Efficient Sparsely Activated Transformers

Salar Latifi [1] **Saurav Muralidharan** [2] **Michael Garland** [2]

Abstract

Transformer-based neural networks have achieved state-of-the-art task performance in a number of machine learning domains including natural language processing and computer vision. To further improve their accuracy, recent work has explored the integration of dynamic behavior into these networks in the form of mixture-of-expert (MoE) layers. In this paper, we explore the introduction of such layers to optimize a different metric: inference latency. We introduce a novel system named PLANER that takes an existing Transformer-based network and a user-defined latency target and produces an optimized, sparsely-activated version of the original network that tries to meet the latency target while maintaining baseline accuracy. We evaluate PLANER on two real-world language modeling tasks using the Transformer-XL network and achieve inference latency reductions of over 2x at iso-accuracy.

1. Introduction

Attention-based deep neural networks (DNNs) such as Transformer (Vaswani et al., 2017) and BERT (Devlin et al., 2018) have been shown to exhibit state-of-the-art performance across a variety of machine learning domains, including natural language processing (Wolf et al., 2020) and computer vision (Dosovitskiy et al., 2020). Due to their size and complexity, they are expensive to train and deploy, especially on resource-constrained hardware. In particular, attention layers, which form the building blocks of such networks, account for the majority of network runtime. From our own independent experiments, we observe that attention layers account for over 80% of total inference latency for the Transformer-XL network (Dai et al., 2019) on two separate NVIDIA GPUs: V100 and A100. Due to their outsize influence on total inference latency, recent work has explored various approaches for runtime performance optimization that specifically target attention layers; this includes works such as PAR Transformer (Mandava et al., 2020) that redistribute attention layers within the network to optimize performance, and various papers on pruning either attention heads and/or entire attention layers (Wang et al., 2020).

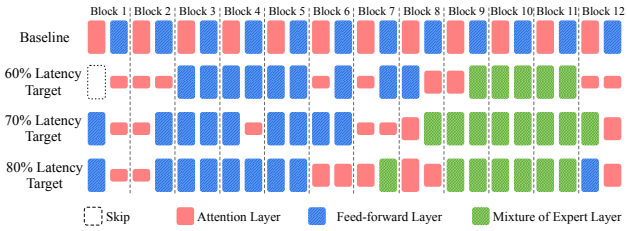


Figure 1: Exploration results for Transformer-XL Base model on enwik8 dataset for different latency targets.

A separate body of work has explored the addition of sparsely activated layers to Transformer models to improve task performance (Shazeer et al., 2017). In particular, mixture-of-expert (MoE) Transformer variants such as Switch Transformer (Fedus et al., 2021) have demonstrated state-of-the-art task performance while simultaneously improving training and inference costs. While most work in this direction has focused on improving task accuracy, in this paper we attempt to answer the following question: *can the addition of sparsely activated layers help **preserve** accuracy in the face of latency-optimizing neural transformations such as skipping/pruning attention layers? And if so, to what extent?*.

To help answer this question, we present PLANER, a novel system for designing latency-aware sparsely activated Transformer networks. Given a Transformer-based model as input, along with an inference latency target expressed as a percentage of the baseline model's latency, PLANER produces a sparsely-activated Transformer model that fulfills the latency objective while preserving baseline accuracy. PLANER employs an efficient two-phase gradient descent-based neural architecture search (NAS) strategy with a dynamic loss formulation to achieve this. During the search process, PLANER efficiently explores the large number of

[1]Department of Computer Science and Engineering, University of Michigan, Ann Arbor, USA [2]NVIDIA Corporation, Santa Clara, USA. Correspondence to: Salar Latifi <salar@umich.edu>, Saurav Muralidharan <sauravm@nvidia.com>, Michael Garland <mgarland@nvidia.com>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

alternative architectures arising from different combinations of feed-forward, attention (with varying number of heads), and mixture-of-expert layers; as a concrete example, PLANER considers over 68 billion unique architectures for the Transformer-XL model in our evaluation. The optimized architecture obtained from NAS is then fine-tuned using a load-balancing loss term to produce the final network. Figure 1 demonstrates how PLANER infers different architectures depending on the user-provided inference latency targets. Here, each of the inferred architectures matches baseline accuracy, but has different inference latencies. Depending on the latency target, we notice that PLANER progressively reduces the number of attention layers and their widths, while using additional MoE and/or feed forward layers to compensate for potential accuracy drops.

