# DECENTRALIZED DIFFUSION LANGUAGE MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Discrete diffusion language models (DLMs) generate text by iteratively denoising masked sequences, and their training objective generalizes autoregressive modeling beyond a predetermined generation order. Like other large language models, however, DLMs are trained as monolithic models, whose data-parallel scaling across GPU nodes is limited by cross-node synchronization and by diminishing returns from larger batches. We propose Decentralized Diffusion Language Models (**D-DLM**), which partition the training corpus into disjoint clusters, train one expert DLM per cluster without cross-expert synchronization, and train a router to predict the cluster membership of a noisy sequence. We prove that the probability velocity of the DLM reverse process decomposes into a mixture of cluster-conditional velocities weighted by the cluster posterior, and that the clean-token posterior admits the same decomposition. Hence, in the realizable limit, D-DLM recovers the same reverse process as centralized DLM sampling, and under masked diffusion, sampling reduces to a router-weighted ensemble of expert predictions. In pre-training at up to 1.2B parameters and 80B tokens, D-DLM improves the average downstream accuracy over monolithic training at every scale, by up to 2.05 points, with no cross-node communication, and it improves training throughput by up to $24\%$ when training is bottlenecked by cross-node communication.

## 1 INTRODUCTION

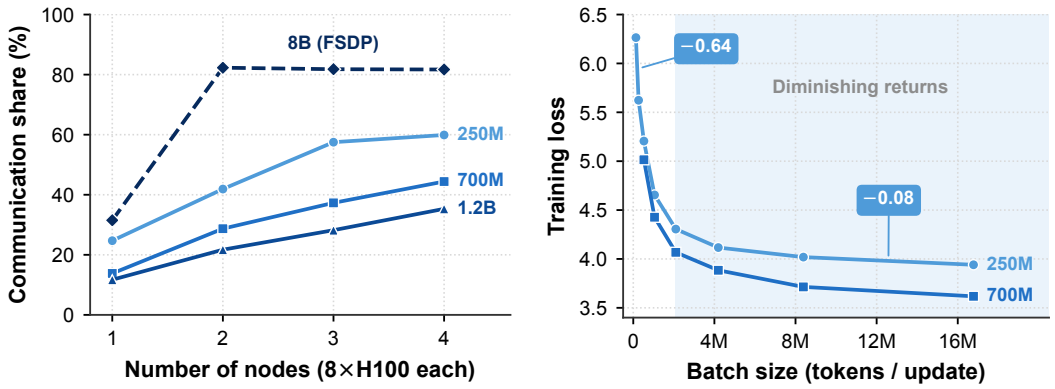
Discrete diffusion language models (DLMs) generate text by iteratively denoising a masked sequence (Austin et al., 2021; Lou et al., 2023; Shi et al., 2024a; Sahoo et al., 2024). They achieve quality comparable to autoregressive (AR) models of similar scale (Nie et al., 2025; Yu et al., 2026) and enable efficient parallel decoding, both as standalone models (Arriola et al., 2025; Bie et al., 2026; Chen et al., 2026b) and as drafters for AR models (Chen et al., 2026a). Moreover, masked diffusion generalizes AR modeling: its objective averages the negative log-likelihood of an any-order AR model over all generation orders (Hoogeboom et al., 2022; Ou et al., 2024; Xue et al., 2026).

Like other large language models, DLMs are trained as monolithic models. Communication-intensive model parallelism, such as tensor and often expert parallelism, is typically confined to a single node, where GPUs are connected by the highest-bandwidth links (e.g., NVLink) (Narayanan et al., 2021). Once intra-node parallelism is saturated, additional nodes primarily increase the degree of data parallelism, which requires gradient synchronization across all nodes at every step. This form of scaling is limited in two respects. First, synchronization over the lower-bandwidth inter-node network accounts for an increasing fraction of each training step as the node count grows (Figure 1a). Second, a higher degree of data parallelism mainly enlarges the global batch, and beyond a critical batch size, each further doubling of the batch yields a diminishing loss reduction (McCandlish et al., 2018; Von Rütte et al., 2026) (Figure 1b). Synchronous training further requires tightly coupled clusters and is vulnerable to stragglers and hardware failures. This raises a simple question: *can we train DLMs without placing all model replicas in a single tightly synchronized run?*

Low-communication methods synchronize less often, but their learners remain coupled through shared parameters (Douillard et al., 2026). Expert-based methods remove synchronization altogether by training AR experts on separate data clusters or domains (Li et al., 2022; Gururangan et al., 2023; Shi et al., 2025), or image diffusion experts combined by a router (McAllister et al., 2025; Jiang et al., 2025; 2026). These methods compose next-token distributions or continuous scores, whereas a DLM defines a categorical reverse process over noisy sequences. It is thus unclear what to compose so that a decentralized DLM sampler remains valid.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.




(a) Communication share vs. number of nodes.

(b) Training loss vs. batch size.

Figure 1: (a) Per-rank fraction of step wall-time spent in NCCL collectives (allreduce for DDP, allgather/reduce-scatter for FSDP) during steady-state pre-training of four LLaDA model sizes, measured with `torch.profiler` over a 80-step window after warmup. Solid lines use DDP and the dashed line FSDP. As training scales out, cross-node IB hops stack on top of the intra-node NVLink ring, so the communication share grows with the number of nodes for every model. (b) Training loss of LLaDA 250M and 700M after the same number of optimizer steps, as the batch size grows from 0.13M to 16.8M tokens per update. The labeled values are the loss reductions from doubling the batch of LLaDA-250M, which shrink from 0.64 at the smallest batch to 0.08 at the largest. Beyond 2.1M tokens/update (shaded), each doubling reduces the loss by less than 0.2 nats for both models, whereas a larger batch requires more nodes and hence more communication.

We propose *Decentralized Diffusion Language Models* (D-DLM), which partition the training corpus into disjoint clusters, train one expert DLM per cluster without cross-expert synchronization, and train a router to predict the cluster membership of a noisy sequence. Our **theoretical result** establishes that this recipe is principled. We prove that the probability velocity governing the reverse process of a DLM decomposes into a mixture of cluster-conditional velocities, weighted by the *cluster posterior*, i.e., the probability that the noisy sequence originates from each cluster (Proposition 3.1). Since the velocity is affine in the *clean-token posterior*, i.e., the distribution of clean tokens given a noisy sequence, the clean-token posterior admits the same decomposition into cluster-conditional posteriors. The experts learn these cluster-conditional posteriors and the router learns the cluster posterior. In the realizable full-mixture limit, D-DLM recovers the same reverse process as centralized DLM sampling. Under masked diffusion, both posteriors are time-independent, and sampling reduces to a router-weighted ensemble of expert predictions that is compatible with standard DLM samplers.

We evaluate D-DLM in pre-training at up to 1.2B parameters and 80B tokens. With **no cross-node communication**, D-DLM improves the average downstream accuracy over monolithic training at every scale, by up to 2.05 points. When training is bottlenecked by cross-node communication, it also improves training throughput by up to $24\%$. Our contributions are:

- We show that the discrete diffusion reverse process admits an exact decentralized form, weighted by a cluster posterior that is time-independent under masked diffusion.
- Based on the decentralized form, we propose D-DLM, a decentralized training and inference framework for masked DLMs that trains experts without synchronization and combines them through a router-weighted ensemble, removing cross-node communication while matching or improving monolithic pre-training at up to 1.2B parameters.
- We find k-means an effective data partitioning strategy, whose clusters yield lower expert loss and a learnable router, unlike random partitions. We build a scalable text embedding and hierarchical k-means pipeline to partition pre-training corpora.

## 2 Preliminaries

**Notations.** We denote the set of one-hot embeddings for $K$ categories by $\mathcal{E}^K = \{\boldsymbol{\delta}_1, \cdots, \boldsymbol{\delta}_K\}$, where $\boldsymbol{\delta}_j$ is the $j$-th canonical basis vector. We define the $K$-simplex as $\Delta^K = \{\boldsymbol{\pi} \in$

$[0,1]^K | \sum_{j=1}^K \boldsymbol{\pi}_j = 1\}$. Given $\boldsymbol{\pi} \in \Delta^K$, let $\mathrm{Cat}(\cdot|\boldsymbol{\pi})$ denote the categorical distribution over $\mathcal{E}^K$, where the probability of the $j$-th class is given by $\boldsymbol{\pi}_j$. We further denote $\mathrm{supp}(\mathrm{Cat}(\cdot|\boldsymbol{\pi}))$ (or $\mathrm{supp}(\boldsymbol{\pi})$ for shorthand) as the support of distribution $\mathrm{Cat}(\cdot|\boldsymbol{\pi})$, i.e., $\mathrm{supp}(\boldsymbol{\pi}) = \{\boldsymbol{\delta} \in \mathcal{E}^K | \, \mathrm{Cat}(\boldsymbol{\delta}|\boldsymbol{\pi}) \neq 0\}$. In particular, $\mathrm{Cat}(\cdot|\boldsymbol{\delta}_j)$ denotes the delta distribution concentrated on the $j$-th class. In this work, we focus on language modeling. Suppose the vocabulary size is $K$ and the sequence length is $N$. A sequence of tokens can be represented as $\boldsymbol{X} \in \mathcal{E}^{N \times K}$, where $\boldsymbol{X}^i \in \mathcal{E}^K$ denotes the $i$-th token. We assume that the `[mask]` token is included in the vocabulary, and use $\boldsymbol{m} \in \mathcal{E}^K$ to denote the corresponding one-hot vector concentrated on `[mask]`.

