CONDITIONAL TRANSFORMER FINE-TUNING BY ADAPTIVE LAYER SKIPPING

Xingjian Zhang[1]* **Jiaxi Tang**[2] **Yang Liu**[2] **Xinyang Yi**[2] **Li Wei**[2] **Lichan Hong**[2] **Qiaozhu Mei**[1] **Ed H. Chi**[2]
[1] University of Michigan, Ann Arbor [2] Google DeepMind
jimmyzxj@umich.edu
{jiaxit,ylyangliu,xinyang,liwei,lichan}@google.com
qmei@umich.edu edchi@google.com

ABSTRACT

In recent years, deep learning have achieved significant success across various domains, such as natural language processing and computer vision. Despite their advancement, most of the deep neural networks assign uniform computation costs to all inputs regardless of their complexity. Focusing on Transformer architecture, our study addresses this challenge by introducing a sequence-level conditional fine-tuning framework through adaptive layer skipping. The proposed framework dynamically adjusts the computation based on the complexity of input sequence and is tailored for modern accelerators like TPU/GPUs. We examined several measurements on input complexity and found one to be very effective on guiding the conditional computation. The experiment results on synthetic and real-world datasets demonstrate the effectiveness of our methodology by achieving a substantial reduction in training time while maintaining the same predictive performance.

1 INTRODUCTION

Transformer-based models are popular across domains such as NLP and CV, thanks to their exceptional capabilities on fitting complex high-dimensional data through over-parameterization. Despite their popularity, training such giant Transformers are computationally expensive, even for fine-tuning on a small set of supervised data. Like other deep models, Transformers typically assumes the computation cost are the same for all inputs regardless of their complexity, resulting in resource inefficiency. This is unlike human cognition, which is believed to adjust computational effort based on task complexity (Franco et al., 2021; Ragni et al., 2011), suggesting a need for models to similarly adapt their computational cost, based on the input complexity.

Related Work Conditional computation, initially proposed by Bengio et al. (2015), is aimed to tackle with the aforementioned problem. Conditional computation approach dynamically adjusts computational resources by selectively activating a subset of the model's parameters based on some property of the input examples. This approach has also been broadly investigated within Transformer architectures (Xin et al., 2020; Schwartz et al., 2020; Liu et al., 2021; Elbayad et al., 2019; Bapna et al., 2020; Ainslie et al., 2023; Zeng et al., 2023; Wang et al., 2022; 2017; He et al., 2021).

Early exit is a common strategy that jointly trains multiple output classifiers across the Transformer blocks (Xin et al., 2020; Schwartz et al., 2020; Liu et al., 2021; Elbayad et al., 2019; Wang et al., 2022; He et al., 2021). Despite its efficacy during inference, this approach introduces significant training overhead and necessitates the balancing of a weighted sum of loss terms, complicating the training process. Our proposed methodology circumvents the need for training multiple output classifiers and achieves efficiency improvements during training phase.

Token-wise routing emerges as another significant paradigm, facilitating conditional computation at the token level (Ainslie et al., 2023; Zeng et al., 2023). This innovative approach offers a refined allocation of computational resources, customizing the processing depth based on individual tokens. However, these methods requires starting from scratch with pre-training, limiting their applicability to existing pre-trained Transformer models. In contrast, our method is designed to be seamlessly

*Work conducted during an internship at Google DeepMind.

integrated with pre-trained Transformers without the need for re-training. Furthermore, the exploration of arbitrary layer skipping (Wang et al., 2022; 2017) and the application of multi-task learning to dynamically manage computational budgets for input processing (Bapna et al., 2020) represent additional avenues of research.

