ECHO-LLaMA: Efficient Caching for High-Performance LLaMA Training

Maryam Dialameh[1,2], Rezaul Karim[2], Hossein Rajabzadeh[1,2], Omar Mohamed Awad[2]
Boxing Chen[2], Walid Ahmed[2], Yang Liu[2], Hyock Ju Kwon[1]
[1]University of Waterloo, Waterloo, Canada
[2]Ascend Team, Huawei Technologies, Toronto, Canada
{maryam.dialameh,hossein.rajabzadeh,hjkwon}@uwaterloo.ca
{omar.mo.awad}@outlook.com
{rezaul.karim3, boxing.chen,walid.ahmed1,yang.liu8}@huawei.com

Abstract

This paper introduces ECHO-LLaMA, an efficient LLaMA architecture designed to improve both the training speed and inference throughput of LLaMA architectures while maintaining its learning capacity. ECHO-LLaMA transforms LLaMA models into shared KV caching across certain layers, significantly reducing KV computational complexity while maintaining or improving language performance. Experimental results demonstrate that ECHO-LLaMA achieves up to 77% higher token-per-second throughput during training, up to 16% higher Model FLOPs Utilization (MFU), and up to 14% lower loss when trained on an equal number of tokens. Furthermore, on the 1.1B model, ECHO-LLaMA delivers approximately 7% higher test-time throughput compared to the baseline. By introducing a computationally efficient adaptation mechanism, ECHO-LLaMA offers a scalable and cost-effective solution for pretraining and finetuning large language models, enabling faster and more resource-efficient training without compromising performance.

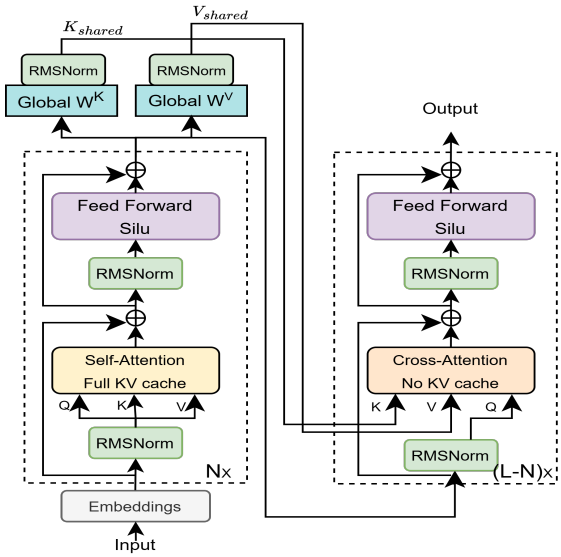


Figure 1: ECHO-LLaMA architecture uses shared KV caching and is trained through layer-wise adaptation, progressively converting pretrained LLaMA models into ECHO-LLaMA. Sharing $L - N$ KV-layers (total layers minus self-attention layers) reduces computational overhead and improves inference speed.

1 Introduction

Large language models (LLMs) have shown remarkable success across a wide range of natural language processing (NLP) tasks, including text generation (Makridakis et al., 2023), summarization (Zhang et al., 2024), and question answering and more (Chang et al., 2024). Despite their capabilities, training and deploying these models is highly resource-intensive, demanding significant computational power and memory (Chowdhery et al., 2023). For instance, a transformers-based LLM requires about three terabyte KV-cached memory for a model of size 500B with 8k context length and 128 batch size (Pope et al., 2023). Furthermore, the process of pretraining for LLMs is both resource and data intensive (Milano et al., 2023; Hoffmann et al., 2022).

Several existing works have been proposed to address the challenge of improving transformer efficiency (Zhang et al., 2023; Yang et al., 2023; Lee et al., 2024; Tang et al., 2024; Adnan et al., 2024; Hajimolahoseini et al., 2023; Ahmed et al., 2023), including GQKVA (Javadi et al., 2023), Beyond KV Caching (Liao and Vargas, 2024) and EchoAtt (Rajabzadeh et al., 2024) that specifically share the attention weights across layers to decrease computation and parameters; however, they still need to cache for V matrices. YOCO (You Only Cache Once) is another similar work that investigates sharing key-value (KV) caches across the second-half of layers (cross-decoder layers) while employing windowed-attention for the first half; thereby gaining computational and memory efficiency during training and inference (Sun et al., 2024). YOCO

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 2252–2269
November 4-9, 2025

models, however, use windowed self-attention, which limits their ability to capture long-range token dependencies (Beltagy et al., 2020). Furthermore, they apply shared KV caches to exactly half of the layers, enforcing a rigid and suboptimal KV sharing strategy. This lack of flexibility is particularly problematic for edge devices, which may benefit from adaptive KV sharing, allowing for shorter or longer KV reuse based on resource constraints and workload demands. Inspired by YOCO, this paper introduces ECHO-LLaMA, an efficient modification of LLaMA architectures that focuses on having efficient training and pretraining processes of LLaMA models. ECHO-LLaMA leverages a caching strategy where KV representations are shared across cross-decoder layers (typically the second half of the LLaMA layers), effectively reducing memory overhead and computational redundancy. Despite cross-decoder layers, the model remains strictly causal and retains the standard autoregressive masking used in LLaMA architectures.

Unlike the original YOCO approach (Sun et al., 2024), which is designed for training models from scratch, ECHO-LLaMA extends this methodology to pre-trained LLaMA models, enabling a more efficient adaptation without requiring full re-training. This is achieved through incremental adaptation, a strategy that incrementally transitions pre-trained LLaMA models into the ECHO architecture while maintaining or improving their language capabilities. The results demonstrate significant improvements in training throughput (up to 77%), Model FLOPs Utilization (MFU, up to 16%), and lower loss (up to 14%) compared to baselines. Moreover, ECHO-LLaMA achieves faster test-time throughput (approximately 7% on the 1.1B model), further showcasing its effectiveness for large-scale model deployment. By reducing training costs, accelerating inference, and preserving model performance, ECHO-LLaMA represents a scalable and practical solution for optimizing pre-trained large language models.

The proposed ECHO-LLaMA framework offers several significant advantages. First, it achieves faster inference without sacrificing language performance. Second, it reduces the need for heavy pre-training, making it a practical approach for building efficient versions of LLaMA models. Third, experiments conducted on "Nvidia-V100 GPUs" and "Huawei Ascend NPU-910B" devices show that ECHO-LLaMA consistently achieves higher training throughput and competitive performance compared to LLaMA baselines. Additionally, the results demonstrate the ability of the proposed framework to maintain or improve model performance while reducing computational costs. Hence, the main contributions of this work are summarized as follows: 1) We propose ECHO-LLaMA, an efficient LLaMA architecture that shares KV caches across selected set of layers to reduce computational redundancy. 2) ECHO-LLaMA employs a layer-wise incremental adaptation for the training strategy to efficiently convert pre-trained LLaMA models into the ECHO-LLaMA structure. 3) This approach significantly improves training throughput and inference speed without compromising language performance. 4) Extensive experiments demonstrate higher model FLOPs utilization, lower loss, and increased tokens-per-second throughput compared to baseline models.

2 Proposed Method

This section introduces our efficient framework for converting pretrained LLaMA models into ECHO-LLaMA architectures through incremental layer-wise adaption. The core idea consists of two parts: 1) the ECHO-LLaMA architecture and 2) layer-wise incremental adaptation strategy. Figure 1 illustrates the workflow of the ECHO-LLaMA, in which the first half of the layers remains full self-attention with full KV caches, and the second half of layers are gradually converted into cross-decoders with only one set of global KV caches. Each cross-decoder layer uses the global KV, computing by linearly transforming the output of the middle layer followed by an RMS normalization (Zhang and Sennrich, 2019).

Given an input sequence of token embeddings $x_1, x_2, \ldots, x_n \in R^d$, where d is the hidden size, the first N layers of an ECHO-LLaMA model (out of a total of L layers) operate similarly to standard LLaMA layers. The parameter $N \in \{L/2, \ldots, L\}$ is a hyperparameter that controls the extent of KV sharing in ECHO-LLaMA. Let's X_{l-1} be the output of layer $l-1$, which would be the input to layer l. Therefore, for an input X_{l-1}, $l \in \{0, 1, ..., N\}$, we have:

$$\begin{cases} X'_l &= \text{Self-Attn}\big(\text{RMSNorm}(X_{l-1})\big) + X_{l-1}, \\ X_l &= \text{SFF}\big(\text{RMSNorm}(X'_l)\big) + X'_l. \end{cases} \quad (1)$$

$$\text{Self-Attn}(X) = \text{softmax}\Big(\frac{XW_Q\,(XW_K)^T}{\sqrt{d_k}}\Big)\,XW_V. \tag{2}$$

where $W_Q, W_K, W_V \in R^{d_{\text{model}} \times d_k}$, d_k represents the dimensionality of the key and query vectors, typically set as $d_k = \frac{d_{\text{model}}}{h}$ and h is the number of attention heads. The LLaMA MLP module is a SiLU (Elfwing et al., 2018) activated feed forward module, SFF(X), consisting of a gate projection, an up projection, and a down projection. Given an input tensor X, the MLP transformation is defined as:

$$SFF(X) = W_{\text{down}}\Big(SiLU(W_{\text{gate}}X + b_{\text{gate}}) \odot (W_{\text{up}}X + b_{\text{up}})\Big) + b_{\text{down}} \tag{3}$$

where $W_{\text{gate}} \in \mathbb{R}^{d \times d_{\text{up}}}$, $b_{\text{gate}} \in \mathbb{R}^{d_{\text{up}}}$ (bias for gate projection parameters), $W_{\text{up}} \in \mathbb{R}^{d \times d_{\text{up}}}$, $b_{\text{up}} \in \mathbb{R}^{d_{\text{up}}}$ (up projection parameters), $W_{\text{down}} \in \mathbb{R}^{d_{\text{up}} \times d}$, $b_{\text{down}} \in \mathbb{R}^{d}$ (down projection parameters), and $\odot$ represents the element-wise Hadamard product.

