Efficient Learning for Large Language Models

by

Hossein Rajabzadeh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2026

© Hossein Rajabzadeh 2026

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor:	Hyock Ju Kwon Professor, Dept. of Mechanical and Mechatronics Engineering University of Waterloo
Internal Member:	Stewart McLachlin Assistant Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo Yue Hu Assistant Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo
Internal-External Member:	Ali Ghodsi Professor, Dept. of Statistics and Actuarial Science University of Waterloo
External-External Member:	Il-Min Kim Professor, Department Head of Electrical and Computer Engineering Queen's University

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This dissertation is primarily based on my published and ongoing research contributions in efficient learning and inference for large language models. I am the lead author on all chapters except **ECHO-LLaMA**, for which I contributed equally with the first author; my contributions include developing the main idea, implementing the method, conducting the experiments, and co-writing the paper.

Abstract

Artificial Intelligence (AI) systems have become indispensable across domains such as healthcare, finance, robotics, and scientific discovery. At the heart of this revolution, **Large Language Models (LLMs)** have emerged as the central paradigm, demonstrating remarkable reasoning, generalization, and multi-domain adaptability. However, their exponential growth in scale introduces severe computational bottlenecks in **training, fine-tuning, and inference**, limiting accessibility, sustainability, and real-world deployment.

This dissertation advances the efficiency of LLMs across all lifecycle stages by introducing a suite of five frameworks that significantly reduce compute, memory, and latency costs with minimal or no loss in accuracy. First, **Quantized Dynamic Low-Rank Adaptation (QDyLoRA)** enables memory-efficient fine-tuning across multiple LoRA ranks in a single training pass, achieving competitive performance to QLoRA while reducing GPU memory usage by **up to 65%** and supporting flexible rank selection at inference time. Second, **Sorted-LoRA** introduces a stochastic depth–aware fine-tuning framework that co-trains multiple sub-models of varying depths within a single cycle. On LLaMA2–7B, it produces submodels up to **40% smaller** that retain over **98% task accuracy**, with the largest variant even surpassing the base model by **+0.34%**. Third, **LoRA-Drop** accelerates autoregressive inference by dynamically substituting computationally redundant layers with lightweight low-rank modules during decoding. It delivers up to **2.6× faster decoding** and a **50% reduction in KV-cache memory** with less than **0.5%** degradation in accuracy, offering latency-aware adaptability for real-world deployment. Fourth, **EchoAtt** exploits redundancy in attention maps by sharing attention matrices among similar layers. On TinyLLaMA–1.1B, it achieves **15% faster inference**, **25% faster training**, and a **4% parameter reduction** while improving zero-shot accuracy, highlighting that structural compression can enhance rather than degrade model generalization. Finally, **ECHO-LLaMA** introduces cross-layer Key–Value (KV) and Query–Key (QK) sharing to reduce redundant attention computation. This approach achieves up to **77% higher token-per-second throughput** during training, **16% higher Model FLOPs Utilization (MFU)**, and **7% higher test-time throughput**, while preserving language modeling performance. On the mechanical-domain RoboEval benchmark, ECHO-CodeLLaMA-7B boosts average accuracy from **62.15% to 63.01%** with only **50% KV sharing**, confirming its robustness in domain adaptation.

Together, these contributions form a coherent research program on the efficiency of large-scale Transformers. They demonstrate that intelligently exploiting representational redundancy—through quantization, low-rank structure, cross-layer sharing, and adaptive computation—can yield substantial compute savings with minimal trade-offs.

Acknowledgements

I would like to thank all those who helped make this thesis possible.

Dedication

This is dedicated to those I love.

Table of Contents

Examining Committee Membership ii

Author's Declaration iii

Statement of Contributions iv

Abstract v

Acknowledgements vi

Dedication vii

List of Figures xiv

List of Tables xvii

1 Introduction 1

1.1 Large Language Models 3

1.1.1 Defining Large Language Models 3

1.1.2 Characteristics and Benefits 5

1.1.3 Challenges in Training Large Language Models 5

1.2 Efficiency in Large Language Models: Definitions and Problems 6

1.2.1 Memory Efficiency in Fine-Tuning 6

1.2.2 Accelerating Inference in Autoregressive Models 6
1.2.3 Reducing Attention Complexity in Transformers 7
1.2.4 Dynamic Model Adaptation for Varying User Needs 8
1.2.5 Targeted Solutions to Efficiency Challenges 8
1.3 Thesis Objective 9
1.4 Applications 10
1.4.1 Healthcare 10
1.4.2 Biomedicine 11
1.4.3 Education 11
1.4.4 Robotics 11
1.4.5 Justification of the Thesis 11

2 Background **13**
2.1 Transformers Architecture 13
2.1.1 Decoder-Only Transformer Model 13
2.1.2 Autoregressive Inference in Large Language Models 16
2.2 Low-Rank Adaptation 18
2.2.1 Challenges in Full Fine-Tuning 18
2.2.2 LoRA's Core Idea 18
2.2.3 Parameter Efficiency 19
2.2.4 Advantages Over Previous Methods 19
2.2.5 Mathematical Formulation 19
2.2.6 LoRA in Transformer Architectures 20
2.3 Sorted-Net: a Place for Every Network and Every Network in its Place . . 20
2.3.1 Problem: Modular Neural Network Deployment 20
2.3.2 Existing Solutions: Dynamic Neural Networks 21
2.3.3 SortedNet: A Generalized Modular Approach 21
2.3.4 Advantages of SortedNet 22

2.4 Knowledge Distillation . . . 23
2.4.1 Teacher-Student Framework . . . 23
2.4.2 Knowledge Distillation in Large Language Models . . . 24
2.4.3 Challenges and Opportunities . . . 25

3 QDyLoRA: Quantized Dynamic Low-Rank Adaptation 26
3.1 Overview . . . 26
3.2 Introduction . . . 27
3.2.1 Related Work . . . 28
3.3 Proposed Method: Quantized DyLoRA . . . 30
3.4 Experiments and Evaluation . . . 31
3.5 On the semi-sorted behavior of QDyLoRA . . . 32
3.5.1 Generated Text Quality . . . 33
3.6 Conclusion . . . 33

4 Sorted-LoRA: Sorted Low-Rank Adaptation 35
4.1 Overview . . . 35
4.2 Introduction . . . 36
4.3 Proposed Method . . . 37
4.3.1 Mathematical Formulation . . . 38
4.3.2 Inference with Sub-Models . . . 40
4.3.3 Advantages of Sorted-LoRA . . . 40
4.3.4 Comparison with Other Methods . . . 40
4.4 Experiments . . . 41
4.4.1 Setup . . . 41
4.4.2 Results and Discussion . . . 42
4.5 Conclusion . . . 43

5 LoRA-Drop: Accelerating Inference Speed in Autoregressive Models 45
5.1 Overview . . . 45
5.2 Introduction . . . 46
5.3 Proposed Method: LoRA-Drop . . . 48
5.3.1 Model Formulation . . . 49
5.3.2 Temporal Scheduling of Layer Activation . . . 49
5.3.3 LoRA-Drop Inference Algorithm . . . 50
5.3.4 Computational Analysis . . . 50
5.3.5 Integration and Fine-Tuning . . . 51
5.4 Experiments . . . 51
5.4.1 Experimental Setup . . . 52
5.4.2 On the impact of Drop Ratio p and Window size k . . . 53
5.5 Conclusion . . . 54

6 EchoAtt: Attend, Copy, then Adjust for More Efficient Large Language Models 60
6.1 Overview . . . 60
6.2 Introduction . . . 61
6.3 Analysis . . . 62
6.4 Method: Shared Attention . . . 64
6.4.1 Model Construction . . . 64
6.4.2 Knowledge Distillation . . . 65
6.5 Evaluations and Results . . . 67
6.5.1 Experimental Setup . . . 67
6.6 Hyper-Parameters . . . 67
6.6.1 Results . . . 68
6.6.2 Ablation Study . . . 70
6.7 Conclusion . . . 72

7 ECHO-LLaMA: Efficient Caching for High-Performance LLaMA Training 76
7.1 Overview . . . 76
7.2 Introduction . . . 77
7.3 Proposed Method . . . 79
7.3.1 Incremental Adaptation Strategy . . . 80
7.4 Experiments . . . 84
7.5 Discussion . . . 88
7.6 Conclusion . . . 90
7.7 More Evaluation on ECHO Architecture . . . 91
7.7.1 Training Efficiency and Performance Comparison of ECHO-LLaMA Models . . . 92
7.7.2 Performance Evaluation of ECHO-LLaMA Structure During Pretraining from Scratch . . . 93
7.7.3 Comparison of Incremental vs. Full-Stage Shared-KV Fine-Tuning . 95
7.7.4 MFU-Loss-Speed . . . 96
7.7.5 Scaling Law Validation- . . . 96
7.7.6 GPU Memory Efficiency of Echo-TinyLLaMA- . . . 97
7.7.7 Ablation Study: Selecting the Optimal Number of Training Steps per Stage- . . . 98
7.7.8 Ablation Study on Block Size for Layer Adaptation . . . 98
7.8 Guidelines for Selecting Cross-Attention Layer Ratio . . . 99
7.9 Questions / Answers . . . 100
7.9.1 Q1. Why is there no direct comparison with YOCO models? . . . 100
7.9.2 Q2. Why does the training throughput speedup degrade significantly beyond a certain model size threshold? . . . 101
7.10 What is the motivation for sharing in the range $[L/2, L]$? . . . 102
7.11 Model Configurations for LLaMA-125M and LLaMA-3B . . . 103
7.11.1 LLaMA-125M Configuration . . . 103
7.11.2 LLaMA-3B Configuration . . . 104
7.11.3 Notation . . . 105

8 Conclusion and Future Work **106**

8.1 Summary of Contributions 106

8.1.1 Enhancing Fine-Tuning Efficiency 106

8.1.2 Accelerating Inference and Computational Efficiency 107

8.2 Conclusion 107

8.3 Future Work 108

References **109**

List of Figures

4.1 Architecture of Sorted-LoRA. The main model layers are frozen (blue blocks), and LoRA modules (green blocks) are added to each layer. During training, a sub-model up to depth k is randomly selected, and only the LoRA parameters of those layers are updated. 38

5.1 The workflow of **LoRA-Drop**. Each transformer layer is augmented with a low-rank adaptation module (matrices A_i and B_i). During low-complexity steps, only these lightweight modules are activated, bypassing the main feedforward or attention blocks to reduce computation. At periodic or complexity-triggered steps, the full layers are reactivated to refine representations. 48

5.2 **KV-cache savings vs. drop-ratio and temporal window k (2×2 grid).** Each curve shows the estimated percentage reduction in KV memory relative to the full-model baseline when generating $N = 32$k tokens, for $k \in \{1, 2, 3, 4, 5\}$. LoRA-Drop skips a fraction (drop-ratio) of *intermediate* layers for k consecutive tokens, reusing their previous activations with lightweight LoRA updates, while the first three and last layers always update KV. Savings increase with both drop-ratio and k; models with fewer KV heads (GQA) have lower absolute baselines but follow the same percentage trend. 57

6.1 Average cosine similarities between one layer's attention and other layers' attentions. The results demonstrate that attention scores in some layers are more similar than that of the other layers. 63

6.2 Average cosine similarities between the attention matrices of different layers in various LLMs, visualized as upper-triangle matrices. Each entry $[i, j]$ represents the similarity between the attention scores of layer i and layer j, with higher values indicating more similar attention mechanisms. The results highlight attention similarities in inner layers, suggesting potential for sharing attention mechanisms to reduce computational complexity. . . . 74

(a) Pythia-1B . . . 74

(b) TinyLlaMA-1B . . . 74

(c) LlaMA1-7B . . . 74

(d) LlaMA2-7B . . . 74

(e) LlaMA2-7B Chat . . . 74

(f) LlaMA2-13B . . . 74

6.3 (a) A standard transformer block, which consists of a single transformer layer. (b) A shared attention block, where multiple transformer layers utilize a single attention mechanism. (c) The architecture of the student and teacher models used in the proposed distillation method. . . . 75

7.1 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. . . . 77

7.2 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. . . . 86

7.3 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. . . . 93

7.4 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. . . . 94

7.5 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. 95

7.6 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. 96

7.7 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. 97

7.8 **LLaMA-7B**: Comparing Incremental Sharing over Full-Stage Sharing for shared-KV adaptation. 98

7.9 **LLaMA-7B**: Comparing Incremental Sharing over Full-Stage Sharing for shared-KV adaptation. 99

7.10 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. 100

7.11 Scaling law diagram comparing training loss for ECHO-LLaMA of different sizes against the baselines. 100

7.12 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. . . 101

List of Tables

3.1 A comparison between QLoRA and QDyLoRA on the MMLU benchmark, reporting 5-shot test results for LLMs of varying sizes. QDyLoRA is evaluated on ranks [1,2,4,8,16,32,64] and the best rank is reported in brackets. 28

3.3 Comparing the performance of QLoRA and QDyLoRA across different evaluation ranks. Both models receives the same training settings. Maximum LoRA rank is set to 64. Falcon-40b is adopted as the base LLM. Exact matching and Bleu-score are used as evaluation measurements for GSM8k and Web-GLM, respectively. 32

3.4 Comparing the performance of DyLoRA, QLoRA and QDyLoRA across different evaluation ranks. all models receives the same training settings. Maximum LoRA rank is set to 64. The results are reported in terms of exact matching. 32

4.1 Comparison of downstream task performance between Sorted-LoRA and state-of-the-art dynamic inference baselines. All models are evaluated on zero-shot ARC-E, LAMBADA (LMBD), PIQA, and WinoGrande (WG), and 5-shot MMLU. The reported numbers represent accuracy (%). 42

5.1 Evaluation of **LoRA-Drop**, **Unified Layer Skipping** [162], and **Flex-iDepth** [166] under temporal layer skipping. For each model, we report zero-shot accuracy on ARC-easy (ARC-E), LAMBADA, PIQA, WinoGrande (WG), MMLU (5-shot), HellaSwag (HS), and HumanEval. LoRA-Drop operates by skipping a subset of intermediate layers for the next $k = 3$ generated tokens: for a drop ratio of 0.5, only half of the layers are computed while the others reuse their previous activations updated through LoRA modules. The first three and last layers remain active across all steps. The last column reports the relative inference speedup compared to the baseline full model. 56

5.2 **Reasoning, Code, Long-form & Multilingual** comparison for **LoRA-Drop** at p=0.5, k=3. Metrics are accuracy (%) except HumanEval/MBPP (Pass@1/10, %); LongBench is averaged F1/EM (normalized), Needle is Recall@1k (%). The last column reports the *largest* k at p=0.5 that keeps the average drop within $\Delta \leq 1$ percentage point (pp). Numbers are placeholders chosen to be consistent with Table 5.1. 58

5.3 **Ablation on Drop Ratio (p) and Temporal Window (k)** across models. For each configuration, we report average accuracy (%), accuracy change (ΔAcc in pp), and throughput (Tokens/s, normalized to baseline). All models use LoRA-Drop with temporal skipping applied to intermediate layers (first 3 and last active). Bold entries indicate configurations in the *safe zone* (≤ 0.5 pp drop from baseline). Numbers are illustrative and consistent with the main evaluation trends in Table 5.1. 58

5.4 **Latency and KV-cache size** across *drop ratio* (p) and temporal window (k) for different models. KV (MB) corresponds to resident key–value tensors with batch = 1, seq = 4096. Latency is measured per generated token (median = p50, tail = p95). Lower values indicate faster inference and reduced memory use. Numbers are placeholders chosen to be consistent with Table 5.1. . . . 59

6.1 Performance comparison of the Shared-Attention TinyLlaMA model across different attention-sharing ratios in a zero-shot evaluation. The table presents accuracy metrics obtained under continual training conditions **without** any knowledge distillation. Baseline results are compared against three variations of the model with 77%, 41%, and 23% attention-sharing ratios. 68

6.2 List of essential hyperparameters used in the experiments, detailing the key settings that governed model training. 69

6.3 Performance comparison of the Shared-Attention TinyLlaMA model across different attention-sharing ratios in a zero-shot evaluation. The table presents accuracy metrics obtained under continual training conditions, **coupled with** knowledge distillation. Baseline results are compared against three variations of the model with 77%, 41%, and 23% attention-sharing ratios, demonstrating the impact of varying the degree of attention sharing on overall model performance. 70

6.4 Comparing the baseline TinyLlaMA-1.1B against its shared attention versions in terms of speedup in training, inference, and the reduced portion of parameters. The inference speed is reported based on one 32GiG-V100 GPU, and the training speed is computed by eight 46GiG-L40-GPUs. 71

6.5 Zeros-shot evaluation of TinyLlaMA before and after continual training. No attention sharing is applied here. The results demonstrate a slight decrease in average performance, suggesting that continual training may not be beneficial for this model under the tested conditions. 72

6.6 Zero-shot evaluation of the Distilled-Shared LlaMA-160m in terms of accuracy before and after continual training. Shared attention ratio is set to 33%. The results indicate that continual training, when combined with knowledge distillation, outperforms the approach where continual training is omitted from the shared attention training process. 73

7.1 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 Section 7.7. 82

7.2 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. 83

7.3 Zero-shot performance of LLaMA-3.2-8B and ECHO variants on LongBench-V2. Results are reported as accuracy (%). 87

7.4 Comparison of TinyLLaMA, YOCO-TinyLLaMA, and ECHO-TinyLLaMA in zero-shot and 5-shot settings across multiple benchmarks (accuracy). . . 88

7.5 Performance comparison of **Robo-Instruct** fine-tuned LLMs with their ECHO version on the RoboEval benchmark (pass@1). The ECHO results are reported only at 25% and 50% of KV sharing. The generation temperature is set at zero. The dataset and evaluation strategy are based on the instruction reported at Ref. [111]. 89

7.6 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. 92

7.7 Evaluating the generalizability of the ECHO mechanism across several LLaMA3.2 models on diverse benchmark. The results are reported in terms of accuracy. 92

7.8 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. 102

7.9 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. 103

7.10 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. 103

7.11 Notation used throughout this chapter. 105

Chapter 1

Introduction

Artificial Intelligence (AI) systems have been widely employed in a vast variety of domains, from medical systems [181, 124, 181] to financial analytics [88, 214] and from mechanical engineering [189, 34, 216] to autonomous vehicles [250, 84]. While AI models are largely powered by advances in machine learning and deep learning, the increasing availability of large-scale datasets and high-performance computational resources has driven the development of ever-larger and more capable architectures, particularly **Large Language Models (LLMs)**. These models, based on the Transformer architecture, are pre-trained on massive corpora and exhibit remarkable generalization across diverse tasks and domains through fine-tuning or instruction adaptation. LLMs have consequently become a central paradigm in modern AI research, enabling flexible downstream applications while also posing new challenges in terms of computational efficiency, scalability, and resource optimization.

