FiRST: Finetuning Router-Selective Transformers for Input-Adaptive Latency Reduction

Anonymous ACL submission

Abstract

Auto-regressive Large Language Models (LLMs) demonstrate remarkable performance across different domains such as vision and language processing. However, due to sequential processing through a stack of transformer layers, autoregressive decoding faces significant computation/latency challenges, particularly in resource-constrained environments like mobile and edge devices. Existing approaches in literature that aim to improve latency via skipping layers have two distinct flavors - 1) Early exit, and 2) Input-agnostic heuristics where tokens exit at pre-determined layers irrespective of input sequence. Both the above strategies have limitations - the former cannot be applied to handle KV Caching necessary for speed-ups in modern framework and the latter does not capture the variation in layer importance across tasks or more generally, across input sequences. To address both limitations, we propose FiRST, an algorithm that reduces inference latency by using layer-specific routers to select a subset of transformer layers adaptively for each input sequence - the prompt (during the prefill stage) decides which layers will be skipped during decoding. FiRST preserves compatibility with KV caching enabling faster inference while being quality-aware. FiRST is model-agnostic and can be easily enabled on any pre-trained LLM. Our approach reveals that input adaptivity is critical - indeed, different task-specific middle layers play a crucial role in evolving hidden representations depending on tasks. Extensive experiments show that FiRST significantly reduces latency while outperforming other layer selection strategies in quality metics. It retains competitive performance to base model (without layer skipping) and in some cases, even improves upon it. FiRST is thus a promising and efficient solution for LLM deployment in low-resource environments.

1 Introduction

Large Language Models (LLM's) have revolutionized the fields of Natural Language Processing and Computer Vision achieving incredible performance on a diverse set of benchmark tasks. However, the scale of these LLM's characterized by billions of parameters hinder their adoption in resource-constrained environments with memory, latency and compute being the main challenges. In this work, we focus on the latency aspect which becomes the most significant bottleneck for tasks such as machine translation, question answering, summarization particularly on devices, such as laptops and mobile phones. As noted by (Schuster et al., 2022), the auto-regressive nature of decoding in LLM's further pronounces the latency bottleneck.

Transformer based LLMs have several stacks of layers (including attention and FFN layers) leading to high latency and compute requirements, making inference very slow or even infeasible in resource constrained settings. This is because of the sequential processing of tokens through all the layers for *every input sequence and task*. However, it is important to note that in the real world, there is a lot of heterogeneity in input sequences and tasks. (Schuster et al., 2022; Sun et al., 2022) noted that the generations made by LLMs can have varying levels of difficulty and certain generations can be solved with reduced compute, by exiting the transformer stack early. At the same time, it has been noted in recent works (Wendler et al., 2024) that inference forward pass proceeds in phases through the layers of transformer based models, with different types of information being extracted or mapped at different phases (sequences of layers) for certain tasks such as translation. Motivated by these and other related works, we hypothesize that *different sequential combinations of layers are important for different input sequences and tasks*. Learning the right sequential combination of layers can help reduce inference latency and compute for on-device scenarios. However, there are several challenges. Any algorithm for determining the "right" combination of layers should minimize any quality loss, be compatible with other latency reduction strategies such as KV cache handling and be learnable with minimal compute/training overhead.

In the last few years, several promising approaches have been proposed in literature that adaptively prune layers at each decoding step. Token-level early exit proposed in (Schuster et al., 2022; Sun et al., 2022) allow tokens to exit the transformer layer stack early based on different strategies to compute the confidence or saturation level. (Elhoushi et al., 2024; Elbayad et al., 2020; Zhang et al., 2019) extended this idea to incorporate layer skipping at a token level during training. While token level early exit is a useful idea in theory, it suffers from a major limitation of incompatible KV caching in practice (Del Corro et al., 2023). The incompatibility stems from having to recompute KV caches for preceding tokens if we have a delayed exit point for latter tokens, often resulting in loss of early exit advantages. This limits its practical adoption since KV cache is crucial in significantly speeding up auto-regressive decoding.

Recently, (Liu et al., 2024; Del Corro et al., 2023; Song et al., 2024) have proposed input-agnostic layer skipping at token level, that handle KV cache appropriately as well as retain the advantage of adaptive partial computation. In these solutions, tokens exit at pre-determined layers irrespective of the input sequence, and for all sequences in a batch, tokens at the same position in a sequence exit at the same layer. Furthermore, tokens at latter parts of the sequence are constrained to exit earlier than the previous tokens to ensure that there is no redundant KV cache re-computation. These solutions are heuristic based and impose hard rules and constraints irrespective of input sequences, which can lead to drop in output quality. Others (Jaiswal et al., 2024; Chen et al., 2024) have proposed skipping layers by identifying redundant ones through computing cosine similarity of (input/output) representations of a layer. However, their strategy does not take into account that several middle layers are crucial (see (Liu et al., 2024)) and furthermore, final prediction capability of full model is not taken into account while deciding which layers to skip. *Importantly, in none of the works described above, the strategy of selecting layers for skipping is sequence dependent. Furthermore, they do not consider finetuning the models in a way such that not only the performance improves but also the model learns to skip layers appropriately.*

Our goal is to design an input-adaptive **learnable layer selection strategy** with quality aware latency gains that is also able to handle the KV cache appropriately. Ideally, for every input sequence and task, we want to predict the optimal (sequential) combination of layers at inference time, such that quality loss is minimum and the latency gains are as high as possible. We want to do this with expending very little compute/additional training and appropriate handling of KV cache. We propose to do this via training **routers**. Based on the output of each layer for a sequence, a router will decide whether or not to skip the subsequent layer in the transformer architecture. Since the decision is at a sequence level, KV cache issues do not arise, as all tokens in a sequence would pass through the same set of layers. Finally, we fine-tune the model combined with trained routers using LoRA adapters to improve the quality significantly while retaining the latency gains. As an added bonus, LoRA fine-tuning smoothens the layer skipping and further highlights the varied importance of layers based on input sequence. We summarize our contributions:

1. We propose a training and inference algorithm FIRST that incorporates layer-specific routers for selecting layers in an input-adaptive manner. The layer selection is uniform for all tokens in a sequence, thus handling KV caches without introducing additional compute/latency. FIRST can be applied on top of any pre-trained model.
2. We propose LoRA based finetuning on top of router based layer selection to improve quality while retaining latency gains. This also smoothens layer selection.
3. Finally, we demonstrate an extensive set of experiments with FIRST on multiple datasets for 3 distinct tasks namely Machine Translation, Summarization and Question Answering, and 2 different open model architectures namely LLaMA-3-8B and LLaMA-3.2-3B. We show that for the same target speed-up, FIRST significantly improves performance across tasks as compared to baselines and prior works.

Due to space constraints, we delegate a study of other related works and orthogonal approaches (for e.g. model compression) for exploring latency/performance tradeoff to Appendix A.1.

2 Problem Statement

Our goal is to exploit the heterogeneity in inputs and tasks to selectively use LLM layers in a quality-aware manner for reducing inference latency and compute for on-device constraints. Ideally, we want to select an *optimal* sub-sequence of layers within a transformer architecture for a given input and task, such that the overall latency, as well as expended computation, are both low, while quality is comparable to the un-modified case where every input sequence passes through every layer. For ease of explanation, without loss of generality, we assume the task is same and simply consider an input sequence for describing the problem.

Let us consider an an input sequence $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ with n tokens. Let there be m transformer layers in the model, where the i^{th} transformer layer is represented as the function $\phi_i()$. As stated lucidly in (Wendler et al., 2024), $\mathcal{X}$ is first converted to an initial latent representation $\mathcal{H}_0 = \{H_0^1, H_0^2, \ldots, H_0^n\}$, where $H_j^0 \in \mathbb{R}^D, \forall j \in [n]$ is a look-up from a learned embedding dictionary corresponding to the j^{th} token. Thereafter, every transformer layer $\phi_i()$ operates on the latent vectors $\mathcal{H}_i$ to generate the embedding for the i^{th} layer as follows. For the j^{th} token,

$$H_i^j = H_{i-1}^j + \phi_i(H_{i-1}^1, H_{i-1}^2, \ldots, H_{i-1}^j) \quad (1)$$

Let the (gold) output or generated sequence for an input sequence $\mathcal{X}$ that passed through all m layers of the model with full computation be $\mathcal{Y}_{\mathcal{X}}^*$. Our hypothesis is that for a given input sequence (and task), there exists an optimal subsequence of functions $\mathcal{F}_{OPT}(\mathcal{X})$ out of the full sequence $\{\phi_i, i \in [m]\}$ such that the output generated by passing through this subsequence: $\mathcal{Y}_{OPT,\mathcal{X}} \approx \mathcal{Y}_{\mathcal{X}}^*$. More formally, if Q is a quantitative quality measure on $\mathcal{Y}$, and $\epsilon \to 0$ is tolerance in deviation in quality from the gold output, then we hypothesize that there exists an optimal subsequence, using the minimum number of layers, $\mathcal{F}_{OPT}(\mathcal{X})$, such that:

$$Q\left(\mathcal{Y}_{OPT,\mathcal{X}}\right) \geq (1-\epsilon) Q\left(\mathcal{Y}_{\mathcal{X}}^*\right), \forall \mathcal{X}. \quad (2)$$

The optimality above is with respect to the minimum subsequence of layers that can help achieve the above, to minimize latency while keeping quality unaffected. Note that, the optimal subsequence $\mathcal{F}_{OPT}(\mathcal{X})$ need to be obey the same autoregressive computation on previous tokens as given in Equation 1. Hence, any algorithm that determines the optimal subsequence, need to be compatible with KV cache handling, to avoid the re-computation of values for tokens preceding the current token.