The output of layer-N, i.e. X_N, is then passed through global W^K and W^V followed by RMS normalization, creating one shared KV for the rest of layers. In other words, layers $l \in \{N+1, ..., L\}$ use the same KV matrices and, therefore, compute cross attention between query and shared KV. This modification allows LLaMA models to save KV cache memory and increase both training and inference speed. Assuming X_N as the output of layer N, the shared KV in ECHO-LLaMA is computed as follows:

$$K_{\text{shared}} = \text{RMSNorm}(W^K_{\text{global}} X_N) \tag{4}$$

$$V_{\text{shared}} = \text{RMSNorm}(W^V_{\text{global}} X_N) \tag{5}$$

where W^K_{global} and W^V_{global} follows the same size as W^K and W^V. We further modify the LLaMA architecture to compute cross attention between query and shared-KV as follows:

$$\text{Cross-Attn}(X) = \text{softmax}\Big(\frac{X_N W_Q\,(K_{shared})^T}{\sqrt{d_k}}\Big)\,V_{shared}. \tag{6}$$

As depicted in Figure 1, the output of cross attention is then added by a residual and passed through RMS normalization and SFF module.

Algorithm 1 Incremental Adaptation for ECHO-LLaMA

Require: Pretrained model $\mathcal{F}_{\theta_{\text{pre}}}$, total layers L, threshold layer N for cross-decoder conversion, token budget per stage $\mathcal{T}_{\text{stage}} \approx 4$M, training steps per stage $S \approx 150$, final token budget $\mathcal{T}_{\text{final}} \approx 4$B

Ensure: Adapted model $\mathcal{F}_\theta$

1: Initialize: $\mathcal{F}_\theta \leftarrow \mathcal{F}_{\theta_{\text{pre}}}$
2: Define adaptation range: $\mathcal{R} \leftarrow \{\ell \mid \ell = L, L-1, \ldots, N\}$
3: **for all** $\ell \in \mathcal{R}$ (in descending order) **do**
4: Convert layer ℓ into a cross-decoder layer in $\mathcal{F}_\theta$
5: Update parameters $\theta^{(\ell)}$ by minimizing

$$\min_{\theta^{(\ell)}} \mathcal{L}\Big(\mathcal{F}_\theta, \mathcal{D}_\ell\Big)$$

for S steps on a token set of size $\mathcal{T}_{\text{stage}}$
6: **end for**
7: Fine-tune $\mathcal{F}_\theta$ for 1 epoch on a token set of size $\mathcal{T}_{\text{final}}$
8: **return** $\mathcal{F}_\theta$

2.1 Incremental Adaptation Strategy

ECHO-LLaMA leverages incremental adaptation from a pre-trained LLaMA model. Denote the parameters of layer ℓ by $\theta^{(\ell)}$. The layers $\{1, \ldots, L\}$ are partitioned into ordered blocks $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$, where each block $\mathcal{B}_m$ is adapted in a separate stage. In our experiments, we assume the size of $\mathcal{B}_m$ is 1. The incremental adaptation process is defined as follows:

1. **Initialization**: For all ℓ, set $\theta^{(\ell)} \leftarrow \theta^{(\ell)}_{\text{pretrained}}$.
2. **Stage** m **Update**: For each $\ell \in \mathcal{B}_m$, update $\theta^{(\ell)}$ via gradient descent on a data subset $\mathcal{D}_m \subseteq \mathcal{D}$, while keeping parameters in layers $\ell \notin \mathcal{B}_m$ frozen.

The stage-m objective is:

$$\min_{\{\theta^{(\ell)}: \ell \in \mathcal{B}_m\}} \mathcal{L}\Big(\mathcal{F}_\theta, \mathcal{D}_m\Big). \tag{7}$$

The overall training objective is defined as the cross-entropy loss:

$$\mathcal{L}(\theta; \mathcal{D}) = \textstyle\sum_{(x_i, y_i) \in \mathcal{D}} \sum_{t=1}^{T_i} -\log P_\theta(y_{i,t} \mid x_i, y_{i,1:t-1}) \tag{8}$$

where T_i is the target sequence length for the ith sample. During incremental adaptation, only the

Model	CQA	BQ	WG	PiQA	Arc_c	Arc_e	OBQA	HS	Avg. Acc	Throughput Improvement
Zero-Shot Results										
TinyLLaMA (Baseline)	20.15	56.02	59.35	72.63	32.68	**55.47**	**36.80**	**61.47**	49.32	NA
TinyLLaMA (Baseline+CT)	20.06	55.84	59.34	72.41	32.53	55.35	36.73	61.35	49.20	NA
ECHO-TinyLLaMA-25%-Shared-KV	**20.72**	**58.86**	**59.75**	**73.45**	**33.19**	54.12	36.40	59.01	**49.44**	**3.35%** ↑
ECHO-TinyLLaMA-50%-Shared-KV	**20.64**	**58.69**	58.96	**73.72**	31.74	53.54	35.00	59.01	48.91	**7%** ↑
5-Shot Results										
TinyLLaMA (Baseline)	19.0	62.11	62.12	74.65	36.43	**69.23**	38.20	62.19	**52.98**	NA
TinyLLaMA (Baseline+CT)	18.88	62.10	62.08	**74.68**	36.41	69.20	**38.21**	**62.20**	52.97	NA
ECHO-TinyLLaMA-25%-Shared-KV	**19.10**	**63.41**	**62.16**	74.67	**36.50**	68.14	36.89	59.35	52.53	**3.35%**↑
ECHO-TinyLLaMA-50%-Shared-KV	18.10	**62.39**	61.48	73.78	36.35	66.20	37.00	59.93	51.79	**7%**↑

Table 1: Zero-shot and 5-shot evaluation results for different models across multiple benchmarks. The last column reports the test throughput improvement compared to the baseline in terms of generated tokens per seconds. NA means not applicable. CT means continual training on the same dataset as ECHO versions. Additional evaluation results on larger models and more diverse datasets are included in Appendix A.

parameters in the current block $\mathcal{B}_m$ are updated. Algorithm 1 outlines the incremental adaptation strategy used to transform a pre-trained LLaMA model into the ECHO-LLaMA architecture. Starting with the pretrained model $\mathcal{F}_{\theta_{\text{pre}}}$, we define an adaptation range

$$\mathcal{R} = \{\ell \mid \ell = L, L-1, \ldots, N\},$$

where L is the total number of layers and N is the target layer at which adaptation stops. In general, each layer $\ell \in \mathcal{R}$ can be grouped as a block $\mathcal{B}_m$ in a block-wise incremental adaptation strategy. For each layer (or block $\mathcal{B}_m$) in $\mathcal{R}$, the standard self-attention is converted into cross-attention by introducing global shared KV matrices. The parameters $\theta^{(\ell)}$ of the converted layer are then updated by minimizing the loss function

$$\mathcal{L}\Big(\mathcal{F}_\theta, \mathcal{D}_\ell\Big)$$

for $S \approx 150$ steps using a token budget of $\mathcal{T}_{\text{stage}} \approx 4$M tokens. This incremental update enables the model to effectively adapt to the new cross-attention mechanism while mitigating catastrophic forgetting (Kirkpatrick et al., 2017).

Once all layers in $\mathcal{R}$ have been incrementally adapted, the entire model $\mathcal{F}_\theta$ undergoes a final fine-tuning phase for one epoch on a larger token set ($\mathcal{T}_{\text{final}} \approx 4$B tokens). This final phase allows the model to further refine its representations, stabilize training dynamics, and enhance overall generalization for downstream tasks. By incrementally updating layers (or blocks) with approximately 150 training steps per stage, the method achieves a favorable balance between computational cost and performance, preserving critical information flow and ensuring robust cross-attention mechanisms.

