DiffSkip: Differential Layer Skipping in Large Language Models

Anonymous ACL submission

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 whole 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.

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 obtained from a pre-trained model.

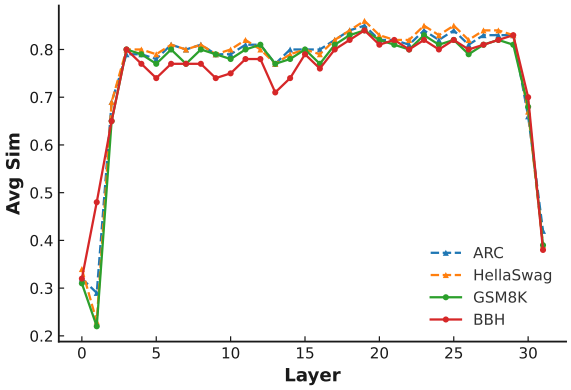

Figure 1: Cosine similarity between Self-attention Input-Output Differences ($\Delta_{Attn} = Attn(x) - x$) and FFN Input-Output Differences ($\Delta_{FFN} = FFN(h) - h$) across different layers in Llama-3-8B-Instruct. The input-output differences of the self-attention and FFN exhibit a high correlation 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. In our experiments, we observe strong correlations between Δ_{Attn} and Δ_{FFN} across various datasets (Figure 1), including single-token generation tasks like ARC (Clark et al., 2018) and HellaSwag (Zellers et al., 2019), as well as multi-token generation tasks such as GSM8K (Cobbe et al., 2021) and BBH (Suzgun et al., 2023). This correlation implies that Δ_{Attn} can be used to predict the transformation that the

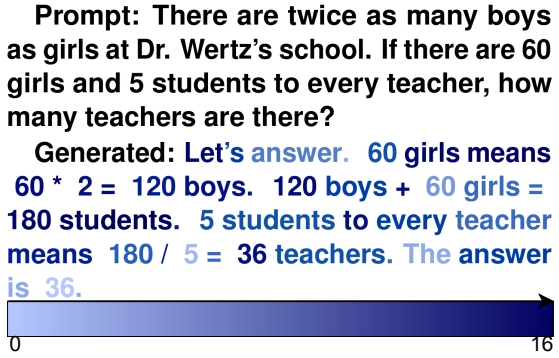

Figure 2: Visualization of token-wise FFN block skipping with DiffSkip. The color gradient from light blue to dark blue indicates the number of FFN blocks utilized during token generation, ranging from 0 to 16.

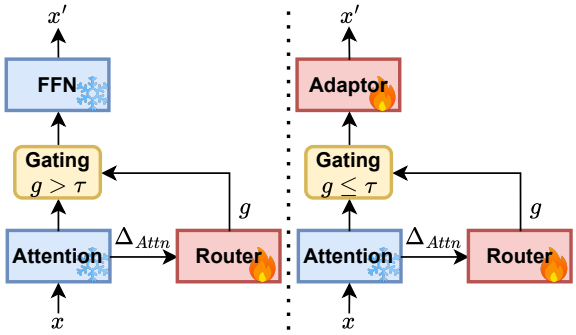

Figure 3: Pipeline for the routing mechanism. The router generates a gating score g based on the self-attention difference Δ_{Attn}. If $g > \tau$, the token representation x is routed to the FFN; if $g \leq \tau$, it is processed by the lightweight adaptor.

FFN would apply, thereby enabling us to determine whether the subsequent FFN transformation is necessary.

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 based on its computational need.

To optimize routing efficiency, we introduce a skipping loss that works in conjunction with the original next-token prediction loss. This loss function is designed to encourage FFN skipping by penalizing the number of FFN blocks utilized. We implement this as an L2 loss on the expected number of blocks preserved per token, which enables routers across different layers to jointly optimize for the skipping objective. In practice, we find that our expectation-based approach leads to improved performance compared to designs that apply penalties to routers independently (Tan et al., 2024; He et al., 2024a). Furthermore, to adapt the skipping strategy to the difficulty of the generation task, we incorporate token-wise weighting into the loss. High-perplexity tokens, indicating harder predictions, receive smaller penalties for preserving FFNs, while low-perplexity tokens, associated with simpler predictions, are encouraged to skip more. Although we adopt next-token perplexity as a lightweight proxy for prediction difficulty in this work, future research could explore more advanced strategies.

