Proceedings of Machine Learning Research **318** (2026)
The 39th Canadian Conference on Artificial Intelligence

EPAS: Efficient Training with Progressive Activation Sharing

Rezaul Karim[†,*], Maryam Dialameh[†‡], Yang Liu[†], Boxing Chen[†], Walid Ahmed[†]
[†] Ascend Team, Huawei Technologies, Toronto, Canada
[‡] University of Waterloo

Abstract

We present a novel method for **E**fficient training with **P**rogressive **A**ctivation **S**haring (EPAS). This method bridges progressive training paradigm with the phenomenon of redundant QK (or KV) activations across deeper layers of transformers. EPAS gradually grows a sharing region during training by switching decoder layers to activation sharing mode. This results in throughput increase due to reduced compute. To utilize deeper layer redundancy, the sharing region starts from the deep end of the model and grows towards the shallow end. The EPAS trained models allow for variable region lengths of activation sharing for different compute budgets during inference. Empirical evaluations with QK activation sharing in LLaMA models ranging from 125M to 7B parameters show up to an 11.1% improvement in training throughput and up to a 29% improvement in inference throughput while maintaining similar loss curve to the baseline models. Furthermore, applying EPAS in continual pretraining to transform TinyLLaMA into an attention-sharing model yields up to a 10% improvement in average accuracy over state-of-the-art methods, emphasizing the significance of progressive training in cross layer activation sharing models.

Keywords: Low Resource NLP, LLM efficiency, efficient inference, parameter-efficient-training, Many-in-one model.

1. Introduction

Recent research in computational efficiency of Transformers has focused on efficient pretraining [1], continual learning [2], fine-tuning [3–5] and inference [6, 7]. However, holistic approaches to efficient training and inference remains underexplored as evident from large accuracy-efficiency trade-offs in this direction [8, 9]. Surprisingly, large transformer models are found to compute redundant activations across deeper layers [10–14]. Therefore, as a promising yet less explored direction, we focus on utilizing redundancy phenomenon towards an unified efficiency solution to training and inference with minimal tradeoffs to accuracy.

Deeper layers of transformer models have been found to exhibit redundancy in activations of the attention block across layers. For example, multiple deeper layers are found to compute mostly similar attention scores [10, 15, 16]. Hence, recent methods reuse QK or KV activations across layers to enhance computational efficiency. These approaches are generally known as activation sharing. Attention sharing approaches compute only the value (V) and reuse the computed attention score directly from a previous layer to make the model compute efficient [10, 15, 16]. Since the attention score from the previous layer is not directly available in block factoring-based efficient attention algorithms, such as Flash-Attention [17, 18], an alternative approach shares QK across layers [19]. Meanwhile, some other approaches have proposed to compute the query (Q) and reuse (KV) from a previous layer [11, 20]. The sharing of QK offers greater computational savings, while sharing KV has greater impact in reducing inference memory footprint.

State-of-the-art activation sharing approaches enhance efficiency mostly by sharing activations across the layers of trained models during the inference [10] or follow model distillation [19]. Few models incorporate efficient design from the training phase, and those that do primarily focus on optimizing inference efficiency while overlooking training efficiency [20]. Since efficiency in both training and inference presents diverse design challenges, there is a growing need for a simple, holistic solution that addresses both aspects.

[*] rezaul.karim3@huawei.com

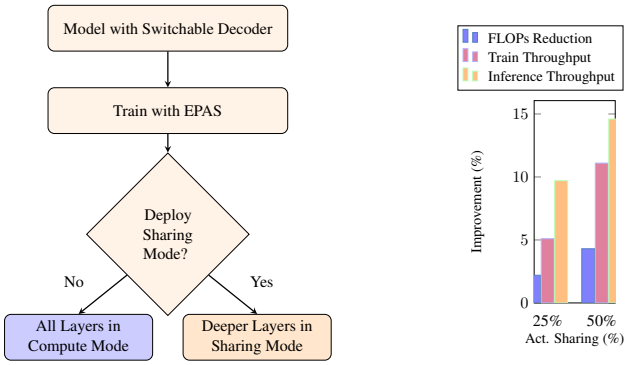


Figure 1. **Left:** Overall solution from efficient training to inference using EPAS. **Right:** TinyLLaMA [33] model FLOPs reduction and train/inference throughput improvement expanding QK activation sharing to 25% and 50% of the layers.

In another direction, efforts for efficient training has presented methods for progressive growth [21–23], progressive layer drop [24], and progressive dataset complexity [25]. Conversely, efficient inference approaches have focused on model pruning [26], distillation [27], progressive low-rank decomposition [28] and step-by-step distillation [29–31]. From a broader perspective, the progressive modification of model during training has proven superior to directly training the modified architecture, often with an additional advantage of many-in-one models [32].

The proposed progressive activation sharing combines progressive training and efficient inference in a unified training method for activation sharing models as shown a high level abstraction in Figure 1. This method allows for utilizing redundancy observed in deeper layers from early phases of training while preserving model accuracy. It improves pretraining throughput to reduce time to accuracy and derives a family of efficient models from a single end-to-end training process. Additionally, it enables flexible transformation of pretrained models into activation sharing models through a single, efficient continual pretraining process without requiring multiple rounds of knowledge distillation. Instead of sharing activations during inference of a pretrained model, an activation-sharing region is progressively expanded during pretraining or continual pretraining. This makes computation lighter as training progresses. This hot switching to activation sharing during training is achieved through a switchable decoder block that can conditionally reuse activation. The training algorithm uses a scheduler that toggles activation-sharing at configured intervals, progressively expanding the sharing block to improve training efficiency. The key contributions of the proposed method are:

- It Enables faster training and flexible efficient model configurations during inference.
- It transforms pretrained models into efficient ones through continual pretraining, eliminating the need for knowledge distillation.
- It improved training and inference throughput while maintaining accuracy.

