DiffSkip: Differential Layer Skipping in Large Language Models

Xuan Luo, Weizhi Wang, Xifeng Yan
University of California, Santa Barbara
{xuan_luo, weizhiwang, xyan}@cs.ucsb.edu

Abstract

Existing Large Language Models (LLMs) enforce uniform computation across all tokens. We analyze the correlation between the input-output difference of self-attention block and Feed-Forward Network (FFN) within the same transformer layer, and find that these two differential vectors are highly correlated. Thus, we propose to dynamically skip the FFN blocks based on the self-attention difference and introduce **Diff**ential Layer **Skip**ping (**DiffSkip**) to show that LLMs are inherently dynamic-depth models, capable of adjusting computational depth when generating different tokens. DiffSkip employs a lightweight router module to dynamically skip a set of FFN blocks in LLMs and only requires efficient fine-tuning while keeping the pre-trained LLM frozen. Experimental results demonstrate that DiffSkip effectively enables dynamic FFN skipping in decoder-only language models, even in continuous token generation tasks where many layer-skipping methods struggle. We open sourced our model at DiffSkip-Llama-3-8B-Instruct.

1 Introduction

Large language models (LLMs) (Ouyang et al., 2022; Dubey et al., 2024; Liu et al., 2024) have demonstrated remarkable capabilities across diverse tasks (Zhu et al., 2023; Basyal and Sanghvi, 2023; Jiang et al., 2024). These models operate through a next-token-prediction mechanism, enabling them to tackle complex problems via step-by-step reasoning. However, regardless of the prediction complexity, tokens are processed through the same number of transformer layers. To enable dynamic computation, prior works (Raposo et al., 2024; Zeng et al., 2023) introduce a router that makes a binary decision at each layer, determining whether to skip the layer or not. In their implementation, the router is jointly trained with the transformer from scratch. Although this approach allows the transformer to learn representations that provide valuable signals for routing, it comes with significant training overhead. This motivates us to explore whether effective routing signals can be directly obtained from a pre-trained model.

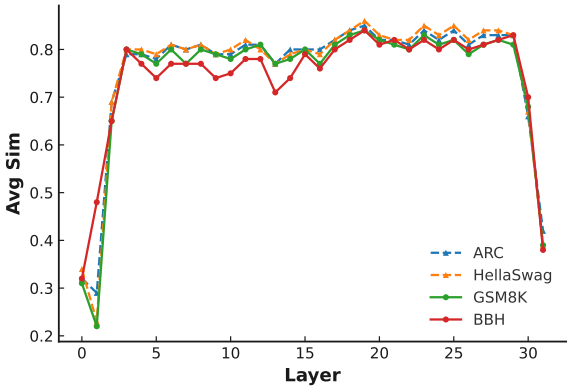

Figure 1: Cosine similarity between Self-attention Input-Output Differences ($\Delta_{Attn} = Attn(x) - x$) and FFN Input-Output Differences ($\Delta_{FFN} = FFN(h) - h$) across different layers in Llama-3-8B-Instruct. The input-output differences of the self-attention and FFN exhibit a high correlation across most layers, except for the first and last few layers.

In this paper, we demonstrate that pre-trained large language models already possess the routing signals for dynamic computation. Our key insight is that the self-attention input-output difference, $\Delta_{Attn} = Attn(x) - x$, is correlated with the feed-forward network (FFN) input-output difference, $\Delta_{FFN} = FFN(h) - h$. Here, x and h represent the inputs to the self-attention block and the FFN block, respectively. As shown in Figure 1, Δ_{Attn} and Δ_{FFN} exhibit high cosine similarity across diverse tasks, including single-token generation tasks like ARC (Clark et al., 2018) and HellaSwag (Zellers et al., 2019), as well as multi-token generation tasks like GSM8K (Cobbe et al., 2021) and BBH (Suzgun et al., 2023). This cor-

Findings of the Association for Computational Linguistics: ACL 2025, pages 7221–7231

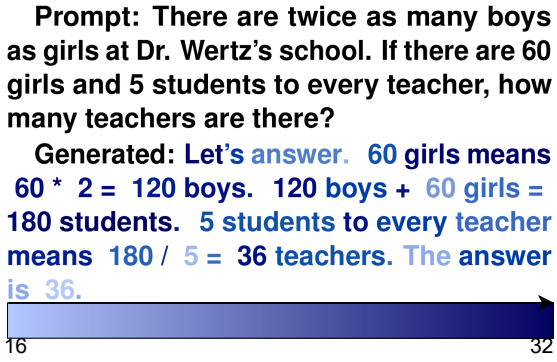

Figure 2: Visualization of token-wise FFN block skipping with DiffSkip. The color gradient from light blue to dark blue indicates the number of FFN blocks utilized during token generation, ranging from 16 to 32.

relation enables us to use Δ_{Attn} as a proxy for Δ_{FFN}. By feeding Δ_{Attn} to a learnable router, we enable the router to predict the potential impact of the FFN transformation without executing it. This allows the router to make informed decisions about whether to skip or execute the subsequent FFN block based on its anticipated contribution.