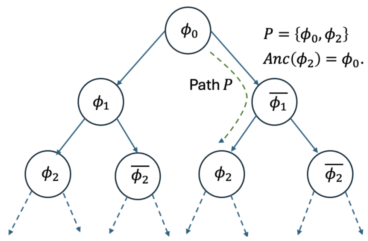

Figure 1: Binary Tree representation of layer selection.

The potential number of subsequences for m layers is 2^m, hence a brute force is infeasible and also beats the purpose of such a layer selection in the first place: reducing latency and compute. In the absence of any known substructure in the behaviour of the latent layers on each input sequence, it is difficult to arrive at the optimal solution polynomially or with low additional latency or compute.

We propose to learn an approximation of the optimal subsequence of layers for any input sequence with low additional latency and minimal training.

3 Proposed Solution: FIRST

Let us first understand what it entails to learn an optimal subsequence of layers for any input. Consider the full transformer sequence to be $\mathcal{F}^* = \{\phi_1, \phi_2, \ldots, \phi_m\}$. Any optimal subsequence for an input $\mathcal{X}$: $\mathcal{F}_{OPT,\mathcal{X}}$ could be thought of as finding an optimal path through a binary tree of functions. Formally, let every level in the binary tree correspond to a transformer layer and the 0^{th} layer corresponds to the initial embedding look up; i.e., at depth $i \in [m]$, there would be 2^i nodes, each corresponding to either ϕ_i or $\overline{\phi_i}$, where the former denotes that a particular transformer layer is included in the optimal path whereas the latter denotes that it is not included. Each (of the 2^{i-1} nodes) ϕ_i or $\overline{\phi_i}$ has two children, corresponding to the next transformer layer: ϕ_{i+1} and $\overline{\phi_{i+1}}$ (See Figure 1). In such a tree structure, for example, the path $\{\phi_i, \overline{\phi_{i+1}}, \phi_{i+2}\}$ indicates the subsequence of transformer layers $\{\phi_i, \phi_{i+2}\}$. For any transformer layer ϕ_i in this tree, let $Anc(\phi_i) = k, 0 \leq k < i$ denote the the lowest ancestor node where the corresponding transformer node ϕ_k is included in the sequence. In the above example, $Anc(\phi_{i+2}) = \phi_i$.

Consider a sequence of functions $\mathcal{F}$, where for level i, $Anc(\phi_i) = \phi_k$. The autoregressive computations for the j^{th} token in the input sequence (originally Eq 1), would now be modified as:

$$H_i^j = \begin{cases} H_k^j, & \text{if } \phi_i \notin \mathcal{F}, \\ H_k^j + \phi_i(H_k^1, H_k^2, \ldots, H_k^j), & \text{if } \phi_i \in \mathcal{F}. \end{cases} \tag{3}$$

Our problem translates to navigating this binary tree to find the optimal path $\mathcal{F}_{OPT}$ for an input sequence and task. Since there are 2^m paths in this tree, we propose to approximate the optimal by making a decision in a greedy fashion at each node. Formally, we add a (lightweight and fast) router R_i before every transformer layer ϕ_i in the model, that will predict whether ϕ_i will be selected or not.

Our aim is to learn to predict the layer choice at a sequence level (not token) to maintain compatibility with the autoregressive computations and avoid re-computation of of KV cache values. Moreover, we should spend minimal compute for learning the R_i functions. Finally, R_i functions should not add any significant latency to the overall computation.

FiRST modifies any off-shelf pre-trained transformer based model by incorporating and training a router or probability function R_i before every transformer layer ϕ_i. The output of R_i is a score ρ_i denoting the probability of selecting ϕ_i in the layer sequence. During inference, ρ_i is rounded to determine selection of ϕ_i. Let $\lfloor \rho_i \rceil = 1$ if $\rho_i \geq 0.5,$ else 0. Equation 1 is now modified as:

$$H_i^j = H_{i-1}^j + \lfloor \rho_i \rceil \cdot \phi_i(H_{i-1}^1, H_{i-1}^2, \ldots, H_{i-1}^j)$$

This recursively approximates Eq 3 for the optimal $\mathcal{F}$ in a probabilistic, greedy manner. We train the functions R_i on datasets and tasks, and further fine tune using LoRA adapters to make the layer selections smooth and improve the output quality.

4 FiRST Framework and Algorithm

In this section, we describe the training and inference frameworks for FiRST in details. We discuss how to train Routers to be adaptive to input sequences. Given an off-the-shelf pre-trained LLM, we propose two training phases. In the first phase, we train a router for each layer that decides whether the input sequence should skip the layer. In the second phase, to tackle the issue of unseen skipping during pre-training, we fine-tune the router-augmented LLM keeping router weights fixed to ensure the model improves performance on the target dataset without reducing the skipping level.

4.1 Adaptive Router Module

The adaptive router module is a single-layer neural network without bias, positioned before every layer in the model. During training of the router, all model parameters except the router weights remain frozen. For the first layer, it takes the tokenized input, and for each of the subsequent layers, it takes the output of the preceding layer as input. Mathematically speaking, for any layer i, given a batch of B tokenized inputs sequences, where each sequence has n tokens and is embedded in to $\mathbb{R}^D$, the adaptive router module takes as input a $B \times n \times D$ tensor output of layer $(i-1)$ and outputs a $B \times n \times 1$ tensor. Subsequently, corresponding to each value (or, token) in the $B \times n \times 1$ tensor, we apply a sigmoid function to ensure that all entries in the tensor are in the interval $[0, 1]$. Following this, we take a mean operation at the sequence level - we take a mean of all the weights in a sequence to output a $B \times 1 \times 1$ tensor. For each sequence in the batch, the corresponding entry is the probability ρ_i with which the sequence passes through the layer i. The input sequence skips the layer i with probability $1 - \rho_i$. During training, the output of a layer is modified using a skip connection, incorporating the probability ρ_i (see Figure: 2).

The routers are trained to encourage skipping by reducing the probabilities $\{\rho_i\}_i$ using a regularizer, to approximate the optimal subsequence for minimizing the latency. The training task is modeled as a language modeling task, specifically next token prediction. The loss function comprises of 3 terms:

- **Cross-entropy loss:** Standard difference between actual and predicted probability distributions to ensure the quality of generation: $\mathcal{L}_{\text{CE}} = -\sum_{x \in \mathcal{X}} \mathcal{Y}^*_{\mathcal{X}} \log(\hat{\mathcal{Y}})$.
- **Regularization loss:** Adds a penalty term to reduce overfitting to noise: $\mathcal{L}_{\text{Reg}} = \sum_{i \in [m]} ||R_i||^2$, where $||R_i||^2$ denotes the ℓ_2 norm of the router weights for the i^{th} layer router, and there are m layers in the model.
- **Non-skip penalization loss:** This is the summation of probability values across all layers of the model architecture: $\mathcal{L}_{\text{PP}} = \sum_{i \in [m]} \rho_i$

The total loss $\mathcal{L}$ is a linear combination of these three terms: $\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \cdot \mathcal{L}_{\text{Reg}} + \alpha \cdot \mathcal{L}_{\text{PP}}$, where α manages the tradeoff between quality and latency.

4.2 LoRA Compensation Module

Skipping layers naturally leads to some performance loss - especially so since the pre-trained

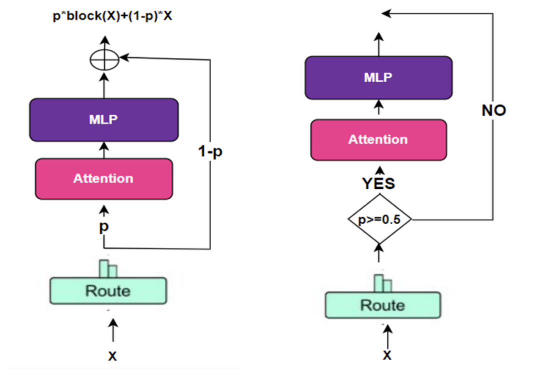

Figure 2: Skip connection used for router training. With probability p, the sequence is processed by the layer and with probability $1-p$, the layer is skipped. During inference, routers make the decision of whether a sequence will skip a particular layer or pass through it.

model was not trained to skip layers. To compensate for the loss in performance caused by skipping layers, we finetune the router-augmented pretrained model on the downstream task [1] using Low Rank Adapters (LoRA). During finetuning, the router parameters are frozen while trainable LoRA adapters are added to both the FFN (Feed-Forward Network) and the attention modules of each layer of the pre-trained model. In order to maintain the skipping level, we again add a non-skip penalization loss component during finetuning with scaling hyper-parameter β. This is essential even though the router weights are frozen because standard finetuning alters the hidden representations of the input sequence in a manner such that no layers are skipped. Note that the LoRA adapters do not lead to any latency overhead during inference.