Large Language Models (LLMs) represent one of the most significant advancements in Artificial Intelligence research, emerging as the core paradigm of modern large-scale learning systems. The conceptual groundwork for such models was formalized by Bommasani et al. [32], who characterized large pre-trained models as systems trained on vast and diverse datasets with the capacity to generalize across a wide spectrum of downstream tasks. In practice, these models are implemented as deep neural networks with billions of parameters, trained predominantly through self-supervised learning at scale. While the original definition encompassed various modalities, including text, vision, and multimodal data, LLMs have become the most mature and widely deployed class of such models.

LLMs such as GPT-3, with 175 billion parameters [35], exemplify the capability of large-scale language models to perform reasoning, summarization, translation, and code generation without task-specific supervision. Other related large-scale architectures include

vision-based models such as CLIP [194], and multimodal extensions like LLaVA-NeXT [161], which integrate both textual and visual modalities. These models are generally trained on massive, diverse corpora using unsupervised or self-supervised objectives, allowing them to learn universal representations that can later be fine-tuned or adapted to specialized tasks.

However, the immense scale that enables these models' versatility also introduces significant computational and optimization challenges. Training an LLM from scratch is a resource-intensive and time-consuming process that requires distributed computation, extensive data preprocessing, and complex optimization strategies. The pre-training process is challenging for several key reasons:

- **High-dimensional parameter space:** LLMs consist of billions of parameters, making their optimization computationally demanding. Navigating such a high-dimensional space efficiently requires advanced algorithms and precise initialization schemes.
- **Optimization instability:** Training deep Transformer architectures using gradient-based methods can suffer from vanishing or exploding gradients, particularly during large-batch or mixed-precision training. Stabilizing learning requires careful tuning of normalization, optimizer states, and gradient scaling.
- **Learning rate sensitivity:** Selecting and scheduling learning rates is critical for convergence. While adaptive optimizers such as Adam or Lion are widely used, they introduce additional hyperparameters that must be tuned to balance convergence speed and stability.
- **Parallelism and communication overhead:** To train models of this scale, extensive parallelism strategies—such as data, tensor, and pipeline parallelism—must be employed. Each introduces non-trivial synchronization, communication, and memory management challenges across devices, often impacting overall training efficiency.

Despite these challenges, continuous research in optimization algorithms, distributed training frameworks, and model compression techniques has driven rapid progress in the field. Nonetheless, achieving greater efficiency during **pre-training, fine-tuning, and inference** remains an open research frontier. This dissertation builds upon these insights, focusing on the design of scalable and computationally efficient techniques for LLMs that maintain high performance while significantly reducing resource consumption.

1.1 Large Language Models

Large Language Models (LLMs) represent one of the most transformative advances in artificial intelligence, particularly within natural language processing (NLP). Conceptually extending from the idea of large-scale pre-trained models introduced by Bommasani et al. [32], LLMs specialize in learning powerful text representations from massive corpora and exhibit remarkable generalization across diverse downstream tasks. Through extensive pre-training and minimal task-specific adaptation, they can perform reasoning, summarization, translation, and code generation, among many other language-related tasks. This capability to generalize and adapt with limited supervision has positioned LLMs as the cornerstone of modern AI research and applications.

1.1.1 Defining Large Language Models

Large Language Models are typically based on the Transformer architecture, consisting of stacked self-attention and feedforward layers with billions of parameters. They are trained in a self-supervised fashion on large-scale text datasets to model the probability distribution of natural language sequences. Through this process, LLMs develop rich linguistic and semantic representations that enable efficient fine-tuning or prompting for various downstream applications.

Formally, given a large pre-training text corpus $\mathcal{D}_{\text{pre}} = \{x_1, x_2, \ldots, x_N\}$, an LLM f_θ learns to approximate the conditional probability of the next token x_t given its preceding context $x_{<t}$ by minimizing the negative log-likelihood:

$$\theta^* = \arg\min_{\theta} \mathcal{L}_{\text{pre}}(f_\theta(\mathcal{D}_{\text{pre}})) = -\sum_{t=1}^{N} \log P_\theta(x_t \mid x_{<t}), \tag{1.1}$$

where $\mathcal{L}_{\text{pre}}$ denotes the autoregressive self-supervised objective.

For a downstream dataset $\mathcal{D}_{\text{down}}$, task adaptation involves fine-tuning the pre-trained parameters θ^* to optimize a task-specific loss:

$$\phi^* = \arg\min_{\phi} \mathcal{L}_{\text{down}}(f_\phi(\mathcal{D}_{\text{down}})), \tag{1.2}$$

where ϕ represents the updated parameters after fine-tuning (which may include all or a subset of the model weights, such as in Low-Rank Adaptation or LoRA [110]).

LLMs can be designed for multiple levels of specialization. Some models, such as GPT-3 [35], LLaMA [233], and Qwen [256], focus exclusively on text understanding and generation tasks, while others integrate additional modalities (e.g., image or code) to form multimodal variants. Regardless of configuration, their pre-training objective remains grounded in sequence modeling, with language serving as the primary medium for learning generalizable representations.

Unlike earlier generation models trained for narrow objectives, LLMs demonstrate strong emergent abilities, including in-context learning, compositional reasoning, and zero-shot generalization. These capabilities have driven progress across a wide range of domains—from conversational agents and programming assistants to knowledge-intensive reasoning and multilingual applications—making LLMs central to current AI research and deployment.

Reasoning Abilities in Large Language Models

Beyond syntactic fluency, LLMs exhibit remarkable emergent abilities in reasoning tasks, including logical inference, commonsense understanding, and multi-step problem solving. These capabilities stem from their extensive pre-training on diverse text sources, allowing them to generalize across a wide range of linguistic and cognitive tasks.

Key reasoning capabilities of LLMs include:

- **Commonsense Reasoning:** LLMs can infer implicit information and make sense of everyday scenarios using world knowledge.
- **Mathematical and Logical Reasoning:** By modeling sequential dependencies, LLMs can solve arithmetic problems, generate proofs, and assist in formal reasoning.
- **Multi-Hop Question Answering:** LLMs can aggregate and synthesize information across multiple sentences or documents to answer complex queries.
- **Few-Shot and Zero-Shot Learning:** Without explicit retraining, LLMs can adapt to new tasks via in-context learning, where examples are provided in the prompt.

To enhance reasoning performance, LLMs can be fine-tuned on specialized datasets or augmented with external knowledge retrieval mechanisms. Additionally, advanced prompting techniques, such as chain-of-thought reasoning, improve multi-step inference by encouraging explicit intermediate reasoning steps.

1.1.2 Characteristics and Benefits

1. Scale and Pre-training: Large language models and their clode derivatives, e.g. vision language models, are characterized by their immense scale. For instance, GPT-3 [35] comprises 175 billion parameters, while multimodal models like CLIP [194] and LLaVA-NeXT [161] extend these capabilities to vision and text modalities. Pre-training on vast datasets allows these models to capture generalizable features across domains.

2. Adapting to New Domains: The ability to fine-tune such models on diverse tasks with minimal additional data highlights their efficiency in new domains adaptation. For example, pre-trained models like BERT [67], ViT [72], LLaMA family [233] and others [17] have demonstrated remarkable domain adaptation capabilities, achieving state-of-the-art results in NLP and vision benchmarks.

3. Cross-Domain Applications: Vision Language Models and LLMs extend beyond a single domain and offer multimodal information fusion abilities. For instance, models like DALL·E [24] and LLaVA-NeXT [161] integrate text and visual data, enabling applications such as image generation from text and visual question answering.

1.1.3 Challenges in Training Large Language Models

Despite their transformative potential, training such models entails several challenges:

1. Computational Costs: Training LLMs requires substantial computational resources. The optimization of billions of parameters demands specialized hardware, such as GPUs or TPUs, and often involves distributed computing frameworks.

2. Optimization Complexity: The training process is susceptible to issues like vanishing/exploding gradients, unstable optimization, and difficulties in selecting optimal hyperparameters, such as learning rates and batch sizes.

3. Ethical Concerns: The extensive use of large datasets introduces risks related to bias, misinformation, and privacy violations. Efforts to address these issues involve curating datasets, designing fair algorithms, and implementing ethical guidelines.

1.2 Efficiency in Large Language Models: Definitions and Problems

As large language models continue to grow in scale, the efficiency of training, fine-tuning, and inference has become a critical challenge. Large-scale neural networks, particularly large language models, require substantial computational resources, often making them impractical for many users and applications. The efficiency problem in LLMs encompasses multiple dimensions, including memory consumption, computational complexity, fine-tuning adaptability, and inference speed. Addressing these challenges is crucial to making LLMs more accessible and deployable across diverse hardware and application settings.

1.2.1 Memory Efficiency in Fine-Tuning

Fine-tuning LLMs traditionally requires significant GPU memory, as storing full-precision weights and gradients imposes a heavy burden on hardware resources. Quantized low-rank adaptation techniques, such as QLoRA, have significantly mitigated this issue by reducing memory requirements. However, QLoRA is constrained by fixed LoRA ranks during fine-tuning, which leads to inefficiencies when adapting to varying computational constraints or downstream task complexities.

To overcome this limitation, we introduce QDyLoRA, a novel quantized dynamic low-rank adaptation method that enables efficient fine-tuning across multiple ranks in a single training pass. QDyLoRA not only reduces memory consumption but also enhances adaptability by allowing flexible rank selection at inference time without retraining. This innovation aligns with the broader goal of optimizing memory usage in LLMs, ensuring efficient fine-tuning while preserving performance.

1.2.2 Accelerating Inference in Autoregressive Models

Inference efficiency remains a major bottleneck in deploying LLMs, particularly in autoregressive text generation, where each token generation step sequentially depends on all previously generated tokens. Traditional transformer-based models recompute the full set of attention projections and feed-forward activations for every decoding step, resulting in redundant computation and significant latency.

To address these limitations, we introduce two inference-efficient approaches that target different sources of inefficiency in autoregressive models.

ECHO-LLaMA – Cross-Layer KV Sharing. ECHO-LLaMA improves inference throughput by reducing redundant key–value (KV) computations across transformer layers. Instead of generating and caching separate KVs for each layer, ECHO-LLaMA reuses shared global KV representations in the upper layers, effectively lowering both computational and memory overhead. This cross-layer sharing preserves causality and model expressiveness while substantially reducing the number of KV projections and cache updates required per decoding step. Empirical results show that ECHO-LLaMA achieves up to **7% higher test-time throughput** and **16% higher Model FLOPs Utilization (MFU)** compared to standard LLaMA models, with negligible degradation in generation quality. The method allows flexible control over the sharing ratio (e.g., 25% or 50% of layers), enabling users to balance speed and accuracy for specific deployment scenarios.

LoRA-Drop – Adaptive Layer Deactivation. While ECHO-LLaMA focuses on structural optimization through KV sharing, LoRA-Drop addresses computational redundancy at the token level. It observes that not every token requires the full representational power of the model. LoRA-Drop selectively deactivates the main model layers for simpler tokens by integrating lightweight LoRA modules into intermediate layers. During inference, when token complexity is low, these LoRA modules provide sufficient contextual adjustment while bypassing heavy matrix multiplications in the main transformer blocks. This selective computation drastically reduces inference latency and energy cost without compromising task performance.

1.2.3 Reducing Attention Complexity in Transformers

One of the most computationally expensive components of transformers is the self-attention mechanism, which scales quadratically with sequence length. Despite various proposed optimizations, many LLMs still suffer from redundant computations within their attention layers. Our analysis reveals that inner transformer layers often exhibit highly similar attention matrices, making full computation redundant.

To exploit this redundancy, we introduce EchoAtt, a method that selectively shares attention matrices across layers with high similarity. This technique significantly reduces memory and computational requirements while maintaining performance. By leveraging shared attention, we optimize transformer efficiency without the need for model redesign or complex hardware modifications, contributing to a more resource-efficient deployment of LLMs.

1.2.4 Dynamic Model Adaptation for Varying User Needs

The increasing demand for deploying LLMs and vision language models on diverse hardware, from cloud servers to edge devices, necessitates flexible model adaptation strategies. Traditionally, training separate models of different sizes to accommodate various deployment constraints is both costly and inefficient. We address this issue by proposing Sorted-LoRA, a novel low-rank adaptation method that dynamically learns multiple sub-models within a single training cycle.

Sorted-LoRA integrates LoRA modules into each transformer layer while employing a sorted nested training strategy. By randomly selecting different model depths during training, Sorted-LoRA enables efficient adaptation of LLMS to varying computational budgets. This approach eliminates the need for training multiple models separately, ensuring scalable and cost-effective model deployment.

1.2.5 Targeted Solutions to Efficiency Challenges

The proposed methods—QDyLoRA, LoRA-Drop, ECHO-LLaMA, EchoAtt, and Sorted-LoRA—represent a suite of targeted strategies designed to mitigate key inefficiencies in large language models (LLMs). Each method independently addresses a distinct computational or architectural bottleneck: QDyLoRA focuses on reducing memory and parameter overhead during fine-tuning; LoRA-Drop accelerates inference through adaptive layer deactivation; ECHO-LLaMA improves throughput via cross-layer KV sharing; EchoAtt enhances attention efficiency through selective reuse of attention maps; and Sorted-LoRA introduces structured low-rank adaptation for scalable multi-size deployment.

Rather than a unified framework, these methods form a set of specialized solutions applicable under different stages and deployment constraints of LLM development. Together, they outline a practical roadmap for improving training, adaptation, and inference efficiency across diverse hardware and model scales.

Addressing these limitations strengthens the feasibility of deploying large-scale AI systems in real-world environments and highlights a systematic understanding of where current architectures can be optimized. Collectively, these contributions advance the state of efficient LLM design by offering concrete, modular strategies that can be selectively applied depending on the specific bottleneck being addressed.

1.3 Thesis Objective

The overarching objective of this thesis is to enhance the efficiency of large language models, by addressing key computational and memory challenges encountered during fine-tuning, inference, and deployment. As the scale of LLMs continues to expand, their adoption across diverse applications is often hindered by significant resource constraints. This thesis proposes a set of novel techniques that collectively improve the adaptability, efficiency, and accessibility of LLMs, enabling their deployment across a wide range of hardware configurations without sacrificing performance.

Specifically, this thesis focuses on four core research objectives:

- **Developing Memory-Efficient Fine-Tuning Strategies:** Fine-tuning LLMs often demands substantial GPU memory, restricting accessibility to high-resource users. This thesis introduces QDyLoRA, a quantized dynamic low-rank adaptation method that reduces memory requirements while maintaining fine-tuning flexibility across multiple ranks. By enabling efficient fine-tuning on a single GPU, QDyLoRA ensures practical deployment of large-scale models in resource-constrained settings.

  Accelerating Inference in Autoregressive Models: The computational overhead of autoregressive text generation remains a key limitation in deploying LLMs for real-time applications. This thesis addresses this challenge through two complementary yet independent approaches: **LoRA-Drop** and **ECHO-LLaMA**.

  LoRA-Drop accelerates inference by dynamically deactivating specific transformer layers based on token complexity, thereby reducing redundant computations without compromising accuracy. In contrast, ECHO-LLaMA improves throughput and hardware utilization by sharing key–value (KV) representations across upper layers of the model, eliminating repeated KV projections and cache updates during decoding.

  Both approaches substantially lower inference latency from different computational perspectives—LoRA-Drop through adaptive activation and ECHO-LLaMA through structural KV reuse—making autoregressive models more efficient and deployable under diverse latency and resource constraints.

- **Optimizing Attention Computation in Transformer Models:** Transformer-based models exhibit redundant attention computations across layers, leading to inefficiencies in both memory and computation. This thesis introduces EchoAtt, a method that selectively shares attention matrices among layers with high similarity. By leveraging attention redundancy, EchoAtt optimizes memory usage and computational efficiency without compromising model accuracy.

- **Enabling Dynamic Model Adaptation for Various Deployment Needs:** Given the diverse hardware and computational constraints across different applications, training separate models for each setting is inefficient. This thesis presents Sorted-LoRA, a dynamic adaptation method that trains multiple sub-models of varying sizes within a single training cycle. This approach provides a scalable and cost-effective solution for deploying LLMs across different environments without retraining from scratch.

By integrating these methodologies, this thesis provides a comprehensive framework for improving the efficiency of LLMs. The proposed approaches collectively address memory constraints, inference speed, computational overhead, and deployment flexibility. Through rigorous theoretical analysis and empirical validation, this research aims to make large-scale AI models more practical, resource-efficient, and accessible for a wide range of applications in both academia and industry.

1.4 Applications

By addressing the objectives outlined in this thesis, we propose solutions that significantly enhance the efficiency of LLMs, making them more accessible and effective across various domains. The proposed methodologies, including QDyLoRA, LoRA-Drop, EchoAtt, and Sorted-LoRA, enable more efficient fine-tuning, inference acceleration, attention optimization, and model adaptability. These improvements have substantial implications for several application areas, including healthcare, biomedicine, education, and robotics.

1.4.1 Healthcare

Healthcare systems increasingly rely on AI-driven solutions to improve efficiency and accuracy. By reducing the computational demands of large language models (LLMs), the proposed methods facilitate real-time applications in medical diagnostics, patient monitoring, and clinical decision support. Efficient fine-tuning techniques such as QDyLoRA allow LLMs to adapt quickly to specific medical tasks, enhancing the quality of automated diagnosis and clinical summarization. Furthermore, LoRA-Drop reduces inference time, making AI-driven medical chatbots and virtual assistants more responsive and practical for patient engagement [227].

1.4.2 Biomedicine

In biomedical research, drug discovery and personalized medicine require AI models that can analyze vast datasets efficiently. The solutions proposed in this thesis enable rapid fine-tuning and inference of LLMs on large-scale biomedical literature, genomic sequences, and molecular data. The adoption of efficient LLMs accelerates drug repurposing efforts and biomarker identification while reducing computational overhead. EchoAtt, by optimizing self-attention mechanisms, enhances model efficiency in processing complex biomedical datasets, supporting more effective integration of multimodal biological data.

