Empirical Analysis of Layer Redundancy in Diffusion Language Models

Yuto Karashima [1] **Hiroaki Ito** [*1] **Hikari Otsuka** [1] **Guanxi Lu** [2] **Tatsuya Kaneko** [1] **Masato Motomura** [1] **Daichi Fujiki** [1]

Abstract

Diffusion language models (DLMs) are promising alternatives to autoregressive models due to their bidirectional attention and parallel decoding. However, their inference cost becomes significantly higher as they scale. To address this challenge, we propose a dynamic layer-skipping framework. Our approach places a lightweight router before each Transformer layer to make unified, sequence-level execution decisions by aggregating masked token representations. Evaluated on LLaDA-8B across six benchmarks, we achieve a better FLOPs-accuracy trade-off than static and random baselines, including a 14.26% FLOPs reduction on PIQA without accuracy loss. Furthermore, analysis reveals that initial layers are consistently critical, and layer redundancy naturally increases as the denoising process progresses.

1. Introduction

Diffusion language models (DLMs) (Austin et al., 2021a; Shi et al., 2024; Nie et al., 2025) have emerged as promising alternatives to autoregressive models (ARMs). While ARMs generate tokens from left to right, DLMs generate text through an iterative denoising process over a masked sequence. This characteristic naturally enables bidirectional attention and parallel decoding, which not only mitigates inherent ARM limitations such as the reversal curse (Berglund et al., 2023), but also offers the potential for significantly faster text generation (Nie et al., 2025). Motivated by these architectural advantages, recent research has successfully scaled DLMs up to 8B parameters. Notably, models such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) demonstrate accuracy comparable to similarly sized ARMs.

Scaling up these models introduces a common challenge across LLMs: high computational cost during inference. Since the dense computations within Transformer layers dominate this burden, we explore the potential of layer skipping in DLMs. This technique exploits inherent layer-depth redundancy by selectively omitting the execution of specific layers, thereby reducing the architectural complexity and overall computational cost. In particular, we focus on dynamic layer skipping. Operating on the principle that tasks or tokens of varying difficulty do not require identical computing resources, this approach adaptively bypasses redundant computations based on the specific input.

While dynamic layer skipping has achieved remarkable success in ARMs (Fan et al., 2024; Raposo et al., 2024; Jiang et al., 2024), its application to DLMs remains largely unexplored, with existing research limited to a static approach (Goel et al., 2026). Directly adapting existing ARM-based dynamic mechanisms to DLMs is non-trivial due to a fundamental architectural difference. In ARMs, layer execution and routing decisions apply only to the last generated token during decoding. In contrast, DLMs process a fixed-length sequence of tokens simultaneously at each denoising step. This parallel paradigm prevents the application of token-level routing, necessitating a shift to sequence-level decisions.

To bridge this gap, we propose a dynamic layer-skipping framework for DLMs. Specifically, we place a lightweight router before each Transformer layer. Instead of per-token routing, the router aggregates masked token representations to make a unified, sequence-level skip decision. To validate our method, we use LLaDA-Base-8B (Nie et al., 2025) and conduct comprehensive experiments across six benchmark datasets. Our empirical analysis reveals a favorable trade-off between computational cost (FLOPs) and accuracy, while we also observe that selectively skipping certain layers often maintains or even improves accuracy. Our main contributions are summarized as follows:

- **Dynamic Layer-Skipping Framework for DLMs:** We propose the first dynamic layer-skipping framework for DLMs, using a lightweight router that aggregates masked token representations for sequence-level skip decisions.

[*]Equal contribution [1]Institute of Science Tokyo, Japan [2]Imperial College London, UK. Correspondence to: Yuto Karashima <karashima.yuto@artic.iir.isct.ac.jp>.

Proceedings of the ICML 2026 Workshop on AdaptFM: Resource-Adaptive Foundation Model Inference, Seoul, South Korea. 2026.

- **Better FLOPs-Accuracy Trade-offs:** Through evaluations, we demonstrate that we generally achieve a better trade-off between computational cost and accuracy compared to static and random layer-skipping methods. Notably, on specific tasks such as PIQA (Bisk et al., 2020), we reduce FLOPs by up to 14.26% while maintaining the original model's accuracy.
- **In-depth Analysis of Dynamic Routing Behavior:** We provide a detailed analysis of how our framework operates within the iterative denoising process. Our investigation reveals that layer execution patterns are highly task-dependent and that initial layers generally exhibit higher importance. Furthermore, we demonstrate the effectiveness of our training objective, which enables the router to adapt to the varying difficulty of each denoising step.

2. Related Work

2.1. Diffusion Language Models

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020) have achieved remarkable success in continuous data generation, including images and video (Rombach et al., 2022; Peebles & Xie, 2023; Ho et al., 2022). In natural language processing (NLP), masked diffusion models (MDMs) (Austin et al., 2021a; Shi et al., 2024; Ou et al., 2024) have emerged as a promising extension of this paradigm. They formulate text generation as an iterative denoising process over an initially masked token sequence. Recent efforts have successfully scaled MDMs up to 8B parameters. Models such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) demonstrate that DLMs can achieve generation quality comparable to that of ARMs (Grattafiori et al., 2024; Yang et al., 2025) of the same size.