4.3 Inference for FIRST

During inference, for the input sequence, each router (corresponding to a layer) outputs a number in the interval $[0, 1]$. If this number is greater than or equal to 0.5, the sequence passes through the layer. Otherwise, the sequence skips the layer (Fig. 2). Below, we discuss some salient points about the functioning of the router during inference to handle KV Cache appropriately:

1. **Prefill phase handling:** Skipping is not allowed during prefill phase. This ensures the first token is generated correctly, which is crucial for WMT tasks, as they are highly sensitive to the correct generation of the first token in the target language. It has been observed in prior works (Liu et al., 2024) that skipping during prefill phase is detrimental to performance during inference.
2. **Fixed router decisions during decoding and handling KV Cache:** During the prefill phase, the decisions made by the routers are cached. During the decoding phase, every token adheres to the cached decision made during prefill. In other words, for a particular layer, if a router outputs a number less than 0.5 during prefill, the number is fixed for the decoding steps and therefore the same layer will be skipped by all tokens during decoding. Similarly, if the router outputs a number more than 0.5 during prefill, the same layer will be processing all tokens during decoding. Such a step ensures that for each decoding step and each layer that is not skipped, the KV cache for all previous tokens is available for that layer. This approach effectively addresses the caching issues encountered in early exit strategies, ensuring consistent decisions across the decoding process.

5 Experiments

We conduct experiments on three benchmark tasks namely Machine Translation, Text Summarization and Question Answering demonstrating both robustness and scalability of FIRST.

Datasets: For machine translation, we use WMT development sets (2017–2020) for English-to-Chinese and English-to-German tasks, evaluating performance on the WMT 2022 test set, which covers diverse domains such as news, social media, e-commerce, and conversational contexts. For summarization, we use the CNN/DailyMail dataset, with 4000 randomly selected training samples and evaluation on the standard test set of 11,490 samples. In Question Answering, we utilize SQuAD v1.0, training on 4000 randomly selected samples from the 100k question-answer pairs and evaluating on the validation set, as test set labels are unavailable. Appendix A.2 contains detailed descriptions.

Evaluation Metrics: We use standard metrics to evaluate the quality of generated output in each of the three tasks. For Machine Translation, we benchmark using BLEU scores and COMET where the latter provides a more nuanced assessment beyond n-gram used in BLEU. For Summarization, we use ROUGE scores and BERT Score - as before, the latter captures meaning-based similarity. For Question Answering, we use Exact Match and F1 Score as the metrics to benchmark the output qual-

[1] similar to Quantization Aware Training such as QLoRA (Dettmers et al., 2024) - compensates for model compression

Skip (%)	Model Type		English-to-German			English-to-Chinese		
			BLEU-1	BLEU-2	COMET	BLEU-1	BLEU-2	COMET
0	Original Model	Base + LoRA	**41.78**	**21.74**	**93**	**56.94**	**35.56**	**82.66**
		Base	37.17	18.57	87.13	38.02	22.46	68.95
15	Skip Decode	Router + LoRA	23.04	10.52	55.62	28.73	15.84	55.98
		Router	3.99	1.18	23.33	4.74	2.33	21.75
	Random Skip	Router + LoRA	30.43	10.98	66.25	47.88	25.75	67.32
		Router	26.54	8.77	60.27	36.60	18.66	59.89
	Unified Skip	Router + LoRA	28.92	10.64	59.34	46.61	25.01	**69.58**
		Router	23.23	7.85	59.26	27.28	13.35	54.57
	FiRST (Ours)	Router + LoRA	**38.01**	**17.89**	**82.14**	**48.35**	**26.57**	68.63
		Router	28.83	11.8	67.74	17.55	8.68	42.76
25	Skip Decode	Router + LoRA	13.67	**6.00**	31.47	20.03	10.85	33.85
		Router	3.24	0.92	21.55	3.78	1.84	20.93
	Random Skip	Router + LoRA	6.01	0.91	29.71	11.69	4.66	27.73
		Router	3.65	0.49	29.95	7.37	2.81	35.16
	Unified Skip	Router + LoRA	15.67	3.36	31.69	34.90	**15.75**	50.59
		Router	12.58	2.65	32.15	17.74	7.35	38.74
	FiRST (Ours)	Router + LoRA	**17.84**	4.14	**34.95**	**35.79**	15.66	**56.92**
		Router	9.67	1.37	26.01	11.01	3.23	25.45

Table 1: Machine Translation Results for LLaMA-3-8B: BLEU-1, BLEU-2, and COMET scores are reported for English-to-German and English-to-Chinese tasks across varying skip levels. FiRST consistently achieves the highest BLEU-1 and BLEU-2 at 15% skipping, for both translation directions. Similarly, high BLEU-1 and COMET scores are obtained for 25% skipping rate.

Skip (%)	Model Type		English-to-German			English-to-Chinese		
			BLEU-1	BLEU-2	COMET	BLEU-1	BLEU-2	COMET
0	Original Model	Base + LoRA	**37.57**	**17.43**	**89.72**	**51.81**	**30.04**	**79.13**
		Base	31.19	13.67	81.66	32.10	17.92	61.84
15	Skip Decode	Router + LoRA	23.24	**11.07**	44.58	38.14	21.68	46.70
		Router	9.93	3.89	32.74	9.59	4.84	34.14
	Random Skip	Router + LoRA	18.99	4.78	47.26	38.75	17.39	57.79
		Router	12.11	2.77	36.30	13.56	5.64	35.86
	Unified Skip	Router + LoRA	21.65	5.93	44.72	36.96	17.35	57.10
		Router	16.41	3.99	39.81	22.43	9.37	45.16
	FiRST (Ours)	Router + LoRA	**24.68**	7.13	**60.29**	**45.66**	**23.66**	**67.45**
		Router	16.47	4.11	43.04	22.69	10.92	54.55
25	Skip Decode	Router + LoRA	16.85	**7.62**	32.33	27.81	**15.74**	42.01
		Router	9.14	3.51	27.64	7.01	3.41	29.51
	Random Skip	Router + LoRA	8.95	1.07	30.97	25.12	9.85	**45.53**
		Router	5.32	0.69	27.22	10.07	3.69	31.50
	Unified Skip	Router + LoRA	15.14	2.65	39.81	30.56	12.27	42.30
		Router	10.21	1.62	30.86	15.21	5.51	31.63
	FiRST (Ours)	Router + LoRA	**18.89**	4.06	**45.38**	**32.92**	13.74	41.66
		Router	9.51	1.44	29.08	10.46	3.55	27.83

Table 2: Machine Translation Results for LLaMA-3.2-3B: BLEU-1, BLEU-2, and COMET scores are shown for English-to-German and English-to-Chinese tasks at different skip levels. FiRST consistently achieves the highest BLEU-1 and COMET scores for English-to-German at both skipping percentages and highest BLEU-1 for English-to-Chinese.

ity. Finally, for benchmarking latency, we look at the TPOT (Time Per Output Token). TPOT evaluates the average time taken to produce each output token and is calculated for GPU to gauge overall decoding performance. Appendix A.3 contains a detailed description of evaluation metrics. Additionally, hyper-parameters used during training and inference can be found in Appendix A.4.

5.1 Baselines for comparison

We report the latency improvement and quality numbers relative to the base models (no skipping).

- **Random Skipping:** We skip a set of k layers randomly where k depends on the target speedup.
- **Skip Decode:** We implement Skip Decode (Del Corro et al., 2023) method that features a monotonic decrease in processing layers, enabling later tokens to leverage the computational resources used for earlier ones.
- **Unified Skipping:** This, to the best of our knowledge, is the state-of-the-art method relies on using a heuristic-based strategy for retaining layers at fixed intervals. We replicate the algorithm in (Liu et al., 2024) and compare performance both with and without LoRA fine-tuning across various skipping percentages.

5.2 Experimental Results on Different Tasks

WMT: Our experiments demonstrate that FiRST performs **consistently well** in different machine translation tasks (EN-to-DE, EN-to-ZH) across different models. For LLaMA-3.8B, for 15% skipping, FiRST achieves a latency improvement of 10-12% on TPOT (see Tables 1 and 5) while being competitive to the base model + LoRA (gold reference) in quality. In most cases, it **significantly outperforms** other layer skipping strategies (Random, Skip Decode, Unified Skipping) and in other

Skip (%)	Model Type		EM	F1
0	Original Model	Base + LoRA	**73.93**	**85.99**
		Base	19.46	36.73
10	Skip Decode	R + LoRA	60.14	65.33
		Router	16.38	31.48
	Random Skip	R + LoRA	65.73	80.08
		Router	18.25	33.75
	Unified Skip	R + LoRA	55.54	74.58
		Router	17.39	32.91
	FiRST (Ours)	R + LoRA	**70.85**	**83.61**
		Router	14.58	31.52
20	Skip Decode	R + LoRA	45.00	55.10
		Router	10.68	26.69
	Random Skip	R + LoRA	47.79	66.37
		Router	6.71	22.46
	Unified Skip	R + LoRA	52.87	69.28
		Router	18.18	32.51
	FiRST (Ours)	R + LoRA	**60.60**	**75.49**
		Router	13.21	27.48

Skip (%)	Model Type		EM	F1
0	Original Model	wLoRA	**73.07**	**84.17**
		Base	18.92	37.74
10	Skip Decode	R + LoRA	60.79	75.00
		Router	20.00	31.55
	Random Skip	R + LoRA	64.78	77.27
		Router	13.76	28.59
	Unified Skip	R + LoRA	65.03	77.53
		Router	13.16	32.31
	FiRST	R + LoRA	**69.44**	**81.35**
		Router	12.79	28.37
20	Skip Decode	R + LoRA	**40.12**	40.00
		Router	20.45	37.62
	Random Skip	R + LoRA	11.32	38.34
		Router	6.75	15.51
	Unified Skip	R + LoRA	37.39	52.49
		Router	7.81	18.20
	FiRST	R + LoRA	39.70	**54.59**
		Router	5.52	15.33

Table 3: Quality Analysis on Question Answering (SQuAD v1.1): LLaMA-3-8B (left) and LLaMA-3.2-3B (right). Exact Match (EM) and F1 scores are reported for varying skipping levels. Note that R + LoRA corresponds to Router Augmentation followed by LoRA fine-tuning (in the proposed FiRST framework) and wLoRA stands for Base Model with LoRA fine-tuning.