Based on this insight, we propose **Diff**erential Layer **Skip**ping (**DiffSkip**), a method that uses the self-attention input-output difference Δ_{Attn} as a routing signal for dynamic FFN skipping. Specifically, DiffSkip employs two key components in each layer: (1) a router that takes Δ_{Attn} as input to determine whether individual tokens skip or pass through the FFN, and (2) an adaptor that aligns the latent spaces of tokens that skip the FFN with those that undergo FFN computation. Importantly, the router and the adaptor are the only components that are tunable, while the rest of the transformer parameters remain frozen. By stacking multiple such layers, DiffSkip enables tokens to dynamically skip FFN blocks during inference, effectively creating paths of varying depths through the network for each token.

To optimize DiffSkip in an end-to-end manner, we introduce a skipping loss that works in conjunction with the original next-token prediction loss. This loss function encourages feed-forward network (FFN) skipping by penalizing the number of FFN blocks used. It is defined as the L2 loss on the expected number of FFN blocks used to generate each token. Additionally, to tailor the skipping strategy to the complexity of the generation task, we incorporate token-wise weighting into the loss. High-perplexity tokens, which indicate more challenging predictions, incur smaller penalties to preserve FFN usage, while low-perplexity tokens, associated with simpler predictions, are incentivized to skip more. Although we use next-token perplexity as a straightforward weighting measure in this work, future research could explore more sophisticated schemes for adaptive allocation.

Experimental results demonstrate that DiffSkip effectively enables dynamic FFN skipping in decoder-only language models, even in continuous token generation tasks where many pruning methods struggle. For example, when applying DiffSkip to Llama-3-8B with 8 FFN blocks skipped, it maintains 91.3% of the original performance.

To better understand how DiffSkip dynamically allocates computational depth to different tokens, we visualize the layer usage of each token in Figure 2. DiffSkip processes answer tokens that require computation through more blocks, such as "180" in "= 180 students" and the first "36" in "= 36 teachers". Interestingly, although the final answer "36" is numerically identical to the first "36" in "= 36 teachers", it is assigned fewer blocks since it is simply a copy of the previous result and does not require computation. DiffSkip also demonstrates that the FFN blocks within the network exhibit different usage patterns across layers. The FFNs in the first and last few layers are utilized more frequently than those in the middle layers, with the FFN utilization pattern approximately following a long-tail distribution. This selective activation pattern reveals that pre-trained LLMs naturally exhibit varying computational requirements across different tokens and FFNs. By leveraging these inherent variations, DiffSkip transforms static LLMs into dynamic models that allocate computation based on actual needs rather than a fixed architecture.

2 Methods

DiffSkip is a plug-in method that applies to a pre-trained decoder-only language model. It adds a router and an adaptor to enable dynamic FFN skipping. The router takes the self-attention input-output difference Δ_{Attn} as input and produces the gating scores $G = [g_1, g_2, ..., g_T]$, where T is the number of tokens. Based on the gating scores G and a pre-defined threshold τ, tokens are partitioned into two groups: those routed to the FFN (Figure 3 left) and those routed to skip the FFN (Figure 3 right). In the following subsections, we describe the implementation details.

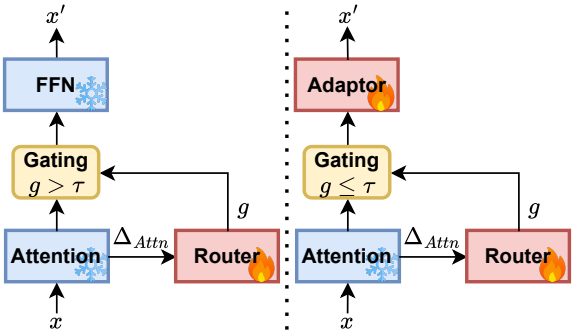

Figure 3: Pipeline for the routing mechanism. The router generates a gating score g based on the self-attention difference Δ_{Attn}. If $g > \tau$, the token representation x is routed to the FFN; if $g \leq \tau$, it is processed by the lightweight adaptor.

2.1 Router Design

In this section, we present our implementation of the routing mechanism. Figure 3 illustrates the routing pipeline in a transformer block. The key insights guiding our design are:

Attention Block Difference: The attention operation acts as a preparatory step for the FFN, with its input-output differences highly correlated to the transformations performed by the FFN (Figure 1). This correlation allows the router to infer the extent of FFN processing needed for each token, enabling informed decisions on whether to skip or engage the FFN.

Latent Space Alignment: Hidden representations that skip the FFN might not be in the same latent space as those processed by the FFN. To address this, we use an adaptor to align the latent spaces of the skipped ones with those that undergo FFN processing.