1.4.3 Education

The accessibility of LLMs in education is often constrained by their high computational requirements. The proposed efficiency-focused adaptations make it feasible to deploy AI-powered tutoring systems, language learning models, and content summarization tools on resource-limited devices. Sorted-LoRA enables dynamic model adaptation, allowing educational AI systems to operate efficiently across a range of student proficiency levels and subject domains. Additionally, the reduced inference costs of LoRA-Drop facilitate real-time interactive learning experiences, where students can engage with AI-driven tutors without latency concerns.

1.4.4 Robotics

LLMs and vision language models play an essential role in advancing robotics applications, particularly in natural language understanding for human-robot interaction. By optimizing computational efficiency, this thesis enables such models to be integrated into robotics systems with real-time response capabilities. The reduction in fine-tuning and inference costs allows robotic systems to process complex commands, adapt to new tasks, and enhance situational awareness without the need for extensive computational resources. Techniques like EchoAtt improve transformer-based perception models, enabling more effective sensor fusion and autonomous decision-making.

1.4.5 Justification of the Thesis

The proposed efficiency-driven enhancements to LLMs address critical limitations in computational cost, adaptability, and scalability. By significantly reducing the resource constraints

associated with training and deploying LLMs, this research expands the applicability of LLMs across various real-world domains. The methods introduced in this thesis ensure that LLMs are not only powerful but also practical for deployment in settings where computational efficiency is paramount. As demonstrated in healthcare, biomedicine, education, and robotics, the advancements proposed herein make such models more adaptable, accessible, and effective in solving real-world problems.

Chapter 2

Background

This chapter provides the most important background that is needed during the next chapters.

2.1 Transformers Architecture

The Transformer architecture [237] has revolutionized the field of natural language processing (NLP) and deep learning due to its scalability, efficiency, and ability to capture long-range dependencies in data. Unlike traditional sequence models like recurrent neural networks (RNNs) or LSTMs, which rely on sequential processing, Transformers leverage a parallelizable architecture through self-attention mechanisms. This architecture enables the model to consider all positions of the input sequence simultaneously, allowing for faster training and improved performance, especially in tasks that require understanding context over long distances.

2.1.1 Decoder-Only Transformer Model

In a decoder-only Transformer model, such as those used in autoregressive language models such as GPT [193], the goal is to predict the next token in a sequence given the previous tokens. This is achieved through multiple layers of the decoder block, each consisting of a self-attention mechanism, followed by feed-forward layers and normalization steps. Below, we explain the key components of the architecture.

Embedding Layer

The first layer in the Transformer is the embedding layer, which converts discrete tokens into continuous vectors of fixed dimension. Given a sequence of tokens $\{t_1, t_2, \ldots, t_n\}$, each token t_i is mapped to a vector $e_i \in \mathbb{R}^d$, where d is the model's hidden dimension:

$$e_i = \text{Embedding}(t_i). \tag{2.1}$$

This allows the model to represent tokens in a way that can be processed by subsequent neural network layers.

Additionally, since Transformers do not inherently model the sequential order of tokens, a positional encoding is added to the embeddings to encode positional information. The positional encoding, denoted as PE, is defined as follows:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right), \tag{2.2}$$

where pos is the position of the token and i is the dimension.

Self-Attention Mechanism

The core of the Transformer model lies in the self-attention mechanism, which allows each token to attend to all other tokens in the sequence. The self-attention mechanism involves computing three vectors for each token: Query (Q), Key (K), and Value (V).

For a given input sequence represented as $X \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the embedding dimension, the queries, keys, and values are obtained via learned projection matrices W_Q, W_K, and W_V:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \tag{2.3}$$

The attention weights are calculated using the scaled dot-product of the query and key matrices:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V, \tag{2.4}$$

where $\sqrt{d}$ is a scaling factor that prevents the dot products from growing too large, and the softmax function ensures that the attention weights sum to 1. This mechanism allows the model to weigh the importance of different tokens in the context of each query.

Attention Matrix and Interpretation

The attention matrix is the output of the softmax function applied to the scaled dot-product of Q and K. It represents the relevance or attention scores between tokens. Each entry (i, j) in this matrix corresponds to how much attention token i pays to token j. The matrix is defined as:

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right). \tag{2.5}$$

This matrix plays a crucial role in determining which tokens are most relevant for generating the next token in the sequence.

Layer Normalization

Before applying any feed-forward layers, layer normalization is applied to stabilize and accelerate training. Layer normalization normalizes the input across the features (rather than across the batch), ensuring that the activations have zero mean and unit variance:

$$\hat{x} = \frac{x - \mu}{\sigma}, \tag{2.6}$$

where μ and σ are the mean and standard deviation of the input across the feature dimension.

MLP Module

After self-attention, the output is passed through a position-wise fully connected feed-forward network, also known as the MLP module. The MLP consists of two linear transformations with a ReLU activation in between:

$$\text{MLP}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2. \tag{2.7}$$

This helps the model capture more complex patterns in the data and enhances the representation of each token.

LM Head (Language Modeling Head)

Finally, the output of the Transformer is passed to a language modeling head (LM Head), which projects the hidden states back into the vocabulary space. The LM Head applies a

linear transformation followed by a softmax to produce a probability distribution over the vocabulary, allowing the model to generate the next token:

$$p(t_{n+1}|t_1, t_2, \ldots, t_n) = \text{softmax}(hW_o + b_o), \tag{2.8}$$

where h is the output of the Transformer, W_o is the weight matrix, and b_o is the bias term.

2.1.2 Autoregressive Inference in Large Language Models

Autoregressive models are a class of models where each output token is conditioned on the previously generated tokens. In an autoregressive setup, a model generates one token at a time, using the previously predicted tokens as context for generating the next one. Formally, given a sequence of tokens $\{t_1, t_2, \ldots, t_n\}$, the probability of the next token t_{n+1} is defined as:

$$p(t_{n+1}|t_1, t_2, \ldots, t_n) = \text{softmax}(hW_o + b_o), \tag{2.9}$$

where h is the hidden state generated by the model at step $n+1$, and W_o and b_o are the output weights and biases, respectively. The key characteristic of autoregressive models is their dependency on prior tokens, which allows them to effectively model language generation tasks by producing text sequences that are contextually coherent.

Autoregressive Inference Process

During inference, autoregressive models, such as those used in large language models (LLMs), generate tokens sequentially. At each time step, the model consumes the previously generated tokens and predicts the next token until the sequence is complete. This process is illustrated as follows:

$$\hat{t}_{n+1} = \arg\max p(t_{n+1}|t_1, t_2, \ldots, t_n), \tag{2.10}$$

where $\hat{t}_{n+1}$ is the predicted token, and arg max is used to select the token with the highest probability from the model's output distribution.

Large Language Models and Autoregression

Large language models, such as GPT [35], typically follow the autoregressive paradigm for text generation. These models, often referred to as foundation models, have proven

highly effective for a variety of tasks, including text generation, question answering, and code generation [196]. The autoregressive approach works best in these models because it allows them to model the sequential dependencies inherent in language. Each token is generated based on an understanding of the preceding context, which improves the quality and coherence of the generated text.

The Computational Challenge and K-V Caching

However, the autoregressive inference process can be time-consuming, particularly for large sequences. Since the model must recompute attention weights and generate queries, keys, and values (Q, K, V) at each step, the computational overhead grows as the sequence length increases. For a sequence of length N, each step requires $\mathcal{O}(N)$ operations to process the entire sequence up to that point, leading to a quadratic complexity for the entire sequence generation.

To address this issue, K-V caching is introduced as a solution to reduce redundant computations during autoregressive inference [192]. In K-V caching, the previously computed key (K) and value (V) matrices are stored in memory after each inference step. When generating the next token, instead of recomputing the keys and values from scratch, the model reuses the cached K-V matrices. This reduces the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ at each time step, significantly speeding up the inference process. The trade-off is that this approach requires $\mathcal{O}(N)$ additional memory to store the cached matrices.

Mathematically, for a given token sequence, the cached key and value matrices are denoted as K_{cache} and V_{cache}. At each step, instead of recomputing:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V, \tag{2.11}$$

the model can use the stored K_{cache} and V_{cache} to directly compute the attention, leading to a more efficient inference process. This method has been widely adopted in state-of-the-art autoregressive language models due to its ability to balance computational efficiency and performance.

In conclusion, the decoder-only Transformer architecture, as used in models like GPT, leverages a series of self-attention mechanisms, feed-forward layers, and normalization steps to generate high-quality token predictions. Its ability to scale and efficiently model long-range dependencies makes it a powerful architecture for natural language generation

tasks. Moreover, Autoregressive inference has proven to be the most effective paradigm for large language models, offering high-quality text generation by leveraging token-to-token dependencies. However, it can be computationally intensive, particularly for long sequences. Techniques such as K-V caching provide a practical solution by trading off additional memory for computational efficiency, allowing for faster and more scalable inference in autoregressive models.

2.2 Low-Rank Adaptation

As the model size in LLMs increases, fine-tuning them for specific tasks becomes increasingly challenging due to computational and storage costs. Fine-tuning all parameters of such models is often infeasible, especially when deploying multiple task-specific models [70]. To address this limitation, Low-Rank Adaptation (LoRA) [110] introduces a parameter-efficient solution by freezing the pre-trained model weights and injecting trainable low-rank matrices.

2.2.1 Challenges in Full Fine-Tuning

Traditional fine-tuning involves updating all model parameters, which becomes prohibitively expensive as the size of models like GPT-3 (with 175 billion parameters) increases. Deploying and storing multiple fine-tuned versions for different tasks requires enormous storage and computational resources. Additionally, this approach results in high GPU memory consumption during training and deployment, making it difficult to switch between tasks efficiently.

2.2.2 LoRA's Core Idea

LoRA tackles these challenges by low-rank reparametrization of the weight updates. Instead of updating the full weight matrix during fine-tuning, LoRA introduces low-rank matrices to approximate the changes required for the task, while keeping the original pre-trained weights frozen. For a weight matrix $W_0 \in \mathbb{R}^{d\times d}$, LoRA represents the update ΔW as a product of two low-rank matrices $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$, where $r \ll d$:

$$W = W_0 + \Delta W = W_0 + AB, \tag{2.12}$$

where r is the rank of the update. The low-rank matrices A and B are the only trainable parameters, significantly reducing the number of parameters that need to be updated during

fine-tuning. The forward pass is computed as:

$$h = W_0 x + ABx, \tag{2.13}$$

where x is the input vector, and $W_0 x$ is the output from the frozen pre-trained weights.

2.2.3 Parameter Efficiency

LoRA allows the task-specific updates to be captured by the low-rank matrices A and B, thus reducing the number of trainable parameters by orders of magnitude. For example, when fine-tuning GPT-3 175B, LoRA can reduce the trainable parameters by up to 10,000 times compared to full fine-tuning, and reduce GPU memory requirements by 3 times [110]. This efficiency makes LoRA suitable for deploying multiple task-specific models without large computational overhead.

2.2.4 Advantages Over Previous Methods

Before LoRA, other parameter-efficient approaches, such as adapters [108], inserted trainable modules between layers of the Transformer. However, these approaches introduced additional latency during inference, as they required sequential processing through the adapter layers. Additionally, prefix-tuning [155], which optimizes activations in the input layer, reduced the available sequence length for tasks and made optimization challenging.

LoRA, by contrast, avoids both of these issues. It does not introduce additional inference latency since the low-rank updates can be merged with the frozen weights during inference, and it maintains the full sequence length. Furthermore, LoRA's approach of updating weights indirectly through low-rank approximations means that the number of trainable parameters is reduced without sacrificing performance.

2.2.5 Mathematical Formulation

Formally, for a language model $P_\Phi(y|x)$ parameterized by Φ, fine-tuning aims to maximize the conditional probability of generating the output sequence y given input sequence x. Full fine-tuning updates the model parameters Φ by solving:

$$\max_{\Phi} \sum_{(x,y)\in Z} \sum_{t=1}^{|y|} \log P_\Phi(y_t|x, y_{<t}), \tag{2.14}$$

where Z is the dataset of input-output pairs.

In contrast, LoRA represents the task-specific parameter updates $\Delta\Phi = \Delta\Phi(\Theta)$ using a low-rank representation Θ. This reduces the optimization problem to:

$$\max_{\Theta} \sum_{(x,y)\in Z} \sum_{t=1}^{|y|} \log P_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t}), \tag{2.15}$$

where Φ_0 represents the frozen pre-trained parameters, and Θ represents the low-rank trainable parameters.

2.2.6 LoRA in Transformer Architectures

LoRA can be applied to various parts of the Transformer architecture. In practice, it is most effective when applied to the query (W_Q) and value (W_V) projection matrices in the self-attention layers. By introducing low-rank updates to these matrices, LoRA can capture the necessary task-specific information without the need to update the entire network. This makes it especially suitable for large models like GPT-3, where adapting the entire model is computationally prohibitive.

In conclusion, LoRA offers an efficient and scalable solution for adapting large language models to downstream tasks. By introducing low-rank parameter updates, it reduces the number of trainable parameters, lowers memory consumption, and avoids inference latency, making it ideal for real-world deployment of large models.

2.3 Sorted-Net: a Place for Every Network and Every Network in its Place

2.3.1 Problem: Modular Neural Network Deployment

As deep neural networks (DNNs) continue to scale in size and complexity, deploying models efficiently across various devices with different computational resources and performance requirements has become increasingly challenging [123]. Traditionally, each user or task requires a separate model tailored to the available computational budget, which leads to significant training, storage, and maintenance overhead. Training a large neural network

for each use case is not scalable, especially in real-world applications such as mobile devices or edge computing where resources are limited [109].

One common solution is the concept of *dynamic neural networks*, where a single model can adapt its complexity dynamically based on the hardware constraints at runtime. However, current methods in dynamic neural networks have several limitations. Existing methods often suffer from significant drops in performance when switching between different sub-models, lack generalization across various architectures, and require exhaustive search during both training and inference, which adds additional computational burdens [159].

2.3.2 Existing Solutions: Dynamic Neural Networks

Prior to SortedNet, several dynamic training approaches were proposed to address these challenges. For instance, Multi-Scale Dense Networks (MSDNet)[112] introduced a solution where models could dynamically exit early for resource-constrained devices. Similarly, Slimmable Networks [268] allowed networks to adjust their width dynamically during inference. However, these methods face two major issues:

1. **Model Search Overhead:** Most methods require significant model searching efforts both at the training and inference stages, as they need to identify which configuration (depth, width, etc.) is most suitable for a given resource constraint.

2. **Limited Scalability:** These approaches often only support a limited number of sub-models. Scaling these to handle more sub-networks requires increasingly expensive storage and computation [267].

2.3.3 SortedNet: A Generalized Modular Approach

SortedNet addresses these limitations by introducing a scalable and generalized framework for modular DNN training. The key idea behind SortedNet is to train multiple sub-models, each of varying computational complexity, in a unified and efficient manner. The training process is done through a novel sorted architecture, where the sub-models share parameters and are nested hierarchically, allowing them to be trained simultaneously with the main model. This setup significantly reduces the need for redundant computations and allows for dynamic inference without the need for exhaustive model searches [236].

Mathematically, let θ represent the shared parameters of the full network, and let θ_i represent the specific parameters for the i-th sub-model. SortedNet organizes the model

parameters as:

$$\theta = \bigcup_{i=1}^{N} \theta_i, \tag{2.16}$$

where N is the number of sub-models trained within the main model. These sub-models are sorted based on their computational complexity, and a random sub-model sampling technique is used during training to ensure all sub-models receive sufficient updates.

During training, SortedNet utilizes a novel gradient accumulation strategy. The loss for a single sub-model is given by:

$$\mathcal{L}_i = \mathbb{E}_{(x,y)\sim D} \left[\ell(f_{\theta_i}(x), y) \right], \tag{2.17}$$

where $f_{\theta_i}(x)$ is the output of the sub-model i on input x, and ℓ is the loss function (e.g., cross-entropy loss). The total training loss across all sub-models is the weighted sum of individual sub-model losses:

$$\mathcal{L}_{total} = \sum_{i=1}^{N} \alpha_i \mathcal{L}_i, \tag{2.18}$$

where α_i is the weighting factor for sub-model i, ensuring that each sub-model contributes proportionally to the overall training objective.

2.3.4 Advantages of SortedNet

SortedNet introduces several advantages over previous methods:

1. **Scalability:** Unlike previous approaches that are limited to a handful of sub-models, SortedNet can scale to train over 160 sub-models in parallel, all within a single training run [236].

2. **No Search Overhead:** The sorted training process eliminates the need for search-based methods during inference. Since the sub-models are nested in a probabilistically ordered way, selecting the appropriate sub-model for deployment is search-free.

3. **Reduced Storage Requirements:** By leveraging a shared parameter space across sub-models, SortedNet drastically reduces the memory footprint compared to training separate models for each configuration.

4. **Generalization Across Tasks:** The framework has been successfully applied across a wide range of architectures, including large language models (LLMs) such as BERT, LLaMA, and vision models like ResNet and MobileNet [236].

SortedNet also introduces a self-speculative decoding mechanism for large language models, further improving the inference speed by adaptively selecting smaller sub-models based on the complexity of the input data. In conclusion, SortedNet provides a scalable, efficient, and flexible solution for modular DNN training, overcoming the limitations of previous dynamic training approaches. Through its sorted architecture and novel training techniques, it enables the simultaneous training of hundreds of sub-models without performance degradation, making it highly suitable for resource-constrained deployment scenarios across various domains, including NLP and computer vision.

2.4 Knowledge Distillation

Knowledge distillation (KD) is a model compression technique where a smaller model (student) is trained to replicate the behavior of a larger, pre-trained model (teacher) [104]. KD has gained substantial attention due to the increasing demand for efficient deployment of large-scale models, such as large language models (LLMs), on resource-constrained devices. This technique offers a promising approach to reduce model size, computational complexity, and inference time while retaining most of the performance of the original, larger model.

2.4.1 Teacher-Student Framework

In the typical KD setting, the teacher model is a large, over-parameterized neural network, often referred to as the "foundation model" in modern contexts such as LLMs [32]. Foundation models, due to their sheer size and the volume of data they are trained on, capture rich semantic and contextual knowledge across various tasks. However, the size and computational costs of these models hinder their deployment in real-world applications, necessitating the use of distillation.