2. Related Works

Accelerating deep neural networks has been a tremendous research attention resulting in various approaches, such as focused on training data [25, 34, 35], parallelism [36], model parameters [37–39], computation graph [24, 40, 41] and activation sharing [42]. Notably, training efficiency has focused on progressively freezing some layers [43], progressive stacking [21], progressive layer drop [24], and progressively increasing training overload in parallel to model growth [25]. Conversely, efficient inference has focused on model pruning [26], low-rank adaptation [29, 30], and distillation approaches [27] where recent works claimed to find progressive low-rank decomposition [28] and step by step distillation [31] as superior than single step low-rank decomposition or distillation.

Cross layer activation sharing to leverage the redundancy observed in representation of deeper layers of transformers has recently emerged as a promising direction for enhancing model efficiency.

Prominent cross-layer activation sharing methods include sharing attention scores, queries and keys ($Q,\ K$), or keys and values ($K,\ V$) across layers. For example, LazyFormer [15] introduced "lazy blocks," where the first layer in a group computes the attention scores and shares them with subsequent layers. ShareAttn [10] later extended this idea by using a single large block during inference following pretrained model analysis. LISA [16] further enhanced attention score sharing by adding transformations to better align shared representations. However, these methods are incompatible with efficient block-factorized attention mechanisms like Flash-Attention [17, 18]. An alternative strategy focused on sharing $Q,\ K$ across layers and employed distillation-based techniques to train the model [19]. Cross-layer $K,\ V$ sharing, on the other hand, reduces inference memory requirements by eliminating the need for separate K/V projections in each layer [11, 20, 44].

While prior activation sharing approaches improved inference speed, there remains challenge in training these models and closing the gap in accuracy. Prior works in progressive training methods have been largely restricted to layer stacking or dropping and have not explored activation sharing. In contrast, EPAS introduces progressive activation sharing by bridging the ideas of the two areas to address the training challenges and severe accuracy drop of state-of-the-art activation sharing models.

3. Progressive Activation Sharing

The proposed **E**fficient training with **P**rogressive **A**ctivation **S**haring (EPAS) method builds transformer models using the proposed switchable activation sharing decoder layer as building block. This decoder layer is a simple yet elegant extension that incorporates conditional activation sharing into the standard decoder layer. The term activation sharing generally refers to reusing some shared QK or KV activations from an early layer. This helps to reduce the computation of somewhat redundant activations in the deeper parts of the model. At a high level, the proposed progressive activation sharing involves gradually growing the number of layers using activation sharing and hence increasing the throughput. In the following, we discuss the details of the newly designed decoder layer and the progressive training algorithm.

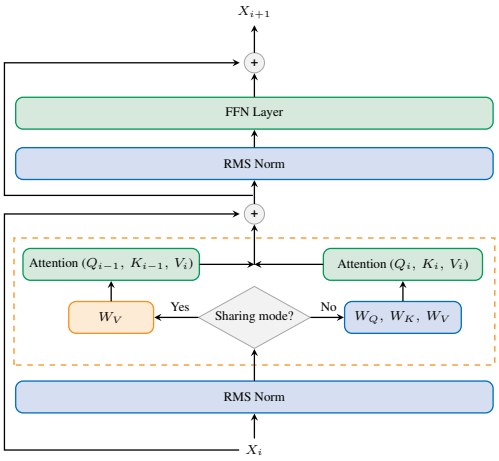
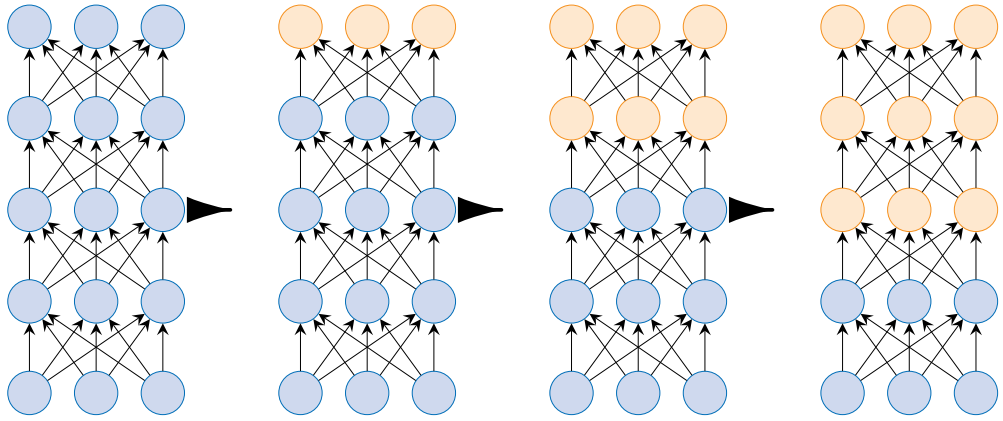


Figure 2. The Switchable Activation Sharing Decoder with an example of attention sharing (Q, K). This decoder layer extends conventional transformer decoder layer by adding a conditional switching branch to reuse Q_{i-1}, K_{i-1} from previous layer instead of computing in current layer(left branch inside dashed box). When not using activation sharing mode, the computation follows the right branch as like convention decoder layer.