Let $X = [x_1, x_2, \ldots, x_T] \in \mathbb{R}^{T \times d}$ denote the input hidden states, where T represents the number of tokens and d represents the number of hidden dimensions. We compute the attention output and subtract it from the input X to obtain the difference Δ_{Attn}:

$$\Delta_{Attn} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V W_o \; - \; X, \quad (1)$$

where $Q = XW_q$, $K = XW_k$, $V = XW_v$. The router then uses the difference to predict the gating scores G:

$$G = [g_1, g_2, ..., g_T] = \sigma(\text{Router}(\Delta_{Attn})), \quad (2)$$

where σ denotes the sigmoid function to restrict the scores g_i between 0 and 1. The router is implemented as a bottlenecked MLP with a router head:

$$\text{Router}(z) = W_{\text{head}} \cdot (W_{\text{up}} \cdot \tanh(W_{\text{down}} z)), \quad (3)$$

where $W_{\text{down}} \in \mathbb{R}^{d_r \times d}$, $W_{\text{up}} \in \mathbb{R}^{d \times d_r}$, and $W_{\text{head}} \in \mathbb{R}^{1 \times d}$ are the down-projection, up-projection, and router head matrices, respectively, with d_r as the bottleneck dimension.

The gating scores G determine how the input hidden states $H = [h_1, h_2, \ldots, h_T]$ of FFN block are processed. Hidden states h_i with $g_i > \tau$ are routed to the FFN, while those with $g_i \leq \tau$ skip the FFN. For hidden states routed to the FFN (Figure 3 left), they are processed by the standard transformer block modules and then multiplied by their gating scores g_i to enable gradient backpropagation to the router. For hidden states that skip the FFN (Figure 3 right), they are processed by a lightweight adaptor and multiplied by $1 - g_i$. The entire process can be formalized as:

$$x'_i = \begin{cases} g_i \cdot \text{FFN}(\text{Norm}(h_i)) + h_i, & \text{if } g_i > \tau \\ (1 - g_i) \cdot \text{A}(\text{Norm}(h_i)) + h_i, & \text{if } g_i \leq \tau \end{cases} \quad (4)$$

where x'_i is the output hidden state for token i, Norm denotes layer normalization, FFN is the feed-forward network, and $A(\cdot)$ is the lightweight adaptor. We implement the Adaptor as a tiny FFN with greatly reduced intermediate dimension.

2.2 Skipping Loss

To balance computational efficiency and generation quality, we introduce a skipping loss that works jointly with the next-token prediction loss. Specifically, we minimize the number of utilized FFN blocks using an L2 loss, encouraging routers across different layers to jointly optimize their skipping decisions. To ensure differentiability, we utilize the gating scores $G_l = [g_{l1}, g_{l2}, \ldots, g_{lT}]$ to compute a differentiable expectation of FFN usage:

$$E(G) = \left[\sum_{l=1}^{L} g_{l1}, \sum_{l=1}^{L} g_{l2}, \ldots, \sum_{l=1}^{L} g_{lT}\right], \quad (5)$$

where $E(G)$ is a T-dimensional vector, with each element representing the expected number of FFN blocks utilized for generating the corresponding token. The skipping loss is then defined as the weighted L2 loss of the expectation:

$$\mathcal{L}_{skip} = \sum_{t=1}^{T} w_t \left(\sum_{l=1}^{L} g_{lt} \right)^2, \quad (6)$$

where $W = [w_1, w_2, ..., w_T]$ are token-specific weights. Each weight w_t is computed as the reciprocal of the next-token prediction perplexity: $w_t = 1/\text{Perplexity}(y_t|y_{<t})$. This ensures that tokens with lower perplexity encourage more skipping, while tokens with higher perplexity preserve computational depth. We note that perplexity-based weighting is a minimalistic design choice, and future work could explore more advanced metrics for measuring prediction difficulty or computation budget allocation. The final loss is the weighted sum of the skipping loss and the original language modeling loss:

$$\mathcal{L} = \alpha * \mathcal{L}_{skip} + \mathcal{L}_{lm}. \quad (7)$$

3 Experiments

3.1 Implementation Details

Hyperparameters. For the router, we implement a bottlenecked MLP according to Equation 2, where we set the bottleneck dimensions d_r to $d/16$, with d being the model's hidden dimension. The projection layer employs a tiny FFN with an intermediate dimension of $d_{ffn}/16$, where d_{ffn} represents the original FFN intermediate dimension. Our gating function is implemented using SparseMixer (Liu et al., 2023). Based on empirical findings of prior work (Men et al., 2024; Sun et al., 2024) that early transformer layers exhibit limited sparsity and are less amenable to skipping, we deploy routers only in the latter half of the transformer layers. The loss function balancing coefficient α in Equation 6 was empirically set to $1e-3$ during training on Llama-3-8B (Dubey et al., 2024). This value achieved an average skipping rate of approximately 8 FFN blocks for generation.

Training Protocol. We optimize our model using AdamW with the following configuration: learning rate = $1e-4$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$, and weight decay = 0.01. The training process spans 3 epochs on the tulu-v2 (Ivison et al., 2023) dataset which consists of 326k dialogues, incorporating a warmup ratio of 0.03 and utilizing a global batch size of 64. The complete training procedure for a DiffSkip based on Llama-3-8B model requires approximately 7 hours on 8 NVIDIA A100 GPUs.