On the other hand, scaling to billions of parameters inevitably introduces significant computational and memory overhead during inference. Prior work has explored various efficient inference strategies. These include accelerating inference via caching strategies (Ma et al., 2025; Liu et al., 2025; Wu et al., 2025; Hu et al., 2025) and step reduction techniques (Wu et al., 2025; Wei et al., 2025; Hu et al., 2025; Israel et al., 2025), as well as enhancing generation quality (Lu et al., 2025; Wang et al., 2025). In contrast, we take a fundamentally different approach: we propose a dynamic layer-skipping framework for DLMs to directly reduce the computational cost by exploiting architectural redundancy.

2.2. Layer Skipping

Layer skipping is an effective technique for reducing computational cost in LLMs by identifying and omitting redundant layers, broadly falling into static (Fan et al., 2019; Yang et al., 2024; Anwar et al., 2017) and dynamic approaches (Schuster et al., 2022; Xin et al., 2020; Kim & Cho, 2021). Static layer skipping permanently removes layers based on predefined metrics. While this simplicity effectively reduces computational cost without routing overhead, high skipping ratios lead to severe accuracy degradation. In contrast, dynamic layer skipping adaptively skips layers based on specific inputs during inference. Early exiting is one such technique (Bolukbasi et al., 2017; Huang et al., 2017; Teerapittayanon et al., 2016), which halts computation entirely at a certain depth for simpler inputs. For NLP, this adaptivity aligns well with the inherent variation in contextual complexity. Consequently, dynamic layer skipping has been widely explored in ARMs (Fan et al., 2024; Raposo et al., 2024; Jiang et al., 2024) , demonstrating remarkable superiority over static approaches (Kim et al., 2024; Gromov et al., 2024) in reducing computational costs.

Despite its success in ARMs, dynamic layer skipping remains largely unexplored in DLMs, where research has so far been limited to a static method (Goel et al., 2026). As we detail in Section 3, directly applying ARM-based routing to DLMs is non-trivial due to their distinct generation paradigms, motivating our sequence-level framework.

3. Method

We first highlight the fundamental challenge of adapting dynamic layer skipping from ARMs to DLMs. In standard ARMs, token generation proceeds sequentially. Because tokens are appended and processed independently at the last position during decoding, existing dynamic methods (Fan et al., 2024; Raposo et al., 2024; Jiang et al., 2024) can easily apply fine-grained, token-level routing. In contrast, DLMs update the entire token sequence simultaneously at each denoising step. Since their Transformer layers compute dense self-attention over this full sequence, executing a layer on only a subset of tokens creates a severe structural mismatch. Consequently, routing decisions in DLMs are more naturally unified at the sequence level. Guided by this constraint, we design a framework that executes or skips a layer for the entire sequence based on the aggregated hidden states of the masked tokens.

3.1. Preliminaries

Diffusion language models (DLMs) formulate text generation as a discrete denoising process. In this framework, "noise" is represented by a special mask token, denoted as $[\mathrm{MASK}]$. Generation proceeds over discrete steps $t = T, T-1, \ldots, 0$, starting from a fully masked target sequence and progressively refining it into a clean output. Using LLaDA (Nie et al., 2025) as an example, we formalize the inference process below.

Let $\mathcal{V}$ be the vocabulary, which includes $[\mathrm{MASK}]$. Given

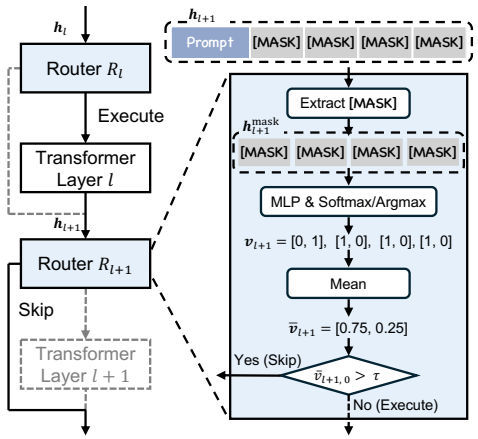

Figure 1. Overview of our proposed framework. To reduce the computational cost, the router dynamically decides whether to skip the subsequent Transformer layer by evaluating the hidden states of the currently masked tokens.

a prompt $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and a user-defined target output length m, the initial state $\boldsymbol{x}^{(T)}$ is constructed by appending m mask tokens to the prompt. Thus, the initial sequence of total length $L = n + m$ is defined as:

$$\boldsymbol{x}^{(T)} = (x_1, \ldots, x_n, \underbrace{[\text{MASK}], \ldots, [\text{MASK}]}_{m \text{ tokens}}). \quad (1)$$