Skip (%)	Model Type		BERT	R-1	R-L
0	Original Model	wLoRA	**84.87**	**28.46**	**16.99**
		Base	82.29	23.49	14.66
15	Skip Decode	R + LoRA	84.74	22.04	17.54
		Router	82.53	13.68	9.30
	Random Skip	R + LoRA	83.70	24.60	15.01
		Router	81.10	19.64	13.07
	Unified Skip	R + LoRA	84.25	24.35	14.3
		Router	80.3	16.61	10.95
	FiRST (Ours)	R + LoRA	**85.14**	**31.8**	**20.13**
		Router	81.25	20.2	13.01
27	Skip Decode	R + LoRA	79.92	10.67	10.32
		Router	77.27	9.59	7.00
	Random Skip	R + LoRA	76.40	11.45	7.89
		Router	77.45	12.56	9.08
	Unified Skip	R + LoRA	**80.28**	**15.94**	9.89
		Router	77.43	10.97	7.68
	FiRST (Ours)	R + LoRA	77.5	14.65	**10.45**
		Router	75.6	9.39	6.92

Skip (%)	Model Type		BERT	R-1	R-L
0	Original Model	wLoRA	**84.89**	**28.37**	**17.02**
		Base	71.85	19.34	12.00
15	Skip Decode	R + LoRA	**83.20**	21.71	13.74
		Router	80.97	9.74	6.87
	Random Skip	R + LoRA	79.52	20.18	12.10
		Router	68.20	10.10	7.10
	Unified Skip	R + LoRA	81.53	18.89	11.72
		Router	70.01	12.49	8.68
	FiRST (Ours)	R + LoRA	83.17	**26.47**	**16.79**
		Router	70.98	16.47	10.51
24	Skip Decode	R + LoRA	78.55	15.83	6.74
		Router	76.91	13.29	8.86
	Random Skip	R + LoRA	80.00	16.33	10.07
		Router	67.88	8.49	5.96
	Unified Skip	R + LoRA	79.31	15.88	10.69
		Router	68.86	9.17	6.97
	FiRST (Ours)	R + LoRA	**80.25**	**21.28**	**13.89**
		Router	69.17	12.36	8.43

Table 4: Quality Analysis on Summarization (CNN/DM dataset) on LLaMA-3-8B (left) and LLaMA-3.2-3B (right): BERT F1, Rouge-1 and Rouge-L scores are reported for varying skipping levels. Note that R + LoRA corresponds to Router Augmentation followed by LoRA fine-tuning (in the proposed FiRST framework) and wLoRA stands for Base Model with LoRA fine-tuning.

cases, it is comparable in quality. In comparison to the gold output (Base + LoRA), FiRST is $\geq 80\%$ in COMET scores (semantic metric), $\geq 85\%$ in BLEU-1 scores, and $\geq 75\%$ in BLEU-2 scores (syntax metrics) for both EN-to-DE and EN-to-ZH translations. For 25% skipping, FiRST achieves significant improvement in quality over other strategies, in almost all metrics, while achieving $\sim 18\%$ reduction in TPOT.

For LLaMA-3.2-3B, the latency improvement is $\sim 10\%$, with quality scores being significantly higher than other layer skipping strategies (Table 6). It is within $65 - 85\%$ of BLEU-1 and COMET scores (Table 2) for EN-DE and EN-ZH of the gold (LoRA fine-tuned base model).

CNN/DailyMail Dataset: For LLaMA-3-8B, at roughly 15% skipping level, our method **outperforms** the base model + LoRA (Table: 4) while obtaining a 12% improvement in TPOT (Table 5). For LLaMA-3.2-3B, at 15%, the quality is comparable ($\sim 98\%$) to gold and other baselines with 12% improvement in TPOT. At 24%, FiRST is significantly better than other layer skipping strategies, while achieving $> 20\%$ improvement in latency.

SQuAD Dataset: For the LLaMA-3-8B model, FiRST is $> 95\%$ in output quality of gold (base + LoRA) (Table 3), with overall latency gains of 6-16% (Table 7). It is **significantly** better in quality than all other baselines across all metrics for different levels of skipping. For LLaMA-3.2-3B, again FiRST is $> 95\%$ in output quality of gold (base + LoRA) for 10% skipping (Table 3) with gains in latency of 6-16% overall (Table 7) over the LoRA fine-tuned base model. Moreover, it is better than all other layer skipping strategies across all metrics. Detailed results for an additional skipping percentage are provided in Appendix A.8.

Layer-wise Skipping Patterns: Layer-wise skip-

Model Type	∼ Skipping (%)	English-to-German TPOT	English-to-Chinese TPOT
Base + LoRA	0	1x	1x
R + LoRA	15	0.90x	0.88x
R + LoRA	25	0.82x	0.83x
R + LoRA	35	0.69x	0.68x

Model Type	∼Skipping (%)	CNN/DM TPOT
Base + LoRA	0	1x
R + LoRA	15	0.88x
R + LoRA	20	0.81x
R + LoRA	27	0.76x

Table 5: TPOT variation of LLaMA-3-8B on WMT (left) and CNN/DM (right) for FiRST. These values are relative to the LoRA fine-tuned base model. Fine-tuning improves TPOT and quality significantly.

Model Type	∼ Skipping (%)	English-to-German TPOT	English-to-Chinese TPOT
Base + LoRA	0	1x	1x
R + LoRA	15	0.90x	0.91x
R + LoRA	25	0.78x	0.75x
R + LoRA	35	0.69x	0.74x

Model Type	∼Skipping (%)	CNN/DM TPOT
Base + LoRA	0	1x
R + LoRA	15	0.88x
R + LoRA	24	0.79x
R + LoRA	28	0.77x

Table 6: TPOT variation of LLaMA-3.2-3B on WMT (left) and CNN/DM (right) for FiRST. These values are relative to the LoRA fine-tuned base model. Fine-tuning improves TPOT and quality significantly.

Model Type	∼Skipping (%)	SQuAD TPOT
Base + LoRA	0	1x
R + LoRA	15	0.94x
R + LoRA	24	0.83x
R + LoRA	28	0.74x

Model Type	∼Skipping (%)	SQuAD TPOT
Base + LoRA	0	1x
R + LoRA	15	0.94x
R + LoRA	24	0.84x
R + LoRA	28	0.72x

Table 7: TPOT variation of LLaMA-3-8B (left) and LLaMA-3.2-3B (right) on SQuAD dataset for FiRST. These values are relative to the LoRA fine-tuned base model. Fine-tuning improves TPOT and quality significantly.

ping varies significantly across tasks, reflecting the task-specific importance of each layer. For LLaMA-3-8B at a 15% skipping rate, layers 7–9 and 21 are fully skipped in English-to-German, with partial skipping in layer 18. In English-to-Chinese, layers 7–9, 16, and 21 are fully skipped, while layer 20 is partially skipped. For Summarization, layers 20, 22, and 23 are fully skipped, with partial skipping in layers 19, 21, and 26. Some layers are skipped less than 10%, indicating their necessity for specific sequences. This task-specificity is also evident in Question-Answering, where only layer 22 is fully skipped, and skipping patterns depend on the input. Detailed statistics are in Appendix A.7.

Computational overhead of Routers: We also examine the computational overhead introduced by training the routers. This overhead remains minimal, as the router parameters are significantly smaller than the total number of trainable parameters in each model (0.0027% for LLaMA-3-8B and 0.0016% for LLaMA-3.2-3B). The proportion of the total computational time spent on the router operations compared to the entire forward pass of the model is close to 0.3% for both models suggesting their overhead does not significantly affect the efficiency of the model during inference or training.

6 Conclusion

We propose a new framework FiRST for layer selection corresponding to input sequence and task towards reducing latency in a quality aware manner. This is sequence dependent and operates in a KV cache compatible manner. For an optimal skipping rate of around 15%, FiRST achieves 10-20% reduction in latency while being quality neutral (approximately 80% or more in quality metrics compared to base model) on multiple tasks/model architectures such as Machine Translation, Summarization and Question Answering, on well known open source datasets, and some cases, even improving upon base model, while significantly outperfoming other layer selection strategies on most quality metrics.

7 Limitations

FIRST algorithm selects layers in a greedy, myopic way one layer at a time, corresponding to sequences (and tasks). A more optimal way of doing this would be to estimate a subsequence of layers to traverse through instead of one layer at a time. We intend to address this in future work. We would like to select a more optimal subset (subsequence) of layers which will increase the output quality while reducing latency even further.

8 Ethical Concerns

There are no ethical concerns to the best of our knowledge.

References

Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *arXiv preprint arXiv:2306.13649*, 2023.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024.