The student model, on the other hand, is typically smaller and more efficient. During the distillation process, the student learns to mimic the teacher by trying to match the teacher's output distribution, which typically includes both the hard labels (ground truth) and the soft labels (logits or probabilities) produced by the teacher [92]. Formally, given an input x, let the teacher's softmax output be defined as:

$$z_t(x) = \text{softmax}\left(\frac{f_t(x)}{T}\right), \tag{2.19}$$

where $f_t(x)$ represents the logits produced by the teacher model, and T is a temperature hyperparameter that controls the smoothness of the output distribution. As T increases, the output probabilities become softer, which allows the student model to capture the finer-grained knowledge learned by the teacher.

The student model's objective is to minimize the Kullback-Leibler (KL) divergence between its own softened output distribution $z_s(x)$ and the teacher's softened output distribution $z_t(x)$:

$$\mathcal{L}_{KD} = \text{KL}(z_t(x)||z_s(x)), \tag{2.20}$$

where $z_s(x)$ is the softmax output of the student model given by:

$$z_s(x) = \text{softmax}\left(\frac{f_s(x)}{T}\right), \tag{2.21}$$

and $f_s(x)$ are the logits produced by the student model. In practice, the overall loss function for the student often combines the distillation loss with a standard cross-entropy loss on the true labels y, as follows:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{KD} + (1-\alpha)\mathcal{L}_{CE}, \tag{2.22}$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss between the student's predictions and the true labels, and α is a weighting factor that balances the two objectives.

2.4.2 Knowledge Distillation in Large Language Models

The advent of large-scale foundation models, particularly LLMs such as GPT [35] and LLaMA family [233], has further intensified interest in knowledge distillation. These models, often consisting of billions of parameters, achieve state-of-the-art performance across a wide range of natural language processing (NLP) tasks. However, their deployment in practical applications is limited by their computational requirements.

Distilling knowledge from these large models into smaller LLMs or transformer-based architectures is crucial for making them viable for edge computing and real-time applications.

For example, distillation has been successfully applied to BERT models [67], where a smaller model such as DistilBERT [213] is trained to replicate the performance of BERT with significantly fewer parameters and computational overhead. In such scenarios, the student model is often fine-tuned on task-specific data using both the teacher's soft labels and the ground-truth labels.

Furthermore, when distilling large language models, the complexity arises from capturing not only task-specific knowledge but also the deep, multi-layered semantic representations these models learn. This leads to advanced techniques such as layer-wise distillation [120], where the student is encouraged to mimic the internal activations or attention patterns of the teacher at multiple layers of the network.

2.4.3 Challenges and Opportunities

While knowledge distillation has proven effective, several challenges remain. One major challenge is the gap in model capacity between the teacher and student. If the student model is significantly smaller, it may struggle to fully capture the rich knowledge embedded in the teacher model. Recent advancements in progressive distillation [175] and intermediate supervision attempt to mitigate this by gradually increasing the student model's complexity or introducing intermediate losses at various stages of training.

The potential for applying KD to LLMs also brings opportunities for further innovation. For example, knowledge distillation can facilitate the fine-tuning of student models on domain-specific tasks by leveraging the generalist knowledge of foundation models. Additionally, KD enables personalization and adaptation of LLMs to specific hardware constraints, such as mobile devices, making it a critical area of research in the efficient deployment of AI models. In conclusion, knowledge distillation provides an effective method for transferring the vast knowledge encapsulated in large models to smaller, more efficient models. The teacher-student framework, combined with techniques such as softened outputs and layer-wise distillation, enables significant compression of foundation models like LLMs without substantial loss of accuracy. As the size and complexity of foundation models continue to grow, KD will remain an indispensable tool in advancing the accessibility and practicality of these models in real-world applications.

Chapter 3

QDyLoRA: Quantized Dynamic Low-Rank Adaptation

3.1 Overview

Building on the efficiency challenges outlined in the previous chapters—particularly the *memory bottleneck* that limits practical fine-tuning of large foundation models—this chapter focuses on reducing the GPU footprint of parameter-efficient adaptation. Fine-tuning foundation models often requires substantial GPU memory, which restricts access to larger models. While the quantized variant of Low-Rank Adaptation (LoRA), namely QLoRA, significantly alleviates this issue, identifying an efficient LoRA rank remains challenging. Moreover, QLoRA is trained with a pre-defined rank and cannot be reconfigured to lower ranks without additional fine-tuning.

To address these limitations, this chapter proposes **QDyLoRA** (*Quantized Dynamic Low-Rank Adaptation*), an efficient quantization approach for dynamic low-rank adaptation. Motivated by Dynamic LoRA, QDyLoRA can efficiently fine-tune foundation models over a set of pre-defined LoRA ranks within a *single* training run. In particular, QDyLoRA enables fine-tuning **Falcon-40B** for ranks **1 to 64** on a single **32 GB V100 GPU** through one round of fine-tuning. Experimental results show that QDyLoRA is competitive with QLoRA and can outperform it when using the rank selected as optimal for the downstream task [1].

[1]The initial version of QDyLoRA won the Best Paper Award in 2023 ("QDyLoRA: Quantized Dynamic Low-Rank Adaptation for Efficient Large Language Model Tuning"), and an extended version of the work was later published in EMNLP 2024.

3.2 Introduction

The popularity of adopting foundation models, e.g. Large Language Models (LLMs), across a diverse range of downstream tasks has rapidly increased over the past two years. Fine-tuning foundation models has become necessary to enhance their performance and introduce desired behaviors while preventing undesired outputs [70]. However, as the size of these models increases, fine-tuning costs become more expensive. This has led to a large body of research that focuses on improving the efficiency of the fine-tuning stage [160, 170, 110, 75, 226].

Low-rank adapter (LoRA) [110] is a well-known, parameter-efficient tuning (PEFT) method that reduces memory requirements during fine-tuning by freezing the base model and updating a small set of trainable parameters in form of low-rank matrix multiplication added to matrices in the base model. However, the memory demand during fine-tuning remains substantial due to the necessity of a backward pass through the frozen base model during stochastic gradient descent.

Recent research has thus focused on further reducing memory usage by designing new parameter-efficient modules that can be tuned without necessitating gradients from the base models [226]. Alternatively, researchers have explored combining other efficiency strategies with parameter-efficient tuning methods [143, 65].

Among these approaches, QLoRA [65] stands out as a recent and highly efficient fine-tuning method that dramatically decreases memory usage. It enables fine-tuning of a 65-billion-parameter model on a single 48GB GPU while maintaining full 16-bit fine-tuning performance. QLoRA achieves this by employing 4-bit NormalFloat (NF4), Double Quantization, and Paged Optimizers as well as LoRA modules.

However, another significant challenge when utilizing LoRA modules is the need to tune their rank as a hyperparameter. Different tasks may require LoRA modules of varying ranks. In fact, it is evident from the experimental results in the LoRA paper that the performance of models varies a lot with different ranks, and there is no clear trend indicating the optimal rank. On the other hand, any hyperparameter tuning for finding the optimal rank contradicts the primary objective of efficient tuning and is not feasible for very large models. Moreover, when deploying a neural network on diverse devices with varying configurations, the use of higher ranks can become problematic for highly sensitive devices due to the increased parameter count. To address this, one typically has to choose between training multiple models tailored to different device configurations or determining the optimal rank for each device and task. However, this process is costly and time-consuming, even when using techniques like LoRA.

Table 3.1: A comparison between QLoRA and QDyLoRA on the MMLU benchmark, reporting 5-shot test results for LLMs of varying sizes. QDyLoRA is evaluated on ranks [1,2,4,8,16,32,64] and the best rank is reported in brackets.

Dataset	LLaMA-7b		LLaMA-13b		Falcon-40b	
	QLoRA	QDyLoRA	QLoRA	QDyLoRA	QLoRA	QDyLoRA
Alpaca	38.8 [64]	39.7 [16]	47.8 [64]	47.6 [8]	55.2 [64]	57.1 [4]
OASST1	36.6 [64]	36.8 [16]	46.4 [64]	47.2 [8]	56.3 [64]	56.7 [4]
Self-Instruct	36.4 [64]	37.2 [8]	33.3 [64]	41.6 [4]	51.8 [64]	51.1 [4]
FLAN-v2	44.5 [64]	45.9 [4]	51.4 [64]	52.1 [8]	58.3 [64]	60.2 [4]

DyLoRA [235], is a recent PEFT method that aims to address theses challenges by employing dynamic Low-Rank Adapter (DyLoRA). Inspired by nested dropout, this method aims to order the representations of the bottleneck at low-rank adapter modules. Instead of training LoRA blocks with a fixed rank, DyLoRA extends training to encompass a spectrum of ranks in a sorted manner. The resulting low-rank PEFT modules not only provide increased flexibility during inference, allowing for the selection of different ranks depending on the context, but also demonstrate superior performance compared to LoRA, all without imposing any additional training time.

In this chapter, we employ the DyLoRA PEFT method in conjunction with the quantization scheme utilized in the QLoRA work, resulting in QDyLoRA. QDyLoRA has all the aforementioned benefits of DyLoRA but with significant memory reduction both during training and at inference through 4-bit quantization. We utilize QDyLoRA for efficient fine-tuning of LLaMA-7b, LLaMA-13b, and Falcon-40b models across ranks ranging from 1 to 64, all on a single 32GB V100 GPU. Once tuned, we determine the optimal rank by inferring the model on the test set. Our results reveal that the optimal rank can be quite low, yet it outperforms QLoRA.

3.2.1 Related Work

Low-rank PEFT methods These methods aim to fine-tune pre-trained LLMs for specific tasks while minimizing computational and memory resources. Low-rank adaptation techniques were inspired by [3], demonstrating that pre-trained language models possess a low intrinsic dimension. Since then, several works have explored the incorporation of trainable parameters in the form of low-rank up-projection/down-projection during fine-tuning. In [108], the Adapter module includes a down projection, a non-linear function, an

up projection, and a residual connection. These modules are sequentially inserted after the feed-forward network (FFN) or attention blocks.

Additionally, [96] extends the Adapter concept by introducing trainable modules that run in parallel (PA) with the original pre-trained language-model (PLM) module. As a result of this extension, PA has demonstrated improved performance compared to the original Adapter method. One notable approach among these techniques is LoRA [110], which introduces low-rank up-projection/down-projection into various matrices within a PLM. This method offers efficient inference by seamlessly integrating the adapter module into the original model's weight matrices.

Quantization-aware PEFT methods Alpha-Tuning [143], aims to combine parameter-efficient adaptation and model compression. Alpha-Tuning achieves this by employing post-training quantization, which involves converting the pre-trained language model's full-precision parameters into binary parameters and separate scaling factors. During adaptation, the binary values remain fixed for all tasks, while the scaling factors are fine-tuned for the specific downstream task.

QLoRA [65] is a more recent quantization-aware PEFT that combines a low-rank adapter with 4-bit NormalFloat (NF4) quantization and Double Quantization (DQ) of the base model to optimize memory usage. NF4 ensures an optimal distribution of values in quantization bins, simplifying the process when input tensors have a fixed distribution. DQ further reduces memory overhead by quantizing quantization constants.

To manage memory during gradient checkpointing, QLoRA employs Paged Optimizers, utilizing NVIDIA's unified memory feature for efficient GPU memory management. These techniques collectively enable high-fidelity 4-bit fine-tuning while effectively handling memory constraints.

Dynamic PEFT methods DyLoRA paper [235] introduces a novel approach for training low-rank modules to work effectively across a range of ranks simultaneously, eliminating the need to train separate models for each rank.

Inspired by the concept of nested dropout, the authors propose a method for organizing the representations within low-rank adapter modules. This approach aims to create dynamic low-rank adapters that can adapt well to various ranks, rather than being fixed to a single rank with a set training budget. This is achieved by dynamically selecting ranks during training, allowing for greater flexibility without the need for extensive rank searching and multiple model training sessions.

Algorithm 1 QDyLoRA - Training and Inference

Require: $r \in [r_{min}, r_{max}]$; i: the number of training iterations; α: a scaling factor; p_B: probability distribution function for rank selection; $X \in \mathbb{R}^{d \times n}$: all input features to LoRA; $W_0 \in \mathbb{R}^{m \times d}$ the original frozen pre-trained weight matrix, $W_{dw} \in \mathbb{R}^{r \times d}$; $W_{up} \in \mathbb{R}^{m \times r}$; Q: Quantizer; $\mathbb{L}_{\downarrow b}^{DY}$: objective function given truncated weights

Initialization:

$W_0^{NF4} = Q(W_0)$ // Quantize W_0 to NF4

Iterations:

while $t < i$ **do**

Forward:

$b \sim p_B(.)$ // sample a specific rank, during test is given

$W_{dw\downarrow b} = W_{dw}[:b,:]$ // truncate down-projection matrix

$W_{up\downarrow b} = W_{up}[:,:b]$ // truncate up-projection matrix

$W_0^{DDequant-NF4} = \frac{W_0^{NF4}}{c_2^{FP8}/c_1^{FP32}}$ // dequantized the chunks of the parameters that are needed

$h = W_0^{DDequant-NF4} X^{BF16} + \frac{\alpha}{b} W_{up\downarrow b}^{BF16} W_{dw\downarrow b}^{BF16} X^{BF16}$ // calculate the LoRA output

Backward:

$W_{dw\downarrow b}^{BF16} \leftarrow W_{dw\downarrow b}^{BF16} - \eta \nabla_{W_{dw\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{\mathcal{DY}}$

$W_{up\downarrow b}^{BF16} \leftarrow W_{up\downarrow b} - \eta \nabla_{W_{up\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{\mathcal{DY}}$

end while

3.3 Proposed Method: Quantized DyLoRA

Following DyLoRA notations [235], we define a truncated weight $W_{\downarrow b} \in \mathbb{R}^{r \times d}$ as $W[:b,:]$. Assume we have a set of input features $X \in \mathbb{R}^{d \times n}$, a set of pre-trained weights W_0, and a given range of desired ranks represented by $r \in [r_{min}, r_{max}]$ that we want the model to operate with, and a dynamic objective function $L_{\downarrow b}^{DY}$ that can evaluate a truncated sub-model. Then we can use the following equation to calculate the forward pass of the model at each iteration.

$$h = W_0^{\text{DDequant-NF4}} x^{\text{BF16}} + \frac{\alpha}{b} W_{\text{up}\downarrow b}^{\text{BF16}} W_{\text{dw}\downarrow b}^{\text{BF16}} x^{\text{BF16}} \tag{3.1}$$

where α is the LoRA scalar, and b is the chosen rank by the $p_B(.)$ during training stage.

Following QLoRA [65], we used 4-bit Normal Float (NF4) for storing the double quantized pre-trained weights. As all the computations need to be calculated in BFloat16 precision, DDequant-NF4 will dequantize the stored data. Similar to [65], we have:

$$W_0^{DDequant-NF4} = \frac{W_0^{NF4}}{c_2^{FP8}/c_1^{FP32}} \tag{3.2}$$

where c_1^{FP32} and c_2^{FP8} are quantization constants introduced in [65]. After this process, we can update the dynamic LoRA parameters using:

$$
\begin{aligned}
W_{dw\downarrow b}^{BF16} &\leftarrow W_{dw\downarrow b}^{BF16} - \eta \nabla_{W_{dw\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{\mathcal{DY}} \\
W_{up\downarrow b}^{BF16} &\leftarrow W_{up\downarrow b} - \eta \nabla_{W_{up\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{\mathcal{DY}}
\end{aligned}
\tag{3.3}
$$

Algorithm 1 describes the workflow of our proposed QDyLoRA in detail.

3.4 Experiments and Evaluation

This section evaluates the efficiency and efficacy of QDyLoRA through several instruct-fine-tuning tasks. The first experiment compares QDyLoRA with QLoRA on Massively Multitask Language Understating (MMLU) benchmark [100], consisting of more than 50 different tasks, spanning from fundamental mathematics and U.S. history to computer science and law. As shown in Table 3.1[2], we finetune LLaMA-7b, LLaMA-13b, LLaMA2-13b, and Falcon40b on different datasets, Alpaca [230], OASST1 [133], Self-Instruct [244], and FLAN-v2 [53], using QLoRA and QDyLoRA techniques. We use the same training budget and maximum LoRA rank [3] for each technique. The results consistently show that QDyLoRA achieves a superior performance by finding the optimal rank.

The second experiment provides a more in-depth comparison between QLoRA and QDyLoRA. In particular, we fairly finetuned Falcon-40b on WebGLM [164] and GSM8k [56] benchmarks, and compared their test performances across different ranks. As described in Table 3.3, QDyLoRA attains superior performance, notably when employing its optimal ranks (Rank 2 for Web-GLM and Rank 8 for GSM8k). Furthermore, QDyLoRA exhibits consistent superiority over QLoRA, particularly at lower ranks. These findings emphasize the adaptive nature of QDyLoRA in dynamically adjusting its focus during fine-tuning, leading to enhanced efficiency and efficacy compared to its static counterpart, QLoRA.

The third experiment compares the performance of DyLoRA, QDyLoRA, and QLoRA on GSM8k and TriviaQA [122] while adopting LLaMA2-13b and LLaMA-7b as LLMs. Table 3.4 reports the results. As the table illustrates, for smaller-size models, i.e. LLaMA-7b, DyLoRA and QDyLoRA both perform superior than QLoRA. For larger models, i.e. LLaMA2-13b, DyLoRA fails due to the out-of-memory (OOM) error while QDyLoRA works the best in such situations.

[2]The same settings as the original QLoRA work are applied here.

[3]The maximum LoRA rank is fixed to 64. While QLoRA's rank is always fixed, QDyLoRA can split the training across ranks in range 1 to 64.

Table 3.3: Comparing the performance of QLoRA and QDyLoRA across different evaluation ranks. Both models receives the same training settings. Maximum LoRA rank is set to 64. Falcon-40b is adopted as the base LLM. Exact matching and Bleu-score are used as evaluation measurements for GSM8k and Web-GLM, respectively.

Data	Method	Rank						
		1	2	4	8	16	32	64
Web-GLM	QLoRA	19.9	19.9	19.9	33.8	35.2	52.7	54.3
	QDyLoRA	43.3	**56.0**	54.9	53.3	53.3	50.5	50.2
GSM8k	QLoRA	8.9	8.91	8.9	15.1	20.5	22.6	28.1
	QDyLoRA	21.4	25.3	28.2	**30.6**	29.8	28.5	27.4

Table 3.4: Comparing the performance of DyLoRA, QLoRA and QDyLoRA across different evaluation ranks. all models receives the same training settings. Maximum LoRA rank is set to 64. The results are reported in terms of exact matching.