During inference, the model progressively denoises this sequence. Let $\boldsymbol{x}^{(t)} \in \mathcal{V}^L$ represent the intermediate sequence at step t. At each denoising step t, a Transformer-based token predictor f_θ, processes the full sequence and outputs token distribution $\boldsymbol{p}^{(t)} \in \mathbb{R}^{L \times |\mathcal{V}|}$:

$$\boldsymbol{p}^{(t)} = f_\theta(\boldsymbol{x}^{(t)}). \quad (2)$$

For each masked position i (where $x_i^{(t)} = [\text{MASK}]$), the most likely token $\hat{x}_i^{(t)}$ is predicted via greedy decoding:

$$\hat{x}_i^{(t)} = \arg\max(\boldsymbol{p}_i^{(t)}). \quad (3)$$

Finally, a transition function S selectively updates a subset of masked positions in $\boldsymbol{x}^{(t)}$ with the predicted sequence $\hat{\boldsymbol{x}}^{(t)}$ (e.g., based on prediction confidence) to form the next state $\boldsymbol{x}^{(t-1)}$:

$$\boldsymbol{x}^{(t-1)} = S(\hat{\boldsymbol{x}}^{(t)}, \boldsymbol{x}^{(t)}). \quad (4)$$

This iterative process continues until the final generated sequence $\boldsymbol{x}^{(0)}$ contains no mask tokens.

3.2. Our proposed framework

Based on the structural constraints discussed above, we place a lightweight router before each Transformer layer. The overall procedure for dynamic routing during inference is summarized in Algorithm 1 in the Appendix. As input to the router, we use only the hidden states of the currently masked tokens. Since these masked tokens are the active targets of generation, this approach naturally aligns with token-level routing in ARMs. By aggregating these masked token states, the router makes a unified decision on whether to execute or skip the subsequent layer for the entire sequence.

To formalize this routing process, let $\boldsymbol{h}_l \in \mathbb{R}^{L \times d}$ be the sequence of continuous hidden states input to the l-th Transformer layer, where d is the hidden dimension. Among these hidden states, let $\boldsymbol{h}_l^{\text{mask}} \in \mathbb{R}^{M_t \times d}$ be the subset of hidden states corresponding to the currently masked tokens, where M_t denotes the number of masked tokens remaining at time step t.

As illustrated in Figure 1, the router R_l consists of three main components: a two-layer MLP (with RMSNorm and SiLU and a hidden dimension of 512), a token-level decision function, and a sequence-level voting gate. As we analyze later in Section 4.6, this MLP is highly lightweight and introduces negligible computational overhead. Specifically, for each token representation in $\boldsymbol{h}_l^{\text{mask}}$, the MLP outputs two-dimensional logits. To render the routing operation differentiable during training while obtaining strict binary decisions, we apply the Gumbel-Softmax reparameterization trick (Jang et al., 2016) to these logits. During inference, this step simplifies to a straightforward argmax operation. Both approaches convert the logits into a discrete token-level one-hot routing vote, $\boldsymbol{v}_l^{(i)} = [v_{l,0}^{(i)}, v_{l,1}^{(i)}]$ for the i-th masked token, where $v_{l,0}^{(i)} = 1$ denotes skip and $v_{l,1}^{(i)} = 1$ denotes execute.

Next, the router aggregates these token-level votes via the mean operation to determine a single, unified sequence-level execution decision. Specifically, we compute the average skip proportion $\bar{v}_{l,0}$ over all M_t masked tokens as follows:

$$\bar{v}_{l,0} = \frac{1}{M_t} \sum_{i=1}^{M_t} v_{l,0}^{(i)}. \quad (5)$$

The final binary gate vector $\boldsymbol{e}_l = [e_{l,0}, e_{l,1}]$ for the entire layer is then determined by comparing this skip proportion with a user-defined threshold hyperparameter τ:

$$e_{l,0} = \mathbb{1}(\bar{v}_{l,0} > \tau), \quad e_{l,1} = 1 - e_{l,0}, \quad (6)$$

where $\mathbb{1}(\cdot)$ is the indicator function. Because this indicator function is non-differentiable, we employ the straight-through estimator (Bengio et al., 2013) during training. This technique allows the model to use the discrete gate values in the forward pass while bypassing the non-differentiable step, routing gradients directly to the continuous proportion $\bar{v}_{l,0}$ during the backward pass. Consequently, the layer execution

gate $\boldsymbol{e}_l$ takes a discrete state of either $[1, 0]$ (skip) or $[0, 1]$ (execute) while remaining fully differentiable end-to-end.

Finally, the hidden states output from the l-th layer, denoted as $\boldsymbol{h}_{l+1}$, are computed based on the discrete gate values:

$$\boldsymbol{h}_{l+1} = \begin{cases} \boldsymbol{h}_l & \text{if } e_{l,0} = 1, e_{l,1} = 0 \quad \text{(skip)} \\ f_l(\boldsymbol{h}_l) & \text{if } e_{l,0} = 0, e_{l,1} = 1 \quad \text{(execute)}, \end{cases} \tag{7}$$

where $f_l(\cdot)$ represents the transformation applied by the l-th Transformer layer.