3.1. Switchable Activation Sharing Decoder

The hot switching of decoder layers to activation sharing mode during the progress of training is performed by switching a conditional branching of computation in the decoder layer. A pictorial illustration of this decoder extension, with a particular example of attention sharing, is presented in Figure 2. This example demonstrates sharing of $Q,\ K$ to make attention sharing compatible with Flash-Attention. The extension is quite simple without requiring any additional parameters. Hence, the modified architecture can reuse previously trained parameters for continual pretraining or post-training. This decoder simply branches out to either reuse some selected activations from a previous layer or to compute with current layer's own parameters. The most conventional activation sharing models use either attention scores, or QK, or KV as the set of activations for this purpose.



(a) Baseline *(b)* Begin activation sharing *(c)* Grow sharing region *(d)* Activation sharing model

Figure 3. An example of EPAS training. Beginning with all L layers in **compute-mode** (e.g., 5 layers here), a region of B layers transition into **sharing-mode** (e.g., 1 layer here) at every I step intervals. The progressive growth continues till maximum S layers (e.g., 3 layers here) are in sharing mode. The trained model can be used either all layer in compute mode or up to S layers in activation sharing mode.

3.2. **Progressive Activation Sharing**

The progressive activation sharing approach aims to gradually expand the activation sharing region by switching decoder layers to activation sharing mode throughout the training steps. The growing of sharing layers follows a deterministic sharing strategy. The rationale behind deterministic sharing is to ensure that shared layers continuously grow, ultimately resulting in a model requiring fewer FLOPs than baseline during the inference.

Initially, training begins with all of the L layers of a model, M, in compute mode. At this stage the model works like a conventional transformer model [45]. A target sharing region, S, defines a list of layers that will progressively transition into activation sharing mode. We found that selecting a set of deeper layers and activated sharing sequentially from deep to shallow layers works well in this scenario. Over the course of T training steps, a group of B layers at the deep end of S is switched to activation sharing mode at every interval of I training steps. Rather than enabling activation sharing of the whole sharing region from the beginning of training, the method gradually expands the sharing region by gently allowing the model to adapt to activation sharing. Following this training strategy, it ends up with a target model of a predefined maximum activation sharing region. The steps are depicted with a schematic example in Figure 3.

In this proposed activation sharing scheme, the layer immediately before the sharing region shares its activations with the layers in sharing group. During the forward pass, if a layer detects that its subsequent layer is in sharing mode, it populates an activation cache with a selected set of activations, A. This progressive activation sharing method is compatible with sharing attention scores, QK, or KV. The trained model benefits from reduced model FLOPs by leveraging the last state of the activation sharing group. Algorithm 1 presents the progressive activation sharing training algorithm instantiated with QK sharing; the adaptation to other forms of activation sharing, such as KV sharing is straightforward.

3.3. **Applications**

Progressively growing the sharing block during training gradually reduces model FLOPs and increases training throughput (tokens/sec). This results in reduced training time and cost, while the gradual change in the computation graph allows for training stability. EPAS enhances computational resource utilization and improves pretraining efficiency by minimizing redundant activations, thereby

Algorithm 1 Progressive Activation Sharing

Input: Model, M, Interval, I, Target Sharing Layers, S_c, Sharing Region Growth Size B
Output: Trained model M
L**:** Layers in Base Model
T**:** Training Steps
S**:** Layers currently in sharing mode
assert $(B \geq 1 \ \ \& \ \ |S_c| \geq 1)$
assert $(|S_c| \mod B = 0)$
$C \leftarrow [], S \leftarrow []$
for $t = 1$ to T **do**
    for $i = 0$ to $|L| - 1$ **do**
        if $L[i] \in S$ **then**
            $Q_i, K_i \leftarrow Q_{i-1}, \ K_{i-1}$ from C
            Compute V_i
        else
            Compute $Q_i, \ K_i, V_i$
        end if
        Do the rest of the computations
        if $L[i+1]$ has sharing on **then**
            $C \leftarrow Q_i, K_i$
        end if
    end for
    if $t\%I == 0 \ \ \& \ \ |S_c| \geq B$ **then**
        $L_c \leftarrow$ pop last B element from S_c
        $S \leftarrow L_c + S$ ▷ Append beginning
    end if
end for
return M

achieving a balanced trade-off between efficiency and model performance. EPAS can also transform existing pretrained models into activation-sharing architectures while used in continual pretraining setting. Moreover, continual pretraining with EPAS enables flexible sub-network selection during inference, allowing a single end to end training on a small dataset to derive a family of efficient models. This eliminates the need for complex multi-model training or repeated distillation.

During the inference phase, the activation sharing models trained with EPAS can achieve superior throughput compared to baseline models while maintaining similar performance. Attention score sharing, or QK sharing, reduces computational overhead and minimizes KV cache memory requirement as the sharing layers only need to cache V. Conversely, KV sharing saves more memory as it eliminates the need to cache both K and V in the sharing layers while offering less computational reduction. In either of the case, the inference process becomes faster and more resource-efficient compared to baseline model.