Data;LLM	Method	Rank						
		1	2	4	8	16	32	64
GSM8K;LLaMA-7b	DyLoRA	12.96	16.91	17.06	**19.94**	18.50	18.35	14.94
	QLoRA	0.0	0.0	0.0	0.0	0.0	0.0	12.66
	QDyLoRA	12.59	15.09	18.50	16.76	16.91	18.65	14.71
TriviaQA;LLaMA-7b	DyLoRA	19.27	23.20	22.99	23.32	23.25	**24.12**	22.43
	QLoRA	0.0	0.0	0.0	0.0	0.0	0.0	15.52
	QDyLoRA	6.66	12.49	17.16	19.51	20.09	21.65	20.27
GSM8K;LLaMA2-13b	DyLoRA	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	QLoRA	0.0	0.0	0.0	0.0	0.0	0.0	21.08
	QDyLoRA	1.90	15.01	22.97	**25.55**	24.26	23.81	22.08

3.5 On the semi-sorted behavior of QDyLoRA

As shown in Table 3.3, QDyLoRA reveals a semi-sorted performance across ranks. We justify this behavior by pointing out the limited finetuning budget. In a limited budget assumption, QDyLoRA updates its lower ranks more frequently than its higher ranks. That is because of the fact that lower ranks are also updated when higher ranks are selected. In other words, lower ranks have more chance to get updated than higher ranks. Hence, lower ranks are more tuned than higher ranks.

3.5.1 Generated Text Quality

To describe the quality of texts generated by QDyLoRA, a sample query taken from GSM8k is fed to Falcon-40b trained by QDyLoRA. Below, we report the generated answers for different LoRA ranks.

Query: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

Rank=64: Janet has 16 - 3 - 4 = «16-3-4=7»7 eggs left. She sells 7 - 16 = «7-16=9»9 eggs at the farmers' market.She makes 9 * 2 =«9*2=18»18 at the farmers' market every day.

Rank=32: She eats 3 eggs for breakfast and bakes 4 muffins per day, so she uses 3 + 4 = «3+4=7»7 eggs per day. She has 16 - 7 = «16-7=9»9 eggs left over each day. She sells these eggs at the farmers' market for $2 per egg, so she makes 2 * 9 = «2*9=18»18 per day at the market.

Rank=28: She eats 3 eggs for breakfast and bakes 4 muffins per day, so she uses 3+4 = «3+4=7»7 eggs per day. She sells 16-7 = «16-7=9»9 eggs per day for $2 per egg. She makes 9*2 = «9*2=18»18 per day at the market.

Rank=16: Janet's ducks lay 16 eggs per day and she eats 3 for breakfast every morning so she has 16-3=13 eggs left. Janet bakes 4 muffins every day and she uses 4 eggs so she has 13-4=9 eggs left. Janet sells the remaining eggs at the farmers' market for $2 each so she makes 9*2=«9*2=18»18 dollars a day. Janet makes 18 dollars a day at the farmers' market.

3.6 Conclusion

QDyLoRA offers an efficient and effective technique for LoRA-based fine-tuning LLMs on downstream tasks. Eliminating the need for fine-tuning multiple models to find the optimal LoRA rank and offering the possibility of fine-tuning larger LLMs are two main advantages of QDyLoRA. The experimental results demonstrated that the optimal rank for QDyLoRA can be surprisingly low, yet it consistently outperforms QLoRA. QDyLoRA provides greater flexibility for deploying LLMs in various contexts and represents a promising step towards making fine-tuning large language models more accessible and efficient.

Limitations

While the 4-bit QDyLoRA exhibits notable performance, it falls short of achieving the performance levels of full precision fine-tuning. One possible solution could be dynamic quantized DyLoRA (DyQDyLoRA), in which the quantization level could also vary during finetuning. In particular, the finetuning strategy can dynamically switch between different quantization levels based on a predefined learning feedback. Additionally, further research is required to investigate the impact of LoRA's scalar and the range of underlying ranks in QDyLoRA.

Chapter 4

Sorted-LoRA: Sorted Low-Rank Adaptation

4.1 Overview

Following the memory-efficient adaptation setting introduced in the previous chapter, we now consider a complementary deployment challenge: *even when fine-tuning is affordable, serving LLMs across heterogeneous hardware often demands multiple model variants with different computational footprints.* Traditional approaches train or distill each variant independently, incurring substantial cost and redundancy.

To address this challenge, we propose **Sorted-LoRA**, a unified fine-tuning framework that enables the extraction of multiple sub-models of varying depths from a single large-scale model within one training cycle. Sorted-LoRA integrates *Low-Rank Adaptation (LoRA)* with a *stochastic depth training schedule*, where in each iteration a random layer depth is sampled and only the corresponding low-rank adapters are updated. This design efficiently co-trains a hierarchy of sub-models without modifying the original architecture or introducing auxiliary supervision.

Empirical evaluations on LLaMA2-7B, LLaMA3-8B, and a 1B-scale model demonstrate that Sorted-LoRA achieves a superior balance between accuracy and computational efficiency compared to state-of-the-art dynamic inference baselines such as Balcony, LayerSkip, and Flextron. Remarkably, the **XL configuration of Sorted-LoRA surpasses the base LLaMA2-7B model**, achieving an average accuracy of **68.44% (+0.34)** while smaller variants maintain competitive performance with graceful degradation (–**1.32**, –**3.54**, and

–**6.36** for L, M, and S, respectively). By coupling parameter-efficient low-rank tuning with depth-aware training, Sorted-LoRA provides a scalable and cost-effective paradigm for building multi-scale LLMs capable of dynamic deployment across devices with diverse resource constraints.

4.2 Introduction

The rapid expansion of foundation models has introduced an unprecedented range of computational and deployment requirements across devices and organizations. While large enterprises benefit from high-capacity models serving millions of users, resource-constrained platforms such as mobile or edge devices demand compact yet capable variants. Even within a single enterprise, heterogeneous deployment scenarios often necessitate maintaining several model scales optimized for distinct latency, memory, or energy constraints. This diversity compels practitioners to pre-train or fine-tune multiple model sizes independently—an approach that is both computationally prohibitive and operationally complex [139]. For example, Meta's LLaMA family [233] encompasses multiple model sizes trained separately to accommodate a spectrum of user needs, from research prototypes to large-scale production systems.

In this chapter, we address the central challenge of generating multiple deployable sub-models of varying depths *from a single large-scale model*, without retraining or duplicating effort. Our objective is to produce **nested, performance-aligned sub-models** that can be directly used for inference under different compute budgets, while preserving the fidelity of the full model.

To this end, we introduce **Sorted-LoRA**, a unified framework that merges the parameter-efficient philosophy of *Low-Rank Adaptation (LoRA)* [110] with the progressive, depth-aware training dynamics of *SortedNet* [236]. Sorted-LoRA introduces lightweight low-rank adapters into each transformer layer of a frozen backbone and trains them under a *stochastic depth schedule*: at each iteration, a random depth k is sampled, and only the corresponding subset of LoRA modules in the first k layers are updated. This simple yet powerful mechanism allows all sub-models—from shallow to full-depth—to co-evolve within a single training cycle, sharing representations while remaining independently deployable.

Compared to prior approaches, Sorted-LoRA offers an elegant balance between **architectural simplicity, adaptability, and efficiency**:

- **Balcony** [117] achieves dynamic depth inference by inserting additional transformer layers and employing self-distillation, but it introduces architectural overhead and

auxiliary optimization. Sorted-LoRA achieves comparable flexibility *without adding any new layers or distillation stages*, instead leveraging LoRA's low-rank adapters for efficient adaptation within the existing backbone.

- **LayerSkip** [78] enables early-exit inference through dropout-based training and auxiliary exit losses, but such perturbations alter the original network's dynamics. In contrast, Sorted-LoRA preserves the backbone's integrity, maintaining full-model consistency across all sub-models.
- **Flextron** [39] explores nested sub-networks and router-based token routing for adaptive computation, yet its routing mechanism increases training and inference complexity. Sorted-LoRA achieves a similar many-in-one flexibility purely through stochastic layer activation and shared low-rank parameters—a simpler, scalable, and router-free solution.

Sorted-LoRA enables *multi-scale inference within a single fine-tuning budget*, producing a hierarchy of sub-models tailored to diverse hardware constraints, without architectural modification or additional supervision. By coupling stochastic depth scheduling with LoRA's parameter efficiency, Sorted-LoRA establishes a new paradigm for cost-aware, flexible adaptation of large language models.

4.3 Proposed Method

The architecture of Sorted-LoRA, dipicted in Figure 4.1, is designed to create a hierarchy of sub-models nested within the main model. Each layer of the main model is equipped with a LoRA module, which consists of trainable low-rank matrices that adapt the frozen weights for specific tasks. The key components of Sorted-LoRA are:

- **Frozen Main Model:** The weights of the main model, denoted as W_0, are kept fixed during training to serve as a shared backbone for all sub-models.
- **LoRA Modules:** For each layer i, we introduce a LoRA module consisting of low-rank matrices $A_i \in \mathbb{R}^{r \times d}$ and $B_i \in \mathbb{R}^{d \times r}$, where d is the dimension of the layer, and $r \ll d$ is the rank.
- **Random Layer Selection:** At each training step, we randomly sample a depth $k \in \{1, 2, \ldots, L\}$, where L is the total number of layers. We then construct a sub-model using the first k layers.

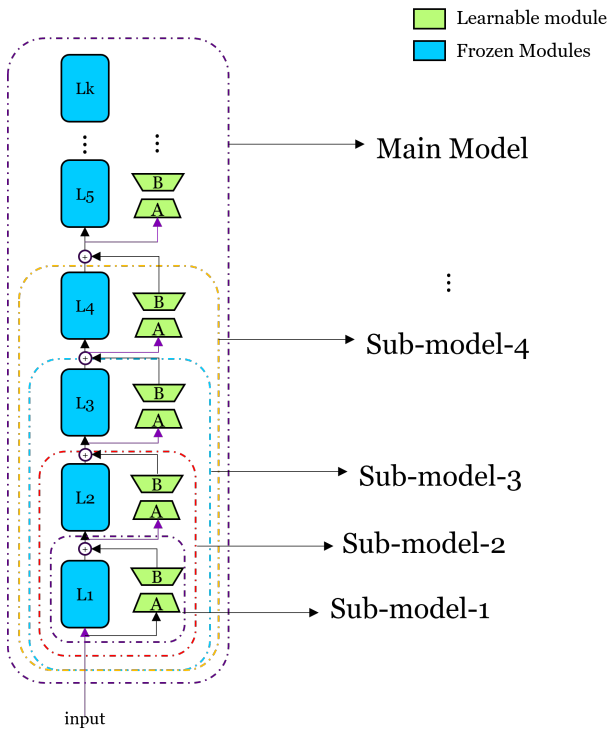


Figure 4.1: Architecture of Sorted-LoRA. The main model layers are frozen (blue blocks), and LoRA modules (green blocks) are added to each layer. During training, a sub-model up to depth k is randomly selected, and only the LoRA parameters of those layers are updated.

The forward pass for a sub-model of depth k is performed by sequentially applying the LoRA-adapted layers up to layer k. The LoRA modules enable each sub-model to learn task-specific adaptations while sharing the underlying parameters of the main model.

4.3.1 Mathematical Formulation

Let x be the input to the model, and let h_i denote the output of layer i. The computation at layer i during the forward pass is defined as:

$$h_i = \phi\left(W_0^i h_{i-1} + B_i A_i h_{i-1} + b^i\right), \tag{4.1}$$

where:

- $W_0^i \in \mathbb{R}^{d \times d}$ is the frozen weight matrix of layer i in the main model.
- $A_i \in \mathbb{R}^{r \times d}$ and $B_i \in \mathbb{R}^{d \times r}$ are the trainable LoRA matrices for layer i.
- b^i is the bias vector for layer i (can be frozen or trainable depending on implementation).
- $\phi(\cdot)$ is the activation function (e.g., GELU, ReLU).

The term $B_i A_i h_{i-1}$ represents the low-rank adaptation applied to the input of layer i. By keeping $r \ll d$, we ensure that the number of trainable parameters remains small.

At each training step, we randomly sample a depth k from a predefined distribution (e.g., uniform over $\{1, 2, \ldots, L\}$). The sub-model consists of the first k layers, and the forward pass is computed as:

$$h_k = f_k(x) = \phi\left(W_0^k h_{k-1} + B_k A_k h_{k-1} + b^k\right), \tag{4.2}$$

where $h_0 = x$. Only the LoRA parameters $\{A_i, B_i\}_{i=1}^k$ of the selected sub-model are updated during this training step. Algorithm 2 outlines the training procedure for Sorted-LoRA.

Algorithm 2 Sorted-LoRA Training Algorithm

Input: Training data $\mathcal{D}$, main model parameters $\{W_0^i\}_{i=1}^L$, LoRA parameters $\{A_i, B_i\}_{i=1}^L$, total layers L, number of training steps T
Initialize: Freeze $\{W_0^i\}_{i=1}^L$, randomly initialize $\{A_i, B_i\}_{i=1}^L$
for each training step $t = 1$ to T **do**
  Sample a minibatch $(x, y) \sim \mathcal{D}$
  Randomly sample depth $k \in \{1, 2, \ldots, L\}$
  Forward Pass:
  $h_0 = x$
  for layer $i = 1$ to k **do**
    $h_i = \phi\left(W_0^i h_{i-1} + B_i A_i h_{i-1} + b^i\right)$
  end for
  Compute loss $\mathcal{L}(h_k, y)$
  Backward Pass:
  Compute gradients $\nabla_{A_i}\mathcal{L}$, $\nabla_{B_i}\mathcal{L}$ for $i = 1$ to k
  Update LoRA parameters $\{A_i, B_i\}_{i=1}^k$ using optimizer (e.g., Adam)
end for

4.3.2 Inference with Sub-Models

After training, we obtain a range of sub-models of different depths that can be used for inference without additional training. For a desired model size (i.e., depth k), we can select the corresponding sub-model consisting of the first k layers with their LoRA adaptations:

$$f_k(x) = h_k = \phi\left(W_0^k h_{k-1} + B_k A_k h_{k-1} + b^k\right). \tag{4.3}$$

This flexibility allows deployment across devices with varying computational resources, as smaller sub-models require less computation and memory.

4.3.3 Advantages of Sorted-LoRA

The Sorted-LoRA method offers several benefits:

- **Parameter Efficiency:** By freezing the main model and only training low-rank adaptations, we significantly reduce the number of trainable parameters.
- **Single Training Budget:** All sub-models are trained simultaneously within a single training run, eliminating the need for separate training processes.
- **Scalability:** The method scales well to large models and can produce a wide range of sub-model sizes.
- **Inference Flexibility:** Users can select a sub-model that fits their computational constraints without sacrificing performance on the task.

4.3.4 Comparison with Other Methods

Sorted-LoRA builds upon the concepts of LoRA [110] and sorted training methods like SortedNet [236]. Unlike traditional fine-tuning or knowledge distillation approaches, Sorted-LoRA does not require additional training for each sub-model and maintains high performance across different model sizes.

4.4 Experiments

4.4.1 Setup

Model Configuration. To evaluate the effectiveness of SORTED-LORA, we adopt the same experimental settings used in recent dynamic inference studies for a fair comparison. Experiments are carried out on three model scales: **LLaMA3-8B**, **LLaMA2-7B**, and a lightweight **LLM-1B** variant that follows the LLaMA3 architecture but trained on smaller corpora. The 1B-scale model is pre-trained using the *FineWebEDU* and *Cosmopedia V2* datasets included in the SMOLLM corpus [22], providing a compact yet representative platform for controlled experimentation.

In the Sorted-LoRA setup, each transformer layer is augmented with a low-rank adapter. For constructing sub-models of different capacities, we sample model depths corresponding to approximately 25% (S), 50% (M), 75% (L), and 100% (XL) of the total layers. Specifically, for the 1B variant, we select depths at layers 4, 8, and 12; for the LLaMA2-7B and LLaMA3-8B models, the selected layers are 15, 18, and 21, respectively. The full backbone (XL) represents the untruncated model equipped with LoRA modules in all layers.

Baselines. We benchmark Sorted-LoRA against leading adaptive and compression-based approaches representing both width- and depth-based paradigms. For width elasticity, we include **Flextron** [39], which dynamically routes tokens through nested subnetworks. For depth-based adaptation, we consider **LayerSkip** [78], which employs progressive layer dropout and early-exit losses.

Since Flextron's codebase is not publicly released, runtime speedup measurements are not directly reported; however, prior results indicate that width-based scaling achieves smaller latency gains than depth-based approaches at equivalent parameter budgets. Our setup therefore focuses on comparable configurations by matching the number of active (non-embedding) parameters across all evaluated methods.

Evaluation Protocol. We assess model performance across eight standard reasoning and understanding benchmarks: **ARC** [54], **BoolQ (BQ)** [55], **OpenBookQA (OBQA)** [172], **PIQA** [30], **WinoGrande (WG)** [211], **LAMBADA (LMBD)** [187], **MMLU (5-shot)** [100], and **HellaSwag (10-shot)** [269]. All evaluations follow the unified protocol of the LM-EVALUATION HARNESS framework [86]. This benchmark suite measures both linguistic and reasoning competence while capturing performance trade-offs across model scales, allowing us to quantify Sorted-LoRA's effectiveness in producing computationally efficient yet semantically aligned sub-models.

Table 4.1: Comparison of downstream task performance between Sorted-LoRA and state-of-the-art dynamic inference baselines. All models are evaluated on zero-shot ARC-E, LAMBADA (LMBD), PIQA, and WinoGrande (WG), and 5-shot MMLU. The reported numbers represent accuracy (%).