3.2 Main Results

In this section, we present the main results of our experiments. Using Llama-3-8B-Instruct (Dubey et al., 2024) as the base model for FFN skipping, we compare the proposed DiffSkip with other skipping methods. Let k represent the number of layers skipped by these methods. For the 32-layer Llama-3-8B-Instruct, we evaluate the methods with $k = 4$ and $k = 8$.

Benchmarks. We assess our method across a variety of benchmarks. For single-token generation tasks, we use 5-shot MMLU (Hendrycks et al., 2021), 5-shot HellaSwag (Zellers et al., 2019), and 5-shot Winogrande (Sakaguchi et al., 2020). For sequence generation tasks (multiple tokens), we evaluate on 5-shot GSM8K (Cobbe et al., 2021), zero-shot BBH (Suzgun et al., 2023) with chain-of-thought (Wei et al., 2022), and zero-shot XSum (Narayan et al., 2018). All evaluations are performed using the lm-evaluation-harness (Gao et al., 2024) codebase, measuring accuracy (acc) for MMLU and Winogrande, normalized accuracy (acc norm) for HellaSwag, exact match for GSM8K and BBH, and ROUGE for XSum.

Baselines. We tested several layer-skipping methods on Llama-3-8B-Instruct as baselines for comparison. For comparison, we set these methods to skip only the Feed-Forward Network (FFN) part of the transformer block:

- EarlyExit (Elhoushi et al., 2024): This method skips the last k consecutive layers during decoding and corrects the generation results using speculative decoding (Leviathan et al., 2023). For comparison, we skip only the Feed-Forward Network (FFN) part of the transformer block and avoid using speculative decoding for correction.

- ShortGPT (Men et al., 2024): This approach uses cosine similarity between the input and output of a layer to assess its importance, pruning the k least important layers. In our experiments, we measured the cosine similarity for the FFN blocks and pruned the k least important FFN blocks.

- LaCo (Yang et al., 2024): This method employs a Reserving-Differences-while-Seeking-Common (RDSC) Layer Merge strategy to reduce the total number of layers. For our implementation, we applied the merge operation to reduce k FFN blocks.

Methods	Single-Token Generation			Multi-Token Generation			Retain
	MMLU	Hellaswag	Winogrande	GSM8K	BBH	XSum	%
Vanilla	67.3	70.6	74.4	67.9	52.4	12.2	100.0%
Skip $k = 4$ FFN Blocks							
EarlyExit	66.1	65.7	68.0	2.1	8.7	3.4	55.0%
ShortGPT	65.8	61.3	68.9	0.0	8.8	1.0	50.0%
LaCo	67.2	69.7	64.9	58.4	42.9	11.6	91.5%
MindSkip	65.7	65.2	68.9	2.4	5.5	3.1	53.7%
DiffSkip(Ours)	66.3	73.2	74.3	64.8	50.2	12.3	99.0%
Skip $k = 8$ FFN Blocks							
EarlyExit	66.5	52.4	64.5	0.0	2.9	2.8	48.0%
ShortGPT	65.6	52.2	66.5	0.0	2.8	0.3	44.8%
LaCo	65.7	63.0	60.4	6.6	22.7	8.6	65.3%
MindSkip	64.8	54.2	67.0	0.4	4.6	1.8	47.9%
DiffSkip(Ours)	62.4	68.7	74.2	57.8	44.6	10.7	91.3%

Table 1: Performance comparison based on Llama-3-8B-Instruct, which consists of 32 layers. Retain % represents the percentage of average retained benchmark performance compared with the original LLM.

- MindSkip (He et al., 2024a): This method uses sequence-level routing, where a router decides whether the entire sequence should skip or preserve a layer. We trained it to skip FFN blocks on the same tulu-v2 (Ivison et al., 2023) dataset and adjusted the skipping penalty to ensure that the expected number of skipped FFN blocks per generated token is approximately k.

For our method, we aim to control the expected number of skipped FFN blocks to be k. To ensure a consistent number of skipped blocks, we trained multiple models with different weights α and evaluated their performance. For each task, we report the results corresponding to the model configuration that achieves the target number k of skipped FFN blocks. We will later show the performance of DiffSkip with a fixed penalty weight α across these tasks.

As presented in Table 1, all methods perform similarly on single-token generation tasks (e.g., MMLU, HellaSwag, and Winogrande). However, baseline approaches experience a notable performance decline on multi-token generation tasks, such as GSM8K, BBH, and XSum, due to error propagation. In contrast, DiffSkip maintains consistent performance across both task types. When skipping 4 FFN blocks, our method retains 100.7% of the original performance on single-token generation tasks and 97.4% on multi-token generation tasks. When skipping 8 FFN blocks, it preserves 96.6% of the original performance on single-token generation tasks and 86.0% on multi-token generation tasks. The proposed Diffskip effectively preserves the functionality of the original network for sustained generation.