**Discrete Flow Models.** Generative diffusion models (Sohl-Dickstein et al., 2015; Song & Ermon, 2019; Song et al., 2020; 2021) and flow models (Lipman et al., 2022; Liu et al., 2022) enable sampling from the target distribution by constructing a probability path between a known initial distribution and the unknown target data distribution. In the discrete-variable setting (Austin et al., 2021; Sun et al., 2022; Lou et al., 2023; Sahoo et al., 2024; Campbell et al., 2024; Gat et al., 2024), such distribution paths can be constructed by defining a forward process that interpolates between a clean data distribution $p_0$ and a samplable initial distribution $p_1 = \mathrm{Cat}(\cdot|\boldsymbol{\pi})$ with $\boldsymbol{\pi} \in \Delta^K$. Given two *independent* samples $\boldsymbol{x}_0 \sim p_0$ and $\boldsymbol{x}_1 \sim p_1$, a time-varying latent variable $\boldsymbol{x}(t)$ can be generated by the following conditional distribution [1]:

$$p_t(\boldsymbol{x}(t)|\boldsymbol{x}_0, \boldsymbol{x}_1) = \mathrm{Cat}\left(\boldsymbol{x}(t)|\alpha_t \boldsymbol{x}_0 + (1-\alpha_t)\boldsymbol{x}_1\right), \tag{1}$$

where $\alpha_t \in [0,1]$ is a strictly decreasing function of $t \in [0,1]$ such that $\alpha_t \to 1$ as $t \to 0$ and $\alpha_t \to 0$ as $t \to 1$. Simulating $\boldsymbol{x}(t)$ from $\boldsymbol{x}_0$ is straightforward: with probability $\alpha_t$, $\boldsymbol{x}(t) = \boldsymbol{x}_0$ remains unchanged; with probability $1-\alpha_t$, $\boldsymbol{x}(t)$ transitions to a random state $\boldsymbol{x}_1$ sampled from $\boldsymbol{\pi}$. This constructed dynamics in distribution space corresponds to a reverse-time sample-space process that transports probability mass from $p_1$ to $p_0$:

$$\boldsymbol{x}(t - \Delta t) \sim \mathrm{Cat}(\cdot|\boldsymbol{x}(t) + \Delta t \cdot u_t(\boldsymbol{x}(t))), \tag{2}$$

where $\Delta t$ is an infinitesimal time interval and vector field $u_t : \mathcal{E}^K \to \mathbb{R}^K$ is referred to as the *discrete velocity field*, satisfying $\sum_{j=1}^K u_t(\boldsymbol{z})_j = 0$ and $u_t(\boldsymbol{z})_j \geq 0$ for $\boldsymbol{\delta}_j \neq \boldsymbol{z}$. The velocity field $u_t$ serves as the core component of the discrete flow model, characterizing the transition probability from time $t$ to time $t - \Delta t$. As shown by Gat et al. (2024), the probability velocity field can be expressed as:

$$u_t(\boldsymbol{z}) = \frac{\alpha_t'}{1-\alpha_t}\left[\boldsymbol{z} - p_{0|t}(\cdot|\boldsymbol{z})\right], \tag{3}$$

where $\alpha_t'$ denotes the time derivative $d\alpha_t/dt$, and $p_{0|t}(\cdot|\boldsymbol{z}) \in \Delta^K$ represents the conditional probability of $\boldsymbol{x}_0$ given $\boldsymbol{x}(t) = \boldsymbol{z}$, induced from the path constructor $p_t(\boldsymbol{z}|\boldsymbol{x}_0, \boldsymbol{x}_1)$. This analytic formulation yields a natural parameterization and training objective for discrete flow models by maximizing the log-likelihood of the posterior $p_{0|t}$ parameterized by time-dependent neural networks $f_\theta(\boldsymbol{z}, t) : \mathcal{E}^K \times [0,1] \to \Delta^K$:

$$\mathcal{L} = -\mathbb{E}_{t, \boldsymbol{x}_0 \sim p_0, \boldsymbol{x}_1 \sim p_1, \boldsymbol{x}(t) \sim p_t(\cdot|\boldsymbol{x}_0, \boldsymbol{x}_1)}[\omega(t) \log \langle f_\theta(\boldsymbol{x}(t), t), \boldsymbol{x}_0 \rangle], \tag{4}$$

where $t \sim \mathcal{U}[0,1]$ is typically sampled from a uniform distribution, and $\omega(t)$ is a time-dependent function reweighting the log-likelihood over time.

**Diffusion Language Models.** Language sequences are naturally discrete data, making discrete flow and diffusion models a compelling foundation for generative language modeling (Shi et al., 2024a; Arriola et al., 2025; Nie et al., 2025; Bie et al., 2026; Ye et al., 2025). Two specializations are required to instantiate the general discrete flow model in this setting. First, we augment the vocabulary with a `[mask]` token and define the source distribution $p_1$ as an i.i.d. delta distribution concentrated on `[mask]` at each token position: $p_1(\boldsymbol{X}) = \prod_{i=1}^N \mathrm{Cat}(\boldsymbol{X}^i|\boldsymbol{m})$. Second, we define the path constructor token-wise, randomly replacing each token with the `[mask]` token independently. Given

[1] Note that we will follow the mathematical formulation of continuous-time discrete flow models in Gat et al. (2024) to introduce discrete flow models and derive our results to give a clearer picture of the theoretical foundation. The discrete-time diffusion models (Austin et al., 2021; Lou et al., 2023; Sahoo et al., 2024) share the same core components and therefore do not affect our results.

$\boldsymbol{X}_0 \sim p_0$ and the one-hot embedding for the mask token $\boldsymbol{m}$, the conditional path distribution is formulated as:

$$p_t(\boldsymbol{X}(t)|\boldsymbol{X}_0) = \prod_{i=1}^{N} \mathrm{Cat}\left(\boldsymbol{X}(t)^i | \alpha_t \boldsymbol{X}_0^i + (1-\alpha_t)\boldsymbol{m}\right). \tag{5}$$

According to Gat et al. (2024), Equation 5 induces a token-wise velocity field

$$u_t(\boldsymbol{Z})^i = \frac{\alpha_t'}{1-\alpha_t}\left[\boldsymbol{Z}^i - p_{0|t}(\cdot|\boldsymbol{Z})^i\right].$$

This leads to a BERT-like training objective $\mathcal{L} = -\sum_{i=1}^{N} \mathbb{E}_{t,\boldsymbol{X}_0,\boldsymbol{X}(t)} \log \left\langle f_\theta(\boldsymbol{X}(t),t)^i, \boldsymbol{X}_0^i \right\rangle$, where the model learns to predict the clean token $\boldsymbol{X}_0^i$ at each position $i$ from the masked sequence $\boldsymbol{X}(t)$. And sampling can then be performed element-wise via $\boldsymbol{X}(t-\Delta t)^i \sim \mathrm{Cat}(\cdot|\boldsymbol{X}(t)^i + \Delta t \cdot u_t(\boldsymbol{X}(t))^i)$.

## 3 Our Approach

In this section, we first show that the discrete velocity field defined above can be decomposed into disjoint domains. This decomposition enables the velocity field to be learned independently on each subdomain while allowing joint inference by aggregating the velocity estimates computed across domains. We then show how this theoretical insight can be incorporated into fully distributed training for diffusion language models.

### 3.1 Decomposing Discrete Velocity Field

Consider dividing the target domain into $C$ clusters through a conditional distribution $p(y|\boldsymbol{x}_0)$, where $y \in [C]$ denotes the cluster label associated with the data point $\boldsymbol{x}_0 \sim p_0$. Our key observation is summarized in the following proposition:

**Proposition 3.1.** *The following equation holds:*

$$u_t(\boldsymbol{z}) = \sum_{l=1}^{C} p_t(y=l|\boldsymbol{z}) u_t^{(l)}(\boldsymbol{z}), \tag{6}$$

*where $u_t^{(l)} : \mathcal{E}^K \to \mathbb{R}^K$ is called the velocity sub-field as it represents the velocity field estimated on the distribution $p_0(\boldsymbol{x}_0|y=l)$ of data cluster $l$: $u_t^{(l)}(\boldsymbol{z}) = \frac{\alpha_t'}{1-\alpha_t}\left[\boldsymbol{z} - p_{0|t}(\cdot|\boldsymbol{z}, y=l)\right]$.*

A detailed derivation is provided in Appendix B. Notably, the term $p_{0|t}(\cdot|\boldsymbol{z}, y=l)$ inside the brackets corresponds to $p_{0|t}$ defined on each subdomain given by $p_0(\boldsymbol{x}_0|y=l)$. Consequently, the full velocity field can be expressed as a weighted average of the velocity fields estimated on these subdomains.

**Training.** Equation 6 indicates that fitting the full velocity field $u_t$ can be decomposed into training each velocity sub-field $u_t^{(l)}$ independently on its data cluster, and training another classifier to approximate $p_t(y=l|\boldsymbol{z})$. Suppose we have a training dataset $p_0(\boldsymbol{x}, y)$, we parameterize each $u_t^{(l)}$ with $u_t^{(l)}(\boldsymbol{z}) = \frac{\alpha_t'}{1-\alpha_t}\left[\boldsymbol{z} - f_\theta^{(l)}(\boldsymbol{z}, t)\right]$, where $f_\theta^{(l)} : \mathcal{E}^K \times [0,1] \to \Delta^K$ is a neural network modeling $p_{0|t}(\cdot|\boldsymbol{z}, y=l)$. We refer to $f_\theta^{(l)}$ as an *expert network*, since it models the probability velocity on a subdomain of data. The training objective is to maximize the log-likelihood on each expert separately:

$$\mathcal{L}_{expert}^{(l)} = -\mathbb{E}_{t,\boldsymbol{x}_0\sim p_0(\cdot|y=l),\boldsymbol{x}_1\sim p_1,\boldsymbol{x}(t)\sim p_t(\cdot|\boldsymbol{x}_0,\boldsymbol{x}_1)}\left[\omega(t)\log\left\langle f_\theta^{(l)}(\boldsymbol{x}(t),t), \boldsymbol{x}_0\right\rangle\right], \quad \forall l \in [C], \tag{7}$$

where $\omega(t)$ is a time-dependent weighting coefficient. Note that each $\mathcal{L}_{expert}^{(l)}$ depends on a completely disjoint set of parameters, thus, can be minimized *in parallel*. In the meantime, we adopt another neural network $g_\phi(\boldsymbol{z}, t) : \mathcal{E}^K \times [0,1] \to \Delta^C$ to learn the data partition. We call $g_\phi(\boldsymbol{z}, t)$ the *router network*, as it assigns the importance score for each expert. Similarly, it can be trained with a standard maximum likelihood objective:

$$\mathcal{L}_{router} = -\mathbb{E}_{t,(\boldsymbol{x}_0,y)\sim p_0,\boldsymbol{x}_1\sim p_1,\boldsymbol{x}(t)\sim p_t(\cdot|\boldsymbol{x}_0,\boldsymbol{x}_1)}\left[\log\left\langle g_\phi(\boldsymbol{x}(t),t), \boldsymbol{\delta}_y\right\rangle\right]. \tag{8}$$

**Inference.** Once experts $f_\theta^{(l)}(\boldsymbol{x}(t), t)$ and router $g_\phi(\boldsymbol{x}(t), t)$ are obtained, the full velocity field can be recovered via Equation 6. Specifically, at inference time, we feed the current latent variable $\boldsymbol{x}(t)$ into each expert in parallel, while obtaining the assignment scores from $g_\phi(\boldsymbol{x}(t), t)$. We then aggregate the output of every expert using these scores to obtain $u_t$, which is then used to sample $\boldsymbol{x}(t - \Delta t)$ following Equation 2. Due to the highly clustered structure of the data, inference can be further accelerated by activating only the top-$k$ experts selected by the router. In practice, samplers such as those of MDLM (Sahoo et al., 2024) and LLaDA (Nie et al., 2025) decode directly from an approximator of $p_{0|t}$ instead of the velocity field. They also apply to our decentralized form, since $u_t$ is affine in $p_{0|t}$ (Equation 3).

### 3.2 Decentralized Diffusion Language Models

We extend this probability velocity field decomposition to decentralized diffusion language model training, resulting in our proposed **D-DLM** framework.