Contribution In this paper, we explore the quantification of sequence-level input complexities and introduce a novel fine-tuning framework capable of dynamically adjusting computational resource in training time for supervised learning. This is achieved by allowing inputs to forward-pass a variable number of Transformer blocks based on their assessed complexity. Our approach incorporates a practical layer-skipping mechanism that incurs a small overhead. In addition, we identify an accurate measurement for gauging input complexities, which is able to guide the layer-skipping of certain input sequences. The proposed approach significantly reduces the training time by at least 25%, while preserving predictive performance across both synthetic and real-world datasets.

2 Problem Formulation

We consider the standard supervised fine-tuning setting. A pre-trained L-layer Transformer f_θ, where θ is its parameters, trained under objectives like masked language modeling, is adapted to specific downstream tasks, such as sequence classification. A critical limitation in conventional fine-tuning approaches is the uniform computational budget assigned to each training input, overlooking the varying complexities inherent in different inputs. Our hypothesis posits that a careful allocation of computational resources, depending on the complexity of individual sequence inputs, could maintain model performance while significantly reducing overall computation. This hypothesis leads us to the following research questions:

RQ1 How can we achieve sequence-wise conditional computation in a Transformer?

RQ2 What measurement could effectively gauge the complexity of each input sequence, thereby guiding the conditional allocation of computational resources?

Dataset and Task We conduct experiments across both synthetic and real-world datasets:[1]

- *Sequence Classification on ListOps*: We use the ListOps dataset (Nangia and Bowman, 2018) for synthetic data analysis. It challenges NLP models with list operations that require understanding and manipulation of hierarchical data structures. A simple input example is [MAX 2 9 [MIN 4 7] 0] and its label is 9. The complexity of the input sequences are controlled by the depth of parsed tree (2 for the example) and the function space of list operations (e.g. $\{$MAX, MIN$\}$). We report the accuracy for classification.
- *Item Retrieval on MovieLens*: We use the MovieLens dataset (Harper and Konstan, 2015), a real-world dataset widely adopted to benchmark recommendation systems. This dataset, comprising extensive movie watching history from users, facilitates the analysis of models' capacity to predict user preferences accurately. We focus on a standard retrieval task, i.e. given the user's watch history and some demographic features, predict the next movie the user will watch. Following other works (Vančura and Kordík, 2021; Kim and Suh, 2019; Shenbin et al., 2020), we report the recall at 10 and 50 for retrieved items.

Preliminary We work on the Pre-Layer Norm (Xiong et al., 2020) version of the Transformer model, which is a typical choice by many work such as GPTs (Brown et al., 2020; Radford et al., 2019; Raffel et al., 2020; Lieber et al., 2021). Given dataset $\mathcal{D}$ consisting of input token sequences $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and labels y, the token sequences are firstly converted to initial representations $\boldsymbol{h}_0 = (e(x_1), e(x_2), \ldots, e(x_n))$ by an embedding layer $e(\cdot)$. Then, at the l-th block, a Transformer block executes a two-step computational process as follows:

$$\begin{aligned} \boldsymbol{h}'_l &\leftarrow \boldsymbol{h}_{l-1} + \text{MHA}_l(\text{LN}_l^{\text{MHA}}(\boldsymbol{h}_{l-1})) \\ \boldsymbol{h}_l &\leftarrow \boldsymbol{h}'_l + \text{FFN}_l(\text{LN}_l^{\text{FFN}}(\boldsymbol{h}'_l)) \end{aligned} \tag{1}$$

Here, FFN denotes the feed-forward network, MHA represents the multi-head attention mechanism, and LN stands for layer normalization. Additionally, $\boldsymbol{h}_l \in \mathbb{R}^{B\times N\times D}$ is the intermediate representation at the l-th block, where B is batch size, N is (maximum) sequence length, and D is hidden dimension of the Transformer. For sequence classification and retrieval task, the representation of a special token `<cls>` is used for downstream classifier.

[1] Details of the datasets are included in the appendix.