4. Experiments

To demonstrate the efficacy of EPAS, we considered QK sharing as a particular instance of activation sharing. The experiments integrate EPAS with open-source transformer-based LLM. In particular, we perform extensive experiment with TinyLLaMA-1.1B [33] as our primary baseline and then further extend our empirical analysis with more LlaMA-based models [46, 47] ranging from 125M to 7B parameters. We used a small subset of the open-source SlimPajama-627B dataset for the pretraining and continual pretraining experiments [48]. Since our pretraining experiment is focused

on efficiency analysis and comparing training dynamics for small number of training steps rather than training till convergence, this set up is sufficient for the intended purpose.

The empirical analysis compared training and inference efficiency as well as learning capacity during training. We measure theoretical FLOPs reduction, training throughput (tokens/sec) and inference throughput (tokens/sec). For evaluation of transformed pretrained model with continual pretraining setup of EPAS, we used lm-eval-harness [49]. Furthermore, to compare the efficiency across diverse devices, experiments are conducted on Nvidia V100 GPU, Ascend 910A NPU, and Ascend 910B NPU.

Furthermore, we present extensive ablation study for critical understanding and justification of our findings and corresponding design choices. While the proposed method is compatible to share various activations, for the scope of this research we limit empirical analysis on attention sharing only. In particular, we used cross-layer query and key (QK) sharing so that the method is compatible with Flash-Attention [17, 18].

4.1. **Training efficiency**

To empirically assess training efficiency, we conduct model FLOPs and training throughput analysis by scaling model sizes from 125M to 7B following LLaMA architectures. Although EPAS presents a generalized training algorithm to train activation sharing architecture, we demonstrate for the example of half of the layers in final sharing mode following recent trends [20]. Table 1 summarizes the reduction of theoretical model FLOPs for each of the model when sharing QK across second half of the layers. We observe up to 8% FLOPs reduction with this configuration. Table 2 presents the improvement in training tokens/second with a distributed training setup on 8 V100 GPU. The models show up to 11% train throughput improvement with the above configuration.

Model	Model FLOPs/sample (TF)		
	Baseline	Q/K-Sharing	Reduction(%)
125M	1.81	**1.72**	4.9%
1.1B	14.98	**14.34**	4.3%
3B	41.41	**38.39**	7.3%
7B	85.62	**79.21**	8.1%

Table 1. Model FLOPs per sample in Terra-FLOPs (TF) for baseline versus QK sharing of 50% of the layers.

Model	Train Tokens/sec		
	Baseline	Q/K-Sharing	Improvement(%)
125M	15458.9	**17143.1**	10.8%
1.1B	3850.2	**4259.8**	11.1%
3B	1783.1	**1961.3**	10.9%
7B	1259.8	**1367.9**	8.6%

Table 2. Model size scaling and training efficiency of QK sharing on V100 GPU with distributed training setup.

We further investigated the learning capacity of activation sharing model when trained with EPAS compared to training baseline model without activation sharing. This experiment was conducted for $0.25M$ tokens per step for 4000 steps resulting in a total of $1B$ tokens. We consider this setup sufficient for comparing the training dynamics of the baseline and activation sharing models by analyzing loss curves without conducting a full LM benchmark evaluation similar to recent literature [20].

Model	**Train Time (hh:mm:ss)**	**Validation Loss**
125M (w/o EPAS)	00:54:12	3.74
125M (w/ EPAS)	**00:50:50**	3.79
1.1B (w/o EPAS)	03:07:18	3.19
1.1B (w/ EPAS)	**02:56:33**	3.22
3B (w/o EPAS)	06:02:46	3.06
3B (w/ EPAS)	**05:39:11**	3.09
7B (w/o EPAS)	11:09:26	2.99
7B (w/ EPAS)	**10:25:17**	3.04

Table 3. Total time and final validation loss for several LLaMA models of varying parameter sizes, demonstrating that EPAS training is significantly **faster** with a negligible difference in final validation loss.

.

Device Name	Device Spec	Single Device			Distributed (8 Device)		
		Baseline	Q/K Sharing	Improvement(%)	Baseline	Q/K Sharing	Improvement(%)
V100 GPU	32GB, 125 TF	4079.6	**4423.6**	10.8%	3850.2	**4259.8**	11.1%
910A NPU	32GB, 278 TF	4321.3	**4874.2**	12.8%	4788.2	**5246.9**	9.57%
910B NPU	64GB, 378 TF	11354.1	**12443.7**	9.6%	12902.1	**13844.5**	7.3%

Table 4. Empirical evidence of training efficiency comparing throughput (tokens/sec) across different hardware for activation sharing (QK) in the second half of the layers of the TinyLLaMA model.

The results in Table 3 compare the total training time and the final validation loss after training. The findings show that EPAS significantly speeds up training, while the difference in final validation loss remains negligibly small (less than 0.05), indicating faster convergence without sacrificing accuracy or increasing the risk of overfitting.

We observe that EPAS has lower loss at equal time and needs less time to achieve equal loss as baseline. We present a loss curve comparison for scaling analysis of training w/ and w/o EPAS in Figure 4. The comparison of train loss vs. time shows faster training and convergence with EPAS while maintaining similar loss curve pattern as the baseline. This is particularly evident from the observation that at any given time during training, EPAS shows a lower loss, especially in the early phases of training.