3.3. Optimization Objective for Dynamic Routing

During training, we freeze the parameters of the backbone model and update only the newly introduced routers. To ensure stable optimization, the weights of these routers are initialized such that training begins with most of the layers active. To reduce computational cost while maintaining accuracy, we optimize these routers to jointly minimize both the cross-entropy loss and the layer-skipping loss. We define the skip loss as follows:

$$\mathcal{L}_{\text{skip}} = |\alpha - \alpha_{\text{target}}|, \tag{8}$$

where α_{target} is a predefined target skip rate, and α is the actual layer skip rate. Here, $|\cdot|$ denotes the absolute value. Assuming the model has a total of N Transformer layers, α is computed as the average of the skip proportions across all layers:

$$\alpha = \frac{1}{N} \sum_{l=1}^{N} e_{l,0}. \tag{9}$$

During training, the input prompt remains unmasked, and we apply masking exclusively to the target output sequence. Specifically, we randomly replace a subset of target tokens with the mask token according to a randomly sampled mask ratio p_{mask}. This randomly set ratio effectively simulates the varying proportions of masked tokens encountered across different denoising steps during inference, rendering the sampling of a denoising step t unnecessary during training. The masked sequence is then fed into the model, where each Transformer layer is dynamically executed or bypassed based on the routers' decisions.

Following the fine-tuning guidelines of LLaDA (Nie et al., 2025), the cross-entropy loss $\mathcal{L}_{\text{CE}}$ for the masked tokens is scaled by the inverse of the mask ratio, $1/p_{\text{mask}}$. Similarly, we introduce an extension by applying the exact same scaling factor to our layer-skipping loss $\mathcal{L}_{\text{skip}}$. The overall objective function $\mathcal{L}$ is thus formulated as:

$$\mathcal{L} = \frac{1}{p_{\text{mask}}} \left(\mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{skip}} \right), \tag{10}$$

where λ is a hyperparameter that controls the trade-off between computational cost and accuracy preservation. This shared scaling mechanism for the skip loss is motivated by the intuition that simpler inputs require fewer layers to process. Smaller sampled values of p_{mask} correspond to the later stages of denoising, where the generative task inherently becomes easier. Consequently, the amplified $1/p_{\text{mask}}$ term imposes a stronger penalty on computation, encouraging the model to skip more layers during these easier stages.

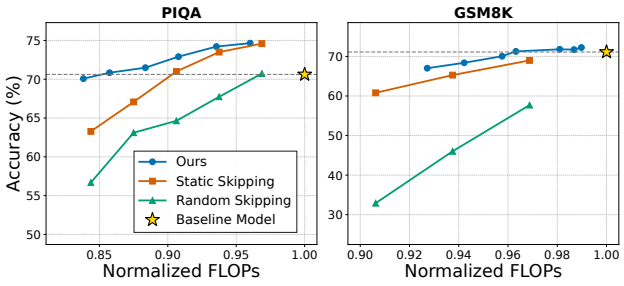

Figure 2. Accuracy vs. normalized FLOPs for PIQA and GSM8K.

4. Experiments

4.1. Setup

We evaluate our framework on LLaDA-Base-8B (Nie et al., 2025), which consists of 32 Transformer layers. For evaluation, we use six diverse datasets: MMLU (Hendrycks et al., 2020), ARC-C (Clark et al., 2018), Hellaswag (Zellers et al., 2019) and PIQA (Bisk et al., 2020) for general knowledge tasks, GSM8K (Cobbe et al., 2021) for mathematical reasoning, and MBPP (Austin et al., 2021b) for code generation. We train a separate router for each dataset.

During the training of the routers, we set the maximum context length to 1024 and the hidden dimension of the two linear layers to 512. The routers are trained for 10 epochs on PIQA, Hellaswag, and GSM8K, for 20 epochs on MBPP, for 30 epochs on MMLU, and for 40 epochs on ARC-C. To investigate the trade-off between computational cost and accuracy, we fix the target rate α_{target} to 0.5 in Eq. (8) and experiment with multiple values of the hyperparameter λ for each dataset. Additionally, the routing threshold τ is consistently set to 0.1 across all experiments. Finally, regarding the Classifier-Free Guidance (CFG) scale during inference, we adopt a scale of 0.0 (i.e., no guidance) for general knowledge tasks to ensure a fair evaluation of skipped layers. Further details regarding the hyperparameters and experimental settings are provided in Appendix Section A.2.

4.2. Efficiency-Accuracy Trade-off

Throughout our evaluation, computational cost is measured in terms of FLOPs. These FLOPs are reported as a ratio relative to the standard baseline model (i.e., the original

model without routing, where executing all layers equals 1.0). By varying the hyperparameter λ, we observe a clear trade-off between FLOPs and accuracy. To demonstrate the effectiveness of our framework, we compare it against two alternative strategies: (1) Static Skipping, which statically removes the k least critical layers (Goel et al., 2026), and (2) Random Skipping, which randomly drops k layers at each denoising step.

Figure 2 illustrates these results for PIQA and GSM8K. As shown in the figure, we generally achieve a better trade-off compared to the static and random layer skipping strategies. A performance comparison across all six datasets is summarized in Table 1. We report results at maximum FLOPs reduction while maintaining baseline accuracy; for MMLU and HellaSwag, where this is not achieved, we report the configuration comparable to static k=1.

Compared to the static layer skipping baseline, we observe that our framework achieves a superior efficiency-accuracy trade-off on the majority of the datasets (GSM8K, PIQA, MMLU, and MBPP). However, we also note that on ARC-C and HellaSwag, static layer skipping yields slightly better results. This indicates that for certain tasks, a static policy—which consistently preserves specific critical layers—can be robust.