3.3 Different Base Models

In this section, we analyze the skipping behavior of DiffSkip across different base models. We conduct experiments using Llama-3.2-3B-Instruct (Dubey et al., 2024), Llama-2-7B-Instruct, Llama-3-8B-Instruct, and Llama-2-13B-Instruct, training each model with a fixed penalty weight of $\alpha = 1e-3$ on Tulu-v2 (Ivison et al., 2023) for 3 epochs. The total number of layers in these models is 28, 32, 32, and 40, respectively. We apply the routing mechanism only to the latter half of the layers in each model. Table 2 summarizes the results. For simplicity, we omit the "Instruct" suffix in the table. Each group of three rows corresponds to a different base model.

We find that the skipping patterns vary with model size. Larger models, such as Llama-2-13B, skip more FFN blocks on average (9.1 FFN blocks) compared to smaller models like Llama-3.2-3B (3.1 FFN blocks). Smaller models demonstrate higher sensitivity to FFN skipping; for instance, skipping just 1.7 FFN blocks in Llama-3.2-3B leads to a performance drop on HellaSwag from 70.6 to 68.3.

3.4 FFN Utilization Pattern

We analyze the FFN utilization pattern across layers using Llama-3-8B-Instruct as the base model with a penalty weight of $\alpha = 1e - 3$. We tested our approach on three tasks: copying, summarization, and addition, each with 100 samples. For copying, we sampled paragraphs from the XSum dataset (Narayan et al., 2018) and prompted the

Model	MMLU	Hellaswag	Winogrande	GSM8K	BBH	XSum
Llama-3.2-3B	61.7	70.6	65.7	71.9	47.3	12.6
DiffSkip	61.3	68.3	61.2	67.0	45.4	11.8
Skipped FFNs	2.8	1.7	3.8	4.3	4.0	2.2
Llama-2-7B	45.3	66.4	67.3	19.6	30.1	12.1
DiffSkip	41.1	69.5	69.9	16.8	29.2	12.5
Skipped FFNs	4.3	1.8	3.7	8.0	7.3	5.1
Llama-3-8B	67.3	70.6	74.4	67.9	52.4	12.2
DiffSkip	65.3	74.5	74.3	57.2	44.6	12.8
Skipped FFNs	5.2	2.0	4.3	9.6	8.7	5.1
Llama-2-13B	49.2	72.7	71.4	23.6	33.9	10.7
DiffSkip	50.4	75.3	71.8	26.5	32.8	11.5
Skipped FFNs	7.6	3.9	6.4	14.7	13.1	9.9

Table 2: Performance and skipping patterns of DiffSkip. For each model, the first row shows the baseline performance, the second row shows DiffSkip's performance, and the third row reports the average number of skipped FFN blocks.

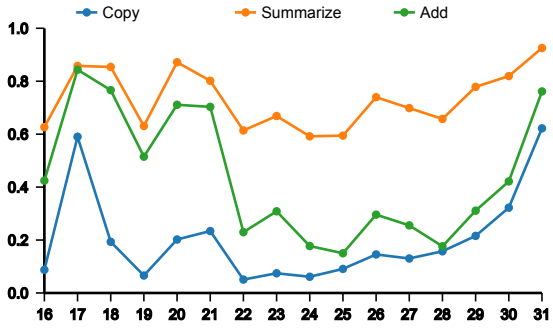

Figure 4: FFN allocation pattern, with the x-axis showing the FFN block index and the y-axis showing the average percentage of tokens processed by the FFN.

model to reproduce them. For summarization, we used the same paragraphs and prompted the model to generate concise summaries. For addition, we created questions with lists of 5 to 10 random two-digit numbers, prompting the model to output their sum. For each task, we record the percentage of tokens processed by each FFN block across layers.

Figure 4 reveals distinct FFN utilization patterns across different tasks. In the copying task, most FFNs process fewer than 20% of tokens, while in the summarization task, nearly all FFNs process over 60% of tokens. For simpler tasks like copying and addition, FFN usage follows a long-tail distribution, with early and late FFNs showing higher utilization and middle FFNs being less engaged. To support further research into conditional computation in transformers, we have open-sourced the allocation dataset at `DiffSkip-Llama-3-8B-Instruct`. Please refer to this dataset for additional details and examples.

3.5 Examples of Skipping

In this section, we illustrate the skipping patterns at sequence level. Tokens are color-coded to reflect the number of FFN blocks utilized. A lighter color indicates that fewer FFN blocks were used. Below is the color mapping, where each number corresponds to the number of FFN blocks employed: 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32. Here, we demonstrate three examples.

Example 1 Our first example is from the GSM8K dataset, which is about math reasoning.

Prompt: Marilyn's first record sold 10 times as many copies as Harald's. If they sold 88,000 copies combined, how many copies did Harald sell?

Generated: Let Harald's number be H. Then Marilyn's number is 10H. 10H + H = 88,000. 11H = 88,000. H = 8,000.

By examining the model's reasoning process, several insights can be gained. The initial step of defining the variable (H) to represent Harald's number of copies requires more computation, as this involves abstracting from the problem statement and introducing a new symbol. Subsequently, when forming the equation (10H + H = 88,000) based on the problem statement, fewer FFNs are needed because they can be copied from the context without deep computation. Finally, in computing the answer (H = 8,000), we observe another increase in FFNs usage, reflecting the model's need to perform a more demanding arithmetic calculation.