Xiaodong Chen, Yuxuan Hu, and Jing Zhang. Compressing large language models by streamlining the unimportant layer. *arXiv preprint arXiv:2403.19135*, 2024.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*, 2023.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

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

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/afdec7005cc9f14302cd0474fd0f3c96-Paper.pdf.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.

Ajay Jaiswal, Bodun Hu, Lu Yin, Yeonju Ro, Shiwei Liu, Tianlong Chen, and Aditya Akella. Ffn-skipllm: A hidden gem for autoregressive decoding with adaptive feed forward skipping. *arXiv preprint arXiv:2404.03865*, 2024.

Wenxiang Jiao, Jen tse Huang, Wenxuan Wang, Zhiwei He, Tian Liang, Xing Wang, Shuming Shi, and Zhaopeng Tu. Parrot: Translating during chat using large language models tuned with human translation and feedback, 2023. URL https://arxiv.org/abs/2304.02426.

Tom Kocmi, R. Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark A. Fishel, Thamme Gowda, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Rebecca Knowles, Philipp Koehn, C. Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Michal Novák, Martin Popel, and Mikulas Popovic. Findings of the 2022 conference on machine translation (wmt22). In *Conference on Machine Translation*, pp. 1–45, 2022.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13355–13364, 2024.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade. *arXiv preprint arXiv:2012.14682*, 2020.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*, 2020.

Yijin Liu, Xianfeng Zeng, Fandong Meng, and Jie Zhou. Instruction position matters in sequence generation with large language models, 2023a. URL https://arxiv.org/abs/2308.12097.

Yijin Liu, Fandong Meng, and Jie Zhou. Accelerating inference in large language models with a unified layer skipping strategy. *arXiv preprint arXiv:2404.06954*, 2024.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023b.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023c.

X Ma, G Fang, and X Wang. On the structural pruning of large language models. *NeurIPS, Llm-pruner*, 2023a.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023b.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

Dheeraj Peri, Jhalak Patel, and Josh Park. Deploying quantization-aware trained networks using tensorrt. In *GPU Technology Conference*, 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL http://arxiv.org/abs/1606.05250.

Rajarshi Saha, Varun Srivastava, and Mert Pilanci. Matrix compression via randomized low rank and low precision factorization. *Advances in Neural Information Processing Systems*, 36, 2023.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Shaohuai Shi, Qiang Wang, and Xiaowen Chu. Efficient sparse-dense matrix-matrix multiplication on gpus using the customized sparse storage format. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 19–26. IEEE, 2020.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. Sleb: Streamlining llms through redundancy verification and elimination of transformer blocks. *arXiv preprint arXiv:2402.09025*, 2024.

Tianxiang Sun, Xiangyang Liu, Wei Zhu, Zhichao Geng, Lingling Wu, Yilong He, Yuan Ni, Guotong Xie, Xuanjing Huang, and Xipeng Qiu. A simple hash-based early exiting approach for language understanding and generation. *arXiv preprint arXiv:2203.01670*, 2022.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM international conference on parallel architectures and compilation techniques*, pp. 31–42, 2020.

Chris Wendler, Veniamin Veselovsky, Giovanni Monea, and Robert West. Do llamas work in English? on the latent language of multilingual transformers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15366–15394, 2024. URL https://aclanthology.org/2024.acl-long.820.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*, 2020.

Linfeng Zhang, Zhanhong Tan, Jiebo Song, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Scan: A scalable neural networks framework towards compact and efficient models. *Advances in Neural Information Processing Systems*, 32, 2019.

Yuxin Zhang, Lirui Zhao, Mingbao Lin, Yunyun Sun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse llms. *arXiv preprint arXiv:2310.08915*, 2023.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

Wei Zhu. Leebert: Learned early exit for bert with cross-level optimization. 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. 2968–2980, 2021.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633*, 2023.

A Appendix

A.1 Related Work

Early Exit: Several works have been proposed in the early exit theme (Zhu, 2021; Zhou et al., 2020; Xin et al., 2020; Liu et al., 2020; Li et al., 2020; Hou et al., 2020; Schuster et al., 2022) where adaptive compute is used for different parts of the token sequence. While these approaches have been popular for encoder-only models which processes the entire sequence as a whole, they have faced challenges in generation tasks. The main limitation of these set of techniques are their inability to handle KV caching appropriately which is crucial for multi-fold speed-ups in current LLM architectures. We emphasize that in our work, we assign varying compute to sequences in different batches but within the same sequence, we assign the same compute to every token.

Input Agnostic Heuristics: In Skip Decoding (Del Corro et al., 2023), initial tokens pass through more layers than later ones, contradicting the observation that later tokens are harder to decode (Liu et al., 2024). Additionally, Skip Decoding skips several bottom layers for most tokens, causing undesirable sub-network imbalance. To address this, Unified Layer Skipping (Liu et al., 2024) proposes a discrete skipping strategy that is uniform for all tokens in a sequence. Based on a latency budget, retained layer ids are passed through by all tokens, ensuring KV Cache handling and retaining key layers. However, the limitation of this approach is that skipping is independent of the input sequence. In contrast, early exit strategies adapt layer skipping to the input sequence, offering more flexibility. In (Fan et al., 2019), a method akin to dropout randomly skips layers during training, but this leads to performance decline during the pre-fill stage. FFN-SkipLLM (Jaiswal et al., 2024) constrains skipping to FFN layers to avoid KV Cache issues but fails to fully exploit redundancy as discussed already. (Song et al., 2024) is a very recent work that also explores greedily identifying layers to skip while preserving the model performance on a calibration dataset - however there are two major limitations of this work which are resolved in our paper - (A) first of all, the layer selection strategy is sequence independent although it can be made task-dependent by calibrating on task-specific datapoints - our approach for skipping layers is sequence dependent and is based on the input to a layer (B) SLEB does not explore the impact of fine-tuning on the layers to be skipped. On the other hand, our skipping strategy incorporates the trained router already - intuitively, the knowledge of skipping is transferred during finetuning. (Chen et al., 2024) is another recent work that compresses models by identifying redundant layers - this is done by computing the average similarity between input/output pairs of a layer. However, as outlined in (Song et al., 2024), such an approach suffers from the limitation that it does not take into account the joint association between the layers while skipping multiple layers. Moreover, like (Jaiswal et al., 2024), this work is neither sequence dependent nor takes the final model predictions into account while identifying the layers to skip.

Model Compression and Quantization Aware Training: Orthogonal approaches to explore the latency/memory-performance trade-off in Large Language Models aim to build smaller models that approximate the performance of larger ones with reduced memory and latency costs. Key techniques include: 1) compressing model parameters into fewer bits (Frantar et al., 2022; Lin et al., 2024; Lee et al., 2024; Saha et al., 2023); 2) pruning the network by removing components like attention heads or neurons based on heuristics (Frantar & Alistarh, 2023; Ma et al., 2023b); and 3) distilling the large model into a smaller, faster counterpart (Agarwal et al., 2023; Gu et al., 2024). For further details, we refer to the survey by (Zhu et al., 2023). A significant body of work (Dettmers et al., 2024; Liu et al., 2023b; Peri et al., 2020; Li et al., 2023) has focused on quantization-aware training to reduce memory footprints and mitigate performance loss, starting with QLoRA (Dettmers et al., 2024). In a similar vein, our work proposes fine-tuning router-augmented models to improve layer skipping and reduce performance degradation, as pre-trained models do not account for layer skipping, leading to higher degradation with vanilla skipping.

Network Pruning: Another orthogonal approach to improve the inference speed-up is to prune redundant network weights by zeroing them out. There has been a significant body of work on pruning model weights (Frantar et al., 2022; Frantar & Alistarh, 2023; Sun et al., 2022; Zhang et al., 2023) - most of these works can be categorized into two clusters namely unstructured pruning and structured pruning. In case of unstructured pruning, there is no structure to the inserted zeros and achieving speedups with modern GPU hardware tailored towards dense matrix multiplication is challenging. In fact, more than 90% sparsity is typically required to achieve any significant speedup (Wang, 2020; Shi et al., 2020). Therefore, structured pruning which is more amenable to GPU hardware has become prominent (2:4 pruning and sub-channel pruning). However, realizing desired speedups through these techniques have been difficult (Song et al., 2024). Moreover, several approaches for dynamically deleting entire rows or columns of weight matrices have been proposed (Ma et al., 2023a; Ashkboos et al., 2024; Liu et al., 2023c) to retain dense matrices but two limitations remain - (A) hardware support is extremely limited for realizing speedup gains (B) extensive finetuning is necessary to align the sparsification with linguistic abilities - this is because, such pruning techniques were not observed by the model during pre-training. Finally, note that several prior works (Tang et al., 2024; Oren et al., 2024; Xiao et al., 2023) have imposed (query aware/ query agnostic) sparsity in the KV cache matrices to speed up self-attention mechanism via clever selection of the critical tokens necessary from the KV cache.

A.2 Details of Datasets

Machine Translation: For translation tasks, namely English-to-Chinese and English-to-German, we employ the WMT development sets from 2017 to 2020 for training/fine-tuning following the methodology outlined in previous studies (Liu et al., 2023a; Jiao et al., 2023). Translation performance is evaluated using the test set from the WMT 2022 dataset (Kocmi et al., 2022) which was developed using recent content from diverse domains. These domains include news, social media, e-commerce, and conversational contexts.