We evaluate PLANER on two different Transformer-based networks drawn from language modeling, and demonstrate an inference latency reduction of at least $2\times$ for each network while maintaining baseline accuracy. We also compare PLANER with prior work such as PAR Transformer (Mandava et al., 2020) and Sandwich Transformer (Press et al., 2019), and with parameter-matched non-MoE implementations of the final optimized networks.

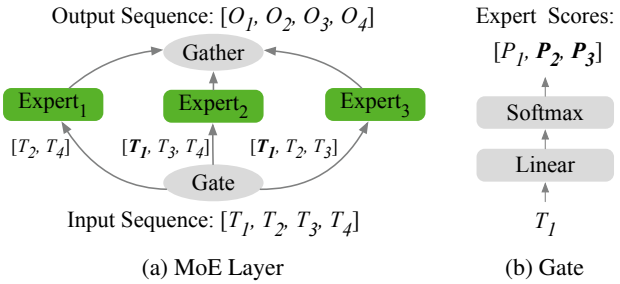
2. Searching for Efficient Transformers

We provide a brief overview of MoEs in this Section, followed by a deep dive into PLANER's 2-phase NAS methodology for finding optimal latency-aware Transformers.

2.1. Background: MoE Networks

Mixture-of-expert (MoE) networks (Masoudnia & Ebrahimpour, 2014) dynamically partition the input domain so that each sub-network or "expert" specializes in one or more input partitions, yielding a sparsely activated network. Recent work has explored the application of MoE layers to efficiently increase the model capacity of Transformer-based architectures (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2021; He et al., 2021). These sparsely-activated architectures are shown to achieve similar accuracy gains without the proportional increase in the computation compared to the traditional scaling of the network parameters (Raffel et al., 2019). In this work, we focus on applying MoE layers to improve inference latency while maintaining baseline accuracy.

Figure 2a depicts a general implementation of an MoE layer with three experts. The sequence of input tokens are distributed among the experts for processing. Each token could be processed either by one or more experts. The number of experts per token is denoted as Top_K in this work. In Figure 2a, Top_K is equal to two. A single-layer linear classifier called a *Gate* (Figure 2b) decides which expert(s) to use to process a specific token. The Gate generates a probability distribution across the experts per token, which will then be used to select the Top_K experts.



(a) MoE Layer (b) Gate

Figure 2: General overview of MoE layers and gate function.

2.2. Phase 1: Exploring the Search Space

Transformer-based models are composed of multiple blocks, where each block consists of multi-head attention (MHA) and feed-forward layers (FFLs) (Vaswani et al., 2017). MoEs could thus be applied to either MHA or FFLs, or both. In this work, we only explore MoE FFLs in the design space; this is primarily due to the runtime overhead introduced by dynamic behavior, which we found to be prohibitively high for the already expensive attention layers. PLANER's first phase explores the large design space composed of different configurations of MHAs, FFLs, and MoE layers. The inputs to the first phase are the design space, the backbone of the baseline network architecture, and a target latency, expressed as a ratio w.r.t. the baseline latency.

For real-world networks, the design space of alternative architectures often gets prohibitively large; for instance, the Transformer-XL Base network on the enwik8 dataset yields a search space size of over 68 billion architectures. To keep the search tractable, we deploy a differentiable NAS strategy, which has been shown to be significantly more efficient than reinforcement-learning-based approaches (Zoph & Le, 2016). We follow a NAS algorithm similar to the one proposed by Wu et al. (2019).

Phase 1 first composes a *search architecture* using the baseline network's backbone. The backbone includes details on the number of blocks (MHA or FFLs) and their configuration (number of heads or hidden dimension). Using the input backbone, each of the MHA or FFL blocks in the baseline network are replaced with *Super Blocks (SB)*, which includes all the search options in the design space. The goal is to find the best option for each block so that overall accuracy is maximized and the latency target is achieved. Figure 3 depicts the formulation of super blocks. Each of the search options $Block_i$ is accompanied by corresponding architectural weights α_i, which are trained using gradient de-

