a hypothesis

`hypothesis` $\rightarrow$ `changed boundary` $\rightarrow$ `minimal path` $\rightarrow$ `matched ablation`

- State which distribution or policy changes—and what stays fixed.
- Define a falsifiable test with quality metrics and diagnostic counters.

MODULE 6

Return to the DLM architecture landscape

Axis	Representative choices
Representation	token, embedding, higher-level latent
Corruption	masking, substitution, edit operations
Generation	full canvas, blocks, variable-length edits
Attention	bidirectional, adapted decoder, encoder–decoder
Training origin	from scratch, A2D adaptation, post-training

Representative taxonomy from DLM surveys and primary model papers.

Open research questions

- How should DLMs revise mistakes and represent variable length?
- When do block diffusion and inference-time scaling deliver real gains?
- How should objectives and evaluation support reasoning, code, and comparability?

Recommended reading: FLARE

FLARE: Diffusion for Hybrid Language Model

- A conversion framework for hybrid-attention LLMs.
- One checkpoint supports AR-Trust verified decoding and Diffusion-Trust parallel denoising.
- Key finding: transfer-data quality is the primary driver of seed-checkpoint capability retention.

Recommended reading: The Flexibility Trap

ICML 2026 Outstanding Paper

- Arbitrary-order samplers can defer high-uncertainty logical-fork tokens.
- For mathematics and code, this is associated with premature collapse of solution coverage.
- JustGRPO forgoes arbitrary order during RL training, yet retains parallel decoding at inference.

Recommended reading: DFlash

DFlash: Block Diffusion for Flash Speculative Decoding

- A lightweight block-diffusion model drafts a whole token block in one forward pass.
- The AR target verifies the draft in parallel, then accepts or corrects tokens before commitment.
- Diffusion serves as a parallel speculative drafter with lossless target-model verification.

Q&A

Thanks for listening