We also considered extending our experiments for various hardware types. For this cross hardware experiment, we fix a model (TinyLLaMA) and perform the same analysis across various hardware. Table 4 presents the observation of training efficiency across different hardware categories. The table shows a negligibly minor variation in train throughput improvement while changing hardware. In particular, the QK sharing model still maintains 8-10% train throughput improvement across the three types of hardware. This evidence further justifies that activation sharing makes a generic efficiency improvement on model's computation that persists across varieties of device specs.

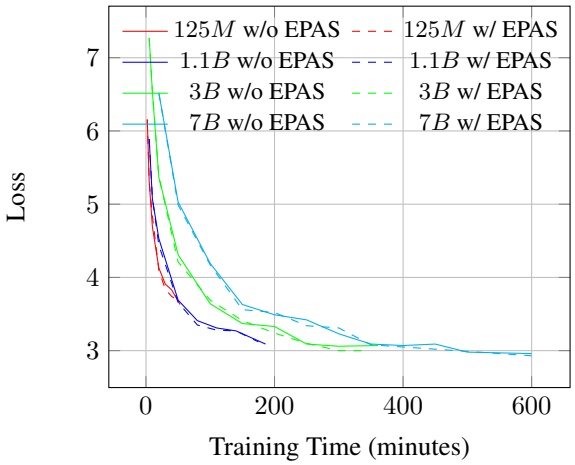


Figure 4. Smoothed loss versus time while scaling model sizes across LLaMA models with $125M$, $1.1B$, and $3B$ parameters. QK sharing models trained with EPAS also exhibit slightly faster convergence during training in addition to have higher throughput during inference.

4.2. **Inference efficiency**

We compare the generation throughput in terms of tokens/sec to measure the inference efficiency improvement of the proposed method. Since models trained with EPAS present a flexible architecture that can be used with various number of activation sharing layers during inference, we present inference throughput for sharing QK activations for 25% and 50% of the layers. For example, LLaMA1.1B has 22 layers, hence 25% and 50% indicates 5 and 11 layers in sharing mode respectively. We observe a 3–22% improvement in generation throughput when sharing 25% of the layers, and a 6–30% improvement when sharing 50% of the layers across different models. The results are summarized Table 5.

4.3. **Language Model Evaluations**

We consider to evaluate model trained with EPAS in a continual pretraining setup to transform base models to attention-sharing models. We follow this direction due to the resource intensity of pretraining from scratch. For fair comparison, we also perform continual pretraining on base models with the same dataset. This experiment begins with a pretrained checkpoint and follows continual pretraining with progressively growing the activation sharing block from the deep end of the model towards the shallow end gradually growing one large activation sharing region. No additional sophisticated training approaches, e.g., knowledge distillation or additional auxiliary loss computation are used. In particular, we consider a pretrained TinyLLaMA model as baseline. Due to limited resources we constrain the activation sharing block to grow up to 25% of the model depth(e.g., 5 out of 22 layers). The training is done in a distributed setup of 8 device with a batch size 8 per device and gradient accumulation steps 16 to match the effective token per step being equal to the baseline pretraining configuration (e.g., $8 \times 8 \times 16 \times 2048 = 2M$). The training is run for $2K$ steps resulting in total 4 billion tokens. We use the lm-eval-harness [49] for evaluating the models on LM benchmarks.

Model	Inference Throughput(tok/sec)		
	Baseline	25% Sharing	50% Sharing
125M	63.8	78.3 (22.7%)	**82.4** (29.2%)
1.1B	39.1	42.9 (9.7%)	**44.8** (14.6%)
3B	38.1	40.9 (7.3%)	**43.2** (13.4%)
7B	24.5	25.27 (3.3%)	**26.0** (6.4%)

Table 5. Inference throughput on V100 GPU with QK sharing on 25% and 50% of the layer in sharing mode.

The checkpoint from continual pretraining with EPAS is evaluated against applying inference time attention sharing, known as Beyond-KV-Cache [10] under various activation sharing configurations. We first compare the baseline and EPAS trained model with both of the models in full-compute mode. Then, we compare by using three and five layers in sharing mode. We observe that the model trained without EPAS shows a large drop in accuracy when evaluated with sharing mode. In contrast, EPAS trained model can retain most of the accuracy in attention sharing mode. When using five layers in sharing mode for both of the models, the EPAS trained model shows around 10% higher accuracy. The results are presented in Table 6, where each row pair shares the same activation sharing configuration.

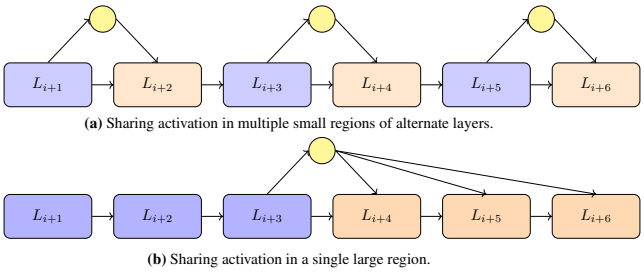
Model	WG	PIQA	BoolQ	ARC-C	ARC-E	OBQA	HS	SciQ	LM(oa)	LM(std)	RTE	Average
w/o EPAS (No Sharing)	59.12	73.56	56.09	32.68	55.51	36.80	61.45	84.20	56.70	50.55	57.04	56.70
w/ EPAS (No Sharing)	58.25	73.01	63.33	31.57	56.44	36.20	58.17	85.90	56.10	52.45	56.32	**57.07**
w/o EPAS (Sharing 3/22)†	60.14	73.50	48.41	32.25	53.91	37.40	60.48	75.80	25.00	23.17	47.65	48.88
w/ EPAS (Sharing 3/22)	59.27	73.07	62.72	33.02	56.14	36.20	58.72	83.60	49.97	47.80	52.35	**55.71**
w/o EPAS (Sharing 5/22)†	55.64	67.03	48.41	27.13	42.55	31.00	51.51	75.80	25.00	23.17	47.65	44.99
w/ EPAS (Sharing 5/22)	58.02	73.18	60.70	31.23	55.85	36.00	57.95	83.10	47.58	44.69	50.90	**54.47**