2.1.1 Memory Footprint

In a standard LLaMA model, each layer ℓ has its own key and value projection matrices, incurring a memory cost proportional to:

$$L \times (2d^2).$$

where p is sharing ratio. For ECHO-LLaMA, with partial KV sharing, the cost becomes:

$$(1-p)L \times (2d^2) + 2d^2.$$

Thus, the ratio of KV memory usage in ECHO-LLaMA versus the baseline is:

$$\frac{(1-p)L \times (2d^2) + 2d^2}{L \times (2d^2)} = (1-p) + \frac{1}{L}.$$

For large L, this ratio approximates $1-p$.

3 Experiments

Benchmarks- The evaluation benchmarks used to assess the performance of ECHO-TinyLLaMA [1] span a wide range of natural language understanding tasks, ensuring comprehensive coverage of different linguistic and reasoning challenges: CommonsenseQA (CQA), BoolQ (BQ), Winogrande (WG), PiQA, ARC_c and Arc_e (challenge and easy), OpenBookQA (OBQA), and HellaSwag (HS). These benchmarks collectively test the model's strengths in commonsense reasoning, linguistic understanding, and scientific knowledge application.

TinyLLaMA MFU- The MFU for the original TinyLLaMA and ECHO-TinyLLaMA was evaluated on different configuration of Nvidia-V100

[1] Converting from TinyLLaMA-1.1B https://huggingface.co/TinyLlama/TinyLlama_v1.1

Model	Train Loss		MFU (%) NPU-910B			MFU (%) V100 (SDPA)		
	NPU	GPU	1	4	8	1	4	8
LLaMA-125M (base)	5.25	5.06	14.35	9.19	8.33	23.59	22.30	21.96
ECHO-LLaMA-125M	5.28	5.08	14.49	9.62	9.45	24.32	22.73	22.25
TinyLLaMA (base)	4.65	4.97	24.17	22.11	22.19	34.34	34.33	LSE
ECHO-TinyLLaMA	4.45	4.67	27.10	27.98	22.51	39.32	34.42	LSE
LLaMA-3B (base)	4.25	4.68	29.36	32.21	30.89	OOM	35.31	34.01
ECHO-LLaMA-3B	4.15	4.70	30.78	48.31	46.34	OOM	37.34	36.05
LLaMA-7B (base)	3.90	4.56	OOM	38.78	35.07	OOM	OOM	31.67
ECHO-LLaMA-7B	3.34	4.00	35.58	LSE	35.53	OOM	OOM	35.74

Table 2: Unified comparison across NPU-910B and V100 (sequence length = 2048). MFU is reported for 1, 4, and 8 devices; OOM = out-of-memory, LSE = loss-scale error. All models were trained under an equal budget; best numbers per row are bold in the source data.

GPUs and Ascend-910B NPUs, different batch sizes, fixed sequence length of 2048, and 100 training steps. We use the MFU calculation script from LLaMA-Factory repository (Zheng et al., 2024) [2]. However, in case of the device types (GPU or NPU), this script needs several modification before computing MFU, including setting precision to fp16, increasing the number of workers for data pre-processing, setting the finetuning_type to full, and updating the theoretical FLOPs based on your computing devices.

Language Model Evaluations- We used LM-Harness repository as an evaluation tool to assess the language performance of different ECHO-LLaMA architecture (Gao et al., 2024). [3]

To evaluate the effectiveness of our incremental training strategy, we incrementally applied the training on approximately 4 billion tokens [4] , initializing the ECHO-TinyLLaMA models from the baseline weights (hug). We experiment for flexibly growing KV sharing architectures allowing the cross-decoders to be used for 25% and 50% of the layers in contrast to a fixed sharing size of YOCO. For 25% configuration, only the last 25% of layers are converted into cross-decoder layers. In the case of the 50% configuration, the entire second half of the layers are transitioned into cross-decoders.

The evaluation results, presented in Table 1, demonstrate that **ECHO-TinyLLaMA with 25% shared cross-decoders consistently outperforms the baseline** in both zero-shot and 5-shot settings, achieving the highest average accuracy in the zero-shot case and performing competitively in the 5-shot scenario. Notably, **ECHO-TinyLLaMA with 50% cross-decoders** closely matches the baseline performance in both evaluation setups, with an average accuracy gap of less than 0.5% in the zero-shot setting and about 1.2% in the 5-shot case. Despite this negligible drop, the 50% configuration delivers a **7% throughput improvement**, highlighting its effectiveness in balancing generation speed with competitive accuracy. To rule out the possibility that the gains are due to training length or configuration, we ran a continual training (CT) experiment on the baseline using the same dataset as ECHO-LLAMA. As shown in Table 1, CT provided no meaningful improvements and in some tasks even slightly reduced performance.

Appendix A includes more results on larger models and more diverse datasets. Moreover, Appendix A.3 provides a comparison results showing that the incremental strategy outperforms the full-stage approach, where all designated layers are converted to shared-KV at once.

Efficiency Comparison-Table 2 provides a detailed comparison between ECHO and non-ECHO versions of LLaMA models of four sizes of parameters: 125M, 1.1B, 3B, and 7B. The experiments were conducted on two devices: Ascend-910B [5] and Nvidia V100, each using up to 8 devices. The comparison metrics include training loss at equal train steps and training MFU. From the results, we observe the following trends: **NPU-910B**: The ECHO-LLaMA models consistently outperform the non-ECHO counterparts across all LLM sizes. ECHO-LLaMA models achieve lower loss values and up to 15% improvements in MFU, as

[2]Please see cal_mfu.py in their repository.

[3]We acknowledge that a direct comparison with YOCO models would be valuable; however, no official checkpoints have been released publicly, preventing such an evaluation. A direct comparison with YOCO models was not possible, as no official checkpoints for YOCO have been released publicly.

[4]Taken from https://huggingface.co/datasets/cerebras/SlimPajama-627B/tree/main/train

[5]Fused Attention is employed as an attention acceleration mechanism

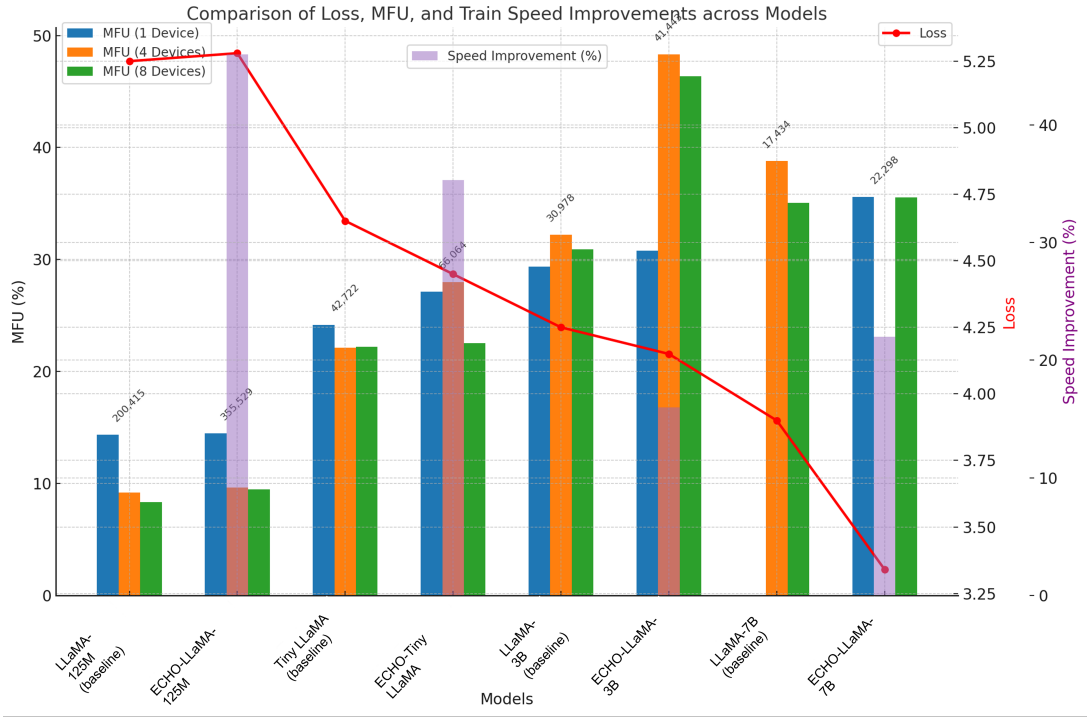


Figure 2: Comparison between of training throughput (Tokens/sec) and final loss for ECHO-LLaMA models and their baselines. ECHO versions consistently achieve lower loss with higher or comparable Tokens/sec speed. Each model is pretrained from scratch on 4B tokens through 1000 steps. Values on top each set of bars shows train token/second throughput.

evidenced by reductions in *Train Speed sec/step* and increases in *Train Token/sec*. For example, ECHO-LLaMA-3B improves the MFU by 15%. **GPU-V100**: The ECHO versions are comparable to or better than the non-ECHO counterparts in training loss. Additionally, ECHO-LLaMA models generally achieve higher training MFU, up to 4%.