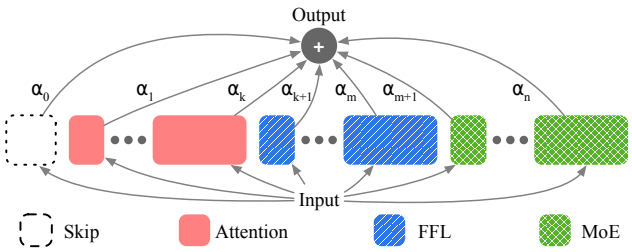


Figure 3: Formulating super blocks from the search space.

scent to represent the benefit factor of the search option (Wu et al., 2019). To make the optimization graph differentiable with respect to the architecture weights, the output of the super block is formulated as:

$$Output = \sum_{i=0}^{n} P_i \times Block_i(Input)$$
$$s.t. \quad P_i = GumbelSoftmax(\alpha_i, [\alpha_0, ..., \alpha_n]) \quad (1)$$

Where the $GumbelSoftmax$ generates probability values by sampling the Gumbel distribution based on α weights.

This formulation yields two sets of parameters to be trained in Phase 1. The first group are the actual network weights ($Block_i$), and the second set are the architectural weights (α_i). Training of each parameter group is done sequentially in each epoch, using separate optimizers.

To incorporate latency optimization in the search phase, an auxiliary loss is formulated based on the latencies of the search and baseline network, as well as the target latency. We use an estimation for the end-to-end latency of the search network as well as baseline in phase 1, using lookup tables filled with individual block latencies similar to prior work (Wu et al., 2019). Equation (2) presents the formulation for the estimated latency which is composed of accumulating the latencies of each super block (Lat_SB).

$$Lat = \sum_{b=0}^{B} Lat\_SB_b, \quad s.t. \quad Lat\_SB_b = \sum_{i=0}^{n} P_{bi} \times Lat_i \quad (2)$$

Here, Lat_i represents the profiled latency of $Block_i$ in isolation, and P_{bi} values correspond to the probability values for super block of b as sampled in Equation (1) with respect to the architecture weights.

The latency loss is implemented as the ratio of the estimated latency of the search network (Lat) over the normalized baseline latency with respect to the target latency.

$$Loss = CE_{Loss} + \beta \times Lat_{Loss}$$
$$s.t. \quad Lat_{Loss} = Lat \ / \ (Lat_{Baseline} \times Target_{Lat})$$
$$s.t. \quad \beta = 1 \quad if \ (Lat_{Loss} > 1) \quad else \ 0 \quad (3)$$

During the training of the architecture weights, the latency loss will be automatically activated depending on whether the estimated latency of the search network is meeting the target latency requirement. This novel dynamic functionality helps the search progress towards the user latency target without the need for additional hyper-parameter tuning.

2.3. Phase 2: Architecture Sampling and Retraining

The optimized architecture obtained from Phase 1 is now instantiated for retraining. Since the weights of this final architecture were shared with other search points during Phase 1, a retraining step is necessary to avoid under-fitting and to obtain optimal accuracy. We construct the optimized architecture by selecting the blocks with the highest architecture weight values in each super block; from our empirical evaluation, this sampling strategy best balances additional training overheads with accuracy compared to other approaches such as the one described in Liu et al. (2018).

After sampling the optimized architecture, we perform a full training from scratch using the same settings as the baseline. Since MoE blocks may be part of the final architecture, we incorporate an auxiliary loss to enforce a balanced load across the experts. While we noticed that a balanced load did not necessarily lead to higher accuracy, it did improve the runtime of the MoE layers by reducing tail latency. We follow the same implementation of the auxiliary loss for load balancing ($Balance_{Loss}$) as prior work (Fedus et al., 2021). The incorporation of the $Balance_{Loss}$ is also done as: $Loss = CE_{Loss} + \beta \times Balance_{Loss}$. The $Balance_{Loss}$ provides an approximation for the score of load balancing across the experts. If the tokens are distributed uniformly across the experts, we obtain an ideal loss value of 1.

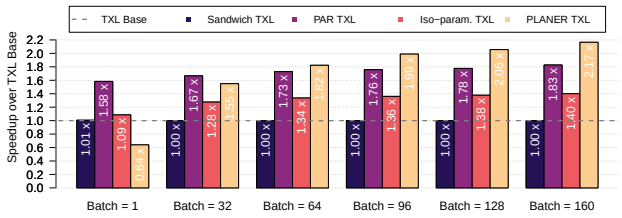
3. Evaluation

We evaluate PLANER on two real-world language modeling tasks and compare the performance of the latency-optimized networks to other state-of-the-art optimized Transformer models. We also provide a detailed analysis of the impact of using our dynamic loss formulation.