Nevertheless, the overall results demonstrate the high potential of dynamic routing. Interestingly, in four out of the six datasets (GSM8K, PIQA, ARC-C, and MBPP), our approach successfully maintains or even outperforms the original baseline accuracy while reducing computational cost. Notably, on the PIQA dataset, we achieve a 14.26% reduction in FLOPs without any degradation in accuracy. These findings confirm the inherent layer redundancy in DLMs and demonstrate the merit of dynamic layer skipping.

4.3. Effect of Dynamic Scaling in the Training Objective

In Section 3.3, we propose a dynamic scaling strategy in our objective function, where the skip loss $\mathcal{L}_{\text{skip}}$ is divided by the current mask ratio p_{mask}. This encourages the routers to skip aggressively when the number of currently masked tokens is small, and to be more conservative when many tokens are still masked. To investigate the effect of this scaling strategy, we compare the efficiency-accuracy trade-off curves with and without the strategy. Here, "without scaling" means that the skip loss $\mathcal{L}_{\text{skip}}$ is applied without the $1/p_{\text{mask}}$ factor, meaning its penalty remains independent of the mask ratio.

Figure 3 shows the resulting trade-off curves on GSM8K and MBPP. As shown in the figure, scaling by the inverse mask ratio yields a better efficiency–accuracy trade-off than the unscaled variant on both datasets. This confirms that early denoising steps are computationally more demanding, requiring more active layers.

Table 1. Overall FLOPs and accuracy comparison.

Task	Method	Normalized FLOPs ↓		Accuracy ↑	
Mathematics					
GSM8K	Base Model	1.0000		71.11	
	+ Static Skipping	0.9688	−3.12%	68.99	−2.12
	+ Ours	0.9632	−3.68%	71.27	+0.16
General Tasks					
PIQA	Base Model	1.0000		70.62	
	+ Static Skipping	0.9063	−9.37%	71.00	+0.38
	+ Ours	0.8574	−14.26%	70.84	+0.22
MMLU	Base Model	1.0000		65.85	
	+ Static Skipping	0.9688	−3.12%	61.10	−4.75
	+ Ours	0.9265	−7.35%	61.40	−4.45
ARC-C	Base Model	1.0000		44.97	
	+ Static Skipping	0.9375	−6.25%	46.16	+1.19
	+ Ours	0.9425	−5.75%	44.97	+0.00
HellaSwag	Base Model	1.0000		70.68	
	+ Static Skipping	0.9688	−3.12%	70.38	−0.30
	+ Ours	0.9799	−2.01%	70.40	−0.28
Code					
MBPP	Base Model	1.0000		40.80	
	+ Static Skipping	0.9688	−3.12%	40.80	+0.00
	+ Ours	0.9700	−3.00%	41.00	+0.20

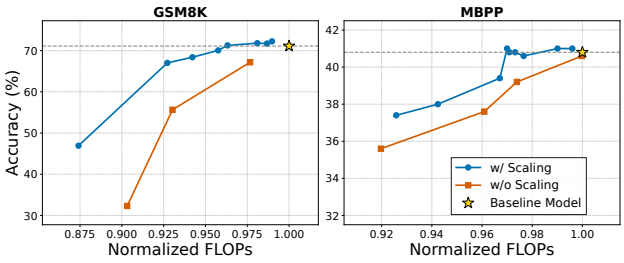

Figure 3. Impact of dynamic scaling on accuracy vs. normalized FLOPs (GSM8K, MBPP). Scaling the skip loss by $1/p_{\text{mask}}$ (blue) maintains higher accuracy by preventing excessive skipping, outperforming the unscaled approach (orange). The gold star and dashed line mark the baseline model.

4.4. Analysis of Layer Execution Patterns

To investigate how our framework operates across different domains, we analyze the execution ratios of each Transformer layer. We select the configurations that achieve the maximum computational reduction (lowest FLOPs) without compromising the baseline accuracy for each of the four datasets. Figure 4 presents the heatmap of the average execution ratios for all 32 layers under these configurations. The results reveal clear domain-specific execution patterns.

First, the initial eight layers (layers 1–8) are consistently executed at exactly 100% across all evaluated tasks. This suggests that these initial layers are essential for extracting fundamental feature representations, regardless of the task type. Beyond this initial phase, the execution patterns begin to diverge. For instance, in PIQA and GSM8K, the execu-

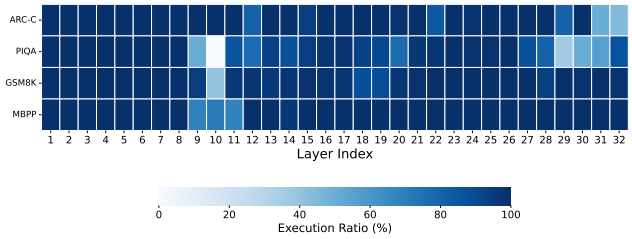

Figure 4. Heatmap of average layer execution ratios across four different datasets. For each task, we visualize the skipping pattern under the configuration that achieves the maximum computational reduction while maintaining baseline performance.