3 RQ1: Sequence-wise Conditional Computation

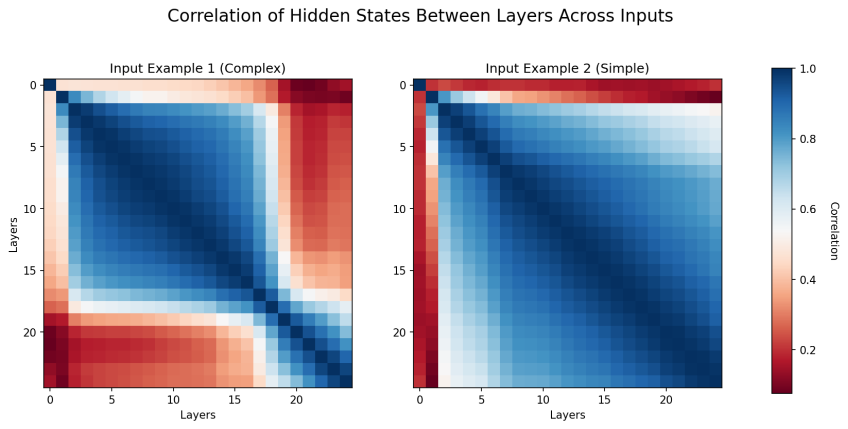

Figure 1: Correlation between representation of the `<cls>` token at different Transformer blocks for a complex input example (left) and a simple input example (right). The representation of a simple input is essentially determined before the computation of the last layer. We use a fine-tuned RoBERTa model to plot this figure on two sequences from ListOps dataset.

The computation within a Pre-Layer Norm Transformer block, as detailed in Eq. 1, can be succinctly expressed as $\boldsymbol{h}_l \leftarrow \boldsymbol{h}_{l-1} + g(\boldsymbol{h}_{l-1})$ for some function g. Here, $g(\boldsymbol{h}_{l-1})$ acts as a minor-scale refinement w.r.t. $\boldsymbol{h}_l$ (Xiong et al., 2020). To minimize computational overhead, a straightforward strategy is to *substitute the Transformer output with an intermediate representation*. This section explores the rationale and methodology for this approach.

Empirical evidence The intuition of this strategy is that the Transformer output are similar to its intermediate representations. Figure 1 shows the correlation of hidden representation of `<cls>` token among all Transformer blocks. We take a complex and a simple input from ListOps (according to the aforementioned complexity criterion) and find high correlations between representations of adjacent blocks. For simple examples, most of the intermediate representations (except the first few) have high similarity (over 0.5) with the output representation. However, this is not the case for complex examples, where the intermediate representations are not similar to the output representation until the last few blocks. This phenomenon is also described in Baldock et al. (2021). Given these observations, we propose the following approach.

Sequence-wise adaptive layer skipping We consider the problem of how many Transformer blocks an input example needs to pass forward, based on the example's complexity. Suppose we have a L-layer Transformer and a function φ that measures the complexity of an input sequence $\boldsymbol{x}$. i.e. $\varphi : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X}$ is the set of all possible sequences. One can simply come up with a heuristic rule γ to classify the numerical input complexity to be the number of Transformer blocks it should pass through, i.e. $\gamma : \mathbb{R} \to \mathcal{L}$ where $\mathcal{L} \subset \{1, \dots, L\}$ can be configured by users. Although versatile and potentially optimal for CPU performance, this approach faces inefficiencies on TPU/GPUs due to variable and conditional computation at runtime. As an alternative for TPU/GPUs, we propose a *ratio-based* rule for a batch of examples to $\gamma_{\mathcal{P}} : \mathbb{R} \to \mathcal{L}$ where $\mathcal{P}$ is a list of portions (sum to 1). For instance, with $\mathcal{P} = (0.5, 0.5)$ and $\mathcal{L} = (12, 24)$, the simpler half of the batch pass through only 12 Transformer blocks while the others pass through all 24 blocks[2]. Essentially, the ratio-based rule compares the complexity of inputs in one batch and filter out more complex examples by a fixed ratio, making the computation graph to be static at runtime.