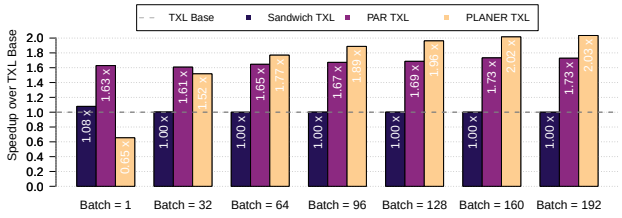
3.1. Methodology

We use Transformer-XL (TXL) Base on the WikiText-103 (WT103) and enwik8 datasets as our baseline networks. The backbone architecture for both datasets uses a model dimension of 512 and an interleaved pattern of MHA with 8 heads and FFLs with an inner dimension of 2048. The total number of blocks (MHA/FFL) is 24 and 32 for enwik8 and WT103, respectively [1]. The search space for phase 1 includes: (1) Skip connection, (2) MHA with 1, 2, 4, or 8

[1]The total number of blocks is 2× of the number of Transformer blocks



(a) WT103



(b) enwik8

Figure 4: Speedups obtained by PLANER w.r.t. various baselines across different batch sizes, profiled on NVIDIA A100.

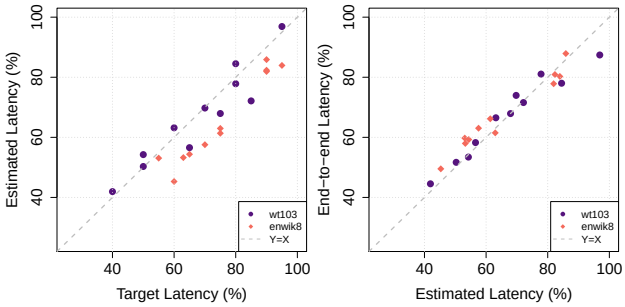
heads, (3) FFL with inner dimension of 2048, and (4) MoE with inner dimension of 2048, 8 experts, where each token is processed by either 1 or 2 experts ($Top_K = 1$ *or* 2). To evaluate the performance of PLANER, we compare the latency and accuracy of the optimized models with the baseline TXL model and two prior papers: Sandwich Transformer and PAR Transformer. We also compare PLANER in an iso-parameter setup, which replaces the MoE with a scaled FFL in the search space. The scaled FFL has an inner dimension of 16384, which results in the same number of parameters as the MoE with 8 experts. The goal of the iso-parameter baseline is to analyze the effectiveness of different model scaling solutions in compensating for accuracy drops caused by aggressive attention pruning. The design space is explored by deploying the 2-phase methodology with varying target latencies (50% to 95%). All training is performed on a cluster with 8 NVIDIA V100 GPUs.

3.2. Accuracy and Performance Trade-offs

We evaluate speedups for each individual architecture across various batch sizes. Figure 4 shows the speedups obtained by PLANER and the various baselines (described in Section 3.1) across both datasets. From the Figure, we notice that PLANER provides speedups of over 2× at larger batch sizes. On smaller batch sizes, PAR Transformer outperforms PLANER; this is primarily due to the unoptimized MoE layers used in our current implementation. Specifically, our current implementation computes the outputs of each expert sequentially, where a batch of sequences with N tokens are sequentially processed in mini-batches of size $\frac{Top_K \times N}{Experts}$. This consequently leads to under-utilization of the compute units. We are currently working on a more optimized parallel implementation of MoE layers, which will help plug this performance gap.

Model	wt103 (PPL)		enwik8 (BPC)	
	Dev	Test	Dev	Test
Transformer-XL Base	22.7	23.4	1.114	1.088
Sandwich Transformer-XL	22.6*	-	1.107	1.083
PAR Transformer-XL	22.7*	-	1.121	1.119
Iso-Parameter Transformer-XL	22.5	23.4	-	-
PLANER Transformer-XL	22.5	23.5	1.109	1.083

Table 1: Accuracy comparison of PLANER with prior work and baselines (scores marked with ∗ are referenced).



(a) Target vs estimated (b) Estimated vs end-to-end

Figure 5: Correlation between target, estimated, and end-to-end latency.