To further examine the effectiveness of KV-sharing strategies, we implemented a YOCO variant of TinyLLaMA by converting the pretrained TinyLLaMA into a YOCO structure (sharing KV across the last half of layers) and then continually training it on 4B tokens using the same dataset as ECHO. As shown in Table 4, YOCO-TinyLLaMA underperforms compared to both the baseline TinyLLaMA and our ECHO-TinyLLaMA variants. In contrast, ECHO consistently improves accuracy across most benchmarks in both zero-shot and 5-shot settings, indicating that incremental adaptation via ECHO provides a more effective approach than YOCO-style sharing at this scale.

To evaluate the impact of ECHO on long-context performance, we further conducted experiments on the LongBench-V2 benchmark using LLaMA-3.2-8B and its ECHO variants. The zero-shot results are reported in Table 3. As reported in the table, with 25% KV sharing, the scores across all difficulty levels remain almost identical to LLaMA-3.2-8B, indicating that ECHO can be applied without sacrificing long-context capability. At 50% sharing, we observe modest drops, particularly on the medium and long subsets, suggesting that more aggressive sharing introduces some trade-offs in handling extended sequences. Additional evaluations on coding and other benchmarks are provided in Table 5 and Table 6 in the Appendix.

4 Discussion

One of the main advantages of ECHO-LLaMA over YOCO models is its flexibility in choosing a balance between attention layers and cross-attention layers based on end-user needs. Unlike YOCO models, which rigidly apply shared-KV caching to a fixed subset of layers, ECHO-LLaMA allows dynamic allocation of self-attention and cross-attention mechanisms. This adaptability enables fine-tuning for different deployment scenarios—whether prioritizing efficiency on resource-constrained edge devices or maximizing performance in high-compute environments. Additionally, the ability to configure attention structures

Model	Easy (%)	Hard (%)	Short (%)	Medium (%)	Long (%)
LLaMA-3.2-8B	30.6	29.5	35.1	27.8	25.7
ECHO-LLaMA-3.2-8B-25%	29.7	28.8	35.1	27.5	25.7
ECHO-LLaMA-3.2-8B-50%	28.4	26.7	33.6	26.1	23.8

Table 3: Zero-shot performance of LLaMA-3.2-8B and ECHO variants on LongBench-V2. Results are reported as accuracy (%).

Model	CQA	BQ	WG	PiQA	Arc-c	Arc-e	OBQA	HS	Avg. Acc
Zero-Shot Results									
TinyLLaMA (Baseline)	20.15	56.02	59.35	72.63	32.68	**55.47**	**36.80**	**61.47**	49.32
YOCO-TinyLLaMA	18.23	53.65	55.19	68.88	28.56	51.43	33.73	57.97	45.95
ECHO-TinyLLaMA-25% KV	**20.72**	**58.86**	**59.75**	**73.45**	**33.19**	54.12	36.40	59.01	**49.44**
ECHO-TinyLLaMA-50% KV	**20.64**	**58.69**	58.96	**73.72**	31.74	53.54	35.00	59.01	48.91
5-Shot Results									
TinyLLaMA-1.1B (Baseline)	19.00	62.11	62.12	74.65	36.43	**69.23**	**38.20**	**62.19**	**52.98**
YOCO-TinyLLaMA	17.25	59.71	58.80	71.45	34.63	63.97	35.12	55.32	49.53
ECHO-TinyLLaMA-25% KV	**19.10**	**63.41**	**62.16**	**74.67**	**36.50**	68.14	36.89	59.35	52.53
ECHO-TinyLLaMA-50% KV	18.10	**62.39**	61.48	73.78	36.35	66.20	37.00	59.93	51.79

Table 4: Comparison of TinyLLaMA, YOCO-TinyLLaMA, and ECHO-TinyLLaMA in zero-shot and 5-shot settings across multiple benchmarks (accuracy).

per task allows ECHO-LLaMA to optimize both inference speed and long-range dependency modeling, making it a more versatile solution across diverse workloads. Another key advantage of ECHO-LLaMA over YOCO models is its cost-effective adaptation strategy. While YOCO models require training from scratch or extensive modifications to integrate their fixed KV-sharing mechanism, ECHO-LLaMA leverages incremental adaptation to efficiently transform a pretrained LLaMA model into its structured format. This eliminates the need for expensive full-scale pretraining while maintaining—or even surpassing—baseline performance. Our experiments on TinyLLaMA demonstrate this efficiency, where a 25% KV-sharing configuration slightly outperformed the baseline in language modeling, and a 50% KV-sharing configuration achieved comparable performance, all while significantly improving inference speed and KV cache efficiency.

The 77% speed-up shown in Figure 2 (from 200k to 355k tokens/sec) corresponds to the ECHO-LLaMA-125M model, a relatively small-scale system with 125M parameters. At this scale, KV projection and caching account for a large share of both memory and compute relative to Feed Forward modules (FFs), making KV-sharing highly impactful for throughput. As model size grows, however, FFs dominate in parameters and computation, reducing the relative benefit of KV-sharing. This trend is reflected in our MFU and throughput results across model sizes, as reported in Table 2 and Appendix B.

5 Conclusion

We present ECHO-LLaMA, a flexible cross-layer KV-sharing framework designed to address key inefficiencies in LLM training and inference. Unlike prior methods that rely on fixed architectures, ECHO-LLaMA enables seamless adaptation of existing pretrained LLaMA models through a layer-wise incremental training strategy. Evaluated across model sizes from 125M to 7B parameters, our approach delivers up to 50% faster training and 7% higher throughput at comparable or better language modeling performance. The 25% and 50% cross-decoder sharing configurations highlight the scalability–efficiency trade-offs available to practitioners.

Limitations

The proposed framework relies on the availability of pre-trained models, which inherently constrains its applicability to scenarios where such models are accessible. Additionally, although the ECHO-LLaMA adaptation reduces computational overhead and improves inference speed, the reliance on shared key-value caching may introduce challenges for extremely long sequence tasks, where memory bottlenecks could still occur. Furthermore, the evaluation primarily focuses on zero/few shot performance and model efficiency, leaving broader generalization capabilities (e.g., domain adaptation or multi-task learning) unexplored.

References

TinyLlama/TinyLlama_v1.1 · Hugging Face — huggingface.co. https://huggingface.co/TinyLlama/TinyLlama_v1.1. [Accessed 27-01-2025].

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127.

Walid Ahmed, Habib Hajimolahoseini, Austin Wen, and Yang Liu. 2023. Speeding up resnet architecture with layers targeted low rank decomposition. *arXiv preprint arXiv:2309.12412*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and 1 others. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

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

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation.

Habib Hajimolahoseini, Walid Ahmed, and Yang Liu. 2023. Training acceleration of low-rank decomposed networks using sequential freezing and rank quantization. *arXiv preprint arXiv:2309.03824*.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2021a. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021b. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021c. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao

Fu, Maosong Sun, and Junxian He. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*.

Farnoosh Javadi, Walid Ahmed, Habib Hajimolahoseini, Foozhan Ataiefard, Mohammad Hassanpour, Saina Asani, Austin Wen, Omar Mohamed Awad, Kangling Liu, and Yang Liu. 2023. Gqkva: Efficient pre-training of transformers by grouping queries, keys, and values. *arXiv preprint arXiv:2311.03426*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and 1 others. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172.

Bingli Liao and Danilo Vasconcellos Vargas. 2024. Beyond kv caching: Shared attention for efficient llms. *arXiv preprint arXiv:2407.12866*.

Spyros Makridakis, Fotios Petropoulos, and Yanfei Kang. 2023. Large language models: Their success and impact. *Forecasting*, 5(3):536–549.

Silvia Milano, Joshua A McGrane, and Sabina Leonelli. 2023. Large language models challenge the future of higher education. *Nature Machine Intelligence*, 5(4):333–334.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

Hossein Rajabzadeh, Aref Jafari, Aman Sharma, Benyamin Jami, Hyock Ju Kwon, Ali Ghodsi, Boxing Chen, and Mehdi Rezagholizadeh. 2024. Echoatt: Attend, copy, then adjust for more efficient large language models. *arXiv preprint arXiv:2409.14595*.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2024. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2023. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. 2024. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

Appedix

A More Evaluation on ECHO Architecture

To further validate the effectiveness and generalizability of our proposed ECHO mechanism, we applied it to multiple publicly available models with LLaMA-style architectures. Table 5 presents a detailed comparison of baseline models and their ECHO-enhanced counterparts across a broad range of standard evaluation benchmarks, including MMLU (Hendrycks et al., 2021b,a), C-Eval (Huang et al., 2023), GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021c), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021; Touvron et al., 2023).

We examine LLaMA2-7B (Touvron et al., 2023). The results show that applying ECHO with 25% shared cross-decoder layers leads to a consistent improvement in average performance (22.56 vs. 22.15), with minor gains across multiple tasks. However, pushing to 50% shared layers introduces degradation (average drops to 21.26), suggesting that over-sharing can hurt task-specific expressivity at this scale.