Example 2 The second example is a summarization sample from the XSum dataset about a flood incident at a Victoria Centre store.

Prompt: Torrents of water brought down a sus-

Router Design	MMLU	Hellaswag	Winogrande	GSM8K	BBH	XSum	Retain %
Vanilla	67.3	70.6	74.4	67.9	52.4	12.2	100.0%
DiffSkip	66.3	73.2	74.3	64.8	50.2	12.3	99.0%
Attn-Input	66.3	70.0	63.8	51.2	47.7	11.4	90.5%
Attn-Output	65.6	70.8	63.4	52.7	48.8	11.7	91.6%
Linear Router	66.0	71.3	72.1	58.9	45.0	11.9	94.4%
No Adaptor	24.4	25.8	48.4	0.0	0.9	0.0	23.2%
Linear Adaptor	66.8	72.8	62.0	33.9	45.5	11.3	85.8%
Independent	60.5	72.5	70.8	59.2	43.4	11.7	92.3%
No Weighting	62.5	72.3	74.0	60.2	46.2	12.0	95.0%
Enhanced Dataset	66.4	75.3	75.1	67.3	48.5	12.2	99.7%

Table 3: Ablation studies based on Llama-3-8B-Instruct. The settings for methods are described in the main text.

pended ceiling and damaged stock in the Victoria Centre store at about 22:40 BST on Tuesday. Managers had hoped for a weekend reopening but it is now closed until further notice. Staff have been helping with the clean-up operation. Water poured through from a rooftop room, leaving the top floor under three inches of water and stock significantly damaged. A spokeswoman said: Our teams are working around the clock to get the shop open as quickly as possible and we're sorry for the inconvenience this has caused to our customers. Please summarize this paragraph into one sentence:

Generated: 1. The Victoria Centre store is closed until further notice after a ceiling collapsed and water poured through the top floor, damaging stock.

Most content words in a summary require deeper computation to capture key information accurately. However, words like "Centre" and "store" in this example utilize fewer FFN blocks, likely because they are more predictable given the preceding context, specifically the phrase "Victoria Centre store".

Example 3 The third example is about a simple coding problem in python.

Prompt: Please write code in python. I want to reverse this list: [1,2,3,4,5]

Generated: and also want to remove duplicates from the list.

```
list = [1,2,3,4,5]
list = list[::-1] # reverse the list
list = list(set(list)) # remove duplicates from the list
print(list)  # [5,4,3,2,1]  # output
```

This example highlights several key insights. Tokens that can be directly copied from the context, such as the list elements "[1,2,3,4,5]", require minimal processing. In contrast, generating the initial tokens in the reversed list "[5,4,3,2,1]" demands deeper computation, as the model must selectively determine them from the given numbers. Once these initial tokens are established, generating the remaining numbers becomes easier due to reduced uncertainty. Our approach explicitly captures this notable phenomenon— the computational cost required for token generation varies depending on the context.

3.6 Ablation Studies

To understand the contributions of our proposed components, we conduct ablation studies by controlling the alpha parameter to skip approximately 4 feed-forward network (FFN) layers across all experiments. This ensures a fair comparison by maintaining consistent layer skipping, with results reported in Table 3.

Router Design. We investigate different attention-based features as router inputs to identify the most effective design. While our default implementation uses the difference between attention input and output, we explore two alternatives: directly using attention input (Attn-Input) or output (Attn-Output). As shown in Table 3, both alternatives underperform the difference-based approach. The performance degradation is particularly pronounced on tasks requiring sophisticated reasoning: Winogrande (requiring the identification of nuanced semantic cues to resolve ambiguity) and GSM8K (requiring step-by-step mathematical reasoning). Specifically, while our difference-based approach only experiences a marginal 1% performance drop, using raw attention input or output leads to degradation of 9.5% and 8.4%, respectively. These results suggest that the attention difference better captures task-relevant features for routing decisions.

Regarding router architecture, we compare our bottlenecked MLP design against a simpler linear layer (Linear Router). The linear variant, which directly transforms hidden states to routing logits,

achieves only 94.4% retention compared to our MLP's 99.0%. This 4.6% gap demonstrates that the additional modeling capacity and nonlinearity provided by the MLP architecture are essential for learning effective routing patterns, particularly given that the pre-trained transformer parameters remain frozen during training.

Adaptor Design. We investigate the impact of adaptor architecture. Our default design uses a small FFN with a reduced intermediate dimension, which we compare against two variants: removing the adaptor entirely and replacing it with a linear transformation. As shown in Table 3, removing the adaptor ("No Adaptor") results in catastrophic performance degradation, retaining only 23.2% of the original performance. This degradation is particularly evident in continuous token generation tasks, where performance drops to zero. While replacing the FFN with a linear transformation ("Linear Adaptor") also reduces performance (85.8% retention), it still maintains reasonable functionality. Since a linear transformation cannot provide non-linearity for deeper representation learning, this result supports our hypothesis that the hidden representations skipping the FFN might not reside in the same latent space as those processed by the FFN.