4 RQ2: Measurement of Sequence Complexity

A key requirement of the proposed method is the knowledge of each sequence's complexity, an aspect that has received limited attention in existing studies to the best of the authors' knowledge.

[2]Therefore, this method is defined by user-specified hyperparameters.

Quantifying sequence complexity Multiple heuristics exist for determining the sequence complexity, as shown in Table 4. Firstly, *Sequence length* indicates the complexity especially for sequences that contains complex reasoning, such as ListOps. This is because longer sequences usually contain more complex structure (Stadler and Neely, 1997). Some other study highlights the importance of capturing unique sequence evolution patterns in retrieval tasks, suggesting a measurement of *sequence diversity* to quantify complexity (Cheng et al., 2016). Lastly, as pointed out by Baldock et al. (2021), *learning difficulty*, the speed at which the model's prediction converges for that input sequences during training, can gauge the sequence complexity. However, one needs to train a model until convergence to obtain the learning difficulty, making this measurement impractical to use. Instead, to reduce the cost, we use *training loss* at a certain step as a proxy.

With adaptive layer skipping and different measurements of sequence complexities, we conduct examples on MovieLens dataset. We fine-tune a 12-layer Transformer[3] and allow the simpler half of the input sequences to exit at the 6th layer. i.e. $\mathcal{P} = (0.5, 0.5)$ and $\mathcal{L} = (6, 12)$. As shown in Table 1, we found using training loss to guide skipping outperforms other heuristics, showing the effectiveness of training loss as an accurate complexity measurement. On the other hand, training loss at an earlier stage is a more effective metric to gauge sequence complexity compared to its counterparts at later stage.

	loss@1k	loss@10k	loss@100k	Sequence diversity	Sequence length	Random
Recall@10	**0.2474**	0.2352	0.2334	0.2235	0.2199	0.2176
Recall@50	**0.5146**	0.4932	0.4891	0.4819	0.4788	0.4748

Table 1: Performance of the proposed method using different measurements of sequence complexity on MovieLens dataset. "loss@1k" denotes the training loss at 1000th step. "Random" is a dummy baseline that uses randomly generated numbers as measurement.

To understand why loss at an early stage works better, we investigate how the example losses change against the training stages. Figure 2 shows the training loss trajectory of a batch of examples in the ListOps dataset, where the colors indicate the example difficulty (red: complex, blue: simple). Losses at early stage (around 1000 steps) are most discriminative for example complexity. In contrast, losses at a later stage (e.g. 10000 steps) may be less informative because it is hard to distinguish simple and medium examples any more and the distribution is pushed to two extremes.

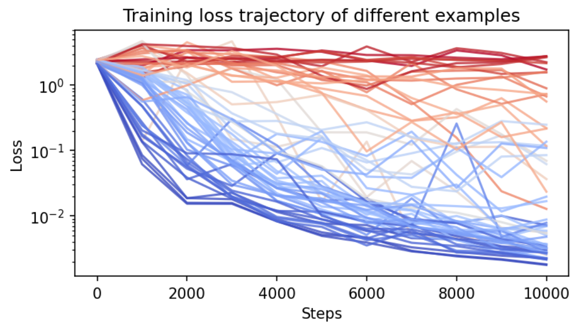

Figure 2: Training loss trajectory of 64 examples in the ListOps dataset. Examples with larger learning difficulty (red) usually has a larger loss at early training step. Loss at early stage (around 1000 steps) are more discriminative. In contrast, it is hard to distinguish simple and medium examples at a later stage (e.g. 10000 steps).

5 Empirical Experiment Results

Baselines and proposed method We run experiments on the following methods:

1. *Full fine-tuning* of all parameters but at a high computation cost.

[3]Please see model details in the appendix.

2. *Drop last k layers* of the pre-trained Transformer before fine-tuning.
3. *Freeze first k layers (and embedding layer)* during fine-tuning.
4. *Skipping by loss@1k*: Proposed adaptive layer skipping configured by $\mathcal{P}$ and $\mathcal{L}$.