**Expert DLM Training.** We instantiate our expert models using masked diffusion language models (MDLMs) (Shi et al., 2024a; Sahoo et al., 2024). As described in Section 2, the initial distribution $p_1$ is a point mass at $[\boldsymbol{m}, \cdots, \boldsymbol{m}]$, where $\boldsymbol{m}$ denotes the one-hot embedding for the `[mask]` token. The forward process randomly masks the tokens in the sequence, while the reverse process reconstructs the original tokens from the masked sequence. For each data subset, the expert network is implemented as a sequence-to-sequence unmasking predictor: $f_\theta^{(l)} : \mathcal{E}^{N\times K} \to \Delta^{N\times K}$. We adopt SUBS parameterization (Sahoo et al., 2024), under which the predictor enforces zero probability to the `[mask]` token, and an unmasked token is not transitioned back to `[mask]`. Moreover, when $p_1$ is a delta distribution on the fully masked sequence, the mask predictor becomes time-independent (Gat et al., 2024; Zheng et al., 2024), allowing us to omit variable $t$ and re-formulate it as $f_\theta^{(l)}(\boldsymbol{X}(t))$ to estimate $p_0(\cdot|\boldsymbol{X}(t), y = l)$. We also adopt the MDLM weighting scheme $\omega(t) = -\frac{\alpha_t'}{1-\alpha_t}$ in the training objective, which boils down to optimizing the NELBO (Sahoo et al., 2024).

**Router Design.** Given a noisy sequence containing `[mask]` tokens from an intermediate denoising step, the router is trained to predict the association probabilities between the noisy sequence and the data clusters. These probabilities are then used as weighting coefficients to aggregate the expert outputs. Consequently, we formulate the router as a sequence model. We use the same backbone architecture as experts and add a routing head, implemented as a linear-layer classifier, to predict logits over all clusters. Specifically, we append a special register token (Dosovitskiy et al., 2020; Darcet et al., 2023) to the noisy sequence and map its hidden state in the final layer to the cluster logits. Since the cluster posterior is time-independent under the masked source (see Appendix C), we can omit the time conditioning in the router. It is worth noting that there is no router-expert communication during training according to our decentralized form (Equation 6).

**Data Clustering.** The formulation in Section 3.1 does not impose specific constraints on the construction of disjoint data clusters. In principle, this allows for any partitioning strategy, including random partitioning. In practice, however, the empirical optimality gaps of subset-specific experts can introduce a performance discrepancy between decentralized and canonical monolithic training. This concern arises from two competing effects of restricting each expert to a subset of the corpus. On the one hand, a more focused data subset may reduce learning complexity and data redundancy, thereby facilitating expert optimization. On the other hand, reducing the amount of training data available to each expert may also limit data diversity, which is particularly crucial for language model training. Following the data clustering strategy in McAllister et al. (2025), we use hierarchical k-means to partition the training corpus. Our ablation study in Section 4.3 shows that k-means clustering is a key to achieving performance comparable to or better than canonical monolithic training baselines, in contrast to random partitioning of the training corpus.

**Decentralized Sampling.** We follow the reverse sampling and decoding process of LLaDA with a low-confidence remasking strategy (Nie et al., 2025), which has been commonly used and adapted for improved text generation quality (Wu et al., 2025; Bie et al., 2025; Ye et al., 2025). Under the MDLM parameterization, combining decentralized experts with the router yields an ensemble form for the final decoding distribution, i.e., $\hat{p}_0(\cdot|\boldsymbol{X}(t)) = \sum_{l=1}^{C} g_\phi(\boldsymbol{X}(t))_l \, f_\theta^{(l)}(\boldsymbol{X}(t))$, following Equation 6.

Evaluating this ensemble form requires querying all expert and router predictors. Since the expert and router evaluations are independent, they can be deployed on separate devices and executed in parallel, reducing the inference-time overhead introduced by expert ensembling. With the router and each expert on its own GPU, a D-DLM forward pass takes nearly the same latency as the monolithic model on a single GPU (Figure 2b).

# 4 EXPERIMENTS

We mainly evaluate D-DLM in pre-training. More results and details are in the appendix.

## 4.1 IMPLEMENTATION DETAILS

**Expert and router models.** Our expert model training follows the training recipe of MDLM (Sahoo et al., 2024), with the parameterization and architecture instantiated from LLaDA configurations (Nie et al., 2025). Our router uses the same model parameterization and backbone architecture, augmented with a routing head that predicts logits over data clusters from the hidden state of the <|expert_cls|> token in the last Transformer block. We also design a router model for Block Diffusion (Arriola et al., 2025), which supports flexible-length decoding and improves inference efficiency through KV caching. See Appendix E for the details of the Block Diffusion router design. Our implementation builds on the recent dLLM codebase (Zhou et al., 2026).

**Data partitioning.** To partition the training corpus for decentralized training, we develop a text embedding and clustering pipeline. We use an open-weight LLM as the text encoder and obtain document-level embeddings by pooling the final hidden states of each document's last token. The resulting embeddings are then clustered using k-means. To scale this procedure to large text corpora, we design a hierarchical k-means framework for both training and assignment. Since embedding and clustering large-scale text datasets require careful system-level design for throughput and I/O efficiency, Appendix D provides additional details on the modular design of our pipeline.

## 4.2 PRE-TRAINING EVALUATION

We evaluate D-DLM in two controlled pre-training settings, each reproducing at our scale a regime that limits large-scale pre-training: 1) training under the data-parallel bound, where enlarging the global batch yields diminishing returns (Section 4.2.1), and 2) training under the communication bound, where cross-node synchronization limits training throughput (Section 4.2.2). All models, from 250M to 1.2B parameters[2], use the Transformer architecture and tokenizer adopted by LLaDA (Nie et al., 2025) and are pre-trained on FineWeb-Edu (Penedo et al., 2024). For decentralized training, we partition the corpus into $C = 4$ clusters with our text clustering pipeline and train one expert on each cluster. Appendix D.2 presents detailed configurations and evaluation of our data clustering.

### 4.2.1 TRAINING UNDER DATA-PARALLEL BOUND

In pre-training, communication-intensive parallelism such as expert and tensor parallelism is typically kept within a GPU node to exploit its fast interconnect (Narayanan et al., 2021). Once a model fits within a node, scaling out to more nodes therefore mainly increases the degree of data parallelism and thus the global batch. Beyond a large global batch, however, further enlarging it yields diminishing returns (Figure 1b), which bounds the benefit of scaling out. We refer to this as the data-parallel bound, and test whether decentralized training can improve performance beyond it under a fixed budget of training tokens. To simulate large-scale pre-training, we saturate each node with a large batch. We train 250M, 700M, and 1.2B models on 80B tokens of FineWeb-Edu with a sequence length of 1024, using four $8\times$ H100 nodes and gradient accumulation to a per-node batch of 8,192 sequences for 250M and 700M models, 16,384 for 1.2B. The monolithic baseline runs DDP across all four nodes. We also compare DiLoCo (Douillard et al., 2023) as a strong baseline, which runs one worker per node and synchronizes them every $H$ steps. D-DLM trains one expert per node on one data cluster, and the router on a separate node.

[2]These parameter counts include only the Transformer backbone and exclude the embedding layers.

Table 1: Pre-training under the data-parallel bound. All methods are trained on four $8\times$H100 nodes with a saturated per-node batch and a matched budget of 80B training tokens. Cross-node comm. is the gradient traffic between nodes, and total NCCL comm. additionally includes the traffic within each node; both are accumulated over the same number of training steps for the four LM training nodes. The best and second-best results for each model size are bolded and underlined, respectively.

| Method | WikiText-2 ppl ↓ | LAMB. ppl ↓ | Arc-C acc_n ↑ | Arc-E acc ↑ | Hella. acc_n ↑ | OBQA acc_n ↑ | TQA mc2 ↑ | Avg. acc ↑ | Cross-node comm. ↓ | Total NCCL comm. ↓ |
|---|---|---|---|---|---|---|---|---|---|---|
| *250M / FineWeb-Edu 80B* | | | | | | | | | | |
| Monolithic (DDP) | $\leq$ 54.25 | $\leq$ 100.14 | 21.3 | 40.4 | 28.5 | <u>25.8</u> | <u>46.7</u> | 42.2 | 22.2 TB | 229.1 TB |
| DiLoCo ($H$=50) | $\leq$ **48.39** | $\leq$ **92.04** | <u>21.6</u> | **43.4** | **29.8** | 24.8 | 45.5 | **42.5** | <u>0.44 TB</u> | <u>211.5 TB</u> |
| D-DLM | $\leq$ <u>51.12</u> | $\leq$ <u>95.93</u> | **21.8** | <u>42.1</u> | <u>29.7</u> | **27.4** | **46.8** | <u>42.3</u> | **0 TB** | **206.9 TB** |
| *700M / FineWeb-Edu 80B* | | | | | | | | | | |
| Monolithic (DDP) | $\leq$ 38.83 | $\leq$ <u>77.16</u> | 22.4 | 47.0 | 33.0 | <u>27.0</u> | <u>44.4</u> | 44.8 | 49.7 TB | 513.7 TB |
| DiLoCo ($H$=50) | $\leq$ **34.80** | $\leq$ **69.56** | **23.6** | <u>49.1</u> | **33.9** | 26.8 | 44.2 | <u>45.4</u> | <u>0.99 TB</u> | <u>474.3 TB</u> |
| D-DLM | $\leq$ <u>38.08</u> | $\leq$ 77.28 | **23.6** | **50.0** | <u>33.8</u> | **28.2** | **44.9** | **45.7** | **0 TB** | **464.0 TB** |
| *1.2B / FineWeb-Edu 80B* | | | | | | | | | | |
| Monolithic (DDP) | $\leq$ <u>43.78</u> | $\leq$ <u>84.75</u> | <u>21.4</u> | <u>43.5</u> | <u>30.5</u> | <u>25.4</u> | 45.8 | <u>42.5</u> | 46.0 TB | 475.2 TB |
| DiLoCo ($H$=25) | $\leq$ 83.53 | $\leq$ 144.50 | 20.7 | 37.2 | 27.2 | 25.2 | **48.2** | 38.7 | <u>1.84 TB</u> | <u>448.3 TB</u> |
| D-DLM | $\leq$ **39.19** | $\leq$ **78.83** | **23.2** | **47.4** | **32.7** | **28.6** | <u>46.1</u> | **44.6** | **0 TB** | **429.2 TB** |

Table 2: Pre-training under the communication bound (Section 4.2.2). Both methods are trained on eight single-GPU GH200 nodes with a matched budget of 24 hours.

| Model | WikiText-2 ppl ↓ | ArXiv ppl ↓ | LM1B ppl ↓ | LAMB. ppl ↓ | Arc-C acc_n ↑ | Arc-E acc ↑ | Hella. acc_n ↑ | SciQ acc ↑ | OBQA acc_n ↑ | Avg. acc ↑ |
|---|---|---|---|---|---|---|---|---|---|---|
| *250M / FineWeb-Edu 20B / 24-hour training budget* | | | | | | | | | | |
| Monolithic | $\leq$ 85.25 | $\leq$ 59.60 | $\leq$ 216.70 | $\leq$ 123.50 | **23.1** | 36.5 | 27.7 | 57.6 | 25.0 | 34.0 |
| D-DLM | $\leq$ **82.14** | $\leq$ **58.61** | $\leq$ **211.64** | $\leq$ **114.98** | 22.6 | **38.2** | **27.9** | **59.3** | **26.6** | **34.9** |
| *700M / FineWeb-Edu 80B / 24-hour training budget* | | | | | | | | | | |
| Monolithic | $\leq$ **68.21** | $\leq$ 56.82 | $\leq$ **170.90** | $\leq$ **104.89** | 22.6 | 36.2 | 27.2 | **57.4** | 24.8 | 33.6 |
| D-DLM | $\leq$ 68.92 | $\leq$ **54.66** | $\leq$ 177.49 | $\leq$ 106.61 | **23.0** | **38.2** | **29.3** | 57.2 | **27.0** | **34.9** |

**Results.** As shown in Table 1, D-DLM improves the average downstream score over the monolithic baseline at all three scales, by up to 2.05 points at 1.2B, **without any cross-node communication**, whereas the monolithic baseline exchanges 22 to 50 TB across nodes. The gain grows with model scale, and at 1.2B, D-DLM achieves the best result on all but one metric. DiLoCo attains lower perplexity at 250M and 700M with comparable downstream accuracy, but it still synchronizes across nodes and falls behind at 1.2B with $H = 25$ in our runs.