Table 1 lists the accuracy numbers obtained by PLANER and compares them with the baseline architectures.

3.3. Validating Estimated and End-to-end Runtime

In this section, we analyze the performance of the dynamic latency loss used in phase-1. Figure 5a shows the correlation of the input target latencies with respect to the estimated latencies of the architectures sampled at the end of phase 1, while Figure 5b depicts the correlation of the estimated latency with the profiled end-to-end latency. From the Figures, we notice that the latency estimated in Equation (2) is highly correlated with real-world latency, making it an appropriate option for PLANER's phase-1 search.

4. Conclusion

This paper has presented PLANER, an automated system for optimizing the inference latency of Transformer-based networks. PLANER employs a two-phase NAS methodology to systematically introduce sparsely activated layers into the given network, and uses a dynamic loss formulation to achieve user-provided latency targets while preserving accuracy. On two real-world NLP models, PLANER achieves inference latency reductions of over 2× at iso-accuracy.

References

Transformer-xl for pytorch: Nvidia ngc. URL https://catalog.ngc.nvidia.com/orgs/nvidia/resources/transformerxl_for_pytorch.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

He, J., Qiu, J., Zeng, A., Yang, Z., Zhai, J., and Tang, J. Fastmoe: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.

Mandava, S., Migacz, S., and Florea, A. F. Pay attention when required. *arXiv preprint arXiv:2009.04534*, 2020.

Masoudnia, S. and Ebrahimpour, R. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2): 275–293, 2014.

Press, O., Smith, N. A., and Levy, O. Improving transformer models by reordering their sublayers. *arXiv preprint arXiv:1911.03864*, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, H., Wu, Z., Liu, Z., Cai, H., Zhu, L., Gan, C., and Han, S. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A. Layerwise Performance Analysis

Figure 6 presents the profiled latencies of individual blocks, normalized to the latency of default multi-head attention configuration in TXL-Base (8 heads). Profiling is done with a model dimension of 512, $target\_len$ of 64, and batch size of 64 using NVIDIA A100 GPU. We observe three key points from the profiling experiment: first, we notice the significant cost of the default attention configuration, which accounts for $6.2\times$ higher runtime compared to the default FFL setup (with inner-dimension of 2048). The second observation is the approximately linear scaling of the attention cost with respect to the number of heads. Therefore, removing attention blocks or pruning attention heads can play a significant role in designing efficient Transformer-based networks. The last observation is the compute efficiency of the MoE blocks compared to both attention and iso-parametric FFL blocks, signifying the promise of using MoE blocks as a cost-effective solution to compensate for the potential accuracy loss caused by aggressive attention pruning.

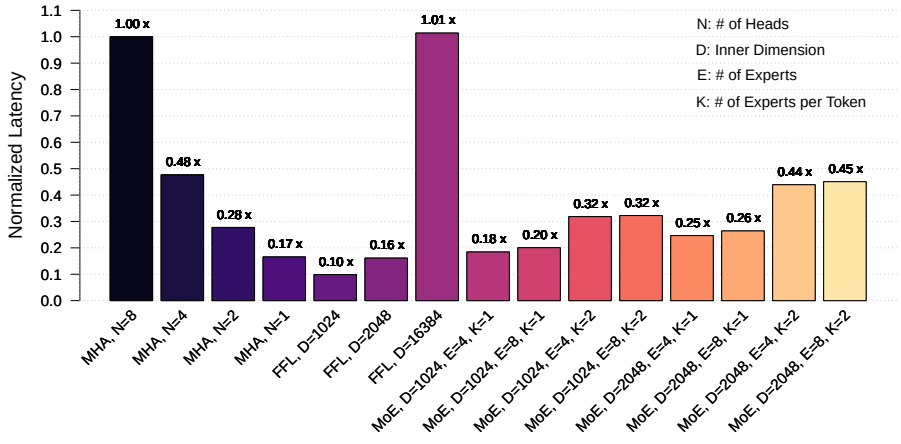


Figure 6: Latency comparison of attention, FFL, and MoE layers normalized w.r.t. attention with 8 heads, profiled on NVIDIA A100 GPU with batch size of 64, sequence length of 192, and half-precision.