Given that the Qwen models (Bai et al., 2023) adopt a transformer architecture closely aligned with LLaMA—including the use of rotary positional embeddings and RMSNorm—we extended the ECHO framework to Qwen-1.8B and Qwen-7B as well. For Qwen-1.8B, applying ECHO with 25% shared decoders marginally improves the average score (27.8 vs. 27.5), while the 50% configuration results in performance decline. Similar trends hold for Qwen-7B, where the 25% variant achieves the best overall average (41.31), modestly surpassing the baseline, while the 50% configuration again incurs noticeable drop-offs.

These results collectively reinforce two key findings: (1) ECHO reliably improves efficiency without sacrificing performance when applied conservatively (e.g., 25% sharing), and (2) the technique is transferable across architectures that share foundational transformer principles, such as LLaMA and Qwen.

A.1 Training Efficiency and Performance Comparison of ECHO-LLaMA Models

To further illustrate the efficiency of the ECHO-LLaMA models, we compare the training throughput (Tokens/sec) and the final loss for different model sizes. Figure 3 shows the performance comparison across all four LLM sizes (125M, 1.1B, 3B, and 7B) and their ECHO-LLaMA counterparts. The results demonstrate that the ECHO-LLaMA models consistently achieve lower final loss compared to the non-ECHO versions while achieving higher or comparable throughput (Tokens/sec). For instance, ECHO-LLaMA-125M improves the training throughput by a significant margin while maintaining a comparable loss, and ECHO-LLaMA-7B reduces the loss substantially with a moderate improvement in throughput. These findings highlight the effectiveness of the ECHO structure in achieving better optimization efficiency during training.

A.2 Performance Evaluation of ECHO-LLaMA Structure During Pretraining from Scratch

In this section, we evaluate the performance of our ECHO-LLaMA structure during pretraining from scratch for different sizes of large language models (LLMs): 125M, 1.1B, 3B, and 7B. The experiments are conducted on two hardware platforms: GPU (V100) and NPU (Ascend-910B). For each LLM size, we analyze the training dynamics using three types of plots:

1. **Train Loss per Steps**: Tracks the training loss as a function of the number of training steps.
2. **Train Loss per Train Time**: Evaluates the relationship between training loss and the elapsed training time.
3. **Train Loss per Number of Train Tokens**: Measures how efficiently the model learns as a function of the number of processed tokens.

To compare GPU and NPU performance across different LLM sizes, we organize the results in a structured figure layout. Figures 4, 5, 6, and 7 present the results, where Each model size has a separate figure, and each figure contains two rows:

- The **first row** contains plots for GPU (V100).
- The **second row** contains plots for NPU (910B).

Within each row, there are three subfigures, corresponding to the three types of plots. The left subfigure illustrate the training loss behavior in terms of training steps, the middle subfigure describes the training loss in terms of time, and the

Model	MMLU (5-shot)	C-Eval (5-shot)	GSM8K (8-shot)	MATH (4-shot)	HumanEval (0-shot)	MBPP (3-shot)	Avg.
LLaMA2-7B (baseline)	46.8	32.5	16.7	3.3	12.8	20.8	22.15
ECHO-LLaMA2-7B-25%	46.9	33.4	17.2	3.6	13.5	20.8	22.56
ECHO-LLaMA2-7B-50%	46.2	31.8	15.6	2.1	12.2	19.7	21.26
Qwen-1.8B (baseline)	45.3	56.1	32.3	2.3	15.2	14.2	27.5
ECHO-Qwen-1.8B-25%	45.8	56.3	32.6	2.4	15.5	14.7	27.8
ECHO-Qwen-1.8B-50%	44.1	54.7	30.8	1.8	13.9	12.7	26.33
Qwen-7B (baseline)	58.2	63.5	51.7	11.6	29.9	31.6	41.08
ECHO-Qwen-7B-25%	58.4	63.8	52.4	12.1	29.7	31.5	41.31
ECHO-Qwen-7B-50%	57.3	62.2	50.4	10.4	28.6	29.8	39.78

Table 5: Evaluating the generalizability of the ECHO mechanism across LLaMA and Qwen architectures on diverse reasoning and coding benchmark. The results are reported in terms of accuracy.

Model	MMLU (5-shot)	AGIEval-En (5-shot)	Arc-c (25-shot)	SQuAD (1-shot)	Avg.
LLaMA3.2-1B (baseline)	32.1	23.1	32.7	49.0	34.22
ECHO-LLaMA3.2-1B-25%	32.4	23.3	32.6	49.3	34.40
ECHO-LLaMA3.2-1B-50%	31.3	21.5	30.4	47.8	32.75
LLaMA3.2-3B (baseline)	58.0	39.1	69.1	67.7	58.47
ECHO-LLaMA3.2-3B-25%	58.3	39.2	69.0	67.6	58.52
ECHO-LLaMA3.2-3B-50%	56.7	37.3	68.1	65.7	56.95
LLaMA3.2-8B (baseline)	66.7	47.7	79.6	69.7	65.92
ECHO-LLaMA3.2-8B-25%	66.8	47.9	79.5	69.9	66.02
ECHO-LLaMA3.2-8B-50%	65.4	47.1	78.4	68.4	64.82

Table 6: Evaluating the generalizability of the ECHO mechanism across several LLaMA3.2 models on diverse benchmark. The results are reported in terms of accuracy.

right one shows the same loss in terms of number of training tokens. As the plots depicts, the ECHO versions suppress or compete with the baselines, particularly in 7B model size. This observation shows that by increasing the size of model, the impact of ECHO-LLaMA becomes more strong. This is because in larger model sizes, the memory and compute demands grow significantly, making efficient KV sharing crucial for reducing memory overhead and improving throughput.

A.3 Comparison of Incremental vs. Full-Stage Shared-KV Fine-Tuning

To evaluate the effectiveness of incremental strategy for applying shared key-value (KV) representations, we compare **Incremental Sharing**, where shared-KV layers are introduced gradually during training starting from the final layer, and **Full-Stage Sharing**, where a fixed subset of layers (25% or 50%) are converted to shared-KV at once and jointly fine-tuned.

Figures 8 and 9 present the results for zero-shot and 5-shot evaluations across diverse benchmark datasets. Incremental Sharing consistently outperforms Full-Stage Sharing at both 25% and 50% sharing ratios, indicating that incremental adaptation enables the model to better preserve pre-trained knowledge while integrating structural modifications. This improvement is most notable in datasets such as Arc-c and OBQA, where the performance gap exceeds 2%.

A.4 MFU-Loss-Speed

Figure 3 compares the training throughput (Tokens/sec), Model FLOPs Utilization (MFU), and final loss across various LLaMA models and their ECHO-LLaMA counterparts. The ECHO-LLaMA versions consistently demonstrate improvements in MFU for configurations with 1, 4, and 8 devices, while achieving lower or comparable final loss compared to their baselines. Notably, ECHO-LLaMA models, such as ECHO-LLaMA-125M and ECHO-TinyLLaMA, exhibit significant speed improvements, as indicated by the purple bars, while maintaining competitive or better loss values. These results explains the effectiveness of ECHO-LLaMA's architecture in enhancing both training efficiency and model performance.

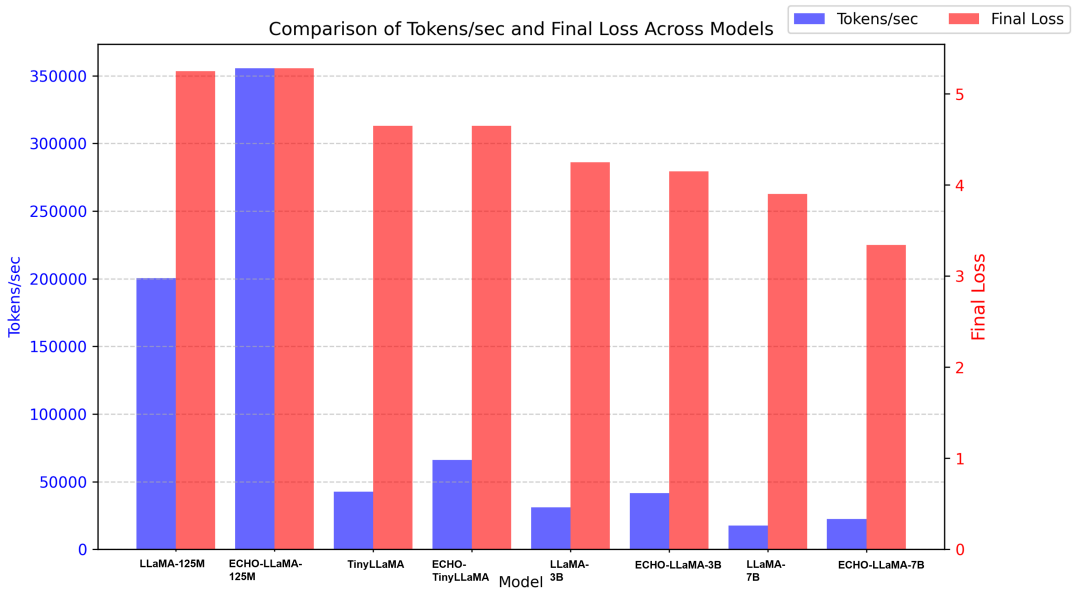


Figure 3: Comparison of training throughput (Tokens/sec) and final loss for ECHO and non-ECHO models on NPU-910B. ECHO-LLaMA versions consistently achieve lower loss with higher or comparable Tokens/sec speed.