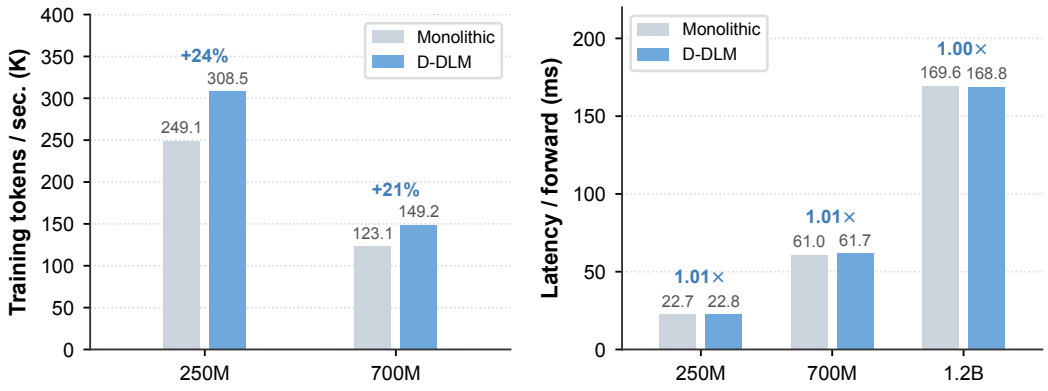
#### 4.2.2 Training under the Communication Bound

Cross-node communication becomes a major training cost for larger models (Figure 1a), but our resources restrict pre-training to relatively small models rather than 100B+ parameter ones. To simulate communication-bound training at small scale, we use a less regular cluster of eight GH200 nodes, each with a single GPU, where cross-node communication is not well-optimized for large-scale LLM pre-training, so that synchronization is costly even for 250M and 700M models. We test whether decentralized training can improve training throughput under a fixed budget of 24 hours. The monolithic model is trained with DDP across all eight nodes, while D-DLM trains four experts on independent 2-node groups and a lightweight router separately. The 250M and 700M models are trained on 20B- and 80B-token subsets of FineWeb-Edu, respectively, following prior work (Shi et al., 2024a; Yang et al., 2024; Zheng et al., 2025), with a sequence length of 2048 and a per-GPU batch size of 64, so the monolithic baseline has a $4\times$ larger global batch than each expert.

**Results.** As shown in Figure 2a, D-DLM improves training throughput by 24% and 21% in tokens per second (24% and 20% in TFLOPS) at the two scales. With the same training time, it also outperforms the monolithic baseline on 4 out of 5 downstream tasks at both scales, improving the average score from 34.0 to 34.9 at 250M and from 33.6 to 34.9 at 700M (Table 2).

### 4.3 Ablation Study

**Random partitioning vs. k-means.** The decomposition in Section 3.1 holds for any data partition, but the partition determines how well the experts and the router can be trained. To isolate this effect,




(a) Training throughput.

(b) Inference latency.

Figure 2: Training and inference efficiency of D-DLM versus the monolithic model. (a) Training throughput under the communication bound (Section 4.2.2) on eight single-GPU GH200 nodes. (b) Inference latency of parallel serving, measured as the latency of one forward pass on 8 sequences of 1,024 tokens on H100 GPUs; the monolithic model runs on one GPU, while D-DLM runs the router and the four experts in parallel on five GPUs.

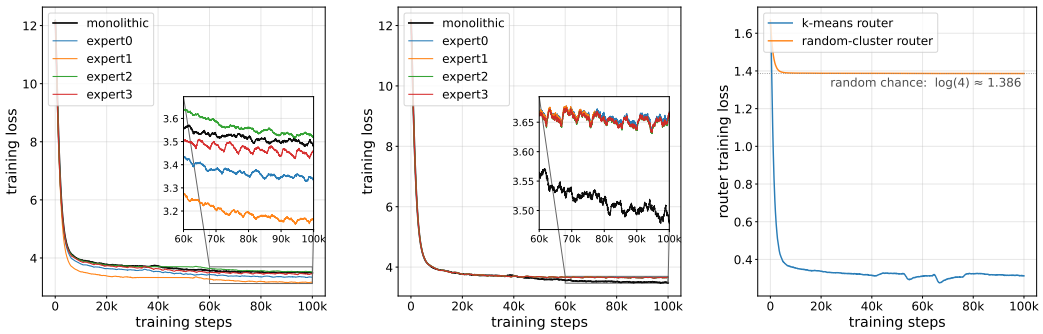
Table 3: D-DLM with k-means experts versus the monolithic model. Each expert and the monolithic model are trained with the same batch size for 100K steps on a single $8\times$H100 node.

| Model | WikiText-2 ppl ↓ | ArXiv ppl ↓ | LM1B ppl ↓ | LAMB. ppl ↓ | Arc-C acc_n ↑ | Arc-E acc ↑ | Hella. acc_n ↑ | SciQ acc ↑ | OBQA acc_n ↑ |
|---|---|---|---|---|---|---|---|---|---|
| *250M / FineWeb-Edu 20B / 100K training steps* | | | | | | | | | |
| Monolithic | $\leq$ 50.29 | $\leq$ 36.17 | $\leq$ 143.27 | $\leq$ 86.56 | 23.0 | 42.0 | 29.9 | 73.4 | 26.0 |
| D-DLM | $\leq$ **44.34** | $\leq$ **31.59** | $\leq$ **126.54** | $\leq$ **79.00** | **23.9** | **46.6** | **31.4** | **75.3** | **27.2** |
| *700M / FineWeb-Edu 80B / 100K training steps* | | | | | | | | | |
| Monolithic | $\leq$ 35.87 | $\leq$ 26.66 | $\leq$ 99.76 | $\leq$ 68.13 | 23.2 | 48.6 | 35.1 | **82.5** | 28.2 |
| D-DLM | $\leq$ **30.39** | $\leq$ **24.36** | $\leq$ **85.62** | $\leq$ **62.59** | **27.3** | **52.4** | **38.0** | 82.0 | **28.6** |

we train each expert with the same batch size and number of training steps as the monolithic model: the monolithic model and each of the four experts are trained on a single $8\times$ H100 node for 100K steps, with the configuration of Section 4.2.2. Figure 3 shows the 250M training curves. With k-means clustering, the experts are better optimized than the monolithic model: two of the four experts reach noticeably lower training loss on their clusters than the monolithic model on the full corpus, one is slightly lower, and one is comparable (Figure 3a). With random partitioning, the experts instead converge to higher losses than the monolithic model, since each random subset reduces the amount of data without reducing its complexity (Figure 3b). K-means clusters also make the router learnable, whereas its loss stays at the level of random chance under random partitioning (Figure 3c). These results identify k-means partitioning as a key component of effective D-DLM training.

The next question is whether aggregating the better-optimized k-means experts through the router weights outperforms the monolithic model, that is, whether the per-cluster gains of the experts boost rather than conflict with each other. As shown in Table 3, D-DLM achieves lower perplexity on all four evaluation corpora at both scales, and improves the average downstream score from 38.9 to 40.9 at 250M and from 43.5 to 45.7 at 700M. This shows that the router combines the experts without cancelling their gains, so that the better optimization of each expert on its own cluster carries over to the aggregated model.

**Impacts of router schemes.** The D-DLM sampling combines predictions from all experts using the router outputs as weights, requiring the router and all experts to be queried at each sampling step. This full-expert evaluation can introduce inference-time overhead in practical applications. To reduce this cost, we consider two alternative routing schemes: Top-$k$ routing and Monte Carlo expert selection. Top-$k$ routing selects the $k$ experts with the highest router-predicted probabilities, while Monte Carlo routing (MC-$k$) samples $k$ experts according to the router-predicted distribution. Another strategy for reducing inference cost is to use a smaller router. Figure 4 compares the performance and inference cost of different routing schemes and router sizes. The 250M expert models pre-trained for 100K




(a) Expert curves (k-means) (b) Expert curves (random) (c) Router curves

Figure 3: Visualization of training curves for the 250M experts, router, and monolithic model under both k-means clustering and random partitioning. All models are trained with the same batch size for 100K steps. The close-up views in (a) and (b) highlight the differences in training loss between the experts and the monolithic model near the end of training.

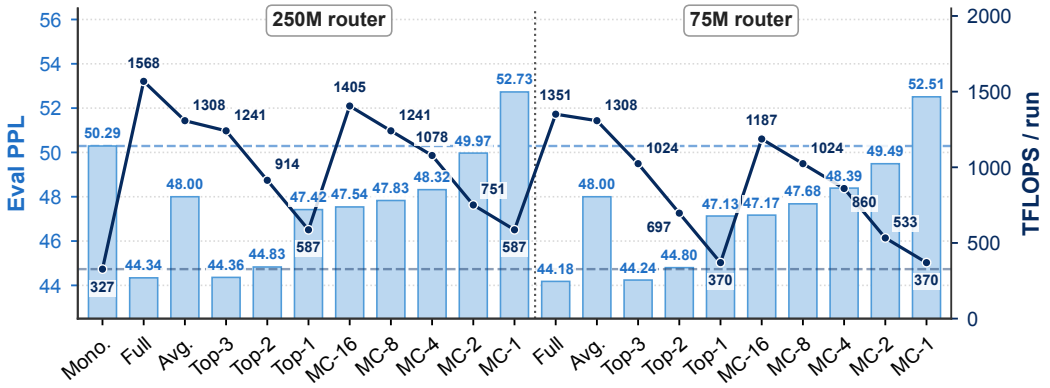



Figure 4: Ablation study of different routing schemes, including full (weighted combination of all experts' predictions), Top-$k$, Monte Carlo (MC-$k$, with sample size k), and average (simply average predictions of all experts). We test these schemes using 75M and 250M routers for 250M experts.

steps are used in this study. Full routing achieves the best evaluation PPL, but incurs more than $4\times$ inference FLOPs. Top-1 routing reduces FLOPs by approximately $60\%$, at the cost of degraded model performance. We also find that replacing the router with a smaller 75M model leads to only a minor performance drop while reducing inference cost. When Top-1 routing is combined with the 75M router, decentralized sampling still achieves competitive performance, outperforming the monolithic baseline with a comparable level of FLOPs.