Results Table 2 and 3 shows the experiment results on the two datasets. On both ListOps and MovieLens, our proposed method achieves better performance than baselines under the same or even less amount of training time, suggesting the effectiveness of our method in performance-cost trade-off. Surprisingly, the proposed method has better performance than full fine-tuning with less computation cost. A potential explanation is that the adaptive skipping essentially divides the model into multiple "experts" so that the deeper blocks learn to specialize in encoding more complex examples.

Method	Full fine-tuning	Drop last k		Freeze first k		Skip by loss@1k
Configuration		$k = 12$	$k = 6$	$k = 12$	$k = 18$	$\mathcal{P} = (0.5, 0.5), \mathcal{L} = (12, 24)$
Accuracy	0.8918	0.8697	0.8821	0.8477	0.7884	**0.8991**
Step-time (ms)	1664	849	1257	1173	924	1297

Table 2: Performance of proposed method and baselines on ListOps dataset by a 24-Layer Transformer. We run experiments for 3 random seeds and omit the standard error since the standard error of accuracy is less than 0.001. This holds true for all following experiments.

Method	Size	Configuration	Recall@10	Recall@50	Step-time (ms)
Full fine-tuning	$L = 8, D = 64$		0.1963	0.4570	45.8
	$L = 12, D = 128$		0.2346	0.4892	120.4
	$L = 24, D = 256$		0.2633	0.5055	511.9
Drop last k	$L = 12, D = 128$	$k = 6$	0.2123	0.4705	65.3
	$L = 12, D = 128$	$k = 3$	0.2264	0.4827	92.0
	$L = 24, D = 256$	$k = 12$	0.2353	0.4855	244.7
	$L = 24, D = 256$	$k = 6$	0.2370	0.4834	345.7
Freeze first k	$L = 12, D = 128$	$k = 9$	0.1922	0.4500	61.2
	$L = 12, D = 128$	$k = 6$	0.2168	0.4737	81.1
	$L = 24, D = 256$	$k = 18$	0.2287	0.4803	225.2
	$L = 24, D = 256$	$k = 12$	0.2439	0.4913	336.7
Skip by loss@1k	$L = 12, D = 128$	$\mathcal{P} = (0.5, 0.25, 0.25)$ $\mathcal{L} = (4, 8, 12)$	0.2421	0.5206	82.8
	$L = 12, D = 128$	$\mathcal{P} = (0.5, 0.5)$ $\mathcal{L} = (6, 12)$	0.2474	0.5146	101.0
	$L = 24, D = 256$	$\mathcal{P} = (0.5, 0.25, 0.25)$ $\mathcal{L} = (8, 16, 24)$	0.2716	0.5395	314.3
	$L = 24, D = 256$	$\mathcal{P} = (0.5, 0.5)$ $\mathcal{L} = (12, 24)$	0.2759	0.5328	370.9

Table 3: Performance of proposed method and baselines on MovieLens dataset. Model sizes are specified in appendix. The loss@1k labels are obtained by a 8-layer Transformer.

6 Conclusion and Limitation

Conclusion In this work, we propose a novel framework to allow sequence-wise conditional computation in Transformers through adaptive layer skipping. We develop an innovative measurement of input sequence complexity that can guide the computation allocation. Experiment results on both synthetic and real-world datasets demonstrate the effectiveness of our method by a substantial 25% reduction in training time while achieving a strong predictive performance.

Limitation and future work A major limitation of our proposed method is the unavailability of the complexity measure "loss@1k" during inference, as it requires the label y to compute the training loss, which is typically inaccessible. To circumvent this, we suggest developing a smaller predictor to estimate complexity measurements, facilitating layer skipping guidance. Moreover, our fixed ratio-based skipping strategy assumes a relatively big batch size to ensure consistent skipping for the same input across various epochs. A more practical way is to adapt the ratio-based skipping into a threshold-based method by setting quantile values of complexity as thresholds. We will address these considerations in future work.