B. PLANER Phase 1 and 2 Training Details

During phase 1, there are two sets of parameters that need to be trained. The first group of parameters are the actual network weights ($Block_i$), and the second set are the architecture weights (α_i). Training of each set is done sequentially in each epoch, using individual optimizers. Thus each epoch of training in phase 1 consists of optimizing the network weights using 100% of the training samples, and then training the architecture weights using 20% of the randomly sampled training data. We use soft sampling for $GumbelSoftmax$ during architecture optimization, and hard-sampling while training the network weights to reduce the overheads associated with the super blocks. To ensure that neither of the network weight sets are starved due to the hard-sampling of $GumbelSoftmax$, the architecture optimization is initially disabled for 10% of the epochs, and an annealing temperature scheduling is used for later epochs. These settings allow the blocks to be randomly sampled for the appropriate number of search epochs.

We use the settings published by NVIDIA (cat) for hyper-parameters. The exact hyper-parameters used for each dataset are:

- **WikiText-103 - Network Weights (Phase 1 and 2):** JITLamb optimizer, learning rate of 0.01, batch size of 256, target and memory length of 192, dropout rate of 0.1 for non-MoE layers and 0.2 for MoE layers, and 40000 iterations.
- **WikiText-103 - Architecture Weights (Phase 1):** Adam optimizer, learning rate of 0.01, initial temperature of 5 for the Gumbel Softmax, and temperature annealing rate of 0.6.
- **enwik8 - Network Weights (Phase 1 and 2):** JITLamb optimizer, learning rate of 0.004, batch size of 64, target and memory length of 512, dropout rate of 0.1 for non-MoE layers and 0.3 for MoE layers, and 120000 iterations.
- **enwik8 - Architecture Weights (Phase 1):** Adam optimizer, learning rate of 0.01, initial temperature of 5 for the Gumbel Softmax, and temperature annealing rate of 0.7.

C. Evaluated Architectures

Figures 7 and 8 present the detailed architecture of all evaluated models in Section 3.2. We notice that PLANER aggressively prunes/skips attention layers, while intelligently introducing sparsely activated layers for accuracy recovery.