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 deep processing. More examples will be discussed in Section 3.4.

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 the lightweight adaptor (Figure 3 right). In the following subsections, we describe the implementation details and the design of our loss function.

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}:

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

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 L2 loss on the expected number of preserved FFN blocks, encouraging routers across different layers to jointly optimize their skipping decisions. The gating scores $G_l = [g_{l1}, g_{l2}, \ldots, g_{lT}]$ for each layer l can be seen as the probability of preserving the FFN block. The expected number of preserved blocks for each token is computed as:

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

where $E(G)$ is a vector of length T, with each element representing the expected number of preserved blocks for the corresponding token. The skipping loss is then defined as the weighted L2 loss of this 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 evaluate our method on diverse benchmarks. For understanding tasks (single token generation), we use 5-shot MMLU (Hendrycks et al., 2021), which consists of multiple-choice questions across STEM and humanities; 5-shot HellaSwag (Zellers et al., 2019), which evaluates commonsense inference through sentence completion; and 5-shot Winogrande (Sakaguchi et al., 2020), which tests commonsense reasoning via Winograd Schema challenges. For sequence generation tasks (multiple tokens), we evaluate on 5-shot GSM8K (Cobbe et al., 2021) for multi-step mathematical word problems, zero-shot BBH (Suzgun et al., 2023) for diverse reasoning tasks ranging from logical deduction to algorithmic thinking, and zero-shot XSum (Narayan et al., 2018) for concise news article summarization. All evaluations are conducted using lm-evaluation-harness (Gao et al., 2024) codebase, with accuracy (acc) for MMLU and Winogrande, normalized accuracy (acc norm) for HellaSwag, exact match for GSM8K and BBH, and ROUGE for XSum. All the datasets are used in accordance with their respective licenses.

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

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.

reduce the total number of layers. For our implementation, we applied the merge operation to reduce k FFN blocks.

- 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 shown in Table 1, while all the methods perform comparably on single-token generation tasks (e.g., MMLU, HellaSwag, and Winogrande), the baseline approaches suffer a significant performance drop on multi-token generation tasks such as GSM8K, BBH, and XSum. In contrast, DiffSkip delivers more consistent performance across both task types. When skipping 4 FFN blocks, our method preserved 100.7% of the original performance on single-token generation tasks and 97.4% on multi-token generation tasks. When skipping 8 FFN blocks, our method preserved 96.6% of the original performance on single-token generation tasks and 86.0% on multi-token generation tasks. The results suggest that single-token generation benchmarks might be more robust to FFN skipping compared to multi-token generation tasks.

3.3 Skipping Pattern Analysis

In this section, we analyze the skipping behavior of DiffSkip across different tasks and model sizes. 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.

DiffSkip performs consistently across different tasks for LLMs trained with a fixed penalty weight. On average, continuous generation tasks (GSM8K, BBH, XSum) skip more FFNs compared to understanding tasks (MMLU, HellaSwag, Winogrande), resulting in larger performance loss.

We find that the skipping patterns vary significantly 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 skip-

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.

ping; 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. In contrast, larger models like Llama-2-13B benefit from FFN skipping. On GSM8K, skipping an average of 14.7 FFN blocks improves performance from 23.6 to 26.5. We hypothesize that Llama-2-13B contains more redundant parameters, allowing it to skip FFN blocks without losing critical information while simultaneously filtering out noisy computations for better performance.

3.4 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: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16. Here, we present three examples generated by Llama-3-8B-Instruct with DiffSkip, using a penalty weight of $\alpha = 1e-3$. For optimal readability, we recommend viewing this section on a screen.

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 suspended 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 con-

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	99.00%
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%
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 Perplexity Weighting	62.5	72.3	74.0	60.2	46.2	12.0	95.0%

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

text, specifically the phrase "Victoria Centre store".

Example 3 The third example is about coding.

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.