References

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, Yun-Hsuan Sung, and Sumit Sanghai. 2023. CoLT5: Faster Long-Range Transformers with Conditional Computation. (March 2023). arXiv:2303.09752 [cs.CL]

Robert J N Baldock, Hartmut Maennel, and Behnam Neyshabur. 2021. Deep learning through the lens of example difficulty. (June 2021). arXiv:2106.09647 [cs.LG]

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2020. Controlling Computation versus Quality for Neural Sequence Models. (Feb. 2020). arXiv:2002.07106 [cs.LG]

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional Computation in Neural Networks for faster models. (Nov. 2015). arXiv:1511.06297 [cs.LG]

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. (May 2020). arXiv:2005.14165 [cs.CL]

Weijie Cheng, Guisheng Yin, Yuxin Dong, Hongbin Dong, and Wansong Zhang. 2016. Collaborative Filtering Recommendation on Users' Interest Sequences. *PLoS One* 11, 5 (May 2016), e0155739.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-Adaptive Transformer. (Oct. 2019). arXiv:1910.10073 [cs.CL]

Juan P. Franco, Karlo Doroc, Nitin Yadav, Peter Bossaerts, and Carsten Murawski. 2021. Task-independent metrics of computational hardness predict performance of human problem-solving. *bioRxiv* (2021). `https://doi.org/10.1101/2021.04.25.441300`

F Maxwell Harper and Joseph A Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (Dec. 2015), 1–19.

Xuanli He, Iman Keivanloo, Yi Xu, Xiang He, Belinda Zeng, Santosh Rajagopalan, and Trishul Chilimbi. 2021. Magic Pyramid: Accelerating inference with early exiting and token pruning. (Oct. 2021). arXiv:2111.00230 [cs.CL]

Daeryong Kim and Bongwon Suh. 2019. Enhancing VAEs for collaborative filtering: flexible priors & gating mechanisms. In *Proceedings of the 13th ACM conference on recommender systems*. 403–407.

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. 2021. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs* 1 (2021), 9.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. 2021. Faster Depth-Adaptive Transformers. *AAAI* 35, 15 (May 2021), 13424–13432.

Nikita Nangia and Samuel R Bowman. 2018. ListOps: A Diagnostic Dataset for Latent Tree Learning. (April 2018). arXiv:1804.06028 [cs.CL]

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21, 1 (Jan. 2020), 5485–5551.

Marco Ragni, Felix Steffenhagen, and T. Fangmeier. 2011. A Structural Complexity Measure for Predicting Human Planning Performance. *Cognitive Science* 33 (2011), 2353–2358.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A Smith. 2020. The Right Tool for the Job: Matching Model and Instance Complexities. (April 2020). arXiv:2004.07453 [cs.CL]

Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I Nikolenko. 2020. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In *Proceedings of the 13th international conference on web search and data mining*. 528–536.

Michael Stadler and Craig B Neely. 1997. Effects of sequence length and structure on implicit serial learning. *Psychol. Res.* 60, 1 (June 1997), 14–23.

Vojtěch Vančura and Pavel Kordík. 2021. Deep variational autoencoder with shallow parallel path for top-N recommendation (VASP). In *International Conference on Artificial Neural Networks*. Springer, 138–149.