A.5 Scaling Law Validation-

To validate the scalability of ECHO-LLaMA models, we conducted experiments across various model sizes (125M, 1.1B, 3B, and 7B) and compared the training losses with their baseline counterparts. Each model is pretrained on 4B tokens through 1000 training steps.

As shown in Figure 11, the ECHO-LLaMA models consistently achieve lower training loss than the baseline models as the model size increases. This observation demonstrates that ECHO architectures adhere to the scaling law while also offering enhanced training efficiency through their innovative design.

A.6 GPU Memory Efficiency of Echo-TinyLLaMA-

To evaluate the efficiency of ECHO-TinyLLaMA, we compare its GPU memory consumption against the baseline TinyLLaMA across various sequence lengths. As shown in Figure 10, ECHO-TinyLLaMA achieves a substantial reduction in memory consumption, requiring almost two times less GPU memory than TinyLLama at all tested sequence lengths. For instance, at a sequence length of 16k tokens, ECHO-TinyLLaMA consumes only 6.4 GB, whereas TinyLLaMA requires 14 GB, resulting in a **2.19x** reduction. This trend continues across longer sequences, with the reduction reaching up to **2.33x** at 32k tokens. Such improvements are particularly critical for enabling longer-context processing on resource-constrained hardware.

A.7 Ablation Study: Selecting the Optimal Number of Training Steps per Stage-

we conducted an ablation study measuring the effect of different training steps S on training loss. The goal was to assess at what point additional steps provide diminishing returns in loss reduction, thereby justifying our selection of 150 training steps. We evaluated training loss for ECHO-TinyLLaMA across six different layers (ranging from layer 22 to layer 12) at increasing training steps: 25, 50, 100, 150, 200, and 300. The loss values at each step were recorded to observe the rate of improvement. The plot in Figure 12 illustrates the training loss progression for each layer as the number of training steps increases. We observe a sharp decrease in loss from 25 to 150 steps, indicating effective learning during this phase. However, beyond 150 steps, the slope of loss reduction significantly flattens, suggesting diminishing returns with additional training. For instance, at 150 steps, Layer 22's loss decreases from 2.8 (25 steps) $\rightarrow$ 1.97 (150 steps), a significant drop. Extending training to 200 or 300 steps results in only marginal improvements (1.85 at 200 steps, 1.80 at 300 steps), making the additional cost inefficient.

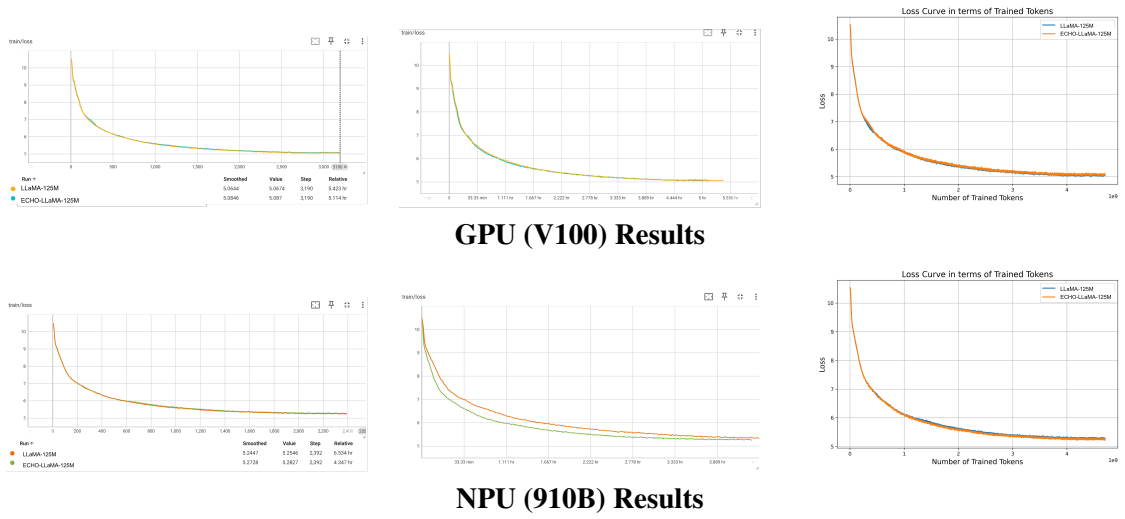


Figure 4: Training results for the LLaMA-125M LLM on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

A.8 Ablation Study on Block Size for Layer Adaptation

To understand the impact of block size in incremental layer adaptation, we conducted an ablation study by varying the number of layers updated in each block. As shown in Table 7, a smaller block size leads to better performance. Specifically, setting the block size to 1 consistently yields the highest average accuracy in both 25% and 50% shared-KV configurations. This supports our hypothesis that gradually converting one layer at a time minimizes disruption to the pretrained weights, reduces information loss, and helps the model avoid catastrophic forgetting. In contrast, larger block sizes (2 or 3) degrade performance and require more training steps to recover.

Table 7: Ablation study on block size during incremental adaptation for TinyLLaMA-1.1B-V1.1. We report zero-shot average accuracy across multiple tasks. Block size 1 consistently outperforms larger sizes.

Zero-shot, 25% Shared-KV		
Block Size	**Avg. Accuracy**	**Observations**
1	**49.44**	Best performance; stable adaptation
2	48.73	Slight degradation; slower recovery
3	46.67	Noticeable drop; higher forgetting

Zero-shot, 50% Shared-KV		
Block Size	**Avg. Accuracy**	**Observations**
1	**52.53**	Best performance; stable adaptation
2	50.49	Slight degradation; slower recovery
3	47.19	Noticeable drop; higher forgetting

B Guidelines for Selecting Cross-Attention Layer Ratio

To balance the trade-off between model efficiency and effectiveness, we provide empirical guidelines for selecting the percentage of layers using shared-KV in ECHO-LLaMA. As shown in Table 8, based on recent experiments using LLaMA-7B with a sequence length of 4096, there exists a clear trade-off between the percentage of shared layers and key performance metrics such as MFU, elapsed time, and total FLOPs.

Moderate KV-sharing ratios (e.g., 18–25%) consistently offer the best trade-off, achieving high normalized MFU and reduced training time and FLOPs without substantial degradation in model throughput. In contrast, very high sharing levels (e.g., 75% and above) lead to diminishing returns or degraded utilization, and 50% sharing shows suboptimal MFU due to load imbalance during KV sharing. These insights provide practical guidance for industry-scale deployments where efficiency and performance must be balanced.

C Questions / Answers

This section addresses several questions relevant to the research.

TinyLLaMA-1.1B Model Training Results

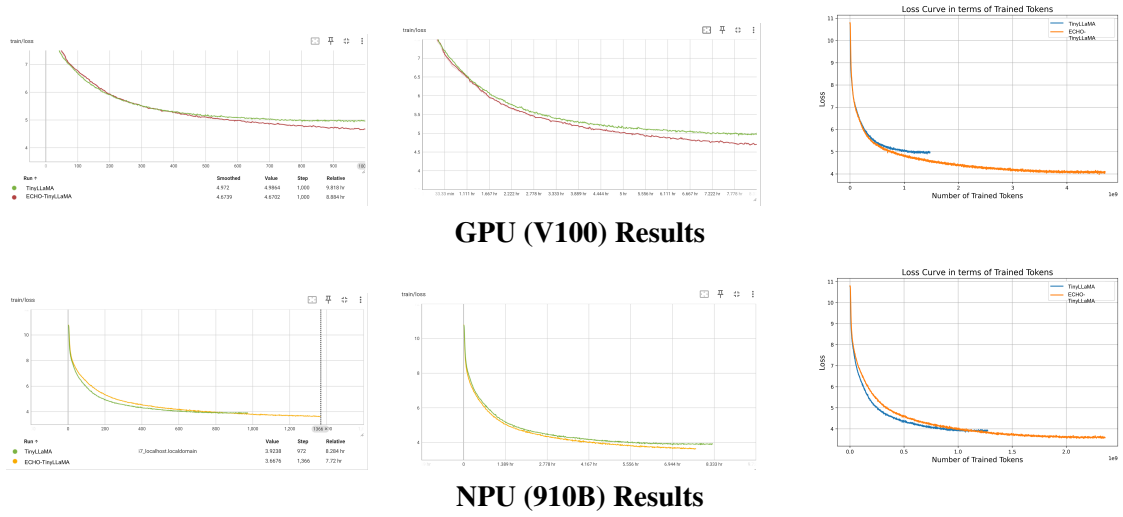


Figure 5: Training results for TinyLLaMA-1.1B LLM on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

Table 8: Trade-off analysis on LLaMA-7B (seq_len = 4096) for varying cross-attention layer sharing ratios. Metrics include actual and normalized MFU, elapsed time, and model FLOPs. Normalization is done relative to the 0%-sharing baseline.