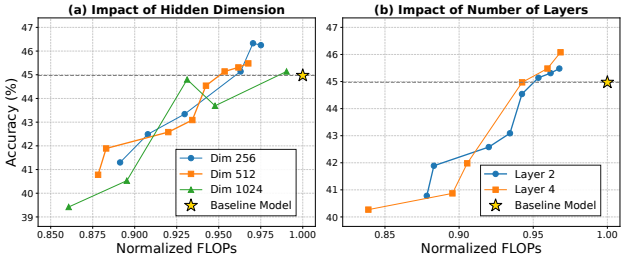

Figure 5. Trade-off between computational cost and accuracy when scaling the router's architecture. **(a)** Comparing hidden dimensions shows that varying the dimension has little to no impact on the trade-off. **(b)** Increasing the number of layers from 2 to 4 similarly yields no significant performance changes.

tion ratio drops sharply at layer 10 (to approximately 1% and 40%, respectively), whereas ARC-C maintains 100% execution up to layer 11.

Furthermore, the execution patterns in the subsequent layers reflect the nature of the tasks. For general knowledge and reasoning tasks like PIQA and ARC-C, the execution ratios decrease significantly in the final layers (e.g., dropping to roughly 36–56% in layers 29–31 for PIQA, and 44–50% in layers 31–32 for ARC-C). This indicates that the model can often complete these tasks without relying on the final refinement layers.

In contrast, for tasks requiring multi-step reasoning, such as mathematics (GSM8K) and code generation (MBPP), the router maintains near 100% execution in the latter half of the layers (e.g., layers 16–32). While these tasks allow for some computational reduction in the intermediate layers, they strictly rely on these latter layers. This demonstrates that complex reasoning inherently requires the final refinement layers to construct accurate outputs.

4.5. Analysis of Router Configuration

To investigate whether our baseline router configuration is sufficient for appropriate routing, we analyze how the efficiency-accuracy trade-off changes as the router architecture scales. In Figure 5, we illustrate the trade-off curves for different hidden dimensions (256, 512, and 1024). The results show that varying the hidden dimension has little impact on the efficiency-accuracy trade-off. Similarly, Figure 5 demonstrates that doubling the number of linear layers from 2 to 4 also does not significantly alter the efficiency-accuracy trade-off. Thus, deeper routers do not improve routing performance. Overall, these observations indicate that our baseline configuration (2 layers with a hidden dimension of 512) is sufficient for accurate routing.

4.6. Router Overhead

We evaluate the overhead of the router relative to the baseline DLM, LLaDA-8B, in Table 2. As shown, the router introduces only a marginal increase in parameter count (under 1.7%). Similarly, the additional peak memory footprint during inference is merely 0.12 GB, which is practically negligible compared to the 16.86 GB required by the baseline DLM. Furthermore, computational FLOPs and latency vary across individual prompts, as they depend on the sequence length and specific decoding dynamics. However, because the router's computation scales proportionally with the backbone's computation, the relative FLOPs overhead remains consistently low at approximately 0.003%. For instance, on the computationally demanding GSM8K dataset, the router adds an average of roughly 120 GFLOPs to the baseline's ∼4,200 TFLOPs per prompt. These results demonstrate that the router overhead is exceptionally modest relative to the backbone cost.

Table 2. Overhead information of the router. The latency and FLOPs metrics are derived from the average performance on the GSM8K dataset.

Metric	Baseline (LLaDA-8B)	Router Overhead	Relative Increase
Parameters	∼8.0 B	∼0.13 B	+ 1.68%
Inference Memory	16.86 GB	+ 0.12 GB	+ 0.71%
FLOPs (per prompt)	∼4,200 TFLOPs	∼120 GFLOPs	+ 0.003%

5. Conclusion

In this paper, we propose a dynamic layer-skipping framework for DLMs that employs a sequence-level routing mechanism, aggregating masked token representations to adaptively skip redundant Transformer layers. Experiments on LLaDA-Base-8B across six benchmarks show that our approach generally achieves a better FLOPs-accuracy trade-off than static and random baselines, indicating that executing all layers is not always necessary. Our analysis further reveals that initial layers are consistently critical across tasks, and that the required number of active layers decreases as the mask ratio drops—suggesting that layer redundancy naturally grows as denoising progresses. We hope these findings inspire the development of more efficient inference strategies for DLMs.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

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

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

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Berglund, L., Tong, M., Kaufmann, M., Balesni, M., Stickland, A. C., Korbak, T., and Evans, O. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*, 2023.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. Adaptive neural networks for efficient inference. In *International conference on machine learning*, pp. 527–536. PMLR, 2017.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Fan, S., Jiang, X., Li, X., Meng, X., Han, P., Shang, S., Sun, A., Wang, Y., and Wang, Z. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*, 2024.

Goel, R., Garrepalli, R., Agrawal, S., Lott, C., Lee, M., and Porikli, F. Skip to the good part: Representation structure & inference-time layer skipping in diffusion vs. autoregressive llms. *arXiv preprint arXiv:2603.07475*, 2026.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., and Fleet, D. J. Video diffusion models. *Advances in neural information processing systems*, 35:8633–8646, 2022.

Hu, Z., Meng, J., Akhauri, Y., Abdelfattah, M. S., Seo, J.-s., Zhang, Z., and Gupta, U. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv e-prints*, pp. arXiv–2505, 2025.

Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., and Weinberger, K. Q. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Israel, D., Broeck, G. V. d., and Grover, A. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jiang, Y., Wang, H., Xie, L., Zhao, H., Zhang, C., Qian, H., and Lui, J. C. D-llm: A token adaptive computing resource allocation strategy for large language models. *Advances in Neural Information Processing Systems*, 37: 1725–1749, 2024.

Kim, B.-K., Kim, G., Kim, T.-H., Castells, T., Choi, S., Shin, J., and Song, H.-K. Shortened llama: Depth pruning for large language models with comparison of retraining methods. *arXiv preprint arXiv:2402.02834*, 2024.

Kim, G. and Cho, K. Length-adaptive transformer: Train once with length drop, use anytime with search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6501–6511, 2021.

Liu, Z., Yang, Y., Zhang, Y., Chen, J., Zou, C., Wei, Q., Wang, S., and Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

Lu, G., Chen, H. M., Karashima, Y., Wang, Z., Fujiki, D., and Fan, H. Adablock-dllm: Semantic-aware diffusion llm inference via adaptive block size. *arXiv preprint arXiv:2509.26432*, 2025.

Ma, X., Yu, R., Fang, G., and Wang, X. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Ou, J., Nie, S., Xue, K., Zhu, F., Sun, J., Li, Z., and Li, C. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Raposo, D., Ritter, S., Richards, B., Lillicrap, T., Humphreys, P. C., and Santoro, A. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

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

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. pmlr, 2015.

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

Teerapittayanon, S., McDanel, B., and Kung, H.-T. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Wang, G., Schiff, Y., Sahoo, S. S., and Kuleshov, V. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025.

Wei, Q., Zhang, Y., Liu, Z., Liu, D., and Zhang, L. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv e-prints*, pp. arXiv–2506, 2025.

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

Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 2246–2251, 2020.

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

Yang, Y., Cao, Z., and Zhao, H. Laco: Large language model pruning via layer collapse. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6401–6417, 2024.

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

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

A. Appendix

A.1. Detailed Inference Algorithm

As discussed in the main text, the overall procedure for dynamic routing is summarized in Algorithm 1.

Algorithm 1 Dynamic Layer-Skipping Framework

Input: Initial hidden states $\boldsymbol{h}_1$, Masked token indices $\mathcal{M}$, Threshold τ, Layers $\{f_l\}_{l=1}^{N}$, Routers $\{R_l\}_{l=1}^{N}$
Output: Final hidden states $\boldsymbol{h}_{N+1}$

$M_t \leftarrow |\mathcal{M}|$ {Number of currently masked tokens}
for $l = 1$ **to** N **do**
  $\boldsymbol{h}_l^{\text{mask}} \leftarrow$ Extract representations at indices $\mathcal{M}$ from $\boldsymbol{h}_l$
  $\boldsymbol{logits}_l \leftarrow \text{MLP}_l(\boldsymbol{h}_l^{\text{mask}})$ {Compute 2D logits via MLP of router R_l}
  $\boldsymbol{v}_l^{(i)} \leftarrow \text{OneHot}(\text{argmax}(\boldsymbol{logits}_l^{(i)}))$ for all $i \in \{1, \ldots, M_t\}$ {Discrete token-level routing vote}
  $\bar{v}_{l,0} \leftarrow \frac{1}{M_t} \sum_{i=1}^{M_t} v_{l,0}^{(i)}$ {Compute sequence-level skip proportion}
  $e_{l,0} \leftarrow \mathbb{1}(\bar{v}_{l,0} > \tau)$ {Determine layer skip decision}
  if $e_{l,0} = 1$ **then**
    $\boldsymbol{h}_{l+1} \leftarrow \boldsymbol{h}_l$ {Bypass layer}
  else
    $\boldsymbol{h}_{l+1} \leftarrow f_l(\boldsymbol{h}_l)$ {Execute layer}
  end if
end for
return $\boldsymbol{h}_{N+1}$

A.2. Hyperparameters and Experimental Settings

We provide further details regarding the experimental settings and hyperparameters used for evaluation. Notably, these configurations are maintained consistently across both the baseline model and our proposed method to ensure a fair and rigorous comparison. Following the evaluation protocol of LLaDA-Base, the general multiple-choice tasks (PIQA, ARC-C, Hellaswag, and MMLU) are evaluated using conditional likelihood estimation, whereas the remaining tasks (GSM8K and MBPP) are evaluated using conditional generation. As summarized in Table 3, the hyperparameters are tailored to these two distinct evaluation formulations.

Table 3. Evaluation settings and hyperparameters for each dataset. "–" indicates that the parameter is not applicable to the evaluation format of the specific task. All settings are identical for both the baseline and our proposed dynamic approach.