Jue Wang, Ke Chen, Gang Chen, Lidan Shou, and Julian McAuley. 2022. SkipBERT: Efficient Inference with Shallow Layer Skipping. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 7287–7301.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2017. SkipNet: Learning Dynamic Routing in Convolutional Networks. (Nov. 2017). arXiv:1711.09485 [cs.CV]

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. (April 2020). arXiv:2004.12993 [cs.CL]

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. 2020. On Layer Normalization in the Transformer Architecture. (Feb. 2020). arXiv:2002.04745 [cs.LG]

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. 2023. Learning to skip for language modeling. (Nov. 2023). arXiv:2311.15436 [cs.CL]

A Complexity Measurements

Measurement	Complexity function φ
Sequence length	$\mathrm{len}\,(\boldsymbol{x})$
Sequence diversity	$\frac{2}{n(n-1)} \sum_{1\le i<j\le n} e(x_i)^\top e(x_j)$
Loss at step t	$\mathrm{loss}(f_{\theta_t}(\boldsymbol{x}), y)$

Table 4: Measurements of sequence complexities. $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ denotes the input sequence. $e(\cdot)$ is the embedding function obtained by a pre-trained Transformer. f_{θ_t} is a fine-tuned Transformer after t steps of training. y is the label corresponding to $\boldsymbol{x}$.

B Experiment Details

B.1 ListOps

Dataset For the dataset corpora, we directly use the ListOps dataset release from `https://github.com/nyu-mll/spinn/blob/master/python/spinn/data/listops/data_d20.txt`. The maximum list depth is constrained to 20 and four list operations are allowed: {MIN, MAX, FIRST, LAST}. We use 90,000 data points as training set and 10,000 data points as testing set.

Model We use the pre-trained Transformer checkpoints from the HuggingFace model hub. In particular, we select the RoBERTa checkpoint `andreasmadsen/efficient_mlm_m0.40` as it is the most downloaded pre-LN Transformer checkpoint on the HuggingFace model hub. The model has 24 layers, 16 attention heads, and 1024 hidden dimensions. All experiments are run on a single NVidia A40 GPU. We train up to 5 epochs with learning rate of $1 \cdot 10^{-5}$ and weight decay of $1 \cdot 10^{-3}$. We fix the maximum length of 500 for the input sequences. A standard linear classification head is attached to the Transformer encoder. The proposed measurements loss@1k is obtained by fine-tuning a 12 layer RoBERTa checkpoint for 1000 steps. For our proposed method, we apply the same adaptive layer skipping in testings sets. The implementation are based on TensorFlow and HuggingFace libraries.

B.2 MovieLens

Dataset We use the standard MovieLens dataset for our training corpora and convert the user interactions into sequence data. The dataset contains 25,000,000 data points (sequences). And we use 20,000,000 for training and 5,000,000 for testing. For sequences larger than maximum sequence length (200), we preserve the latest sub-sequence as inputs.

Size	Layers	Heads	Hidden Dimensions
M	8	4	64
L	12	8	128
XL	24	16	256

Table 5: Three sizes are used in our experiments on the MovieLens dataset.

Model We pre-train standard Transformers of three different sizes for our experiments on MovieLens. The sizes of them are specified in Table 5. All experiments are run on 32 TPUs with a total batch size of 8192. We train up to 100,000 steps with learning rate of $3 \cdot 10^{-5}$ and fix a maximum length of 200 for the input sequences. The pre-training objective is standard MLM objective, with a mask ratio of 0.15. For downstream retrieval task, we train up to 300,000 steps with learning rate of $3 \cdot 10^{-5}$ and also fixes a maximum length of 200 for the input sequences. The proposed measurement loss@1k is obtained by fine-tuning a size-M Transformer for 1000 steps. For our proposed method, we also apply the same adaptive layer skipping in testings sets. A two layer MLP retrieval tower is attached to the Transformer encoder. The implementation are based on TensorFlow.

C Extra Results

	loss@1k	loss@2k	loss@3k	Sequence length	Random
Method	XL-12(.5)	XL-12(.5)	XL-12(.5)	XL-12(.5)	XL-12(.5)
Accuracy	**0.8991**	0.8882	0.8860	0.8770	0.8733

Table 6: Performance of the proposed method using different measurements of sequence complexity on ListOps dataset. “loss@1k” denotes the training loss at 1000th step. “Random” is a dummy baseline that uses randomly generated numbers as measurement.