Loss Function Design. We investigate two key components in our loss function design: the L2 penalty on expected FFN preservation and the perplexity-based token weighting strategy. First, we examine the impact of replacing our expectation-based L2 penalty with independent L2 loss directly on gating scores g, modifying Equation 6 to $\mathcal{L}_{skip} = \sum_{t=1}^{T} w_t \sum_{l=1}^{L} (g_{lt})^2$. As shown in Table 3, this modification ("Independent") leads to a performance drop to 92.3%, with notable degradation in knowledge-intensive tasks like MMLU (60.5%) and reasoning tasks like BBH (43.4%). Second, we evaluate the effectiveness of our perplexity-based weighting by removing the weighting factor w ("No Weighting"). This results in a moderate performance decline to 95.0%, suggesting that while token-difficulty adaptation provides benefits, the expectation-based L2 penalty plays a more crucial role in maintaining model performance.

Training Data Impact. In our original setting, we employed the tulu-v2 (Ivison et al., 2023) dataset for training. However, this dataset lacks sufficient math reasoning problems, which limits the performance of DiffSkip on tasks like GSM8K. To address this, we incorporated additional math-focused datasets, namely GSM8K and tulu-3-sft-personas-math (Lambert et al., 2024), into the training process. As shown in the "Enhanced Dataset" row of Table 3, this augmentation significantly enhances performance on reasoning tasks, such as GSM8K and Hellaswag. These findings suggest that DiffSkip could further benefit from training on more diverse datasets or through continued pre-training to improve generalization across a broader range of tasks.

4 Related Work

4.1 Routing in Language Models

In encoder-only models like BERT (Devlin et al., 2019), methods such as PoWER-BERT (Goyal et al., 2020), LTP (Kim et al., 2022), and LoT (Kim et al., 2023), make layer-by-layer decisions on whether tokens should be processed or skipped. For encoder-decoder models like T5, methods such as CoDA (Lei et al., 2023) and COLT5 (Ainslie et al., 2023) introduce conditional computation mechanisms for the encoder. They conditionally replace the original attention or FFN with a lightweight adaptor. However, these methods are primarily designed for the encoder portion of the model, as they rely on noncausal routing mechanisms. Consequently, they are not directly applicable to decoder-only causal language models.

While significant progress has been made in encoder-based models, enabling dynamic depth in decoder-only language models remains an understudied area. Mixture of Depth (MoD) (Raposo et al., 2024) and SkipLayer (Zeng et al., 2023) were the first to introduce depth routing in causal language models, demonstrating that a simple router combined with extensive pre-training can train a dynamic depth model from scratch. DLO (Tan et al., 2024) further explored this area by using a layer-expanding and layer-skipping mechanism, showing that supervised fine-tuning can adapt LLaMA-3 into a dynamic depth model. However, these methods require the model to be fully tunable. In this work, we enable dynamic depth in a pre-trained large language model without changing any of its original parameters.

4.2 Layer Pruning

Layer pruning reduces computational overhead by identifying and eliminating redundant layers in neural networks. One prevalent approach analyzes the

Method	MMLU	Hellaswag	Winogrande	GSM8K	BBH	XSum
Vanilla	33.12	11.71	28.26	5.52	12.09	0.71
DiffSkip	35.24	11.93	28.91	5.69	12.45	0.69

Table 4: Throughput (iterations/s) comparison across benchmarks.

similarity between layer inputs and outputs to detect redundancy. Men et al. (2024) measured cosine similarity between transformer block inputs and outputs, pruning modules that exhibit high similarity based on calibration data. He et al. (2024b) conducted a finer-grained analysis by separately evaluating the similarity between inputs and outputs of attention and FFN modules, revealing distinct redundancy patterns across components. Building on this foundation, Razzhigaev et al. (2024) incorporated cosine similarity-based regularization into the pruning process, while Ashkboos et al. (2020) achieved pruning by replacing dense matrices with smaller counterparts.

To overcome the limitations of static pruning approaches, Zhang et al. (2024) developed an iterative framework that greedily removes layers with minimal impact, quantified by the output difference between original and pruned models. He et al. (2024a) took a different approach by employing sequence-level routers to dynamically skip attention or feed-forward modules. Similarly, Yang et al. (2024) reduced layer count by merging adjacent layers that demonstrate high cosine similarity. Moreover, early-exit strategies offer an alternative by halting computation for tokens once they satisfy confidence thresholds. Schuster et al. (2022) advanced this paradigm by training early-exit classifiers to detect local consistency in encoder-decoder architectures. For decoder-only models, Varshney et al. (2024) enhanced early-exit quality by training intermediate layers to enable mid-network decoding, while Elhoushi et al. (2024) combined layer dropout during training with speculative decoding at inference to improve overall robustness.

5 Conclusion

In this paper, we show that large language models inherently possess a capacity for dynamic-depth processing without modifying their original parameters. We introduced **Diff**erential Layer **Skip**ping (**DiffSkip**). This method leverages the self-attention input-output difference as a routing signal, enabling a lightweight router mechanism to decide whether each token's hidden state should undergo or skip FFN transformations. Consequently, computational depth is allocated dynamically based on token difficulty and show interesting phenomenon on various tasks. We hope these findings encourage further research into exploiting the inherent signals and dynamic-depth properties of large language models to create more adaptive and efficient AI systems.