Parameter	PIQA	ARC-C	Hellaswag	MMLU	GSM8K	MBPP
Output Length	–	–	–	–	256	256
Denoising Steps	–	–	–	–	256	256
Block Size	–	–	–	–	256	256
Few-shot	0	0	0	5	5	3
Greedy Decoding	False	False	False	False	–	–
CFG Scale	0.0	0.0	0.0	0.0	–	–
MC Num	128	128	128	1	–	–

Details of the Static Layer Skipping Baseline For the static skipping baseline (Goel et al., 2026), we apply their proposed layer-evaluation algorithm to our base model, LLaDA-Base-8B, to identify the least critical Transformer layers. The layers identified by the algorithm are then statically skipped across all denoising steps. Note that the 32 Transformer layers are numbered from 1 to 32. As we progressively increase the number of skipped layers k, the specific combinations of skipped layer indices are determined as follows:

Number of skipped layers (k)	Skipped Layer Indices
1	$\{5\}$
2	$\{5, 7\}$
3	$\{5, 7, 9\}$
4	$\{3, 5, 7, 9\}$

Training Details and Computational Resources All models are implemented in PyTorch and run on a server equipped with four NVIDIA H100 GPUs. Specifically, we utilize all 4 GPUs for training, while inference is conducted on a single GPU. During training, the base model parameters are kept strictly frozen to exclusively update the proposed dynamic router. We train using `bfloat16` precision, a maximum sequence length of 1024, and an effective batch size of 64 achieved via gradient accumulation. The router is optimized using AdamW ($\beta = (0.9, 0.95)$, weight decay 0.02, max gradient norm 1.0) with a peak learning rate of 5×10^{-5}. The learning rate schedule consists of a 2-epoch linear warmup followed by a half-cycle cosine decay. Depending on the dataset, a complete training run on our 4-GPU setup takes from 15 minutes up to roughly 2 hours for Hellaswag, the most computationally intensive task.

Furthermore, to illustrate the learning dynamics of the router, Figure 6 presents the trajectory of the average layer execution ratio evaluated on the validation set during training for the ARC-C dataset. As shown in the figure, the execution ratio smoothly transitions and converges to a stable value as the training steps progress. This clear convergence indicates that the router effectively learns a consistent, reliable layer-skipping policy during training.

Figure 6. Trajectory of the layer execution ratio on the validation set during the training of the router on the ARC-C dataset. The ratio consistently converges to a stable value as training progresses.

A.3. Detailed Latency Profiling and Implementation Efficiency

To demonstrate the real-world efficiency of DIALS and address potential concerns regarding GPU scheduling overhead, we conducted a deep layer-level latency profiling. Throughout our experiments, we observed that a naive dynamic routing implementation in modern frameworks (e.g., PyTorch) incurs significant overhead due to memory slicing (*gather/scatter*) and dynamic control flow synchronization. To mitigate this, we employed a hardware-friendly dense execution strategy. Because the router consists of point-wise linear layers, extracting mask tokens before the MLP is mathematically equivalent to applying the MLP across the dense sequence and extracting the mask token outputs afterward. By adopting the latter, we effectively bypass the memory irregularity before the matrix multiplications, maximizing GPU Tensor Core utilization.

Profiling Results: Under this optimized implementation, taking the GSM8K dataset as an example, the total execution overhead introduced by DIALS per Transformer block is approximately 0.232 ms. Considering that the execution of a pure standard LLaDA-8B block takes roughly 1.453 ms, the dynamic routing introduces merely a $\sim$16% latency overhead per block. Currently, our framework requires approximately a 16% layer skip rate to break even in actual wall-clock time.

Hardware and Framework Challenges: A detailed breakdown of this 0.232 ms overhead reveals significant future optimization potential, as presented in Table 4. Crucially, the pure neural network computation of the router (linear projections, activations,

Table 4. Detailed latency breakdown of the DIALS overhead per Transformer block (taking GSM8K as an example). The pure neural network computation accounts for only ∼37.5% (0.087 ms) of the total overhead, representing a mere ∼6.0% of the baseline block's latency. The majority of the overhead is consumed by framework-level artifacts required for dynamic batch routing, specifically block-level memory slicing (gather/scatter) and CPU-GPU synchronization.

Operation Category	**Latency (ms)**	**Proportion**
Pure NN Compute (Linear, Act, Softmax, Logic)	0.087	37.5%
Batch-level Slicing (Block Gather / Scatter for active batches)	0.065	28.0%
Synchronization (`torch.nonzero`, GPU-CPU sync)	0.060	25.9%
Token-level Slicing (Router internal Gather / Scatter)	0.006	2.6%
Other Framework Overhead	0.014	6.0%
Total DIALS Overhead	0.232	100.0%
Reference: Pure Baseline Transformer Block	1.453	-
Ratio: Pure NN Compute / Baseline Block	-	6.0%

and softmax) consumes only ∼0.087 ms. This accounts for merely 37.5% of the total DIALS overhead, representing a mere ∼6.0% of the baseline Transformer block's execution time. The remaining majority of the latency (∼62.5%) is not computational but is entirely dominated by framework-level artifacts required to manage dynamic execution within PyTorch. Specifically, supporting dynamic routing across multiple sequences in a batch requires block-level memory slicing (gather and scatter, ∼0.065 ms) to extract only the active batch indices and write back their results. Furthermore, evaluating these dynamic batch sizes requires GPU-to-CPU synchronization (e.g., `torch.nonzero`, ∼0.060 ms). These profiling results highlight that the pure algorithmic complexity of DIALS is remarkably minimal. The current latency-efficiency trade-offs imposed by framework overheads are not fundamental limits, but rather engineering challenges to be overcome. As custom kernels for dynamic execution evolve to fuse batch masking and routing natively, we anticipate the actual wall-clock latency of DIALS will converge closely toward its theoretical FLOPs reduction bounds.