Table 6. Evaluation of multiple inference configurations from a single trained model. The EPAS model uses continual pretraining with a target sharing region of five layers while varied sharing region length during inference. † represents our re-implementation of [10] for smaller model for various QK sharing settings.

Full model evaluation during inference without activation sharing improves accuracy in the EPAS-trained model. As expected, baseline accuracy declines more rapidly as additional layers are included in the sharing block. In contrast, the EPAS-trained model maintains robust accuracy as the activation sharing block expands. Notably, with a QK sharing block size of five layers (25% of model depth), the EPAS-trained model outperforms the baseline by approximately 12%. These findings highlight EPAS's effectiveness in preserving accuracy while progressively sharing activations, offering a robust approach for efficient transformer model training and inference.

4.4. **Ablation Experiment**

Single vs. Multiple Sharing Block. Recent approaches of attention, $Q,\ K$ and $K,\ V$ sharing demonstrate two distinct methodologies for applying activation sharing across multiple layers. One line of research advocates for small activation-sharing blocks [15, 19] while others argue that such



(a) Sharing activation in multiple small regions of alternate layers.

(b) Sharing activation in a single large region.

Figure 5. Schematic illustration of layer grouping: multiple small blocks versus a single large block. The colors indicate **compute-** and **sharing-mode**. The number of layers shown is for illustrative purposes; both approaches can accommodate a variable number of layers.

fine-grained partitioning adds unnecessary complexity and computational overhead per block and favors instead a large activation-sharing block in the deeper end of the model [20]. To examine the trade-off between simplicity and computational efficiency, we conduct a comparative experiment with an equal number of layers in activation reuse mode for TinyLLaMA. One setup employs three groups of two activation-reusing layers, while the computationally equivalent model with a single block arranges three activation-reusing layers sequentially, as schematically depicted in Fig. 5. Results in Table 7 consistently show superior performance with a single large activation-sharing block, attributed to deeper-layer sharing. This trend holds across training settings with and without EPAS. Adhering to Occam's razor, we favor the simplicity of a single large sharing block, as additional complexity yields no clear advantage.

Model	PIQA	WG	BoolQ	OBQA	HS
MB (w/o EPAS)	72.25	58.01	**49.02**	36.20	60.15
SB (w/o EPAS)	**73.50**	**60.14**	48.41	**37.40**	**60.84**
MB (w/ EPAS)	**73.67**	57.93	60.06	36.40	58.28
SB (w/ EPAS)	73.45	**58.41**	**60.31**	**37.00**	**58.55**

Table 7. Results comparing using a single large sharing block (SB) vs. multiple small sharing blocks (MB)

Impact of Last Layer. Recent research presents diverging perspectives on activation sharing regarding the role of the final layer. One approach argues for excluding the last layer or retaining it solely during inference due to its distinct attention pattern [10], while an alternative view supports its inclusion within the activation-sharing block [20]. To assess the optimal configuration, we conducted an analysis using continual pretraining of TinyLLaMA with EPAS, evaluating LM benchmark performance on a small dataset under two conditions: exclusion versus inclusion of the last layer within sharing blocks. As summarized in Table 8, the results indicate minimal performance differences, with a slight advantage observed when incorporating the last layer during training. Consequently, we adopt this strategy in our approach.

Model	PIQA	WG	BoolQ	OBQA	HS
Excluding last layer	73.29	58.01	52.94	36.00	58.66
Including last layer	**73.39**	**59.12**	**56.51**	**36.80**	**59.11**

Table 8. LM benchmark evaluation of continual pretraining reveals a minor difference between including or excluding the last layer in activation sharing.

5. Conclusion

EPAS presented an efficient transformer training algorithm incorporating a switchable decoder layer. It integrates cross-layer activation sharing as a generic property of the model and training process rather than a drop in modification of trained model. This design showcased a comprehensive solution for improving efficiency in both training and inference, balancing optimization to maintain

performance while reducing computation. By applying EPAS during continual pretraining, pretrained models can be efficiently converted into activation-sharing models with adaptable computational budgets. Empirical evaluations demonstrated that EPAS enhances training and inference throughput with accuracy similar to the baseline. These findings underscore the potential of redundancy-aware training and inference as a scalable approach to optimizing transformer models. Furthermore, EPAS holds promise for extending to post-training as well as vision and speech domains, which merit further exploration in future research.

Acknowledgements

The authors gratefully acknowledge the support of the Toronto Ascend team.