## 5 Conclusion

We introduce Decentralized Diffusion Language Models (D-DLM), a framework that trains expert DLMs independently on disjoint data clusters and combines their predictions with a router at inference time. D-DLM builds on a decentralized form of discrete diffusion: the clean-token posterior, and hence the probability velocity, decomposes into a mixture of cluster-conditional posteriors weighted by the cluster posterior of the noisy sequence. Each expert therefore only needs to model its own cluster, and the router only needs to infer which cluster a noisy sequence belongs to. Although this decomposition holds for any data partition, the partition determines how well the experts and the router can be learned. We find that text k-means clustering, unlike random partitioning, allows each expert to be better optimized on its own cluster and makes the cluster membership learnable by the router. In our experiments, D-DLM achieves competitive or improved performance compared with monolithic baselines on multiple evaluation tasks while reducing the communication burden of multi-node optimization.

### AI USE STATEMENT

In this work, we used generative AI tools only to polish the writing, namely proofreading the text, fixing typos, and correcting grammatical errors. We have not used generative AI tools for research ideation, method development, experiment design, implementation, or result analysis. We have reviewed all AI-assisted edits and take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work proposes a training framework for diffusion language models and does not involve human subjects, personally identifiable information, or the release of new datasets. All experiments use publicly available datasets and open-weight models. As with other language models, models trained with D-DLM may reflect biases present in their training data. We are not aware of other ethical concerns specific to this work.

### REPRODUCIBILITY STATEMENT

For the theoretical results, Section 3.1 states the setting of Proposition 3.1, Appendix B provides its complete derivation, and Appendix C proves the time-independence of the cluster posterior. For the experiments, Section 4.1 describes the expert and router models and our data partitioning, Sections 4.2.1 and 4.2.2 specify the pre-training settings, and Appendix H describes the supervised fine-tuning setup. Appendix D details the text embedding and hierarchical k-means pipeline with its configurations, Appendix E describes the router for Block Diffusion, and Appendix F defines the routing schemes used at inference time. All datasets and base models are publicly available, and our implementation builds on the open-source dLLM codebase (Zhou et al., 2026).

## REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

Ruisi Cai, Yeonju Ro, Geon-Woo Kim, Peihao Wang, Babak Ehteshami Bejnordi, Aditya Akella, and Zhangyang Wang. Read-ME: Refactorizing LLMs as router-decoupled mixture of experts with system co-design. In *Advances in Neural Information Processing Systems*, 2024.

Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974. doi: 10.1080/03610927408827101.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.

Ziwei Chai, Guoyin Wang, Jing Su, Tianjie Zhang, Xuanwen Huang, Xuwu Wang, Jingjing Xu, Jianbo Yuan, Hongxia Yang, Fei Wu, et al. An expert is worth one token: Synergizing multiple expert llms as generalist via expert token routing. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11385–11396, 2024.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *International Conference on Machine Learning*, 2026a.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. DMax: Aggressive parallel decoding for dLLMs. *arXiv preprint arXiv:2604.08302*, 2026b.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. *arXiv preprint arXiv:2309.16588*, 2023.

David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979. doi: 10.1109/TPAMI.1979.4766909.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, et al. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. *arXiv preprint arXiv:2504.13161*, 2025.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Arthur Douillard, Keith Rush, Yani Donchev, Zachary Charles, Nova Fallen, Ayush Dubey, Ionel Gog, Josef Dean, Blake Woodworth, Zachary Garrett, et al. Decoupled diloco for resilient distributed pre-training. *arXiv preprint arXiv:2604.21428*, 2026.

Shangbin Feng, Weijia Shi, Yuyang Bai, Vidhisha Balachandran, Tianxing He, and Yulia Tsvetkov. Knowledge card: Filling LLMs' knowledge gaps with plug-in specialized language models. In *International Conference on Learning Representations*, 2024.

Tao Feng, Haozhen Zhang, Zijie Lei, Haodong Yue, Chongshan Lin, Ge Liu, and Jiaxuan You. Llmrouter: An open-source library for llm routing. `https://github.com/ulab-uiuc/LLMRouter`, 2025. GitHub repository.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37: 133345–133385, 2024.

Suchin Gururangan, Margaret Li, Mike Lewis, Weijia Shi, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Scaling expert language models with unsupervised domain discovery. *arXiv preprint arXiv:2303.14177*, 2023.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2022.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models, 2024. URL `https://arxiv.org/pdf/2411.04905`.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Zhiying Jiang, Raihan Seraj, Marcos Villagra, and Bidhan Roy. Paris: A decentralized trained open-weight diffusion model. *arXiv preprint arXiv:2510.03434*, 2025.

Zhiying Jiang, Raihan Seraj, Marcos Villagra, and Bidhan Roy. Heterogeneous decentralized diffusion models. *arXiv preprint arXiv:2603.06741*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

David McAllister, Matthew Tancik, Jiaming Song, and Angjoo Kanazawa. Decentralized diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 23323–23333, 2025.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Jishnu Mukhoti, Yarin Gal, Philip HS Torr, and Puneet K Dokania. Fine-tuning can cripple your foundation model; preserving features may be the solution. *arXiv preprint arXiv:2308.13320*, 2023.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data, 2024.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. doi: 10.1016/0377-0427(87)90125-7.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Yunfan Shao, Linyang Li, Zhaoye Fei, Hang Yan, Dahua Lin, and Xipeng Qiu. Balanced data sampling for language model training with clustering. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 14012–14023, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024a.

Weijia Shi, Sewon Min, Maria Lomeli, Chunting Zhou, Margaret Li, Gergely Szilvasy, Rich James, Xi Victoria Lin, Noah A. Smith, Luke Zettlemoyer, Scott Yih, and Mike Lewis. In-context pretraining: Language modeling beyond document boundaries. In *International Conference on Learning Representations*, 2024b.

Weijia Shi, Akshita Bhagia, Kevin Farhat, Niklas Muennighoff, Pete Walsh, Jacob Morrison, Dustin Schwenk, Shayne Longpre, Jake Poznanski, Allyson Ettinger, et al. FlexOlmo: Open language models for flexible data use. In *Advances in Neural Information Processing Systems*, 2025.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 34:1415–1428, 2021.

Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. Score-based continuous-time discrete diffusion models. *arXiv preprint arXiv:2211.16750*, 2022.

Dimitri Von Rütte, Janis Fluri, Omead Pooladzandi, Bernhard Schölkopf, Thomas Hofmann, and Antonio Orvieto. Scaling behavior of discrete diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 36638–36663, 2026.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Nuoya Xiong, Yuhang Zhou, Hanqing Zeng, Zhaorun Chen, Furong Huang, Shuchao Bi, Lizhu Zhang, and Zhuokai Zhao. Token-level llm collaboration via fusionroute. *arXiv preprint arXiv:2601.05106*, 2026.

Shuchen Xue, Tianyu Xie, Tianyang Hu, Zijin Feng, Jiacheng Sun, Kenji Kawaguchi, Zhenguo Li, and Zhi-Ming Ma. Any-order GPT as masked diffusion model: Decoupling formulation and architecture. In *International Conference on Machine Learning*, 2026.

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

Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Xiaoze Fan, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Kurt Keutzer, et al. Flash-kmeans: Fast and memory-efficient exact k-means. *arXiv preprint arXiv:2603.09229*, 2026.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

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

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, et al. Introspective diffusion language models. *arXiv preprint arXiv:2604.11035*, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Chuanyang Zheng, Jiankai Sun, Yihang Gao, Enze Xie, Yuehao Wang, Peihao Wang, Ting Xu, Matthew Chang, Liliang Ren, Jingyao Li, et al. Understanding the mixture-of-experts with nadaraya-watson kernel. *arXiv preprint arXiv:2509.25913*, 2025.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.

Zhanhui Zhou, Lingjie Chen, Hanghang Tong, and Dawn Song. dllm: Simple diffusion language modeling. *arXiv preprint arXiv:2602.22661*, 2026.

# A Related Work

**Diffusion models for language modeling.** D3PM (Austin et al., 2021) established a general CTMC-based formulation of discrete diffusion models, covering corruption processes such as mask absorbing (masked diffusion) and uniform token replacement (uniform-state diffusion). Subsequent work, including MD4 (Shi et al., 2024a) and MDLM (Sahoo et al., 2024), improved masked diffusion through refined parameterizations and noise schedules, while LLaDA (Nie et al., 2025; Bie et al., 2026) scaled masked diffusion language models and achieved autoregressive-like performance. Recent hybrid approaches further combine masked diffusion with autoregressive modeling, such as block diffusion with block-causal attention (Arriola et al., 2025) and I-DLM, which moves diffusion language modeling toward causal autoregressive generation through causal attention and introspective verification (Yu et al., 2026). Unlike these methods, which typically couple training-time attention-mask design with specialized inference procedures, our method is agnostic to the corruption process and attention pattern. Instead, it provides a scalable training infrastructure for a broad class of discrete diffusion language models, including both masked and uniform-state diffusion.

**Data clustering for large-scale model training.** Semantic clustering has been used to organize large-scale pre-training data across vision and language modeling: Decentralized Diffusion Models (McAllister et al., 2025) and subsequent extensions (Jiang et al., 2025; 2026) partition image data into clusters for independent diffusion experts, while Balanced Data Sampling (Shao et al., 2024) and Nemotron-CLIMB (Diao et al., 2025) use text clusters to rebalance or search training-data mixtures for a single language model, and Shi et al. (2024b) groups related documents into the same pre-training context. Unlike these single-model methods, and similar to Gururangan et al. (2023), which trains one autoregressive expert per k-means cluster, we use clusters to partition training data across independently trained experts, reusing the cluster id as router supervision; partitioning is done with Flash-KMeans (Yang et al., 2026), a fast exact GPU k-means.

**Decentralized training.** Large-scale language model training combines data parallelism with model parallelism (Shoeybi et al., 2019; Narayanan et al., 2021), which synchronizes devices at every step, so stragglers, failures, and limited bandwidth stall the global step (Douillard et al., 2026). Low-communication methods such as DiLoCo (Douillard et al., 2023) and Decoupled DiLoCo (Douillard et al., 2026) synchronize less often, but their learners still exchange parameters to train a single model. Full gradient isolation removes this coupling; previously explored for image diffusion (McAllister et al., 2025; Jiang et al., 2025; 2026), we establish it for diffusion language models, where experts and the associated routers are trained with no cross-model communication.