(Details in Appendix: A.5, Table: 8). **Summarization:** We use the popular CNN-DailyMail (CNN/DM) (Hermann et al., 2015) dataset which is a large collection (over 300k) of text summarization pairs, created from CNN and Daily Mail news articles. Each datapoint in this dataset comprises of an **article** (the body of the news article with 683 words on average) and the corresponding **highlights** (article summary as written by the article author). While the training set contains more than

287k samples, we have randomly chosen 4k samples for training both routers and LoRA. During training in our framework, the number of trainable parameters is small in both phases - therefore a small subset of data points is sufficient for training.

Inference is performed on the standard test set with 11,490 samples.

Question Answering: We use the popular Stanford Question Answering Dataset (SQuAD v1.0) (Rajpurkar et al., 2016), a widely-used benchmark for machine Question Answering. The dataset consists of over 100k question-answer pairs posed by crowd-workers on a set of over 500 Wikipedia articles. Each sample comprises a **context** (a passage from a Wikipedia article), a **question** (crafted to test comprehension of the passage) and the corresponding **answer** (a text span from the corresponding reading passage). Similarly to the CNN/DM dataset, 4k samples are chosen as random to train both routers and LoRA. The training and validation split contains 87,599 and 10,570 samples respectively. Evaluation is performed on the validation set (Schuster et al., 2022) as the test set labels are not publicly released.

A.3 Evaluation Metrics

Quality-Based Metrics for Translation task:

- **BLEU Score:** BLEU (Bilingual Evaluation Understudy) scores are used to measure the quality of translations. BLEU compares n-grams of the candidate translation to n-grams of the reference translation, providing a score between 0 and 1, with higher scores indicating better translations. In this evaluation, NLTK BLEU is employed, focusing on BLEU-1 and BLEU-2 scores.
- **COMET:** COMET (Cross-lingual Optimized Metric for Evaluation of Translation) is used to assess translation quality further. COMET evaluates translations using a model trained to correlate well with human judgments. Specifically, Unbabel/XCOMET-XL [2] is used in this evaluation. COMET provides a more nuanced assessment of translation quality by considering the intricacies of both source and target languages, beyond the n-gram matching used in BLEU.

Quality based Metrics for Summarization Task:

- **BERTScore:** This metric quantifies semantic similarity between texts by leveraging contextual word embeddings.
  BERTScore captures meaning-based similarity rather than relying on exact word matches, providing a nuanced evaluation of text generation quality.
- **ROUGE:** (Recall-Oriented Understudy for Gisting Evaluation) is a common metric - ROUGE-1 refers to overlap of unigrams between the system summary and reference summary. Similarly, ROUGE-L measures longest matching sequence of words.

Quality based Metrics for Question Answering Task:

- **Exact Match:** This metric measures the percentage of predictions that exactly match the ground truth answer.
- **F1 score:** Since EM is a highly stringent metric, we also report the F1 score which provides a more flexible evaluation of answer prediction. This metric also takes into account near-matches.

A.4 Training and Inference Setup

- **Training settings:** We perform extensive experiments on two models, namely LLaMA-3-8B and LLaMA-3.2-3B from Meta, which consist of 32 and 28 layers, respectively. Training of routers and LoRA adapters is conducted on A100 80GB GPUs, with training/inference is performed in full precision to avoid performance degradation due to quantization. The training process employs our custom loss function and continues for a fixed number of epochs, terminating when the validation loss fails to improve over 4 consecutive steps. The learning rate is set between $1e^{-4}$ and $3e^{-4}$ - a cosine scheduler is used to adjust the learning rate. Gradients are accumulated after 5 steps and the regularization coefficient λ is fixed at 0.01. For LoRA fine-tuning, we employ a rank of 8, a dropout rate of 0.1, and a scaling factor (lora_alpha) of 32. We set the non-skipping penalization regularizer hyper-parameter $\beta = \alpha/3$ after tuning to maintain skipping level - where α is the non-skipping penalization hyperparameter used in the first phase and set according to the target speedup ratio. For translation, the maximum sequence length is set to 128 for router training and 256 for LoRA training. Similarly, for summarization, the maximum sequence length is set to 500 and 700 respectively. For Question Answering, this length is set to 512. Prompts for the different tasks regarding training/inference are shown in Appendix A.6.
- **Inference settings:** For all the tasks, we set the

[2] https://github.com/Unbabel/COMET

temperature to 0.8 and enable top-k sampling over 10 tokens. The maximum number of tokens to be generated is set to 80 for WMT, 200 for CNN/DM and 32 for SQuAD. Caching is turned on during inference.

A.5 Training and testing split

	WMT		Summarization	RC
	Eng-to-German	Eng-to-Chinese	CNN/DM	SQuAD
Train	3505	8983	3400	3400
Validation	876	998	600	600
Test	2038	2038	11490	10570

Table 8: Train-Validation-Test split for WMT, CNN/DM and SQuAD datasets

A.6 Prompt Details

The prompt structures used for both training and inference are as follows:

- For the machine translation task (English-to-German or English-to-Chinese), the following general prompt structure is used to train the routers and during final inference:

```
### Instruction:
   Translate the following sentences from English
to German.

### Input:
    {Text to be translated}

### Response:
```

- For the summarization task (used in CNN/DailyMail dataset), the prompt structure used is:

```
### Instruction:
   Summarize the news article in around 100-200
words.

### Input:
    {Article to be summarized}

### Response:
```

- For the Question Answering task (used in SQuAD dataset), the following prompt structure is utilized:

```
### Instruction:
   Answer the question based on the given passage.

### Passage:
    {context}

### Question:
    {Question to be answered}
### Response:
```

During the training of the LoRA module, task-aware training is applied. The expected translation or summary is appended after the ### Response section, making the model predict the response tokens following the "Response:\n".

A.7 Layer-wise Skipping Statistics

Table 15 present the fraction of sequences that skip a particular block during the task for the LLaMA-3-8B model. If the corresponding cell in a row shows a value of 80.00, it implies that 80% of the sequences skip this block. It is important to note that the decision regarding which block to skip varies across different datasets and tasks. Additionally, partial skipping in some blocks, with varying percentages, suggests that while some sequences consider the layer important, others do not and therefore skip it during the decoding phase.

Figures 3, 4, 5, and 6 depict the distribution of skipped blocks when the model is configured to skip approximately 15% of the layers. These plots demonstrate the variation in block skipping based on the task and dataset, highlighting the differences in block importance. The bar plots further show that the determination of important layers depends not only on the dataset but also on the model architecture. For instance, Figure 3 illustrates that layers 7 and 9 are entirely skipped in LLaMA-3-8b, while they are rarely skipped in LLaMA-3.2-3b.

A.8 Detailed Result Table

Tables 9 and 11 present the detailed results for the LLaMA-3.2-3B model, while Tables 10 and 12 summarize the performance of the LLaMA-3-8B model for an additional skipping percentage on Machine Translation Task. The results are reported using BLEU (BLEU-1, BLEU-2) and COMET metrics, highlighting performance across different skipping percentages. Similarly, Table 13 presents cumulative results for both models reporting BERT F1, ROUGE-1 and ROUGE-L. Lastly, Table 14 presents Exact Match (EM) and F1 scores for both models for three skipping percentage variations.

Skipping (%)	Model Type	Configuration	BLEU-1	BLEU-2	COMET
0	Original Model	Base + LoRA	**37.57**	**17.43**	**89.72**
		Base	31.19	13.67	81.66
15	Skip Decode	Router + LoRA	23.24	**11.07**	44.58
		Router	9.93	3.89	32.74
	Random Skip	Router + LoRA	18.99	4.78	47.26
		Router	12.11	2.77	36.30
	Unified Skip	Router + LoRA	21.65	5.93	44.72
		Router	16.41	3.99	39.81
	FiRST (Ours)	Router + LoRA	**24.68**	7.13	**60.29**
		Router	16.47	4.11	43.04
25	Skip Decode	Router + LoRA	16.85	**7.62**	32.33
		Router	9.14	3.51	27.64
	Random Skip	Router + LoRA	8.95	1.07	30.97
		Router	5.32	0.69	27.22
	Unified Skip	Router + LoRA	15.14	2.65	39.81
		Router	10.21	1.62	30.86
	FiRST (Ours)	Router + LoRA	**18.89**	4.06	**45.38**
		Router	9.51	1.44	29.08
35	Skip Decode	Router + LoRA	1.62	0.36	23.30
		Router	1.71	0.27	19.34
	Random Skip	Router + LoRA	4.72	0.15	25.08
		Router	1.96	0.06	21.18
	Unified Skip	Router + LoRA	1.18	0.05	19.65
		Router	0.90	0.03	20.36
	FiRST (Ours)	Router + LoRA	**9.47**	**1.29**	**27.45**
		Router	5.74	0.56	25.03

Table 9: Machine Translation Results for English to German on LLaMA-3.2-3B: BLEU-1, BLEU-2 and COMET scores for various skipping strategies.