3.5 Ablation Studies

Router Input. 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.

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 pro-

vides benefits, the expectation-based L2 penalty plays a more crucial role in maintaining model performance.

4 Limitations

While DiffSkip reduces computational FLOPs by dynamically skipping FFN blocks, it does not achieve speedup on the current GPU hardware. This is because, in batched decoding scenarios, some tokens are routed to FFN while others might go to the adapter within each layer. As a result, both modules need to be fetched and executed. The IO overhead of managing them outweights the computational savings, preventing throughput improvements.

5 Related Works

5.1 Dynamic Computation in Language Models

In encoder-only models like BERT, 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.

5.2 Layer Pruning

Layer pruning aims to reduce computational overhead by identifying and skipping redundant layers. A common approach involves analyzing the similarity between layer inputs and outputs to detect redundancy. For instance, Men et al. (2024) proposed measuring cosine similarity between the input and output of transformer blocks, pruning modules with high similarity based on a calibration dataset. 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. To address limitations in static pruning, Zhang et al. (2024) introduced an iterative pruning framework that greedily removes layers with minimal impact, measured by the difference between outputs of the original and pruned models. He et al. (2024a) used a sequence-level router to skip the Attention or FFN module. Yang et al. (2024) merged adjacent layers with high cosine similarity to reduce the number of layers.

In addition to these methods, early-exit strategies halt further computation for tokens once they meet certain confidence criteria. Schuster et al. (2022) advanced this by training an early-exit classifier to detect local consistency in encoder-decoder architectures. For decoder-only models, Varshney et al. (2024) improved early-exit quality by training intermediate layers to enable decoding in the middle, while Elhoushi et al. (2024) combined layer dropout during training with speculative decoding at inference to enhance robustness.

6 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.

References

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David C. Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, Yun-Hsuan Sung, and Sumit Sanghai. 2023. Colt5: Faster long-range transformers with conditional computation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5085–5100. Association for Computational Linguistics.

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, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

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

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, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12622–12642. Association for Computational Linguistics.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating BERT inference via progressive word-vector elimination. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3690–3699. PMLR.

Shwai He, Tao Ge, Guoheng Sun, Bowei Tian, Xiaoyang Wang, Ang Li, and Dong Yu. 2024a. Router-tuning: A simple and effective approach for enabling dynamic-depth in transformers. *CoRR*, abs/2410.13184.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. 2024b. What matters in transformers? not all attention is needed. *CoRR*, abs/2406.15786.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. Camels in a changing climate: Enhancing lm adaptation with tulu 2. *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, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 784–794. ACM.

Yeachan Kim, Junho Kim, Jun-Hyung Park, Mingyu Lee, and SangKeun Lee. 2023. Leap-of-thought: Accelerating transformers via dynamic token routing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 15757–15769. Association for Computational Linguistics.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Y. Zhao, Yuexin Wu, Bo Li, Yu Zhang, and Ming-Wei Chang. 2023. Conditional adapters: Parameter-efficient transfer learning with fast inference. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

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, Bin Yu, and Jianfeng Gao. 2023. Bridging discrete and backpropagation: Straight-through and beyond. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *CoRR*, abs/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, Brussels, Belgium, October 31 - November 4, 2018*, pages 1797–1807. Association for Computational Linguistics.

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, Timothy P. Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *CoRR*, abs/2404.02258.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2024. Transformer layers as painters. *CoRR*, abs/2407.09298.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.

Zhen Tan, Daize Dong, Xinyu Zhao, Jie Peng, Yu Cheng, and Tianlong Chen. 2024. DLO: dynamic layer operation for efficient vertical scaling of llms. *CoRR*, abs/2407.11030.

Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. 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, Mexico City, Mexico, June 16-21, 2024*, pages 3656–3677. Association for Computational Linguistics.

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, Miami, Florida, USA, November 12-16, 2024*, pages 6401–6417. Association for Computational Linguistics.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. 2023. Learning to skip for language modeling. *CoRR*, abs/2311.15436.

Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and Kenji Kawaguchi. 2024. Finercut: Finer-grained interpretable layer pruning for large language models. *CoRR*, abs/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*.