References

[1] N. He, W. Xiong, H. Liu, Y. Liao, L. Ding, K. Zhang, G. Tang, X. Han, and Y. Wei. "SoftDedup: an Efficient Data Reweighting Method for Speeding Up Language Model Pre-training". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 4011–4022. DOI: 10.18653/v1/2024.acl-long.220. URL: https://aclanthology.org/2024.acl-long.220/.

[2] W. Zhao, S. Wang, Y. Hu, Y. Zhao, B. Qin, X. Zhang, Q. Yang, D. Xu, and W. Che. "Sapt: A shared attention framework for parameter-efficient continual learning of large language models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 11641–11661.

[3] Z. Zhang, D. Zhao, X. Miao, G. Oliaro, Z. Zhang, Q. Li, Y. Jiang, and Z. Jia. "Quantized Side Tuning: Fast and Memory-Efficient Tuning of Quantized Large Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 1–17. DOI: 10.18653/v1/2024.acl-long.1. URL: https://aclanthology.org/2024.acl-long.1/.

[4] Z. Liu, S. Kundu, A. Li, J. Wan, L. Jiang, and P. Beerel. "AFLoRA: Adaptive Freezing of Low Rank Adaptation in Parameter Efficient Fine-Tuning of Large Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 161–167. DOI: 10.18653/v1/2024.acl-short.16. URL: https://aclanthology.org/2024.acl-short.16/.

[5] X. Ge, A. Mousavi, E. Grave, A. Joulin, K. Qian, B. Han, M. Arefiyan, and Y. Li. "Time Sensitive Knowledge Editing through Efficient Finetuning". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 583–593. DOI: 10.18653/v1/2024.acl-short.53. URL: https://aclanthology.org/2024.acl-short.53/.

[6] K. Alizadeh, S. I. Mirzadeh, D. Belenko, S. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, and M. Farajtabar. "LLM in a flash: Efficient Large Language Model Inference with Limited Memory". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 12562–12584. DOI: 10.18653/v1/2024.acl-long.678. URL: https://aclanthology.org/2024.acl-long.678/.

[7] H. Wu and K. Tu. "Layer-Condensed KV Cache for Efficient Inference of Large Language Models". In: *Annual Meeting of the Association for Computational Linguistics*. 2024. URL: https://api.semanticscholar.org/CorpusID:269899988.

[8] Y. Kim and S. Lee. "SparseFlow: Accelerating Transformers by Sparsifying Information Flows". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 5937–5948. DOI: 10.18653/v1/2024.acl-long.323. URL: https://aclanthology.org/2024.acl-long.323/.

[9] J. Song, K. Oh, T. Kim, H. Kim, Y. Kim, and J.-J. Kim. "SLEB: Streamlining LLMs through Redundancy Verification and Elimination of Transformer Blocks". In: *arXiv preprint arXiv:2402.09025* (2024).

[10] B. Liao and D. V. Vargas. "Beyond kv caching: Shared attention for efficient llms". In: *arXiv preprint arXiv:2407.12866* (2024).

[11] W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. R. Kelly. "Reducing Transformer Key-Value Cache Size with Cross-Layer Attention". In: *arXiv preprint arXiv:2405.12981* (2024).

[12] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami. "Kvquant: Towards 10 million context length llm inference with kv cache quantization". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 1270–1303.

[13] A. Tomar, C. Hooper, M. Lee, H. Xi, R. Tiwari, W. Kang, L. Manolache, M. W. Mahoney, K. Keutzer, and A. Gholami. "XQuant: Breaking the Memory Wall for LLM Inference with KV Cache Rematerialization". In: *arXiv preprint arXiv:2508.10395* (2025).

[14] A. Qiao, Z. Yao, S. Rajbhandari, and Y. He. "SwiftKV: Fast prefill-optimized inference with knowledge-preserving model transformation". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. 2025, pp. 25745–25764.

[15] C. Ying, G. Ke, D. He, and T.-Y. Liu. "Lazyformer: Self attention with lazy update". In: *arXiv preprint arXiv:2102.12702* (2021).

[16] Y. Mu, Y. Wu, Y. Fan, C. Wang, H. Li, Q. He, M. Yang, T. Xiao, and J. Zhu. "Cross-layer attention sharing for large language models". In: *arXiv preprint arXiv:2408.01890* (2024).

[17] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. "Flashattention: Fast and memory-efficient exact attention with io-awareness". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 16344–16359.

[18] T. Dao. "Flashattention-2: Faster attention with better parallelism and work partitioning". In: *arXiv preprint arXiv:2307.08691* (2023).

[19] H. Rajabzadeh, A. Jafari, A. Sharma, B. Jami, H. J. Kwon, A. Ghodsi, B. Chen, and M. Rezagholizadeh. "Echoatt: Attend, copy, then adjust for more efficient large language models". In: *arXiv preprint arXiv:2409.14595* (2024).

[20] Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei. "You only cache once: Decoder-decoder architectures for language models". In: *arXiv preprint arXiv:2405.05254* (2024).

[21] L. Gong, D. He, Z. Li, T. Qin, L. Wang, and T. Liu. "Efficient training of bert by progressively stacking". In: *International conference on machine learning*. PMLR. 2019, pp. 2337–2346.

[22] C. Wu, Y. Gan, Y. Ge, Z. Lu, J. Wang, Y. Feng, Y. Shan, and P. Luo. "Llama pro: Progressive llama with block expansion". In: *arXiv preprint arXiv:2401.02415* (2024).