KV Sharing (%)	# Shared Layers	MFU (%)	Elapsed Time (s)	Model FLOPs	Norm. MFU	Norm. Time	Norm. FLOPs
0	0	62.450	104.140	1.965	0.000	1.000	1.000
3	1	63.246	102.620	1.957	0.948	0.881	0.968
6	2	63.438	101.870	1.949	0.957	0.822	0.937
9	3	63.613	101.150	1.940	0.965	0.765	0.901
15	5	64.098	99.500	1.932	0.987	0.636	0.870
18	6	**64.390**	**99.270**	**1.923**	**1.000**	**0.618**	**0.834**
21	7	64.160	99.250	1.915	0.990	0.616	0.802
25	8	63.900	99.090	1.906	0.978	0.604	0.767
28	9	63.461	99.180	1.898	0.958	0.611	0.735
50	16	62.137	98.140	1.839	0.897	0.529	0.502
75	24	63.494	92.070	1.763	0.959	0.053	0.202
85	27	63.039	91.400	1.737	0.938	0.001	0.099
99	31	62.131	91.390	1.712	0.897	0.000	0.000

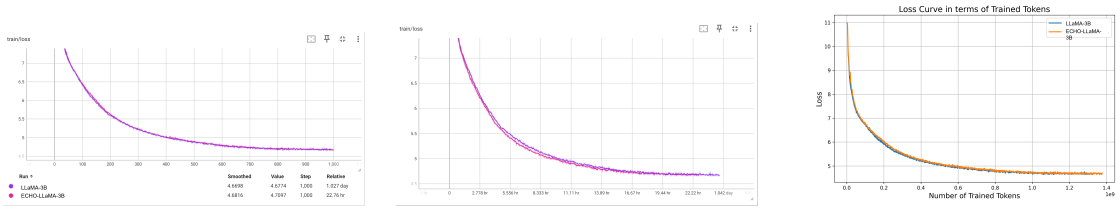
C.1 Q1. Why is there no direct comparison with YOCO models?

First, no official checkpoints for YOCO models have been released publicly, and a full pretraining of YOCO models from scratch is beyond the scope of this work. Second, while YOCO and ECHO-LLAMA both adopt a shared-KV mechanism, their methodologies differ fundamentally. YOCO requires a heavy pretraining phase from scratch, whereas ECHO-LLAMA proposes a lightweight fine-tuning approach that transforms existing pretrained models into ECHO architectures. For these reasons, we did not include a direct comparison with YOCO in our evaluation.

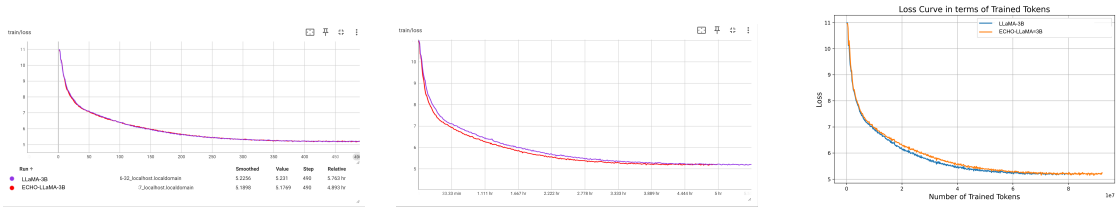
C.2 Q2. Why does the training throughput speedup degrade significantly beyond a certain model size threshold?

The reduced speedup observed for models exceeding a certain size threshold is primarily due to the increasing dominance of non-KV components—such as MLP layers—in the overall compute cost. As the model size grows, the proportion of parameters and computation attributed to these non-shared components becomes significantly larger than that of the shared KV layers. Consequently, the relative benefit of KV-sharing diminishes, leading to a lower overall throughput speedup.

LLaMA-3B Model Training Results



GPU (V100) Results



NPU (910B) Results

Figure 6: Training results for LLaMA-3B on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

D What is the motivation for sharing in the range $[L/2, L]$?

In ECHO-LLAMA, KV-sharing is always applied in a top-down pattern, i.e. incrementally converting layers into shared-KV starting from the last transformer layer. Specifically, in a model with L total layers, we begin converting the topmost layer ($\#L$) to the shared-KV format, followed by layer $L-1$, and so on. The process continues until a user-defined cutoff layer N is reached. In this paper, we experimented with two practical configurations: sharing the last 25% and the last 50% of layers (i.e., $N = L/4$ and $N = L/2$, respectively). These settings reflect meaningful trade-offs between memory throughput and performance, and were selected to showcase the effectiveness of our method. However, the approach is flexible and supports other thresholds (e.g., $L/3$, $L/6$) depending on the deployment constraints. Moreover, we added an ablation study showing that the top-down pattern works the best, in comparison with bottom-up pattern. Table 9 shows the results for 25% and 50% KV-sahring, explaining that the the Top-Down pattern performs better than the Bottom-Up pattern. The Bottom-Up pattern starts sharing-KV from a lower layer and continue upward. All the results are conducted on TinyLLaMA1.1B-V1.1.

E Model Configurations for LLaMA-125M and LLaMA-3B

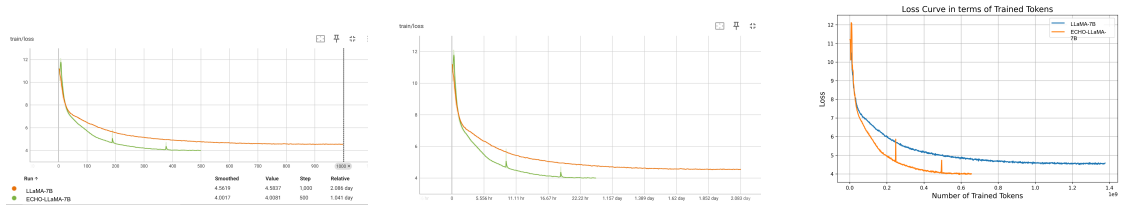
We provide detailed configuration settings for the LLaMA-125M and LLaMA-3B models, which were developed solely for this research. These configurations are designed by following the architectural patterns of TinyLLaMA, LLaMA-7B, and other higher versions of the LLaMA family. We adhered to the scaling strategy used in LLaMA models, ensuring proportional ratios for hidden size, intermediate size, number of layers, and attention heads as the models scale up.

E.1 LLaMA-125M Configuration

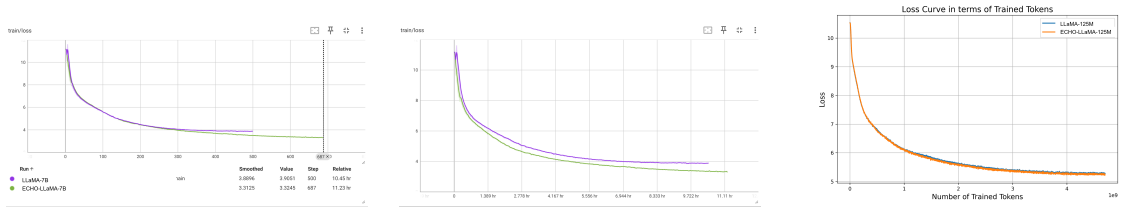
The following configuration outlines the settings for the LLaMA-125M model:

```
{
  "architectures": ["ECHOLLaMAForCausalLM"],
  "hidden_size": 768,
  "intermediate_size": 2048,
  "num_hidden_layers": 12,
  "num_attention_heads": 12,
  "max_position_embeddings": 2048,
  "vocab_size": 32000,
  "rotary_emb_base": 10000,
  "tie_word_embeddings": False,
  "use_cache": True,
  "layer_norm_epsilon": 1e-5,
  "init_std": 0.02,
  "torch_dtype": "float16",
  "model_type": "echo_llama",
  "pad_token_id": None,
  "bos_token_id": 1,
  "eos_token_id": 2
}
```

LLaMA-7B Model Training Results



GPU (V100) Results



NPU (910B) Results

Figure 7: Training results for LLaMA-7B on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

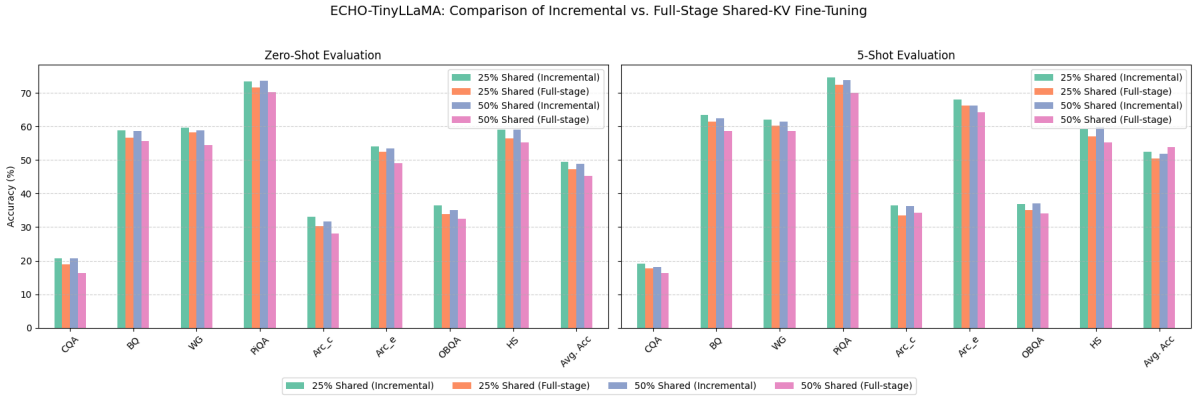


Figure 8: **LLaMA-7B**: Comparing Incremental Sharing over Full-Stage Sharing for shared-KV adaptation.