**Multi-capability mastery via expert routing.** Multi-capability mastery in a single model is constrained by catastrophic forgetting (Kirkpatrick et al., 2017; Mukhoti et al., 2023; Luo et al., 2025), manifesting in industrial LLMs as the "see-saw effect" documented by the MiMo-V2-Flash technical report (Xiao et al., 2026), where improving one skill regresses another. Such mastery is foundational to general intelligence and long-horizon self-evolution, motivating distribution across multiple specialists rather than packing into one. In autoregressive LLMs, Knowledge Card (Feng et al., 2024) augments a general model with plug-in specialized LMs; query-level routers (Ong et al., 2024; Feng et al., 2025) commit each prompt to a single specialist; finer-grained token-level routers (Xiong et al., 2026; Chai et al., 2024) switch experts per decoding step for compound tasks where different segments need different specialists. We bring this fine-grained per-step routing to discrete diffusion language models.

**Mixture-of-experts and decentralized expert architectures.** Most Transformer mixture-of-experts (MoE) architectures (Shazeer et al., 2017; Jiang et al., 2024; Yang et al., 2025; DeepSeek-AI, 2026) couple experts through a shared backbone and joint training. Read-ME (Cai et al., 2024) decouples the router from the MoE backbone to enable pre-computed routing, but its experts are still extracted from a single pre-trained dense model. Our framework departs from this paradigm on every architectural axis: there is no shared backbone, each expert is a full DLM rather than a sub-module

(and may itself adopt an MoE architecture), and the router is trained post hoc on cluster identities with no signal exchange between experts. As a byproduct, the no-data-sharing property pursued by decentralized MoE designs such as FlexOLMo (Shi et al., 2025) is naturally achievable here.

## B Derivations

We present a detailed version of derivations in Sec. 3.1.

$$
\begin{aligned}
p_{0|t}(\cdot|\boldsymbol{z}) &= \sum_{l=1}^{C} \sum_{\boldsymbol{x}_0 \in \mathcal{T}} \sum_{\boldsymbol{x}_1 \in \mathcal{S}} p_t(\boldsymbol{x}_0, \boldsymbol{x}_1, y = l|\boldsymbol{z}) \boldsymbol{x}_0 \\
&= \sum_{l=1}^{C} \sum_{\boldsymbol{x}_0 \in \mathcal{T}} \sum_{\boldsymbol{x}_1 \in \mathcal{S}} \frac{p_t(\boldsymbol{z}|\boldsymbol{x}_0, \boldsymbol{x}_1, y = l) p_0(\boldsymbol{x}_0) p(y = l|\boldsymbol{x}_0) p_1(\boldsymbol{x}_1)}{p_t(\boldsymbol{z})} \boldsymbol{x}_0 \\
&= \sum_{l=1}^{C} \sum_{\boldsymbol{x}_0 \in \mathcal{T}} \sum_{\boldsymbol{x}_1 \in \mathcal{S}} \frac{p_t(\boldsymbol{z}|\boldsymbol{x}_0, \boldsymbol{x}_1) p(\boldsymbol{x}_0|y = l) p(y = l) p_1(\boldsymbol{x}_1)}{p_t(\boldsymbol{z})} \boldsymbol{x}_0 \\
&= \sum_{l=1}^{C} \sum_{\boldsymbol{x}_0 \in \mathcal{T}} \sum_{\boldsymbol{x}_1 \in \mathcal{S}} \frac{p_t(\boldsymbol{z}|\boldsymbol{x}_0, \boldsymbol{x}_1) p(\boldsymbol{x}_0|y = l) p(y = l) p_1(\boldsymbol{x}_1)}{p_t(\boldsymbol{z}|y = l)} \frac{p_t(\boldsymbol{z}|y = l)}{p_t(\boldsymbol{z})} \boldsymbol{x}_0 \\
&= \sum_{l=1}^{C} \sum_{\boldsymbol{x}_0 \in \mathcal{T}} \sum_{\boldsymbol{x}_1 \in \mathcal{S}} \frac{p_t(\boldsymbol{z}|\boldsymbol{x}_0, \boldsymbol{x}_1) p(\boldsymbol{x}_0|y = l) p_1(\boldsymbol{x}_1)}{p_t(\boldsymbol{z}|y = l)} \frac{p_t(\boldsymbol{z}, y = l)}{p_t(\boldsymbol{z})} \boldsymbol{x}_0 \\
&= \sum_{l=1}^{C} \sum_{\boldsymbol{x}_0 \in \mathcal{T}} \sum_{\boldsymbol{x}_1 \in \mathcal{S}} \frac{p_t(\boldsymbol{z}|\boldsymbol{x}_0, \boldsymbol{x}_1) p(\boldsymbol{x}_0|y = l) p_1(\boldsymbol{x}_1)}{p_t(\boldsymbol{z}|y = l)} p_t(y = l|\boldsymbol{z}) \boldsymbol{x}_0 \\
&= \sum_{l=1}^{C} p_{0|t}(\cdot|\boldsymbol{z}, y = l) p_t(y = l|\boldsymbol{z}),
\end{aligned}
$$

where $\mathcal{T} = \mathrm{supp}(p_0)$ and $\mathcal{S} = \mathrm{supp}(p_1)$ denote the supports of the target and source distributions, respectively. The second equality follows from Bayes' rule. The third uses the conditional independence of $\boldsymbol{z}$ and $y$. In the fourth, we introduce $p_t(\boldsymbol{z} \mid y = l)$ in both the numerator and the denominator. The fifth simply regroups the terms, and the final equality applies Bayes' rule once more.

Moving forward to velocity field $u_t(\boldsymbol{z})$, we use the result above to show Equation 6.

$$
\begin{aligned}
u_t(\boldsymbol{z}) &= \frac{\alpha_t'}{1 - \alpha_t} \left[\boldsymbol{z} - p_{0|t}(\cdot|\boldsymbol{z})\right] \\
&= \frac{\alpha_t'}{1 - \alpha_t} \left[\boldsymbol{z} - \sum_{l=1}^{C} p_{0|t}(\cdot|\boldsymbol{z}, y = l) p_t(y = l|\boldsymbol{z})\right] \\
&= \sum_{l=1}^{C} \frac{\alpha_t'}{1 - \alpha_t} \left[\boldsymbol{z} - p_{0|t}(\cdot|\boldsymbol{z}, y = l)\right] p_t(y = l|\boldsymbol{z}).
\end{aligned}
$$

## C Time-Independence of the Cluster Posterior

**Proposition C.1.** *Suppose $p_1$ is the delta distribution on the fully masked sequence and $\boldsymbol{X}(t)$ follows Equation 5. Then for any $\boldsymbol{Z} \in \mathcal{E}^{N \times K}$ and $l \in [C]$, the cluster posterior $p_t(y = l|\boldsymbol{Z})$ is independent of $t$ over all $t \in [0, 1]$ with $p_t(\boldsymbol{Z}) > 0$.*

Let $\mathcal{T} = \mathrm{supp}(p_0)$, $\mathcal{U}(\boldsymbol{Z}) = \{i \in [N] | \boldsymbol{Z}^i \neq \boldsymbol{m}\}$ be the unmasked positions of $\boldsymbol{Z}$, and $\mathcal{A}(\boldsymbol{Z}) = \{\boldsymbol{X}_0 \in \mathcal{T} | \boldsymbol{X}_0^i = \boldsymbol{Z}^i, \forall i \in \mathcal{U}(\boldsymbol{Z})\}$. Since clean tokens are never `[mask]`, Equation 5 gives, for any $\boldsymbol{X}_0 \in \mathcal{T}$,

$$
p_t(\boldsymbol{Z}|\boldsymbol{X}_0) = \prod_{i \in \mathcal{U}(\boldsymbol{Z})} \alpha_t \mathbb{I}\left[\boldsymbol{Z}^i = \boldsymbol{X}_0^i\right] \prod_{i \notin \mathcal{U}(\boldsymbol{Z})} (1 - \alpha_t) = c_t(\boldsymbol{Z}) \mathbb{I}\left[\boldsymbol{X}_0 \in \mathcal{A}(\boldsymbol{Z})\right],
$$

where $c_t(\boldsymbol{Z}) = \prod_{i \in \mathcal{U}(\boldsymbol{Z})} \alpha_t \prod_{i \notin \mathcal{U}(\boldsymbol{Z})} (1 - \alpha_t)$ does not depend on $\boldsymbol{X}_0$. Moreover, $c_t(\boldsymbol{Z}) > 0$ since $p_t(\boldsymbol{Z}) = c_t(\boldsymbol{Z}) \sum_{\boldsymbol{X}_0 \in \mathcal{A}(\boldsymbol{Z})} p_0(\boldsymbol{X}_0) > 0$. Therefore,

$$
\begin{aligned}
p_t(y = l|\boldsymbol{Z}) &= \frac{\sum_{\boldsymbol{X}_0 \in \mathcal{T}} p_t(\boldsymbol{Z}|\boldsymbol{X}_0, y = l) p(y = l|\boldsymbol{X}_0) p_0(\boldsymbol{X}_0)}{\sum_{\boldsymbol{X}_0 \in \mathcal{T}} p_t(\boldsymbol{Z}|\boldsymbol{X}_0) p_0(\boldsymbol{X}_0)} \\
&= \frac{\sum_{\boldsymbol{X}_0 \in \mathcal{T}} p_t(\boldsymbol{Z}|\boldsymbol{X}_0) p(y = l|\boldsymbol{X}_0) p_0(\boldsymbol{X}_0)}{\sum_{\boldsymbol{X}_0 \in \mathcal{T}} p_t(\boldsymbol{Z}|\boldsymbol{X}_0) p_0(\boldsymbol{X}_0)} \\
&= \frac{\sum_{\boldsymbol{X}_0 \in \mathcal{A}(\boldsymbol{Z})} p(y = l|\boldsymbol{X}_0) p_0(\boldsymbol{X}_0)}{\sum_{\boldsymbol{X}_0 \in \mathcal{A}(\boldsymbol{Z})} p_0(\boldsymbol{X}_0)},
\end{aligned}
$$

where the first equality follows from Bayes' rule, the second uses the conditional independence of $\boldsymbol{Z}$ and $y$ given $\boldsymbol{X}_0$, and the third substitutes $p_t(\boldsymbol{Z}|\boldsymbol{X}_0)$ and cancels $c_t(\boldsymbol{Z})$. The final expression does not depend on $t$.

# D TEXT DATASET CLUSTERING

In this section, we describe our text embedding and hierarchical clustering pipeline used to construct data partitions. The pipeline consists of two stages: first, dense document embeddings are produced with a Qwen3-embedding model (Zhang et al., 2025); second, a hierarchical k-means model is trained over the resulting embedding corpus. Both stages are designed for large-scale document collections and are evaluated on H100 GPUs.