6 Limitations

While our approach reduces computational FLOPs by dynamically skipping Feed-Forward Network (FFN) blocks, it does not achieve speedup on current GPU hardware during the decoding stage. In batched decoding scenarios, tokens can be routed to both FFN and adapter modules within each layer, requiring both to be fetched and executed. The resulting I/O overhead outweighs computational savings, limiting throughput improvements. However, our proposed DiffSkip accelerates the prefilling stage, where computation is the bottleneck. We evaluated throughput (iterations/s) on the paper's benchmarks using 8 A6000 GPUs, a batch size of 8, and 5-shot examples. For multi-token tasks, output length was fixed at 5 tokens for comparison. Results are shown in Table 4.

Despite demonstrating modest throughput gains during prefilling, DiffSkip's router and adapter overhead limits speedup in batched decoding due to I/O constraints. Consequently, it cannot deliver acceleration for continuous generation tasks like GSM8K, BBH, and XSum. Future hardware optimizations—such as enhanced memory management or specialized accelerators—could address these bottlenecks and unlock additional performance gains.

7 Acknowledgements

This work was partially supported by the BioPACIFIC Materials Innovation Platform of the National Science Foundation under Award No. DMR-1933487. We extend our sincere gratitude to Meta (formerly Facebook) for their generous donation of an A100 GPU cluster, which was instrumental in conducting our experiments.

References

Joshua Ainslie, Tao Lei, Michiel de Jong, et al. 2023. Colt5: Faster long-range transformers with conditional computation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5100.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2020. Slicegpt: Compress large language models by deleting rows and columns, 2024. *URL https://arxiv.org/abs/2401.15024*.

Lochan Basyal and Mihir Sanghvi. 2023. Text summarization using large language models: a comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models. *arXiv preprint arXiv:2310.10449*.

Peter Clark, Isaac Cowhey, Oren Etzioni, et al. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, et al. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 12622–12642.

Leo Gao, Jonathan Tow, Baber Abbasi, et al. 2024. A framework for few-shot language model evaluation. *Zenodo*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, et al. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 3690–3699.

Shwai He, Tao Ge, Guoheng Sun, et al. 2024a. Router-tuning: A simple and effective approach for enabling dynamic-depth in transformers. *arXiv preprint arXiv:2410.13184*.

Shwai He, Guoheng Sun, Zheyu Shen, et al. 2024b. What matters in transformers? not all attention is needed. *arXiv preprint arXiv:2406.15786*.

Dan Hendrycks, Collin Burns, Steven Basart, et al. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations*.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, et al. 2023. Camels in a changing climate: Enhancing lm adaptation with tulu 2. *arXiv preprint arXiv:2311.10702*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Sehoon Kim, Sheng Shen, David Thorsley, et al. 2022. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 784–794.

Yeachan Kim, Junho Kim, Jun-Hyung Park, et al. 2023. Leap-of-thought: Accelerating transformers via dynamic token routing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15757–15769.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, et al. 2024. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Tao Lei, Junwen Bai, Siddhartha Brahma, et al. 2023. Conditional adapters: Parameter-efficient transfer learning with fast inference. In *Advances in Neural Information Processing Systems*, volume 36.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liyuan Liu, Chengyu Dong, Xiaodong Liu, et al. 2023. Bridging discrete and backpropagation: Straight-through and beyond. In *Advances in Neural Information Processing Systems*, volume 36.

Xin Men, Mingyu Xu, Qingyu Zhang, et al. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

David Raposo, Samuel Ritter, Blake A. Richards, et al. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*.

Anton Razzhigaev, Matvey Mikhalchuk, Elizaveta Goncharova, Nikolai Gerasimenko, Ivan Oseledets, Denis Dimitrov, and Andrey Kuznetsov. 2024. Your transformer is secretly linear. *arXiv preprint arXiv:2405.12250*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, et al. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 8732–8740.

Tal Schuster, Adam Fisch, Jai Gupta, et al. 2022. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35.

Qi Sun, Marc Pickett, Aakash Kumar Nain, et al. 2024. Transformer layers as painters. *arXiv preprint arXiv:2407.09298*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, et al. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051.

Zhen Tan, Daize Dong, Xinyu Zhao, et al. 2024. Dlo: Dynamic layer operation for efficient vertical scaling of llms. *arXiv preprint arXiv:2407.11030*.

Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, et al. 2024. Investigating acceleration of llama inference by enabling intermediate layer decoding via instruction tuning with 'lite'. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3656–3677.

Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6401–6417.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, et al. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 4791–4800.

Dewen Zeng, Nan Du, Tao Wang, et al. 2023. Learning to skip for language modeling. *arXiv preprint arXiv:2311.15436*.

Yang Zhang, Yawei Li, Xinpeng Wang, et al. 2024. Finercut: Finer-grained interpretable layer pruning for large language models. *arXiv preprint arXiv:2405.18218*.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2023. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675*.