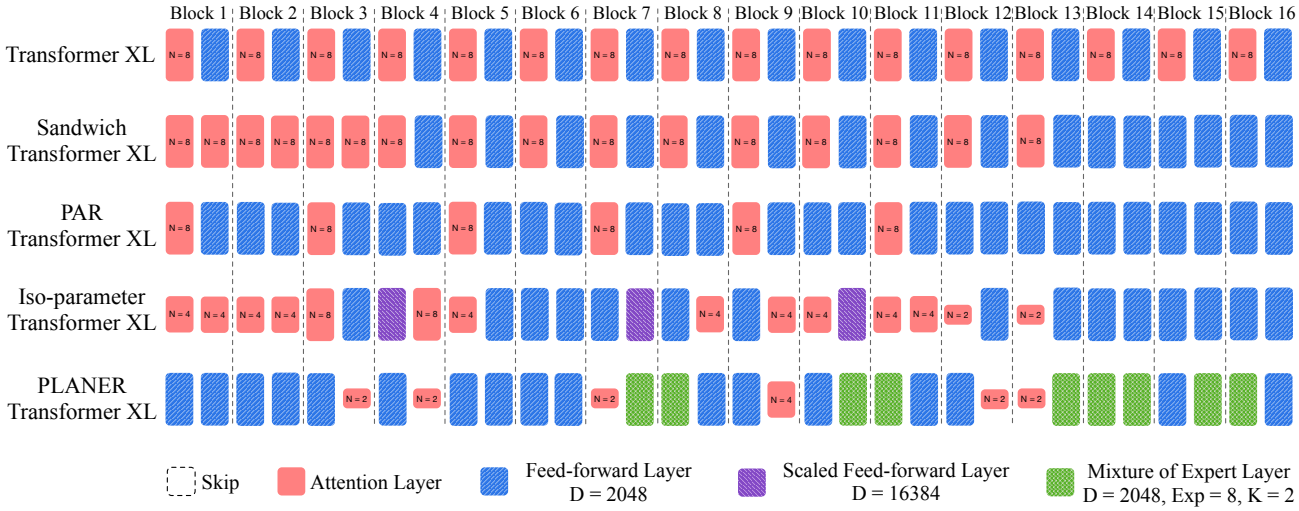


Figure 7: Evaluated architectures for WT103 dataset

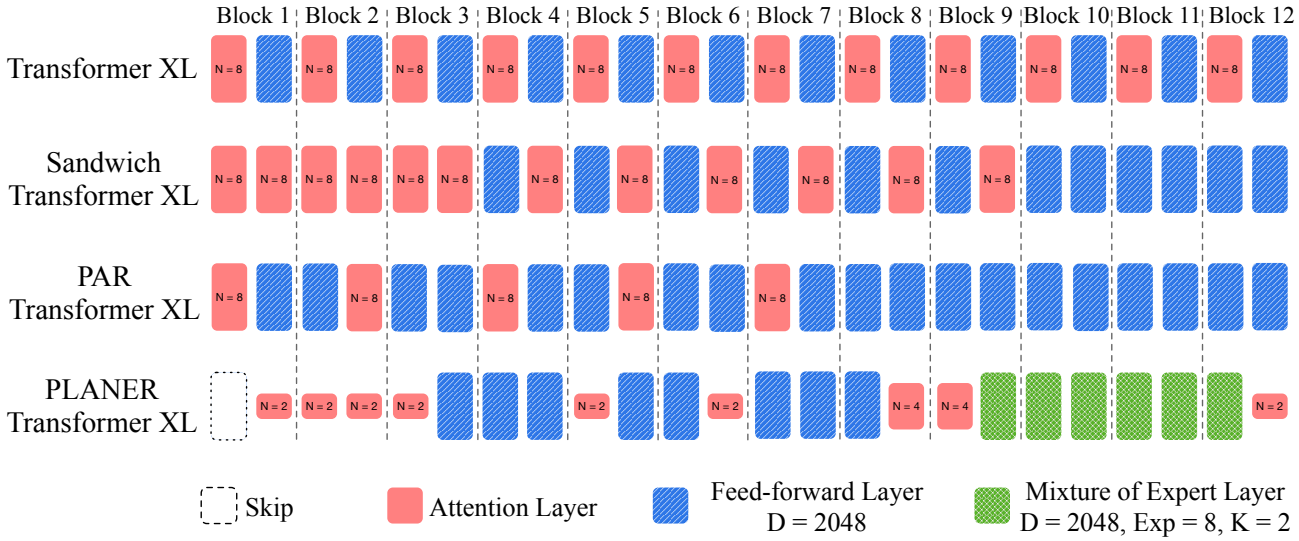
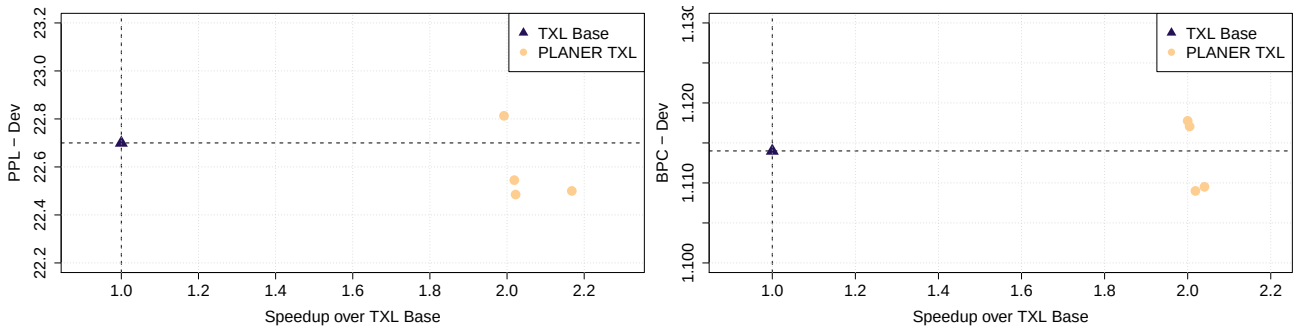


Figure 8: Evaluated architectures for enwik8 dataset

D. Repeatability Experiments

We repeat the PLANER optimization of the designs evaluated in Section 3.2 to validate the reproducibility of the experiments, and to observe any variations in the final architectures. Figure 9 presents all the achieved accuracy and speedup numbers. We notice that all the accuracy numbers are within 0.5% of the baseline, and speedups of over $2\times$ are achieved. Figures 10 and 11 also present the variations in the final architectures across the two datasets. Although the architectures do not match exactly, we notice a strong similarity in the number of heads in the attention layers. Another interesting observation from the Figures is that MoE layers are concentrated towards the end of the networks across both datasets.