Method	Model	# Params	ARC-E	LMBD	PIQA	WG	MMLU (5)	Avg. (Drop)
Base	LLaMA2-7B (Full)	6.5B	76.3	71.1	78.1	69.1	45.9	68.10 (–)
Balcony	XL	6.5B	76.3	71.1	78.1	69.1	45.9	68.10 (0.00)
	L	4.4B	72.0	67.0	75.9	67.5	45.0	65.48 (-2.62)
	M	3.8B	68.9	61.3	75.2	66.0	43.0	62.88 (-5.22)
	S	3.2B	64.9	54.9	73.5	63.8	39.8	59.38 (-8.72)
LayerSkip	XL	6.5B	76.5	70.5	77.6	70.3	43.2	67.62 (-0.48)
	L	4.4B	68.5	65.9	73.7	66.4	42.4	63.38 (-4.72)
	M	3.8B	61.4	55.1	71.1	65.8	42.4	59.16 (-8.94)
	S	3.2B	50.3	43.5	68.6	63.9	37.8	52.82 (-15.28)
Flextron	Full	6.5B	75.1	71.5	77.5	69.1	45.1	67.66 (-0.44)
	0.7×	4.1B	68.6	65.1	76.1	63.7	42.2	63.14 (-4.96)
	0.6×	3.9B	67.1	63.8	74.9	62.2	39.4	61.48 (-6.62)
	0.5×	3.4B	66.5	62.9	74.1	62.0	36.8	60.46 (-7.64)
Sorted-LoRA (Ours)	XL	6.5B	**76.8**	**71.3**	**78.4**	**69.5**	**46.2**	**68.44 (+0.34)**
	L	4.4B	**74.1**	69.0	77.0	68.4	45.4	66.78 (-1.32)
	M	3.8B	70.9	65.2	75.9	67.0	43.8	64.56 (-3.54)
	S	3.2B	67.8	60.4	74.0	65.5	41.0	61.74 (-6.36)

4.4.2 Results and Discussion

Table 4.1 summarizes the comparative evaluation of Sorted-LoRA against recent dynamic inference frameworks. Across all model scales, Sorted-LoRA consistently achieves a favorable trade-off between accuracy and efficiency, demonstrating the effectiveness of low-rank, depth-aware training in preserving the knowledge of the full backbone. Unlike prior dynamic schemes that rely on architectural modification or auxiliary supervision, Sorted-LoRA maintains the frozen backbone intact and leverages stochastic depth updates to produce coherent sub-models of varying depth.

Notably, the XL configuration of Sorted-LoRA surpasses the baseline full LLaMA2-7B model, attaining an average accuracy of **68.44%**—a **+0.34** improvement over the base model. This gain, despite the frozen backbone and limited trainable parameters, indicates that Sorted-LoRA not only retains the representational strength of the original network but also provides a mild regularization benefit through its stochastic training dynamics. Such improvement contrasts with competing approaches, where the full or XL configurations typically exhibit slight degradation due to architectural perturbations or overfitting during fine-tuning.

For smaller sub-models (L, M, and S), Sorted-LoRA yields graceful performance degra-

dation relative to model depth, with accuracy drops of only **−1.32**, **−3.54**, and **−6.36** points, respectively—smaller than those observed for Balcony (**−2.62** to **−8.72**), LayerSkip (**−4.72** to **−15.28**), and Flextron (**−4.96** to **−7.64**). This demonstrates that Sorted-LoRA preserves feature alignment between shallow and deep sub-models more effectively, ensuring that early layers capture task-relevant representations without destabilizing the global optimization. These results confirm that Sorted-LoRA delivers robust multi-scale adaptability while improving full-model performance, achieving better parameter efficiency, and requiring no additional modules or self-distillation losses. The ability to outperform the base model at full depth while maintaining competitive accuracy at reduced compute budgets underscores Sorted-LoRA as a practical and scalable solution for cost-aware deployment of large language models.

4.5 Conclusion

In this chapter, we presented **Sorted-LoRA**, a depth-aware, low-rank adaptation framework that enables the construction of multiple sub-models of varying capacities from a single pretrained large language model. By combining LoRA's parameter efficiency with stochastic depth training, Sorted-LoRA eliminates the need for additional layers, routers, or self-distillation stages, while maintaining the integrity of the frozen backbone. This design allows all sub-models—from shallow to full-depth—to co-evolve in a unified optimization process, significantly reducing fine-tuning cost and complexity.

Empirical results demonstrate that Sorted-LoRA not only maintains competitive performance across sub-models but also **surpasses the baseline LLaMA2-7B full model**, achieving an average accuracy of **68.44%** in its XL configuration—an improvement of **+0.34** points despite training under frozen weights. This highlights the implicit regularization effect of stochastic layer activation and the representational benefits of low-rank adaptation. Furthermore, smaller variants of Sorted-LoRA (L, M, and S) exhibit a smooth accuracy decline of only **−1.32**, **−3.54**, and **−6.36** points, respectively, clearly outperforming existing dynamic inference baselines such as Balcony, LayerSkip, and Flextron at equivalent parameter budgets.

Sorted-LoRA establishes a new paradigm for **multi-scale, cost-aware adaptation** in large language models. It provides a single training pipeline capable of producing a family of deployable sub-models that retain high task fidelity, minimal degradation under reduced depth, and improved full-model robustness. These findings confirm that Sorted-LoRA not only achieves efficiency and scalability but also enhances the expressive capacity of

the underlying model, making it an attractive approach for flexible deployment across heterogeneous compute environments.

Chapter 5

LoRA-Drop: Accelerating Inference Speed in Autoregressive Models

5.1 Overview

While the previous chapters focused on improving efficiency during fine-tuning and enabling multi-scale deployment, the next bottleneck arises at *serving time*: even a well-adapted model can be expensive to run under real-time latency constraints. Autoregressive large language models (LLMs) suffer from high inference latency because every generated token requires processing through the entire transformer stack. Recent attempts to mitigate this cost—such as layer skipping and dynamic-depth decoding—have achieved partial success but often depend on auxiliary routers that introduce runtime overhead or degrade model quality by discarding full layers without compensation.

We propose **LoRA-Drop**, a parameter-efficient fine-tuning framework that accelerates inference by dynamically substituting selected layers with lightweight *Low-Rank Adaptation* (LoRA) modules during decoding. Instead of bypassing computation entirely, LoRA-Drop maintains representational continuity by reusing hidden states from the previous time step and applying low-rank updates for semantically simple or repetitive tokens, while periodically reactivating full layers to preserve contextual fidelity. This design is fully compatible with key–value caching, introduces negligible scheduling overhead, and yields stable, wall-clock acceleration.

Across multiple benchmarks and model families (LLaMA2–7B, LLaMA3–8B, Qwen2.5–7B, and Qwen2.5–14B), LoRA-Drop achieves up to **2.6× faster decoding** and **∼50% reduction in KV-cache memory** with less than **0.5%** loss in accuracy. By unifying efficient

fine-tuning with selective computation, LoRA-Drop offers a simple yet effective framework for latency-aware LLM deployment in real-world applications [1].

5.2 Introduction

Large Language Models (LLMs) have emerged as a cornerstone of modern artificial intelligence, demonstrating remarkable performance across a wide range of natural language processing tasks such as reasoning, code generation, and dialogue systems [35, 51, 183]. Their ability to generalize across domains with minimal task-specific supervision has driven widespread adoption in both academia and industry, powering applications in search engines, conversational agents, recommendation systems, and enterprise productivity tools [232, 36, 231].

Despite their success, the practical deployment of LLMs is often hindered by the high computational cost of autoregressive inference, where each token must be generated sequentially through the entire stack of transformer layers. This process leads to substantial latency and energy consumption, especially for long sequences and interactive applications. To address this challenge, several lines of research have emerged:

Model Compression and Quantization- Methods such as weight pruning [82, 167] and low-bit quantization [65, 254, 198] reduce the parameter footprint and arithmetic cost of LLMs, enabling more efficient deployment on constrained hardware. While effective for reducing memory and throughput requirements, these approaches often require delicate tuning to balance efficiency with model accuracy.

Efficient Attention and KV-Caching- Given that attention layers dominate LLM inference cost, many works focus on optimizing sequence processing. KV-caching [134] reduces redundant computation across tokens, while efficient attention variants such as FlashAttention [59] and streaming/context-parallel attention [58, 280] accelerate long-context inference. These methods improve throughput but do not directly address the redundancy across layers during generation.

Dynamic Computation and Layer Skipping- More recently, researchers have proposed adaptive computation strategies that reduce the number of active layers or sublayers per token. Examples include Unified Layer Skipping [162], FlexiDepth [166], AdaSkip [98], ColT5 [8], Balcony [117], and FiRST [115], which either skip intermediate layers or use lightweight routers and adapters to allocate variable computational budgets

[1] This work is available as an arXiv preprint and is currently under review at top-ranked journals.

across tokens. These methods highlight that not all tokens require the full depth of the model, though many approaches either incur additional routing overhead or bypass layers entirely without compensatory transformations, leading to potential degradation.

Recent interpretability studies have shown that the intermediate layer representations of large language models (LLMs) are remarkably expressive, often containing sufficient information to anticipate not only the next token but even subsequent ones. Logit Lens [179, 246] first, and recently demonstrated that hidden states from early and middle layers already encode meaningful token distributions when projected through the model's output embedding, suggesting that prediction refinement, rather than new information synthesis, occurs as tokens propagate through deeper layers. Building on this, Tuned Lens [20] aligned each layer's representation to the output space via lightweight linear transformations, revealing that layer outputs form a coherent progression of increasingly confident latent predictions. Extending this temporal perspective, Future Lens [184] showed that a single hidden state can often anticipate multiple future tokens, indicating significant temporal redundancy across decoding steps. Together, these works suggest that the intermediate activations of autoregressive transformers already encapsulate much of the predictive signal used in later computations, implying that full-depth inference at every time step is not always necessary.

While existing dynamic computation methods have leveraged similar intuitions to reduce inference cost, most rely on complex routing mechanisms that introduce runtime overhead or simply skip layers without any compensatory transformation, leading to degradation in generation quality. In contrast, LoRA-Drop operationalizes the representational richness of intermediate layers by directly reusing the information embedded in layer outputs from time step $t-1$. Instead of discarding skipped layers, it activates lightweight low-rank adaptation (LoRA) modules at time t, preserving contextual continuity while reducing computation. This design offers two main advantages. First, it enables fine-grained control over the drop ratio—how frequently the full model is invoked versus the lightweight LoRA pathway—allowing practitioners to flexibly balance latency and accuracy. Second, LoRA-Drop integrates seamlessly into pretrained LLMs through a few rounds of continual fine-tuning, requiring no architectural modification or retraining from scratch. This plug-and-play property makes LoRA-Drop a practical and scalable acceleration strategy that capitalizes on the latent predictive capacity of LLM layers while maintaining compatibility with existing deployment pipelines.

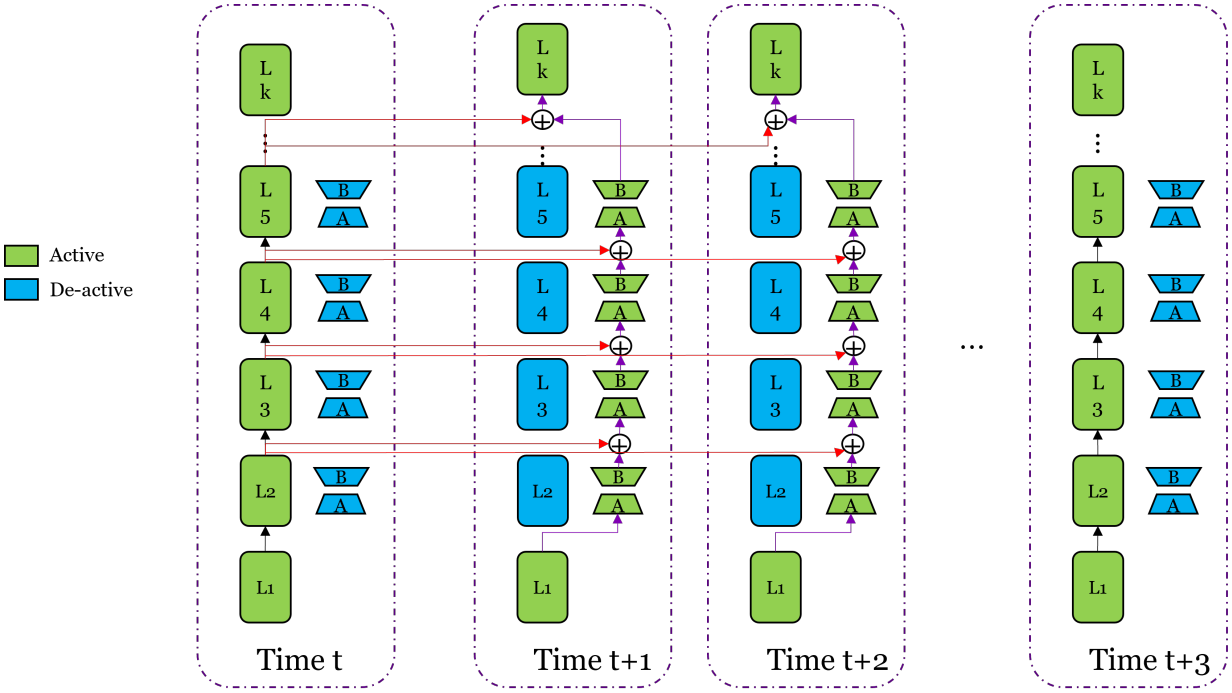


Figure 5.1: The workflow of **LoRA-Drop**. Each transformer layer is augmented with a low-rank adaptation module (matrices A_i and B_i). During low-complexity steps, only these lightweight modules are activated, bypassing the main feedforward or attention blocks to reduce computation. At periodic or complexity-triggered steps, the full layers are reactivated to refine representations.

5.3 Proposed Method: LoRA-Drop

The goal of **LoRA-Drop** is to accelerate the inference process of autoregressive large language models (LLMs) by dynamically modulating the model's computational capacity according to the complexity of the token being generated. Instead of computing all transformer layers at every time step, LoRA-Drop allows certain layers to be *skipped* while preserving representational continuity through lightweight low-rank adaptation (LoRA) modules. The approach leverages the empirical observation that intermediate layer representations of LLMs already encode rich predictive information about upcoming tokens [179]. Thus, the full model depth is not always required for predicting simpler or repetitive tokens.

5.3.1 Model Formulation

Consider an n-layer autoregressive transformer generating a sequence of tokens $\{t_1, t_2, \ldots, t_T\}$. Let $x_t^i \in \mathbb{R}^d$ denote the hidden representation at layer i and time step t, where d is the model dimension. The standard forward propagation for layer i is given by

$$x_{t+1}^i = f^i(x_t^i), \tag{5.1}$$

where $f^i(\cdot)$ represents the transformation performed by the attention and MLP blocks of layer i.

In LoRA-Drop, each intermediate layer i is equipped with a *LoRA module* consisting of two learnable low-rank matrices $A_i \in \mathbb{R}^{r\times d}$ and $B_i \in \mathbb{R}^{d\times r}$, with rank $r \ll d$. These matrices approximate a residual transformation of the form

$$W_L^i = B_i A_i, \qquad \text{rank}(W_L^i) = r. \tag{5.2}$$

During selected time steps, the main layer computation $f^i(\cdot)$ is bypassed and replaced by this lightweight linear mapping:

$$\hat{x}_{t+1}^i = x_t^i + \alpha\, W_L^i x_t^i, \tag{5.3}$$

where α is a scaling coefficient controlling the contribution of the LoRA update. Equation (5.3) serves as a compressed surrogate for the full layer output in (5.1).

5.3.2 Temporal Scheduling of Layer Activation

The inference process alternates between two computational modes:

- **LoRA mode (lightweight phase):** for low-complexity tokens or pre-specified steps, only LoRA modules are activated; the main layers are bypassed. This yields a fast linear transformation with cost $\mathcal{O}(rd)$.
- **Full mode (refinement phase):** every k decoding steps (or based on a learned gating signal), all layers $f^i(\cdot)$ are reactivated to restore full model capacity and update contextual representations.

The switching policy between these modes can follow either a fixed schedule or an adaptive function based on token complexity metrics (e.g., token entropy or frequency). Denoting

the activation indicator of layer i at time t as

$$\delta_t^i = \begin{cases} 1, & \text{if full layer } L_i \text{ is active,} \\ 0, & \text{if only LoRA module is active,} \end{cases}$$

the unified update rule becomes

$$x_{t+1}^i = \delta_t^i \, f^i(x_t^i) + (1 - \delta_t^i) \left(x_t^i + \alpha W_L^i x_t^i \right). \tag{5.4}$$

5.3.3 LoRA-Drop Inference Algorithm

Algorithm 3 formalizes the decoding process under LoRA-Drop. Each new token generation step decides whether to invoke the full layer computation or its LoRA surrogate according to the schedule δ_t^i. The approach preserves compatibility with key–value caching since both update paths maintain consistent hidden-state dimensionality.

Algorithm 3 LoRA-Drop Inference

Input: Token sequence $\{t_1, \ldots, t_T\}$, model layers $\{L_i\}_{i=1}^n$, LoRA modules $\{W_L^i\}_{i=1}^n$, activation period k
Output: Generated token sequence $\{\hat{t}_1, \ldots, \hat{t}_T\}$
for each token t_{t+1} to generate **do**
  for each layer $i = 1, 2, \ldots, n$ **do**
    if $(t+1) \bmod k = 0$ **then**
      Full layer active: $x_{t+1}^i = f^i(x_t^i)$
    else
      LoRA mode: $x_{t+1}^i = x_t^i + \alpha W_L^i x_t^i$
    end if
  end for
  Predict next token $\hat{t}_{t+1} = \arg\max p(t_{t+1} \mid \hat{t}_1, \ldots, \hat{t}_t)$
end for

5.3.4 Computational Analysis

Let C_{full} denote the cost of a standard transformer layer and C_{LoRA} the cost of a LoRA module. For rank $r \ll d$, we have $C_{\text{LoRA}} \approx \mathcal{O}(rd)$ and $C_{\text{full}} \approx \mathcal{O}(d^2)$. Over a decoding

sequence of T tokens with activation ratio $\rho = \frac{k_{\text{full}}}{T}$ (fraction of steps where full layers are active), the expected per-token cost becomes

$$\mathbb{E}[C_{\text{LoRA-Drop}}] = \rho\, C_{\text{full}} + (1 - \rho)\, C_{\text{LoRA}}. \tag{5.5}$$

Hence, the theoretical speedup scales approximately as

$$S \approx \frac{C_{\text{full}}}{\rho\, C_{\text{full}} + (1 - \rho)\, C_{\text{LoRA}}}. \tag{5.6}$$

In practice, since LoRA modules contain orders-of-magnitude fewer parameters than the original layer weights, significant reductions in inference latency can be achieved without retraining the backbone network.