[23] Y. K., T. S., K. S., K. S., and J. Suzuki. "Efficient Construction of Model Family through Progressive Training Using Model Expansion". In: *Conference on Language Modeling*. 2025.

[24] M. Zhang and Y. He. "Accelerating training of transformer-based language models with progressive layer dropping". In: *Advances in neural information processing systems* 33 (2020), pp. 14011–14023.

[25] C. Li, B. Zhuang, G. Wang, X. Liang, X. Chang, and Y. Yang. "Automated progressive learning for efficient training of vision transformers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12486–12496.

[26] H. Cheng, M. Zhang, and J. Q. Shi. "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[27] C. Yang, Y. Zhu, W. Lu, Y. Wang, Q. Chen, C. Gao, B. Yan, and Y. Chen. "Survey on knowledge distillation for large language models: methods, evaluation, and application". In: *ACM Transactions on Intelligent Systems and Technology* (2024).

[28] H. Hajimolahoseini, M. Hassanpour, F. Ataiefard, B. Chen, and Y. Liu. "Single parent family: A spectrum of family members from a single pre-trained foundation model". In: *arXiv preprint arXiv:2406.19995* (2024).

[29] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. "Lora: Low-rank adaptation of large language models." In: *ICLR* 1.2 (2022), p. 3.

[30] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen. "Dora: Weight-decomposed low-rank adaptation". In: *Forty-first International Conference on Machine Learning*. 2024.

[31] C.-Y. Hsieh, C.-L. Li, C.-k. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister. "Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller

Model Sizes". In: *Findings of the Association for Computational Linguistics: ACL 2023*. 2023, pp. 8003–8017.

[32] R. Cai, S. Muralidharan, G. Heinrich, H. Yin, Z. Wang, J. Kautz, and P. Molchanov. "Flextron: Many-in-One Flexible Large Language Model". In: *International Conference on Machine Learning*. PMLR. 2024, pp. 5298–5311.

[33] P. Zhang, G. Zeng, T. Wang, and W. Lu. "Tinyllama: An open-source small language mode". In: *arXiv preprint arXiv:2401.02385* (2024). URL: https://arxiv.org/abs/2401.02385.

[34] H. Hajimolahoseini, O. M. Awad, W. Ahmed, A. Wen, S. Asani, M. Hassanpour, F. Javadi, M. Ahmadi, F. Ataiefard, K. Liu, et al. "SwiftLearn: A Data-Efficient Training Method of Deep Learning Models using Importance Sampling". In: *arXiv preprint arXiv:2311.15134* (2023).

[35] F. Ataiefard, W. Ahmed, H. Hajimolahoseini, S. Asani, F. Javadi, M. Hassanpour, O. M. Awad, A. Wen, K. Liu, and Y. Liu. "SkipViT: Speeding Up Vision Transformers with a Token-Level Skip Connection". In: *arXiv preprint arXiv:2401.15293* (2024).

[36] H. Amer, R. Karim, A. Pourranjbar, W. Zhang, W. Ahmed, and B. Chen. "Distributed Hybrid Parallelism for Large Language Models: Comparative Study and System Design Guide". In: *arXiv preprint arXiv:2602.09109* (2026).

[37] H. Hajimolahoseini, W. Ahmed, and Y. Liu. "Training acceleration of low-rank decomposed networks using sequential freezing and rank quantization". In: *arXiv preprint arXiv:2309.03824* (2023).

[38] W. Ahmed, H. Hajimolahoseini, A. Wen, and Y. Liu. "Speeding up resnet architecture with layers targeted low rank decomposition". In: *arXiv preprint arXiv:2309.12412* (2023).

[39] H. Hajimolahoseini, W. Ahmed, A. Wen, and Y. Liu. "Accelerating the Low-Rank Decomposed Models". In: *arXiv preprint arXiv:2407.20266* (2024).

[40] A. Fan, E. Grave, and A. Joulin. "Reducing Transformer Depth on Demand with Structured Dropout". In: *International Conference on Learning Representations*. 2020.

[41] S. He, G. Sun, Z. Shen, and A. Li. "What matters in transformers? not all attention is needed". In: *arXiv preprint arXiv:2406.15786* (2024).

[42] F. Javadi, W. Ahmed, H. Hajimolahoseini, F. Ataiefard, M. Hassanpour, S. Asani, A. Wen, O. M. Awad, K. Liu, and Y. Liu. "GQKVA: Efficient Pre-training of Transformers by Grouping Queries, Keys, and Values". In: *arXiv preprint arXiv:2311.03426* (2023).

[43] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. "Freezeout: Accelerate training by progressively freezing layers". In: *arXiv preprint arXiv:1706.04983* (2017).

[44] M. Dialameh, R. Karim, H. Rajabzadeh, O. M. Awad, B. Chen, H. J. Kwon, W. Ahmed, and Y. Liu. "ECHO-LLaMA: Efficient Caching for High-Performance LLaMA Training". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*. 2025, pp. 2252–2269.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[46] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[47] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288* (2023).

[48] J. Doe. *SlimPajama-627B*. URL: https://huggingface.co/datasets/cerebras/SlimPajama-627B/tree/main/train (visited on 01/02/2025).

[49] L. Gao et al. *A framework for few-shot language model evaluation*. Version v0.4.3. July 2024. DOI: 10.5281/zenodo.12608602. URL: https://zenodo.org/records/12608602.