## D.1 TEXT EMBEDDING GENERATION

We generate document-level embeddings using Qwen3-embedding 8B (Zhang et al., 2025). Each input document is independently encoded, so the embedding stage is naturally parallel over input shards and does not require inter-device collective communication. In our large-scale setting, each GPU processes a disjoint subset of input documents with an independent model replica.

The embedding representation is obtained from the pooled final-token hidden state. We store the resulting vectors as 4096-dimensional embeddings. For processing a large pre-training corpus with a considerable throughput, we use a vLLM (Kwon et al., 2023) pooling backend. The embedding-generation stage uses Qwen3-embedding 8B with 4096-dimensional output embeddings. Model inference is performed in bfloat16 precision. The maximum sequence length is set to 8192 tokens, and the per-GPU text batch size is set to 32. For large-scale runs, we use the vLLM pooling backend with GPU memory utilization set to $85\%$.

To support embedding generation at pre-training data scale, we develop an efficient text embedding pipeline with particular efforts for data loading and model-inference throughput. We next describe the key system-level design choices that make large-scale embedding practical.

**Storage format and data loading.** The embedding vectors are stored using a fixed-size vector representation rather than a variable-length list representation. This choice is important for downstream clustering because the k-means stage must repeatedly read and construct large embedding data matrices. A fixed-size vector layout allows the embedding column to be loaded as a contiguous two-dimensional array, avoiding per-row Python object conversion and reducing I/O overhead. In empirical measurements, the fixed-size float32 representation is substantially faster than an older variable-list double-precision schema, which was observed to incur a 2–4$\times$ I/O slowdown.

**Embedder backend throughput.** In Table 4, we compare the Hugging Face Transformers (Wolf et al., 2019) and the vLLM pooling backend on 2048 SlimPajama-6B (Soboleva et al., 2023) documents using a single H100 GPU, with maximum sequence length 8192 and bfloat16 inference. The Hugging Face backend becomes slower as the caller batch size increases, mainly because sequence padding is determined by the longest sequence in each batch. In contrast, vLLM maintains stable throughput once the batch size is sufficiently large, benefiting from continuous batching and its scheduling mechanism. The vLLM backend reaches approximately 30 texts/s on a single H100 GPU. In contrast, the Hugging Face backend achieves its highest throughput at batch size 1, with approximately 14 texts/s, and runs out of memory at larger batch sizes.

Table 4: Single-GPU embedding throughput on H100 across different embedder batch sizes.

| Batch size | HF Transformers texts/s | vLLM texts/s | vLLM speedup |
|---|---|---|---|
| 1 | 14.08 | 24.61 | 1.75× |
| 4 | 7.94 | 28.28 | 3.56× |
| 8 | 5.43 | 28.80 | 5.30× |
| 16 | 3.72 | 30.07 | 8.08× |
| 32 | OOM | 30.29 | – |

Table 5: Hierarchical k-means configurations for FineWeb-Edu 20B and 80B clustering.

| Component | FineWeb-Edu 20B | FineWeb-Edu 80B |
|---|---|---|
| Embedding dimension | 64 | 64 |
| Distance metric | cosine | cosine |
| Vector normalization | $\ell_2$ normalization | $\ell_2$ normalization |
| Number of hierarchy levels | 3 | 3 |
| Centroids per layer, $C = 4$ | 80,000 $\rightarrow$ 800 $\rightarrow$ 4 | 400,000 $\rightarrow$ 4,000 $\rightarrow$ 4 |
| Centroids per layer, $C = 8$ | 80,000 $\rightarrow$ 800 $\rightarrow$ 8 | 400,000 $\rightarrow$ 4,000 $\rightarrow$ 8 |
| Sample size | 8,000,000 | 40,000,000 |
| Number of k-means iterations | 100 | 100 |
| Tolerance of kmeans | 1e-4 | 1e-4 |

### D.2 Hierarchical K-Means Clustering

After embedding generation, we train a hierarchical k-means model over the document embeddings. The hierarchy reduces the number of representatives across successive levels. At the first level, a large sample of document embeddings is clustered into many centroids. Each subsequent level clusters the centroids from the previous level, and the final level produces the desired number of leaf clusters.

The clustering stage uses cosine distance with $\ell_2$-normalized vectors. To reduce memory and compute cost, clustering is performed on the first 64 dimensions of the 4096-dimensional Qwen3 embeddings. This dimensional truncation is supported by the Matryoshka representation learning property (Kusupati et al., 2022) adopted in the Qwen3-embedding model. The text embedding clustering hierarchy uses three levels. For FineWeb-Edu 20B, we use $80{,}000 \rightarrow 800 \rightarrow C$, where $C$ is the final number of leaf clusters. For FineWeb-Edu 80B, we use $400{,}000 \rightarrow 4{,}000 \rightarrow C$. This keeps the layer-wise reduction ratio approximately constant across corpus sizes. We choose Flash-Kmeans (Yang et al., 2026), a Triton-accelerated k-means implementation, as our clustering backend. Table 5 presents the k-means configurations for text clustering in our experiments on the FineWeb-Edu dataset.

**Compute scaling and memory usage.** We evaluate k-means compute scaling using pseudo-random 64-dimensional vectors on a single H100 GPU. Table 6 shows the k-means runtime and memory profiling across different numbers of input vectors. The wall-clock time scales approximately linearly with the number of input vectors once the number of centroids and embedding dimension are fixed, consistent with the dominant distance-computation cost. The clustering stage is not memory-bound on a 96 GB H100. The configuration with 40M samples, 400K centroids, and 64 dimensions peaks at approximately 12 GB of GPU memory, leaving substantial headroom. The dominant cost is therefore compute and autotuning time rather than device memory.

**Clustering evaluation.** In Table 7, we evaluate clustering quality using three standard internal validity indices that do not require ground-truth labels. The silhouette coefficient (Rousseeuw, 1987) measures how much closer each sample is to its assigned cluster than to the nearest alternative cluster, with larger values indicating better separation. The Davies–Bouldin index (Davies & Bouldin, 1979) averages the worst-case similarity between each cluster and another cluster, where lower values indicate more compact and better separated clusters. The Calinski–Harabasz index (Caliński & Harabasz, 1974) compares between-cluster dispersion with within-cluster dispersion, with larger

Table 6: K-means compute scaling on a single H100 GPU.

| Samples | Centroids | Wall-clock | Peak GPU memory | K-means sec. / iter. |
|---|---|---|---|---|
| 5M | 400,000 | 910 s | 1.92 GB | ≈1.1 s |
| 10M | 400,000 | 1817 s | 3.32 GB | ≈2.2 s |
| 20M | 400,000 | 3636 s | 6.12 GB | ≈4.4 s |
| 40M | 400,000 | 7279 s | 11.96 GB | ≈8.8 s |
| 60M | 400,000 | 10,922 s | 17.88 GB | ≈13.2 s |
| 80M | 400,000 | 14,585 s | 23.80 GB | ≈17.6 s |

Table 7: Clustering quality on FineWeb-Edu 20B and FineWeb-Edu 80B.

| Dataset | $C$ | Silhouette ↑ | Davies–Bouldin ↓ | Calinski–Harabasz ↑ | Mean dist. ↓ | Dist. std. ↓ |
|---|---|---|---|---|---|---|
| FineWeb-Edu 20B | 4 | 0.033 | 4.361 | $1.54 \times 10^4$ | 1.037 | 0.012 |
| FineWeb-Edu 20B | 8 | 0.018 | 4.516 | $9.38 \times 10^3$ | 1.006 | 0.031 |
| FineWeb-Edu 80B | 4 | 0.034 | 4.300 | $1.59 \times 10^4$ | 1.034 | 0.015 |
| FineWeb-Edu 80B | 8 | 0.025 | 4.429 | $1.02 \times 10^4$ | 1.002 | 0.017 |

values indicating stronger cluster separation. We also report the mean cosine distance to the assigned centroid as a direct measure of local compactness in Table 8.

## E Router Design of Decentralized Block Diffusion

In this section, we describe how we implement router models for Block Diffusion. We use the same transformer architecture as Block Diffusion with a routing head, predicting the logits for different experts. To realize inference-time routing based on decoded clean tokens and the current denoising block, we slightly modify the training attention maps to align the training scheme with the attention for inference-time decoding.

**Training-time routing tokens.** Block Diffusion (Arriola et al., 2025) splits a sequence $\boldsymbol{X} \in \mathcal{E}^{N \times K}$ into $M = N/B$ blocks of $B$ tokens. We write $\mathcal{I}_b = \{(b-1)B+1, \ldots, bB\}$ for the token positions of block $b \in [M]$ and $\boldsymbol{X}^{\mathcal{I}_b}$ for its tokens. To learn a block-wise routing policy on top of Block Diffusion, we augment the noised half of the input with one special token $\mathtt{c} = \texttt{<|expert_cls|>}$ per block while leaving the clean half unchanged. For each noised block, we prepend one such token. The resulting training input has layout $[\,\mathtt{c}, \boldsymbol{X}(t)^{\mathcal{I}_1}, \mathtt{c}, \boldsymbol{X}(t)^{\mathcal{I}_2}, \ldots, \mathtt{c}, \boldsymbol{X}(t)^{\mathcal{I}_M} \parallel \boldsymbol{X}_0^{\mathcal{I}_1}, \boldsymbol{X}_0^{\mathcal{I}_2}, \ldots, \boldsymbol{X}_0^{\mathcal{I}_M}\,]$, where $\boldsymbol{X}(t)^{\mathcal{I}_b}$ and $\boldsymbol{X}_0^{\mathcal{I}_b}$ denote the noisy and clean versions of block $b$, respectively. Since each noised block contains $B$ regular tokens plus one routing token, the full input length becomes $M(2B+1)$. The routing token for block $b$ is assigned the same positional index as the first regular token of that block, namely $(b-1)B+1$. Figure 5 (left) presents the attention map for router training. The block-wise attention mask extends naturally to the routing-token setting. The routing token of each block is placed in the noised half and inherits the attention pattern of its corresponding noised block. It can attend to the regular tokens in its own noised block through the block-diagonal attention component (in blue) and to all preceding clean blocks through the offset block-causal component (in orange and pink). Importantly, routing tokens are isolated from one another: the routing token of a block neither attends to nor is attended by the routing token of any other block. Clean tokens also do not attend to routing tokens.