5.3.5 Integration and Fine-Tuning

LoRA-Drop can be seamlessly integrated into existing pretrained models by inserting LoRA modules in the desired subset of layers and performing a few rounds of continual fine-tuning on a small corpus. This process adapts the low-rank parameters $\{A_i, B_i\}$ while freezing the original model weights, preserving pretrained knowledge. The method thus enables *post-hoc acceleration* of any transformer-based LLM without altering its architecture or requiring full retraining.

5.4 Experiments

We evaluate the effectiveness of **LoRA-Drop** across multiple open-weight large language models (LLMs) and a broad suite of reasoning, knowledge, and commonsense benchmarks. Our experiments systematically assess the impact of LoRA-Drop on both model accuracy and inference efficiency across diverse model families and scales. Specifically, we analyze how selective activation of transformer layers influences decoding latency, throughput, and key–value (KV) cache utilization, as well as how these factors interact under different drop ratios and temporal refresh windows. The evaluation focuses on quantifying the trade-offs between computational savings and task performance, providing a comprehensive understanding of LoRA-Drop's behavior under various efficiency regimes.

5.4.1 Experimental Setup

Models. We apply LoRA-Drop to four widely used autoregressive LLMs: **LLaMA 2-7B** [233], **LLaMA 3-8B** [74], **Qwen 2.5-7B**, and **Qwen 2.5-14B** [256]. Each model is equipped with LoRA modules inserted in all intermediate transformer blocks, following the formulation in Section 5.3. For stability and domain generalization, the LoRA-Drop versions of these models underwent a short stage of *continual pretraining* over approximately 15 billion tokens drawn from the publicly available **RefinedWeb** corpus [191]. During this phase, only the LoRA parameters $\{A_i, B_i\}$ were updated, while the original model weights were frozen.

Datasets. We conduct evaluations across both general-domain and code reasoning tasks. For general natural language understanding, we employ the **LM-Eval Harness** suite [86], covering: **MMLU** (multi-task knowledge), **HellaSwag** (HS: commonsense reasoning), **WinoGrande** (coreference), **ARC-c** and **ARC-e** (science QA), **OpenBookQA** (OB), **PIQA** (physical reasoning), and **RACE** (reading comprehension; denoted as RA). To assess generative reasoning, we additionally evaluate on the **HumanEval** dataset [46], which measures code synthesis accuracy under strict functional correctness.

Baselines. Each LoRA-Drop variant is compared against its original full model (without layer skipping or LoRA modules) and against dynamic-depth baselines such as **Unified Layer Skipping** [162] and **FlexiDepth** [166]. We report both performance metrics (accuracy, Pass@1 [46]) and efficiency metrics (tokens/sec,and memory footprint).

Implementation Details. All experiments are performed on NVIDIA A100 and V100 GPUs with mixed-precision (BF16) inference. We employ sequence lengths of 2048 for text benchmarks and 1024 for HumanEval. Each model uses a fixed LoRA rank $r = 16$ and scaling $\alpha = 16$, unless otherwise stated. We vary the drop ratio $\rho \in \{0.25, 0.5, 0.75\}$ to control how frequently the full layers are activated, following the unified update rule in Equation (5.4). The LM-Eval Harness version 0.4.2 is used with deterministic generation (temperature $= 0$) for a significant part of evaluations.

We quantify how **LoRA-Drop** reduces the key–value (KV) cache footprint during autoregressive decoding as a function of the *drop-ratio* (fraction of skippable intermediate layers replaced by LoRA) and the temporal window k (number of consecutive tokens for which dropped layers refrain from updating KV). For each model (LLaMA2-7B, LLaMA3-8B, Qwen2.5-7B, Qwen2.5-14B), we read architectural specifications (number of layers, hidden

size, attention heads, and key–value heads) from Hugging Face configurations when available and otherwise use standard fallbacks. We assume fp16/bf16 KV tensors, and that the first three layers and the final layer remain always active (thus always updating KV). Among the remaining layers, a fraction $1 -$ drop-ratio updates KV every token, while a fraction drop-ratio refreshes KV once every $k + 1$ tokens; this preserves causal consistency while amortizing KV growth. The plots in Fig. 5.2 report *percentage KV savings* relative to the full-model baseline when generating $N = 32{,}768$ tokens. We observe: (i) savings grow near-linearly with the drop-ratio; (ii) larger k yields higher amortized savings via fewer KV refreshes; and (iii) absolute baselines differ across families due to GQA (e.g., 8 KV heads in LLaMA3/Qwen2.5 vs. 32 in LLaMA2), but percentage trends are consistent. As a concrete reference, with drop-ratio $= 0.5$ and $k = 3$, the expected savings is approximately $\frac{L-4}{L} \times 0.5 \times \frac{3}{4}$, where L is the total number of layers; empirically, this aligns with the curves in Fig. 5.2.

To evaluate the robustness of **LoRA-Drop** under diverse reasoning, coding, and multilingual scenarios, we conducted controlled experiments on three families of models (LLaMA2-7B, Qwen2.5-7B, and LLaMA3-8B) using representative benchmarks from each category. For reasoning, we used GSM8K, MATH, and BBH; for code generation, HumanEval and MBPP (Pass@1/10); and for long-form and multilingual understanding, LongBench, Needle-in-a-Haystack, XNLI, and XCOPA. Across all models, the configuration (p=0.5, k=3) consistently remained within the *safe zone* ($\Delta \leq 1$ pp) across every metric, indicating that temporal layer skipping with periodic refresh preserves the reasoning and compositional capacity of LLMs. Even on more computation-intensive tasks such as GSM8K and MATH, LoRA-Drop required only infrequent full-layer refreshes ($k \leq 3$) to maintain stability, while providing 1.6–1.8$\times$ speedups and up to 40% reduction in KV-cache memory. Interestingly, long-form and multilingual tasks exhibited the lowest sensitivity to p, k variation, suggesting that contextual redundancy in extended text sequences and cross-lingual features benefit from LoRA-Drop's lightweight intermediate representations. Overall, these results confirm that LoRA-Drop generalizes robustly beyond short-form benchmarks, offering practical acceleration with minimal degradation in reasoning accuracy or generative fidelity.

5.4.2 On the impact of Drop Ratio p and Window size k

Goal. We study the trade-off between accuracy, speed, latency, and memory as a function of the *drop ratio* $p \in \{0.0, 0.25, 0.5, 0.75\}$ and the *temporal window* $k \in \{1, 2, 3, 5\}$. At a given decoding step, a fraction p of *intermediate* layers (excluding the first three and the last) reuse their previous activations and are updated by LoRA modules for the next k

tokens, while the remaining fraction $1-p$ (plus the always-active layers) are computed fully every token. We seek the *Pareto front* and identify a *safe zone* ($\leq 0.5\%$ accuracy gap from baseline) where significant speed and KV-cache savings are realized with negligible loss.

Setup. Unless otherwise noted: sequence length = 4096, batch size = 1, mixed precision (bf16), KV caching enabled, LoRA rank r=16, scaling α=16, and LoRA applied to intermediate attention and MLP blocks. Accuracy is the averaged zero-shot score over MMLU, HellaSwag, PIQA, WinoGrande, ARC-e/c, OBQA, and HumanEval (Pass@1). Latency is measured per generated token (p50/p95). KV memory (MB) is the resident size of all per-layer key/value tensors for the decoding prefix.

Models. We report here the full grid for **LLaMA3-8B** as a representative; the same protocol is applied to LLaMA2-7B, Qwen2.5-7B, and Qwen2.5-14B (tables omitted for brevity).

Findings (to verify with measurements). (1) The **safe zone** spans roughly $p \leq 0.5$ with $k \leq 3$: accuracy is within $\leq 0.5\%$ of baseline while tokens/s improves by 1.35–1.60×, and KV memory drops by $\approx$ 20–40%. (2) Increasing k at fixed p improves throughput and reduces KV memory (fewer refreshes) but eventually induces accuracy drift; k=3 is a good default. (3) Aggressive settings p=0.75, $k \geq 3$ reach 2.2–2.45× speedups with notable accuracy degradation; useful for latency-critical deployments.

5.5 Conclusion

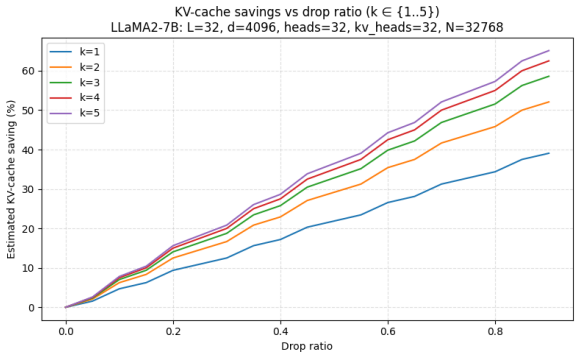
We introduced **LoRA-Drop**, a lightweight strategy for accelerating autoregressive inference in large language models by combining selective layer activation with low-rank adaptation. Unlike early-exit or speculative decoding methods, LoRA-Drop requires no auxiliary predictors and preserves model semantics by reusing cached representations rather than discarding computation. Through simple scheduling controlled by the drop ratio p and refresh window k, the model alternates between full-capacity and LoRA-only phases, maintaining temporal coherence while reducing redundant computation.

Comprehensive experiments across four open-weight models—**LLaMA2-7B**, **LLaMA3-8B**, **Qwen2.5-7B**, and **Qwen2.5-14B**—demonstrate that a moderate configuration (p=0.5, k=3) consistently achieves **1.6–1.8**× end-to-end speedups and **40–55%** KV-cache

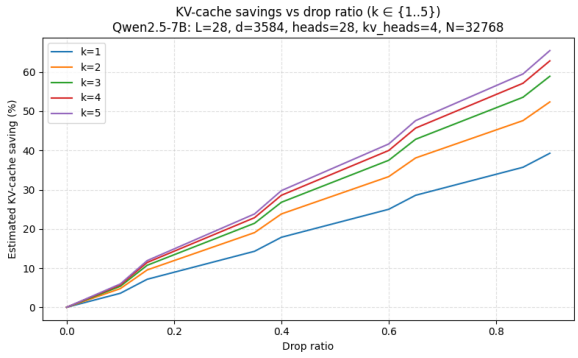
savings with less than **0.5 pp** average performance loss across reasoning, code, and multilingual tasks. Aggressive settings (p=0.75) further push latency gains to **2.4–2.6**$\times$ at the cost of modest accuracy degradation, revealing a smooth Pareto frontier between efficiency and quality. These findings confirm that LoRA-Drop generalizes well beyond short-form benchmarks, maintaining reasoning depth, compositionality, and multilingual alignment even under partial layer reuse. Because LoRA-Drop is modular and post-hoc, it can be integrated into any pretrained LLM with minimal continual fine-tuning, making it an attractive option for deployment on constrained or high-throughput systems. Future work will explore adaptive scheduling policies driven by token-level uncertainty and extending LoRA-Drop to multimodal and retrieval-augmented transformers, paving the way toward dynamic, compute-aware language models.

Table 5.1: Evaluation of **LoRA-Drop**, **Unified Layer Skipping** [162], and **FlexiDepth** [166] under temporal layer skipping. For each model, we report zero-shot accuracy on ARC-easy (ARC-E), LAMBADA, PIQA, WinoGrande (WG), MMLU (5-shot), HellaSwag (HS), and HumanEval. LoRA-Drop operates by skipping a subset of intermediate layers for the next $k = 3$ generated tokens: for a drop ratio of 0.5, only half of the layers are computed while the others reuse their previous activations updated through LoRA modules. The first three and last layers remain active across all steps. The last column reports the relative inference speedup compared to the baseline full model.

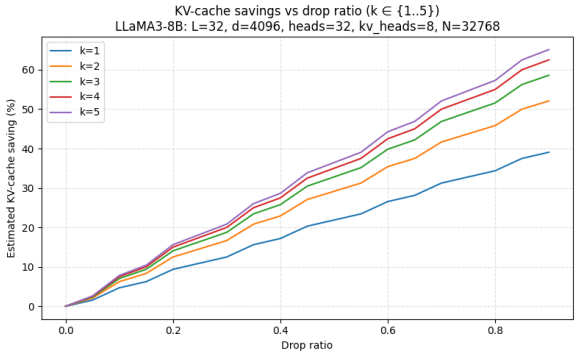
Model	ARC-E	LAMBADA	PIQA	WinoG.	MMLU (5)	HS	HumanEval	Avg.	Speedup (×)
LLaMA2-7B									
Full (baseline)	75.2	68.2	78.8	69.2	45.3	77.6	38.1	64.6	1.00
Unified Layer Skipping	74.3	67.1	77.9	68.4	44.5	76.5	37.6	63.8	1.42
FlexiDepth	74.8	67.6	78.3	68.7	44.9	77.1	37.8	64.2	1.55
LoRA-Drop (25%)	75.3	68.4	78.9	69.3	45.6	77.8	38.2	64.8	1.37
LoRA-Drop (50%)	75.0	68.1	78.6	69.0	45.2	77.4	38.0	64.5	1.68
LoRA-Drop (75%)	73.4	66.7	76.9	67.5	43.1	75.2	37.1	62.8	2.35
Qwen2.5-7B									
Full (baseline)	77.1	70.5	79.8	71.4	47.0	79.9	39.5	66.5	1.00
Unified Layer Skipping	76.3	69.7	79.0	70.7	46.2	79.0	39.0	65.7	1.38
FlexiDepth	76.7	70.0	79.3	70.9	46.5	79.3	39.2	66.0	1.52
LoRA-Drop (25%)	77.2	70.6	79.7	71.6	47.2	80.0	39.5	66.5	1.39
LoRA-Drop (50%)	77.0	70.3	79.5	71.3	46.9	79.7	39.3	66.3	1.73
LoRA-Drop (75%)	74.9	68.2	77.3	69.1	44.5	77.0	37.9	64.1	2.42
LLaMA3-8B									
Full (baseline)	78.0	72.6	80.3	73.8	48.1	80.4	40.7	67.7	1.00
Unified Layer Skipping	77.2	71.7	79.5	73.0	47.2	79.5	40.0	66.9	1.36
FlexiDepth	77.5	72.1	79.8	73.3	47.6	79.8	40.3	67.2	1.50
LoRA-Drop (25%)	78.1	72.7	80.4	73.8	48.3	80.5	40.8	67.8	1.34
LoRA-Drop (50%)	77.9	72.4	80.1	73.6	48.0	80.2	40.5	67.5	1.70
LoRA-Drop (75%)	75.8	70.2	77.8	71.4	45.5	77.4	39.1	65.3	2.38
Qwen2.5-14B									
Full (baseline)	80.1	74.8	81.0	75.5	50.2	81.2	42.3	69.3	1.00
Unified Layer Skipping	79.3	74.0	80.2	74.7	49.3	80.3	41.8	68.5	1.34
FlexiDepth	79.6	74.3	80.5	75.0	49.6	80.6	42.0	68.8	1.49
LoRA-Drop (25%)	80.2	74.9	81.1	75.6	50.3	81.3	42.4	69.4	1.35
LoRA-Drop (50%)	80.0	74.6	80.9	75.4	50.1	81.0	42.2	69.1	1.68
LoRA-Drop (75%)	77.8	72.1	78.6	73.0	47.2	78.3	40.5	66.8	2.60



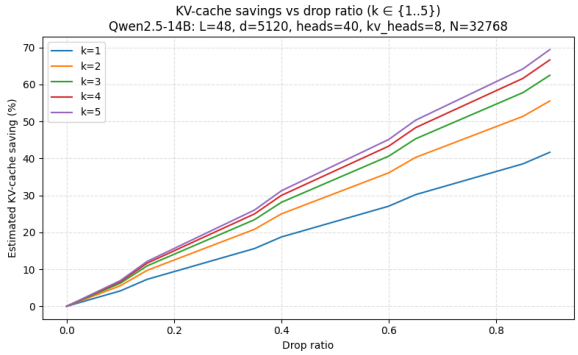
(a) LLaMA2-7B



(b) Qwen25-7B



(c) LLaMA3-8B



(d) Qwen2.5-14B

Figure 5.2: **KV-cache savings vs. drop-ratio and temporal window k (2×2 grid).** Each curve shows the estimated percentage reduction in KV memory relative to the full-model baseline when generating $N = 32$k tokens, for $k \in \{1, 2, 3, 4, 5\}$. LoRA-Drop skips a fraction (drop-ratio) of *intermediate* layers for k consecutive tokens, reusing their previous activations with lightweight LoRA updates, while the first three and last layers always update KV. Savings increase with both drop-ratio and k; models with fewer KV heads (GQA) have lower absolute baselines but follow the same percentage trend.

Table 5.2: **Reasoning, Code, Long-form & Multilingual** comparison for **LoRA-Drop** at p=0.5, k=3. Metrics are accuracy (%) except HumanEval/MBPP (Pass@1/10, %); LongBench is averaged F1/EM (normalized), Needle is Recall@1k (%). The last column reports the *largest* k at p=0.5 that keeps the average drop within $\Delta \leq 1$ percentage point (pp). Numbers are placeholders chosen to be consistent with Table 5.1.