(a) WT103 (b) enwik8

Figure 9: Speedup and accuracy results of repeatability experiments.

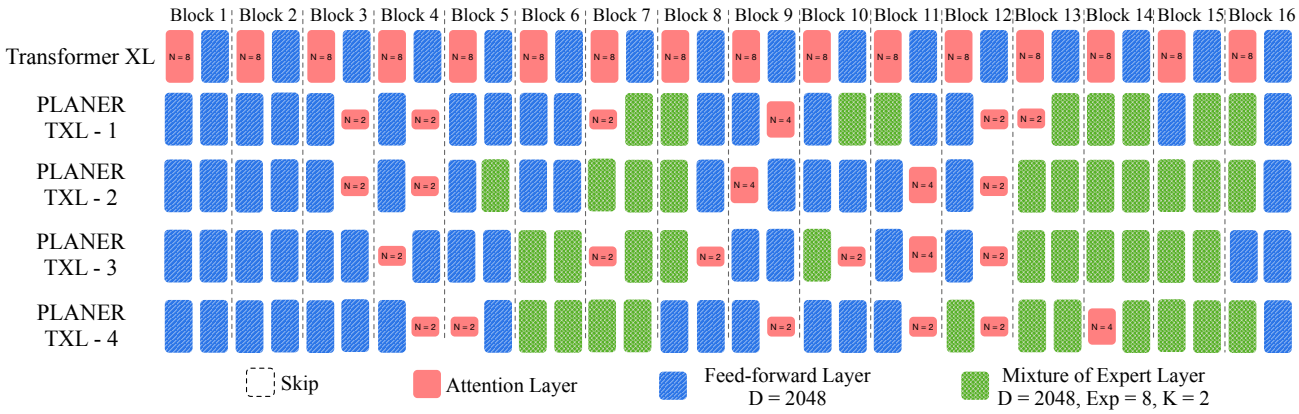


Figure 10: Explored architectures through repeatability experiment on WikiText-103 dataset.

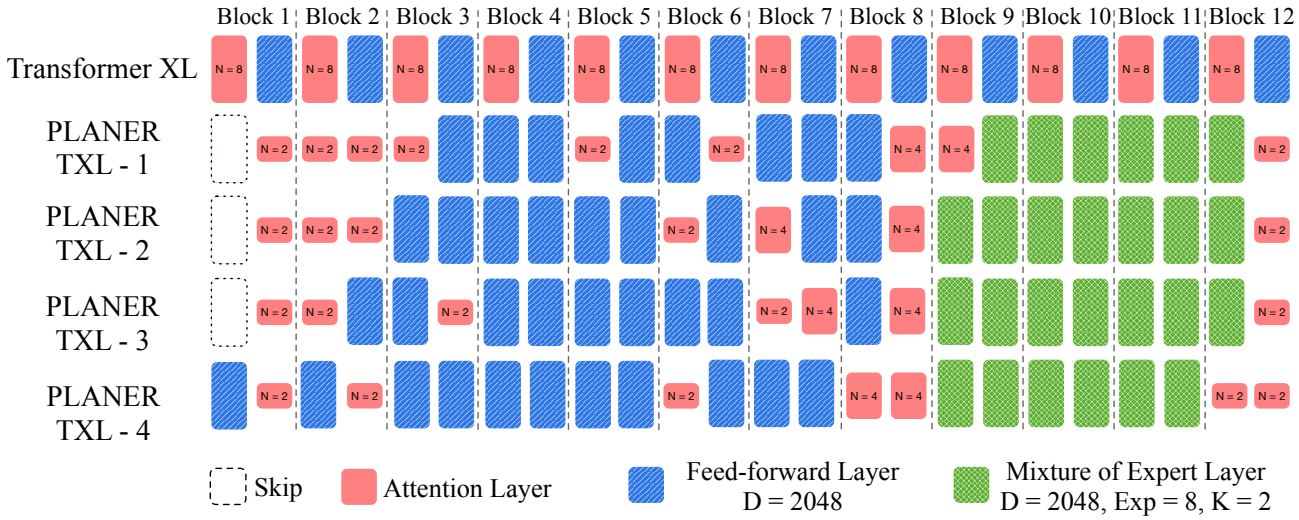


Figure 11: Explored architectures through repeatability experiment on enwik8 dataset.