**Decoding with block-wise routing.** At inference time, text generation via decentralized Block Diffusion proceeds block by block. The decoding procedure preserves the conditioning pattern used to train the routing token. We maintain $C$ expert KV caches and one router KV cache, each corresponding to the committed prefix, including the prompt and previously generated blocks. When generation enters a new block, the committed prefix is processed once by each expert and once by the router using the standard block-causal attention mask. The resulting prefix KV caches are then fixed for the current block denoising. During the decoding for the active block, only the active block tokens are re-evaluated. Each expert processes the current noised block conditioned on its cached prefix. In parallel, the router processes the short sequence $[\,\mathtt{c}, \boldsymbol{X}(t)^{\mathcal{I}_b}\,]$, where $b$ is the current block

Table 8: Per-cluster mean cosine distance for the evaluated hierarchical k-means configurations.

| $C = 4$ | | $C = 4$ | |
|---|---|---|---|
| **FineWeb-Edu 20B** | | **FineWeb-Edu 80B** | |
| **Cluster** | **Mean dist.** | **Cluster** | **Mean dist.** |
| 0 | 1.058 | 0 | 1.060 |
| 1 | 1.030 | 1 | 1.034 |
| 2 | 1.037 | 2 | 1.020 |
| 3 | 1.025 | 3 | 1.024 |
| $C = 8$ | | $C = 8$ | |
| **FineWeb-Edu 20B** | | **FineWeb-Edu 80B** | |
| **Cluster** | **Mean dist.** | **Cluster** | **Mean dist.** |
| 0 | 1.032 | 0 | 1.006 |
| 1 | 0.979 | 1 | 1.011 |
| 2 | 1.023 | 2 | 0.999 |
| 3 | 1.037 | 3 | 1.014 |
| 4 | 1.006 | 4 | 0.997 |
| 5 | 1.017 | 5 | 0.999 |
| 6 | 1.017 | 6 | 1.029 |
| 7 | 0.937 | 7 | 0.965 |

and $\boldsymbol{X}(t)^{\mathcal{I}_b}$ is still partially masked. Figure 5 (right) illustrates the step-by-step decoding process for a router and experts.

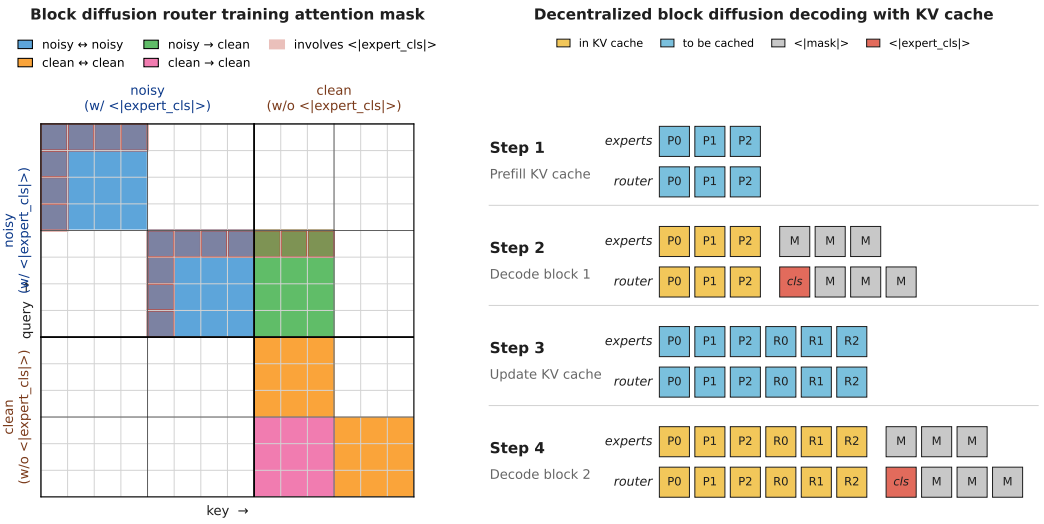



Figure 5: Illustration of the proposed router design for block-diffusion language modeling. Left: the training-time attention mask for the router model, where one routing token is inserted for each noised block while preserving the block-aware attention structure of Block Diffusion. Right: the decoding procedure for decentralized Block Diffusion.

## F Inference-Time Routing Schemes

Table 9 details the routing schemes compared in Figure 4. At each sampling step, let $\boldsymbol{g} = g_\phi(\boldsymbol{X}(t)) \in \Delta^C$ denote the router output and $\boldsymbol{f}^{(l)} = f_\theta^{(l)}(\boldsymbol{X}(t))$ the prediction of expert $l$. Each scheme forms the decoding distribution $\hat{p}_0(\cdot|\boldsymbol{X}(t))$ used by the sampler. Full routing is the ensemble form in Section 3.2, and Monte Carlo routing is an unbiased estimate of it. Since Monte Carlo routing draws experts with replacement, $k$ may exceed $C$.

Table 9: Routing schemes compared in Figure 4.

| Scheme | Experts queried | Decoding distribution $\hat{p}_0(\cdot\|\boldsymbol{X}(t))$ |
|---|---|---|
| Full | all $C$ experts | $\sum_{l=1}^{C} g_l \boldsymbol{f}^{(l)}$ |
| Average | all $C$ experts | $\sum_{l=1}^{C} \boldsymbol{f}^{(l)}/C$ |
| Top-$k$ | $\mathcal{A}$: the $k$ experts with the largest $g_l$ | $\sum_{l\in\mathcal{A}} g_l \boldsymbol{f}^{(l)} / \sum_{l\in\mathcal{A}} g_l$ |
| MC-$k$ | $l_1, \ldots, l_k$ drawn i.i.d. from $\boldsymbol{g}$ | $\sum_{j=1}^{k} \boldsymbol{f}^{(l_j)}/k$ |

## G Router Entropy decreases over unmasking process

Figure 6 shows the entropy change of the router output $g_\phi(\boldsymbol{X}(t))$ as tokens are unmasked through the denoising process. A heavily masked sequence reveals little about which cluster it belongs to, so the router is uncertain, and with all tokens masked its output is nearly uniform. As more tokens are unmasked, the cluster membership of the sequence becomes clearer and the router becomes more confident: its entropy drops sharply within the first 5% of unmasked tokens and keeps decreasing.

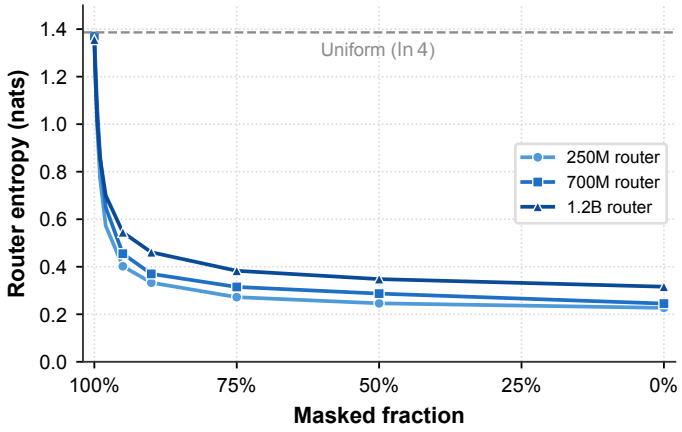



Figure 6: Mean router entropy on WikiText-2 versus the masked fraction. The dashed line marks the uniform distribution over the four experts ($\ln 4$).

## H Supervised Fine-tuning Evaluation

A compelling application scenario is supervised fine-tuning of DLMs from a pretrained language model for downstream tasks, such as math and coding. This setting raises a natural practical question for D-DLM: whether fine-tuning pretrained models on different SFT data subsets to obtain specialized experts, and aggregating their predictions at inference time, can improve downstream performance. To evaluate D-DLM in this setting, we follow the setup of Tiny-A2D (Zhou et al., 2026), which converts an AR model into a Block Diffusion model (Arriola et al., 2025) through post-training. We use Qwen3-0.6B (Yang et al., 2025) as the pretrained base model and fine-tune it on Tulu3-SFT (Lambert et al., 2024), opc-sft stage 1&2 (Huang et al., 2024), and SmolTalk (Allal et al., 2025). Similar to our pre-training setup, we cluster the SFT data via our k-means pipeline into two disjoint clusters. In addition to k-means clustering, we consider a domain-based partitioning strategy for the SFT setting. This is a natural choice when the SFT corpus contains subsets with clear domain annotations. For example, Tulu 3 Persona MATH and GSM are designed for mathematical reasoning, while the opc-sft series targets coding tasks. In our experiments, we construct two domains, math and coding, and curate the corresponding SFT corpora by mixing in general instruction-following data while ensuring that the two domain-specific subsets remain disjoint. The monolithic baseline and the two types of D-DLMs are both fine-tuned for 4K iterations using a single $8\times$ H100 node.

**Results.** Table 10 compares the SFT results of the two D-DLM variants with the monolithic baseline. Overall, D-DLM with k-means achieves the best performance. Compared with domain-based experts,

Table 10: SFT evaluation results of D-DLM. We compare the monolithic baseline with D-DLM using domain-based and k-means partitioning. The best non-expert-biased result for each task is bolded.

| | Model | GSM8K | BBH | MMLU | Hella. | HumanEval | MBPP | MMLU Pro | MATH |
|---|---|---|---|---|---|---|---|---|---|
| *Baseline* | Monolithic | 32.15 | 26.20 | 40.61 | 37.03 | 30.49 | **25.8** | 13.28 | 9.96 |
| *via domain* | math expert | 49.28 | 23.36 | 39.75 | 37.56 | 19.51 | 19.60 | 10.50 | 11.740 |
| | code expert | 32.83 | 19.67 | 39.36 | 37.56 | 25.00 | 19.80 | 12.40 | 10.98 |
| | D-DLM | 43.59 | 23.14 | 40.82 | 38.44 | 25.61 | 20.6 | 12.01 | **11.58** |
| *via kmeans* | kmeans expert 0 | 45.94 | 26.69 | 39.15 | 32.65 | 31.10 | 25.20 | 12.08 | 12.04 |
| | kmeans expert 1 | 44.05 | 27.77 | 41.28 | 37.48 | 31.10 | 25.2 | 14.28 | 10.14 |
| | D-DLM | **44.96** | **28.18** | **41.69** | **37.42** | **34.76** | **25.8** | **13.77** | 10.6 |

k-means experts are less biased to specific task domains, whereas domain experts tend to perform better on tasks aligned with their own fine-tuning domain than on the complementary domain. We also observe that expert aggregation can yield a performance gain beyond a simple interpolation of individual expert performance, indicating that the router can boost the predictions by combining experts effectively.

# I LIMITATION

Our approach requires querying multiple expert models, which introduces additional serving challenges. First, although running the router and experts in parallel keeps the latency close to that of a monolithic model (Figure 2b), it requires $C + 1$ devices, and full routing costs more than $C\times$ the inference compute of a monolithic model, which Top-1 routing with a smaller router reduces to about $1.1\times$ at some cost in perplexity (Section 4.3). Second, the total storage cost scales with the number of experts: a D-DLM with $C$ experts requires approximately $C\times$ the storage of a monolithic model of the same size. Third, the current decentralized sampling procedure is not directly aligned with existing LLM serving systems, such as SGLang and vLLM, which are primarily designed for single-model or standard batched inference.

Our experiments are also limited by available compute resources. In this work, we train models up to the 1.2B scale, as larger-scale training is beyond our current compute budget. As a result, the scalability of D-DLM and its behavior under larger model and data scales remain open questions.

Overall, D-DLM is still an exploratory framework. While our results provide initial evidence for the feasibility of decentralized DLM training, additional work is needed to improve serving efficiency, reduce storage overhead, and validate the approach at larger scales before it can be considered for practical deployment.