E.2 LLaMA-3B Configuration

The following configuration outlines the settings for the LLaMA-3B model:

```
{
  "architectures": ["ECHOLLaMAForCausalLM"],
  "hidden_size": 3072,
  "intermediate_size": 8192,
  "num_hidden_layers": 26,
  "num_attention_heads": 24,
  "max_position_embeddings": 2048,
  "vocab_size": 32000,
  "rotary_emb_base": 10000,
  "tie_word_embeddings": False,
  "use_cache": True,
  "layer_norm_epsilon": 1e-5,
  "init_std": 0.02,
  "torch_dtype": "float16",
  "model_type": "echo_llama",
  "pad_token_id": None,
  "bos_token_id": 1,
  "eos_token_id": 2
}
```

E.3 Notation

Table 10 summarizes the key notations used throughout the paper. These notations are consistent with standard transformer-based LLM literature and are used to describe our cross-layer KV-sharing architecture and training procedure.

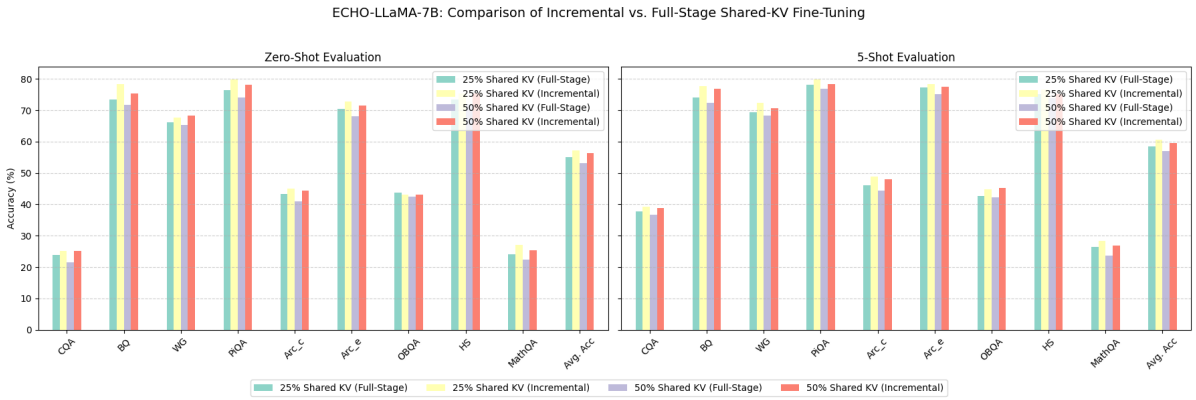


Figure 9: **LLaMA-7B**: Comparing Incremental Sharing over Full-Stage Sharing for shared-KV adaptation.

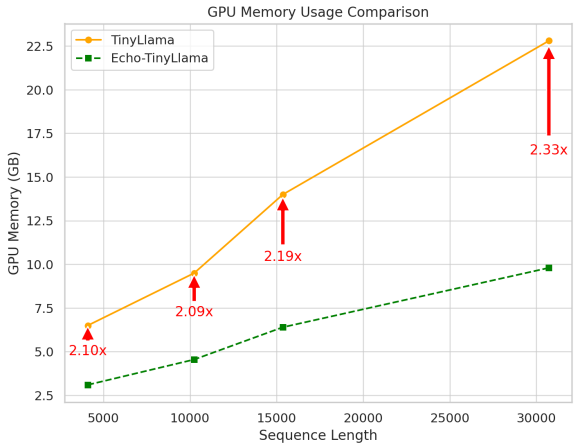


Figure 10: GPU memory usage comparison between the baseline TinyLLaMA and ECHO-TinyLLaMA across different sequence lengths. ECHO-TinyLLaMA consistently consumes nearly half the memory compared to TinyLLaMA, as indicated by the annotated reduction ratios.

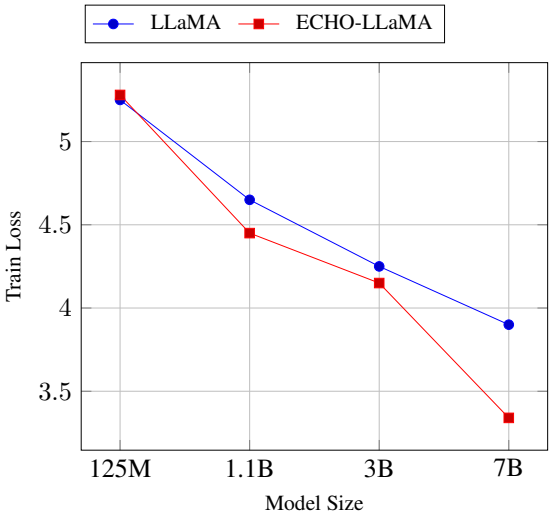


Figure 11: Scaling law diagram comparing training loss for ECHO-LLaMA of different sizes against the baselines.

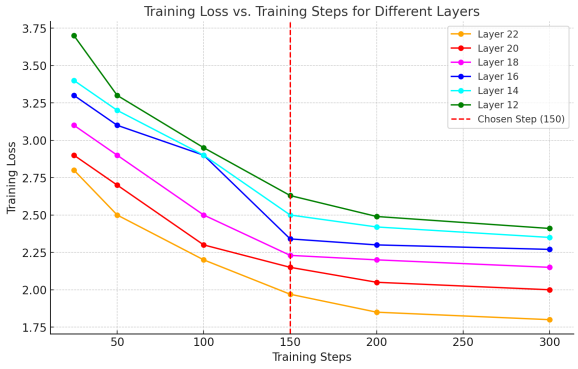


Figure 12: Training Loss vs. Training Steps for Different Layers in ECHO-TinyLLaMA. The loss decreases significantly up to 150 steps, after which the rate of improvement diminishes. This justifies our selection of $S \approx 150$ training steps as the optimal point for balancing computational cost and performance.

Model	CQA	BQ	WG	PiQA	Arc_c	Arc_e	OBQA	HS	Avg. Acc
Top-Down (25%-Shared-KV)	20.02	55.18	56.31	68.01	30.25	55.18	30.84	44.35	45.02
Bottom-Up (25%-Shared-KV)	19.11	54.35	54.84	67.35	30.04	54.67	30.26	43.96	44.32
Top-Down (50%-Shared-KV)	18.13	53.45	55.71	66.34	28.63	54.97	30.74	44.83	44.09
Bottom-Up (50%-Shared-KV)	17.05	52.11	53.85	65.21	27.18	53.67	29.48	43.51	42.75

Table 9: Comparison of top-down vs. bottom-up shared-KV patterns on various language understanding benchmarks. Accuracy (%) is reported for each dataset, and the average accuracy is shown in the last column.

Notation	Description
L	Total number of layers in the model
N	Threshold layer index where KV-sharing starts (user-defined stopping layer)
X_{l-1}	Input to layer l
X'_l	Output of the self-attention sublayer at layer l (before feedforward)
W_Q, W_K, W_V	Query, key, and value projection matrices
d	Model hidden size
d_k	Dimensionality of key/query vectors (d/h)
h	Number of attention heads
p	Proportion of layers converted to shared-KV (e.g., 0.25 or 0.5)
$\mathrm{SFF}(X)$	SiLU-activated feed-forward module
$K_{\text{shared}}, V_{\text{shared}}$	Shared global key and value matrices
$W^K_{\text{global}}, W^V_{\text{global}}$	Projection matrices for computing shared key and value
$\mathcal{L}$	Set of layers adapted into cross-decoders in incremental adaptation
$\mathcal{L}_m$	Block of layers updated in stage-m
$\theta^{(\ell)}$	Parameters of layer ℓ
$\mathcal{M}_\theta$	Model parameterized by θ
$\mathcal{D}$	Training dataset
S	Number of training steps per incremental stage
$\mathcal{B}_{\text{stage}}$	Token budget per adaptation stage
$\mathcal{B}_{\text{final}}$	Token budget for final fine-tuning

Table 10: Notation used throughout the paper.