Skipping (%)	Model Type	Configuration	BLEU-1	BLEU-2	COMET
0	Original Model	Base + LoRA	**41.78**	**21.74**	**93**
		Base	37.17	18.57	87.13
15	Skip Decode	Router + LoRA	23.04	10.52	55.62
		Router	3.99	1.18	23.33
	Random Skip	Router + LoRA	30.43	10.98	66.25
		Router	26.54	8.77	60.27
	Unified Skip	Router + LoRA	28.92	10.64	59.34
		Router	23.23	7.85	59.26
	FiRST (Ours)	Router + LoRA	**38.01**	**17.89**	**82.14**
		Router	28.83	11.8	67.74
25	Skip Decode	Router + LoRA	13.67	**6.00**	31.47
		Router	3.24	0.92	21.55
	Random Skip	Router + LoRA	6.01	0.91	29.71
		Router	3.65	0.49	29.95
	Unified Skip	Router + LoRA	15.67	3.36	31.69
		Router	12.58	2.65	32.15
	FiRST (Ours)	Router + LoRA	**17.84**	4.14	**34.95**
		Router	9.67	1.37	26.01
35	Skip Decode	Router + LoRA	5.55	0.33	23.85
		Router	3.03	**0.82**	20.03
	Random Skip	Router + LoRA	1.80	0.12	**25.56**
		Router	1.44	0.06	25.34
	Unified Skip	Router + LoRA	**6.44**	0.77	22.05
		Router	3.92	0.51	22.88
	FiRST (Ours)	Router + LoRA	6.39	0.42	19.96
		Router	3.7	0.14	21.41

Table 10: Machine Translation Results for English to German on LLaMA-3-8B: BLEU-1, BLEU-2 and COMET scores for various skipping strategies.

Skipping (%)	Model Type	Configuration	BLEU-1	BLEU-2	COMET
0	**Original Model**	**Base + LoRA**	**51.81**	**30.04**	**79.13**
		Base	32.10	17.92	61.84
15	**Skip Decode**	**Router + LoRA**	38.14	21.68	46.70
		Router	9.59	4.84	34.14
	Random Skip	**Router + LoRA**	38.75	17.39	57.79
		Router	13.56	5.64	35.86
	Unified Skip	**Router + LoRA**	36.96	17.35	57.10
		Router	22.43	9.37	45.16
	FiRST (Ours)	**Router + LoRA**	**45.66**	**23.66**	**67.45**
		Router	22.69	10.92	54.55
25	**Skip Decode**	**Router + LoRA**	27.81	**15.74**	42.01
		Router	7.01	3.41	29.51
	Random Skip	**Router + LoRA**	25.12	9.85	**45.53**
		Router	10.07	3.69	31.50
	Unified Skip	**Router + LoRA**	30.56	12.27	42.30
		Router	15.21	5.51	31.63
	FiRST (Ours)	**Router + LoRA**	**32.92**	13.74	41.66
		Router	10.46	3.55	27.83
35	**Skip Decode**	**Router + LoRA**	2.84	1.10	22.30
		Router	2.36	1.04	18.32
	Random Skip	**Router + LoRA**	4.17	1.52	**32.81**
		Router	2.11	0.76	23.63
	Unified Skip	**Router + LoRA**	3.26	0.52	19.12
		Router	2.20	0.37	19.82
	FiRST (Ours)	**Router + LoRA**	**20.01**	**6.96**	28.10
		Router	6.54	1.78	23.13

Table 11: Machine Translation Results for English to Chinese on LLaMA-3.2-3B: BLEU-1, BLEU-2 and COMET scores for various skipping strategies.

Skipping (%)	Model Type	Configuration	BLEU-1	BLEU-2	COMET
0	**Original Model**	**Base + LoRA**	**56.94**	**35.56**	**82.66**
		Base	38.02	22.46	68.95
15	**Skip Decode**	**Router + LoRA**	28.73	15.84	55.98
		Router	4.74	2.33	21.75
	Random Skip	**Router + LoRA**	47.88	25.75	67.32
		Router	36.60	18.66	59.89
	Unified Skip	**Router + LoRA**	46.61	25.01	**69.58**
		Router	27.28	13.35	54.57
	FiRST (Ours)	**Router + LoRA**	**48.35**	**26.57**	68.63
		Router	17.55	8.68	42.76
25	**Skip Decode**	**Router + LoRA**	20.03	10.85	33.85
		Router	3.78	1.84	20.93
	Random Skip	**Router + LoRA**	11.69	4.66	27.73
		Router	7.37	2.81	35.16
	Unified Skip	**Router + LoRA**	34.90	**15.75**	50.59
		Router	17.74	7.35	38.74
	FiRST (Ours)	**Router + LoRA**	**35.79**	15.66	**56.92**
		Router	11.01	3.23	25.45
35	**Skip Decode**	**Router + LoRA**	12.24	**6.27**	25.23
		Router	3.64	1.74	22.84
	Random Skip	**Router + LoRA**	7.38	2.19	27.35
		Router	4.47	1.12	**29.46**
	Unified Skip	**Router + LoRA**	7.51	2.10	20.25
		Router	3.87	1.06	21.24
	FiRST (Ours)	**Router + LoRA**	**15.66**	3.95	26.80
		Router	6.13	1.54	22.89

Table 12: Machine Translation Results for English to Chinese on LLaMA-3-8B: BLEU-1, BLEU-2 and COMET scores for various skipping strategies.

Skip (%)	Model Type		BERT	R-1	R-L
0	Original Model	wLoRA	**84.87**	**28.46**	**16.99**
		Base	82.29	23.49	14.66
15	Skip Decode	R + LoRA	84.74	22.04	17.54
		Router	82.53	13.68	9.30
	Random Skip	R + LoRA	83.70	24.60	15.01
		Router	81.10	19.64	13.07
	Unified Skip	R + LoRA	84.25	24.35	14.3
		Router	80.3	16.61	10.95
	FiRST (Ours)	R + LoRA	**85.14**	**31.8**	**20.13**
		Router	81.25	20.2	13.01
20	Skip Decode	R + LoRA	82.57	20.41	14.87
		Router	81.62	13.48	9.19
	Random Skip	R + LoRA	81.39	21.57	13.83
		Router	79.23	15.51	10.93
	Unified Skip	R + LoRA	**82.93**	22.3	13.37
		Router	80.32	16.51	11.15
	FiRST (Ours)	R + LoRA	82.8	**27.65**	**17.84**
		Router	79.32	16.28	10.85
27	Skip Decode	R + LoRA	79.92	10.67	10.32
		Router	77.27	9.59	7.00
	Random Skip	R + LoRA	76.40	11.45	7.89
		Router	77.45	12.56	9.08
	Unified Skip	R + LoRA	**80.28**	**15.94**	9.89
		Router	77.43	10.97	7.68
	FiRST (Ours)	R + LoRA	77.5	14.65	**10.45**
		Router	75.6	9.39	6.92

Skip (%)	Model Type		BERT	R-1	R-L
0	Original Model	wLoRA	**84.89**	**28.37**	**17.02**
		Base	71.85	19.34	12.00
15	Skip Decode	R + LoRA	**83.20**	21.71	13.74
		Router	80.97	9.74	6.87
	Random Skip	R + LoRA	79.52	20.18	12.10
		Router	68.20	10.10	7.10
	Unified Skip	R + LoRA	81.53	18.89	11.72
		Router	70.01	12.49	8.68
	FiRST (Ours)	R + LoRA	83.17	**26.47**	**16.79**
		Router	70.98	16.47	10.51
24	Skip Decode	R + LoRA	78.55	15.83	6.74
		Router	76.91	13.29	8.86
	Random Skip	R + LoRA	80.00	16.33	10.07
		Router	67.88	8.49	5.96
	Unified Skip	R + LoRA	79.31	15.88	10.69
		Router	68.86	9.17	6.97
	FiRST (Ours)	R + LoRA	**80.25**	**21.28**	**13.89**
		Router	69.17	12.36	8.43
28	Skip Decode	R + LoRA	70.69	8.76	6.74
		Router	40.99	2.05	1.23
	Random Skip	R + LoRA	**79.45**	14.69	9.23
		Router	67.48	8.14	5.64
	Unified Skip	R + LoRA	78.57	11.74	7.47
		Router	68.23	8.12	5.66
	FiRST (Ours)	R + LoRA	77.48	**15.98**	**11.14**
		Router	67.14	8.09	6.00

Table 13: Quality Analysis on Summarization (CNN/DM dataset) on LLaMA-3-8B (left) and LLaMA-3.2-3B (right): BERT F1, Rouge-1 and Rouge-L scores are reported for varying skipping levels. Note that R + LoRA corresponds to Router Augmentation followed by LoRA fine-tuning (in the proposed FiRST framework) and wLoRA stands for Base Model with LoRA fine-tuning. FiRST with fine-tuning, improves upon Unified Skipping for all skipping levels on both Rouge-1 and Rouge-L and is competitive on BERT F1.

Skip (%)	Model Type		EM	F1
0	Original Model	wLoRA	**73.93**	**85.99**
		Base	19.46	36.73
10	Skip Decode	R + LoRA	60.14	65.33
		Router	16.38	31.48
	Random Skip	R + LoRA	65.73	80.08
		Router	18.25	33.75
	Unified Skip	R + LoRA	55.54	74.58
		Router	17.39	32.91
	FiRST (Ours)	R + LoRA	**70.85**	**83.61**
		Router	14.58	31.52
20	Skip Decode	R + LoRA	45.00	55.10
		Router	10.68	26.69
	Random Skip	R + LoRA	47.79	66.37
		Router	6.71	22.46
	Unified Skip	R + LoRA	52.87	69.28
		Router	18.18	32.51
	FiRST (Ours)	R + LoRA	**60.60**	**75.49**
		Router	13.21	27.48
30	Skip Decode	R + LoRA	30.77	48.38
		Router	10.67	28.52
	Random Skip	R + LoRA	25.45	42.68
		Router	3.55	15.49
	Unified Skip	R + LoRA	25.61	38.55
		Router	15.19	28.11
	FiRST (Ours)	R + LoRA	**38.20**	**52.68**
		Router	3.64	13.30

Skip (%)	Model Type		EM	F1
0	Original Model	wLoRA	**73.07**	**84.17**
		Base	18.92	37.74
10	Skip Decode	R + LoRA	60.79	75.00
		Router	20.00	31.55
	Random Skip	R + LoRA	64.78	77.27
		Router	13.76	28.59
	Unified Skip	R + LoRA	65.03	77.53
		Router	13.16	32.31
	FiRST	R + LoRA	**69.44**	**81.35**
		Router	12.79	28.37
20	Skip Decode	R + LoRA	**40.12**	40.00
		Router	20.45	37.62
	Random Skip	R + LoRA	11.32	38.34
		Router	6.75	15.51
	Unified Skip	R + LoRA	37.39	52.49
		Router	7.81	18.20
	FiRST	R + LoRA	39.70	**54.59**
		Router	5.52	15.33
30	Skip Decode	R + LoRA	20.68	23.08
		Router	0.78	3.83
	Random Skip	R + LoRA	0.30	8.87
		Router	0.62	5.88
	Unified Skip	R + LoRA	7.39	13.72
		Router	0.37	6.15
	FiRST	R + LoRA	**33.99**	**50.37**
		Router	2.55	10.26

Table 14: SQuAD performance on LLaMA-3-8B (left) and LLaMA-3.2-3B (right): EM (Exact Match) and F1 scores are reported for varying skipping levels. Note that R + LoRA corresponds to Router Augmentation followed by LoRA fine-tuning (in the proposed FiRST framework) and wLoRA stands for Base Model with LoRA fine-tuning.

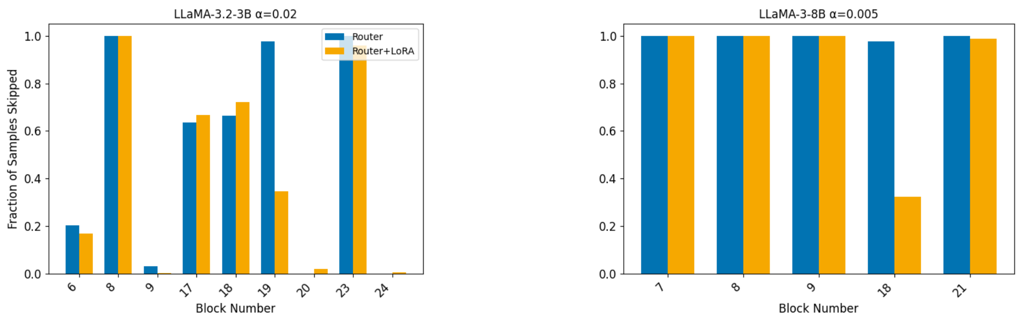

Figure 3: Comparison of LLaMA-3.2-3B (left) and LLaMA-3-8B (right) at 15% skipping rate on English-to-German Machine Translation Task. The graph shows how different layers contribute to the skipping behavior for the same dataset. Layers with no skipping, indicated by a 0% skipping rate, are not represented in the plot.

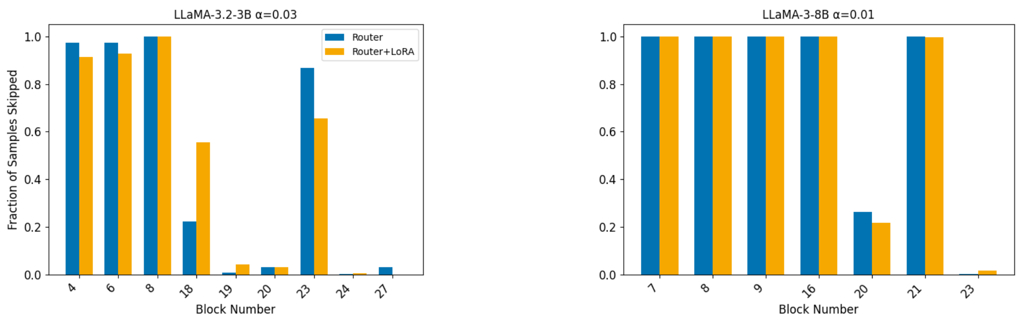

Figure 4: Comparison of LLaMA-3.2-3B (left) and LLaMA-3-8B (right) at 15% skipping rate on English-to-Chinese Machine Translation Task. The graph shows how different layers contribute to the skipping behavior for the same dataset. Layers with no skipping, indicated by a 0% skipping rate, are not represented in the plot.

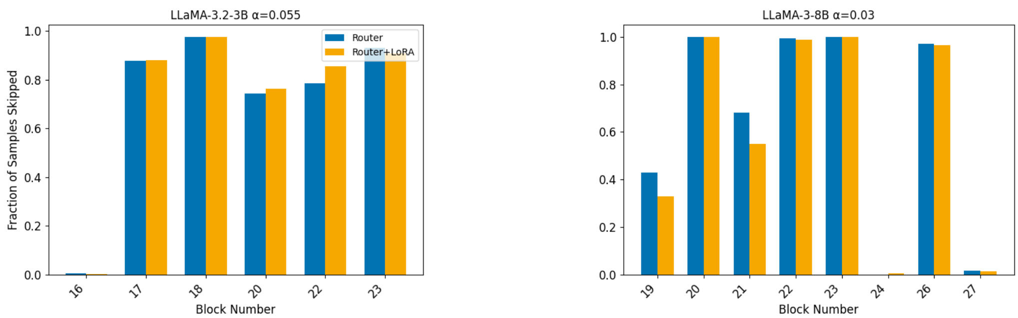

Figure 5: Comparison of LLaMA-3.2-3B (left) and LLaMA-3-8B (right) at 15% skipping rate on CNN Summarization Task. Layers with no skipping, indicated by a 0% skipping rate, are not represented in the plot.

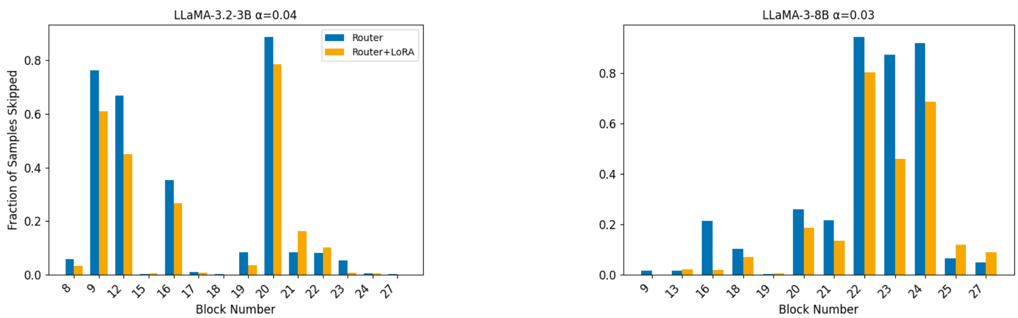

Figure 6: Comparison of LLaMA-3.2-3B (left) and LLaMA-3-8B (right) at 15% skipping rate on SQuAD Question-Answering Task. Layers with no skipping, indicated by a 0% skipping rate, are not represented in the plot.

Layer ↓ α →	**R** **0.005**	**R+L** **0.005**	**R** **0.01**	**R+L** **0.01**	**R** **0.025**	**R+L** **0.025**
0	0.00	0.00	0.00	0.00	0.00	0.00
1	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00
7	100.00	100.00	100.00	100.00	100.00	100.00
8	100.00	100.00	100.00	100.00	100.00	100.00
9	100.00	100.00	0.00	0.00	0.10	1.32
10	0.00	0.00	0.00	0.00	89.25	72.67
11	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	100.00	100.00	100.00	100.00
13	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.59	0.00	0.25
15	0.00	0.00	100.00	99.95	100.00	99.36
16	0.00	0.00	0.10	2.45	100.00	99.95
17	0.00	0.00	0.00	0.00	0.00	0.00
18	97.79	32.24	100.00	100.00	100.00	100.00
19	0.00	0.00	99.85	91.17	100.00	99.61
20	0.00	0.00	100.00	100.00	100.00	99.46
21	99.85	98.72	100.00	100.00	100.00	100.00
22	0.00	0.00	0.00	0.00	0.00	0.00
23	0.00	0.00	24.14	0.79	99.75	91.66
24	0.00	0.00	0.00	0.00	0.00	0.00
25	0.00	0.00	0.00	0.00	0.00	0.00
26	0.00	0.00	57.31	2.45	100.00	86.02
27	0.00	0.00	0.00	0.00	0.00	0.00
28	0.00	0.00	0.00	0.00	0.00	0.05
29	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.00	0.00	0.00	0.00
31	0.00	0.00	0.00	0.00	0.00	0.00
Avg	15.55	13.47	27.54	24.92	37.16	35.95

Table 15: Variation in skipping percentage (15-35%) with the Non-skip Penalization Loss coefficient α for LLaMA-3-8B on Machine Translation (English-to-German). As α increases, the skipping percentage also increases for both the Router-Only and Router-Augmented LoRA fine-tuned models. Similar trends are observed for other datasets as well.