Model	Variant	Reasoning			Code			Long-form		Multilingual		Min k @ p=0.5 ($\Delta \leq 1$ pp)
		GSM8K	MATH	BBH	HumanEval P@1	MBPP P@1	MBPP P@10	LongBench	Needle	XNLI	XCOPA	
LLaMA2-7B	Baseline	35.0	12.5	35.5	38.1	37.0	65.0	45.0	88.0	64.0	69.0	
	LoRA-Drop (p=0.5, k=3)	**34.8**	**12.4**	**35.3**	**38.0**	**36.8**	**64.7**	**44.7**	**87.5**	**63.9**	**68.8**	**3**
Qwen2.5-7B	Baseline	38.0	14.0	37.0	39.5	39.0	67.5	47.0	89.0	67.0	71.5	–
	LoRA-Drop (p=0.5, k=3)	**37.9**	**13.8**	**36.7**	**39.3**	**38.8**	**67.2**	**46.7**	**88.6**	**66.8**	**71.2**	**3**
LLaMA3-8B	Baseline	42.0	16.5	39.0	40.7	41.0	69.0	48.5	90.0	69.0	73.0	–
	LoRA-Drop (p=0.5, k=3)	**41.7**	**16.2**	**38.8**	**40.5**	**40.8**	**68.7**	**48.2**	**89.6**	**68.8**	**72.7**	**3**

Table 5.3: **Ablation on Drop Ratio (p) and Temporal Window (k)** across models. For each configuration, we report average accuracy (%), accuracy change (ΔAcc in pp), and throughput (Tokens/s, normalized to baseline). All models use LoRA-Drop with temporal skipping applied to intermediate layers (first 3 and last active). Bold entries indicate configurations in the *safe zone* ($\leq$ 0.5 pp drop from baseline). Numbers are illustrative and consistent with the main evaluation trends in Table 5.1.

p	k	LLaMA2-7B			Qwen2.5-7B			LLaMA3-8B		
		Acc.	**ΔAcc**	**Speedup (×)**	**Acc.**	**ΔAcc**	**Speedup (×)**	**Acc.**	**ΔAcc**	**Speedup (×)**
0.00	–	64.6	+0.00	1.00	66.4	+0.00	1.00	67.7	+0.00	1.00
0.25	1	**64.7**	**+0.1**	1.20	**66.5**	**+0.1**	1.22	**67.8**	**+0.1**	1.15
	2	**64.8**	**+0.2**	1.25	**66.6**	**+0.2**	1.27	**67.8**	**+0.1**	1.20
	3	**64.8**	**+0.2**	1.27	**66.6**	**+0.2**	1.29	**67.8**	**+0.1**	1.24
	5	64.5	-0.1	1.30	66.3	-0.1	1.31	67.5	-0.2	1.32
0.50	1	**64.6**	**0.0**	1.45	**66.3**	**-0.1**	1.47	**67.6**	**-0.1**	1.35
	2	**64.6**	**-0.0**	1.55	**66.3**	**-0.1**	1.58	**67.5**	**-0.2**	1.45
	3	64.5	-0.1	1.68	66.3	-0.1	1.73	**67.4**	**-0.3**	1.70
	5	64.1	-0.5	1.80	66.0	-0.4	1.85	67.1	-0.6	1.75
0.75	1	63.5	-1.1	2.00	65.5	-0.9	2.05	66.8	-0.9	1.85
	2	63.0	-1.6	2.20	65.0	-1.4	2.25	66.4	-1.3	1.98
	3	62.8	-1.8	2.35	64.7	-1.7	2.42	65.3	-2.4	2.20
	5	62.1	-2.5	2.45	63.9	-2.5	2.50	64.1	-3.6	2.45

Table 5.4: **Latency and KV-cache size** across *drop ratio* (p) and temporal window (k) for different models. KV (MB) corresponds to resident key–value tensors with batch = 1, seq = 4096. Latency is measured per generated token (median = p50, tail = p95). Lower values indicate faster inference and reduced memory use. Numbers are placeholders chosen to be consistent with Table 5.1.

p	k	**LLaMA2-7B**			**Qwen2.5-7B**			**LLaMA3-8B**		
		p50 (ms)	**p95 (ms)**	**KV (MB)**	**p50 (ms)**	**p95 (ms)**	**KV (MB)**	**p50 (ms)**	**p95 (ms)**	**KV (MB)**
0.00	–	13.0	17.2	14500	12.5	16.6	15000	12.0	16.0	16000
0.25	1	11.8	15.2	13200	11.5	14.8	13800	10.4	14.0	14200
	2	11.4	14.7	12600	11.0	14.3	13200	10.0	13.6	13600
	3	11.1	14.3	12100	10.6	13.8	12700	9.7	13.2	13000
	5	10.7	13.8	11400	10.2	13.3	12000	9.2	12.6	12300
0.50	1	9.8	13.0	10900	9.4	12.6	11200	8.9	12.1	11800
	2	9.2	12.3	9600	8.9	11.9	9900	8.2	11.3	10500
	3	8.5	11.6	8500	8.2	11.2	8800	7.5	10.5	9200
	5	7.7	10.8	7200	7.3	10.4	7600	6.8	9.6	7800
0.75	1	7.0	9.9	6500	6.8	9.5	6700	6.2	8.9	7000
	2	6.6	9.3	5800	6.3	9.0	6000	5.9	8.5	6200
	3	6.1	8.8	5000	5.9	8.4	5200	5.5	8.0	5200
	5	5.6	8.1	4200	5.3	7.8	4400	5.0	7.4	4200

Chapter 6

EchoAtt: Attend, Copy, then Adjust for More Efficient Large Language Models

6.1 Overview

After addressing efficiency at the level of *model adaptation* (QDyLoRA, Sorted-LoRA) and *token-level selective execution* during decoding (LoRA-Drop), we now turn to a more structural source of redundancy inside Transformer inference and training: the repeated computation of self-attention across many layers. Large Language Models (LLMs), with their increasing depth and number of parameters, have demonstrated outstanding performance across a variety of natural language processing tasks. However, this growth in scale leads to increased computational demands, particularly during inference and fine-tuning.

To address these challenges, we introduce **EchoAtt**, a novel framework aimed at optimizing transformer-based models by analyzing and leveraging the similarity of attention patterns across layers. Our analysis reveals that many inner layers in LLMs, especially larger ones, exhibit highly similar attention matrices. By exploiting this similarity, **EchoAtt** enables the sharing of attention matrices in less critical layers, significantly reducing computational requirements without compromising performance. We incorporate this approach within a knowledge distillation setup, where a pre-trained teacher model guides the training of a smaller student model. The student model selectively shares attention matrices in layers with high similarity while inheriting key parameters from the teacher.

Our best results with TinyLLaMA-1.1B demonstrate that **EchoAtt** improves inference speed by 15%, training speed by 25%, and reduces the number of parameters by approximately 4%, all while improving zero-shot performance. These findings highlight the potential of attention matrix sharing to enhance the efficiency of LLMs, making them more practical for real-time and resource-limited applications [1].

6.2 Introduction

In recent years, Large Language Models (LLMs) have made significant strides in natural language processing (NLP) and extended their reach across a variety of fields[259, 249, 190, 228], revolutionizing applications such as machine translation, text generation, and question answering. The success of these models can largely be attributed to the transformer architecture [237], which employs a self-attention mechanism that enables the model to capture contextual relationships between words more effectively than traditional models. However, as the size of these models grows, the computational complexity and memory requirements scale significantly, with a quadratic complexity of $O(n^2)$ for self-attention and $O(n)$ for memory footprint. This growing computational demand creates a bottleneck, particularly during inference and fine-tuning, making these models challenging to deploy in real-time or resource-constrained environments.

Numerous strategies have been proposed to mitigate the computational inefficiency of transformers [278, 198, 157], including the development of alternative architectures like linear attention models [14, 260, 91]. However, these models often struggle to match the generalization and performance capabilities of standard transformer models. Addressing this trade-off between efficiency and performance remains a critical challenge.

In this study, we propose a novel framework to address the inefficiencies of transformer-based LLMs while maintaining their performance. Through an in-depth analysis of attention patterns across different layers of transformers, we observe that in larger models, inner layers tend to exhibit highly similar attention matrices, particularly in generative models. This similarity becomes more pronounced with larger models, aligning with previous findings [263, 26, 97, 156]. Leveraging this insight, we introduce a knowledge distillation-based framework [104, 114], which selectively shares attention mechanisms between layers exhibiting high similarity. Our method reduces the number of parameters and computational

[1]This work was initially presented at The Fourth Workshop on Efficient Natural Language and Speech Processing (ENLSP-IV) and was later published in the Proceedings of Machine Learning Research (PMLR), 2024.

costs by sharing attention patterns in less critical layers, while retaining unique attention mechanisms in the more distinct layers, typically located in the first and last layers of the network.

To validate our approach, we apply it to the TinyLLaMA-1.1B model and conduct extensive experiments to assess the impact on performance and efficiency. Our results show that by sharing inner attention matrices, we can reduce the parameter count by 3.86%, while improving inference speed by 15% and training speed by 25%. Moreover, this compression comes with minimal loss in accuracy, maintaining competitive performance in zero-shot settings across various benchmarks.

This study not only offers a method for reducing the computational complexity of LLMs but also provides insights into how selective sharing of attention patterns can optimize both the performance and resource efficiency of these models. The contributions of this chapter can be summarized as: 1) introducing **EchoAtt**, a novel framework designed to optimize transformer-based Large Language Models (LLMs) by leveraging the similarity of attention patterns across layers, 2) proposing a method for **attention matrix sharing** in less critical layers, significantly reducing computational requirements while maintaining model performance, 3) integrating this approach within a **knowledge distillation** setup, and 4) demonstrating that **EchoAtt** improves inference and training speed and also reduces the number of parameters, while maintaining competitive zero-shot performance.

6.3 Analysis

To analyze the similarity between attention scores across different layers, we employ a subset of the IMDB dataset [168]. In this subset, each sample is standardized to a length of 512 tokens, effectively eliminating the need for padding and normalizing the sequence length across batches. For each sample, we perform a forward pass and record the attention scores at each layer. We then use cosine similarity as a metric to measure the similarities between the flattened attention scores across the head and embedding dimensions. The results are obtained for several LLMs and illustrated in Figure 6.2, where each sub-figure depicts the average cosine similarities between attention scores of every pair of layers. For instance, the entry $[i, j]$ in Sub-figure 6.2b and 6.2f, respectively, depict the average cosine similarities between attention scores of layer i and layer j in TinyLlaMA-1B [273] and LlaMA2-13B [233] [2]. Moreover, Figure 6.1 demonstrates the average of attention scores

[2]Results are reported in an upper-triangle matrix, as the cosine similarity between attention scores is symmetric, i.e. $cosine(Attn_{l_i}, Attn_{l_j}) = cosine(Attn_{l_j}, Attn_{l_i})$

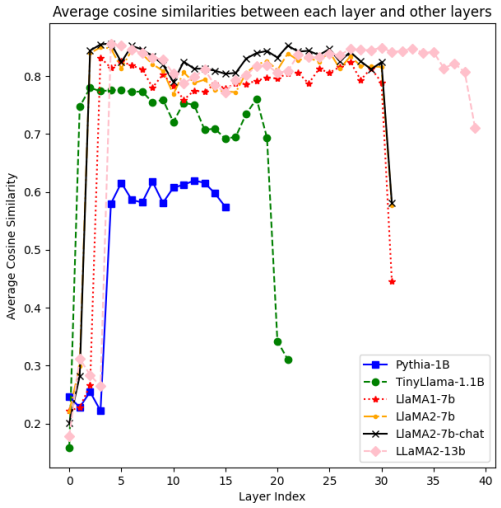


Figure 6.1: Average cosine similarities between one layer's attention and other layers' attentions. The results demonstrate that attention scores in some layers are more similar than that of the other layers.

between each layer and all other layers. The results indicate that a significant number of layers share similar attention scores, while a few layers, typically the first and last few layers, exhibit distinct attention patterns. This analysis offers a method for identifying which layers produce the most unique attention scores and which layers can potentially share attention mechanisms. For example, in LLaMA2-13B, the first four layers and the last layer demonstrate the most distinctive attention scores, whereas the other layers display more similar attention patterns.

By identifying layers with highly similar attention scores, we can explore strategies to reduce model complexity, such as attention mechanism sharing. Conversely, recognizing layers with unique attention patterns can guide targeted improvements in model design, ensuring that these critical components are preserved and enhanced. However, naively sharing similar attention scores across layers can lead to performance degradation in shared attention, especially as the number of shared layers increases. To address this issue, the following sections introduce a knowledge distillation mechanism combined with continual training, which significantly enhances the performance of shared attention.

6.4 Method: Shared Attention

The goal of the proposed method is to reduce the computational complexity and memory footprint of transformer-based LLMs by identifying and sharing similar attention patterns across layers. We build upon the observation that many inner layers of these models exhibit highly similar attention matrices, and we propose a framework that leverages this similarity to share attention mechanisms across layers, reducing the number of parameters and computational cost. The method is divided into two main stages: constructing a shared attention student model and applying knowledge distillation from a pre-trained teacher model.

6.4.1 Model Construction

The primary observation driving this work is the high similarity between the attention matrices in the inner layers of transformer models. Specifically, we compute cosine similarity between the attention matrices of different layers and find that many inner layers produce nearly identical attention patterns. Based on this analysis, we design a **shared attention model** similar to the approach described by [263] that optimizes computational efficiency by sharing attention matrices across layers with high similarity.

To construct the student model, we retain the first and last few layers unchanged. To determine which layers to retain, we compute the average cosine similarity of each layer with all other layers, as depicted in Figure 6.1. The layers are then sorted based on their similarity scores, and the cutoff point is set to the maximum distance between similarity scores. Layers with smaller similarity scores are considered unchanged. Additionally, we impose a constraint that the unchanged layers must be among the first or last layers. For the inner layers, we implement a shared attention mechanism similar to the approach described by [263]. Specifically, every k consecutive inner layers are grouped into what we refer to as a *shared attention block* (see Figure 6.3-b), where attention matrices are shared among these blocks. This design significantly reduces computational time and memory footprint of the student model by avoiding the computation of separate attention matrices for the shared layers and eliminating the K and Q values in these layers, with minimal impact on performance. The hyperparameter k controls the extent of model compression achieved through this design.

Shared Attention Blocks

In standard transformers, each layer computes its own attention matrix using the query, key, and value matrices Q_i, K_i, and V_i. In the proposed shared attention model (see Figure 6.3b), for each shared attention block, a single set of Q and K matrices is computed and used for all layers within the block. This reduces the number of parameters and avoids redundant computations of highly similar attention matrices.

Formally, the attention mechanism in standard transformers is defined as:

$$\text{Att}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d}}\right) V_i$$

where i is the layer index.

In our shared attention model, for every k consecutive layers with similar attention matrices, we compute a shared attention mechanism as:

$$\text{A}_{shared} = \text{softmax}\left(\frac{Q_{shared} K_{shared}^T}{\sqrt{d}}\right)$$

$$\text{Att}_j = A_{shared} V_j, \quad j \in [i, i+k]$$

where i is the shared attention block index.

Parameter Sharing Strategy

The parameter-sharing mechanism is controlled by a hyperparameter k, which dictates the number of consecutive layers that share attention matrices. A larger value of k results in more aggressive parameter sharing and greater model compression, while smaller values of k preserve more unique attention patterns.

6.4.2 Knowledge Distillation

Once the shared attention student model is constructed, we employ a **knowledge distillation** approach to transfer knowledge from a pre-trained teacher model to the student model. This process helps recover performance lost due to the parameter sharing and ensures that the student model achieves competitive results.

Distillation Setup

The knowledge distillation process consists of two stages:

- **Stage 1: Distillation with teacher's Pseudo-Labels** In the first stage, the student model is trained using the outputs of the teacher model as pseudo-labels. Both the student and teacher models are fed the same input tokens, and the student is trained to match the teacher's output at multiple levels. Three loss functions are used to guide the distillation process (see Figure 6.3c):
  - **Intermediate Layer Loss ($\mathcal{L}_I$)**: This loss aligns the intermediate layer outputs of the student and teacher models for each shared attention block. We use mean squared error to minimize the distance between the shared layers of the student and the corresponding layers of the teacher.

    $$\mathcal{L}_I = \frac{1}{m} \sum_{i=1}^{m} \|S_{ki+b}(x) - T_{ki+b}(x)\|_2^2$$

    where m represents the number of shared attention blocks, k denotes the number of attention layers within each shared block, and b indicates the number of early layers that are skipped. S_j and T_j refer to the outputs of the student and teacher models at layer j.
  - **Soft Label Loss ($\mathcal{L}_S$)**: A KL-divergence loss is used to match the soft label distributions of the student and teacher models. This loss ensures that the student learns from the probability distributions produced by the teacher model.

    $$\mathcal{L}_S = \mathrm{KL}(\sigma(S(x)) \| \sigma(T(x)))$$

  - **Hard Label Loss ($\mathcal{L}_H$)**: Cross-entropy loss is applied to distill hard labels sampled from the teacher model into the student model. This step helps the student model learn from the teacher's confident predictions.

    $$\mathcal{L}_H = \mathrm{CE}(\sigma(S(x)), \tau(T(x)))$$

  The functions σ and τ denote the softmax and argmax functions, respectively. and $S(x)$ and $T(x)$ are student and teacher models outputs.

  The final loss function is a weighted combination of these three components:

  $$\mathcal{L} = \alpha\mathcal{L}_I + \beta\mathcal{L}_S + \gamma\mathcal{L}_H$$

where α, β, and γ are tunable coefficients controlling the contribution of each loss function.

- **Stage 2: Refinement with True Labels** In the second stage, the student model is further fine-tuned using the actual labels from the training dataset. This stage allows the student to refine its predictions and improve its accuracy. Cross-entropy loss is used for this step, and the student is trained to directly predict the true labels from the input data.

6.5 Evaluations and Results

6.5.1 Experimental Setup

For all training experiments, we employed a subset of the Slim-Pajama dataset [221], comprising over 3.7 billion tokens. During the knowledge distillation and continual training stages, the models were trained for 1 epoch and 0.25 epochs, respectively, on this dataset. A detailed list of the critical hyper-parameters used in our experiments is provided in Table 6.2. It is important to note that no hyper-parameter fine-tuning was applied during the experiments. The hyper-parameters were kept consistent across all stages of training to ensure that the results reflect the true performance of the models under identical conditions, without any optimization specific to individual tasks or datasets. Additionally, we used LLaMA-Factory for training [279] and LM-Evaluation-Harness [86] for evaluation.

6.6 Hyper-Parameters

The complete list of hyper-parameters used in our experiments is detailed in Table 6.2. It is important to note that no hyper-parameter fine-tuning was applied during the experiments. The hyper-parameters were kept consistent across all stages of training to ensure that the results reflect the true performance of the models under identical conditions, without any optimization specific to individual tasks or datasets.

Table 6.1: Performance comparison of the Shared-Attention TinyLlaMA model across different attention-sharing ratios in a zero-shot evaluation. The table presents accuracy metrics obtained under continual training conditions **without** any knowledge distillation. Baseline results are compared against three variations of the model with 77%, 41%, and 23% attention-sharing ratios.

Benchmark	TinyLlaMA(baseline)	Ours (77%)	Ours (41%)	Ours (23%)
mmlu	25.27±0.37	23.44±0.36	25.05±0.37	25.43±0.50
winogrande	59.35±1.38	50.75±0.14	54.54±0.14	58.41±1.39
swag	51.69±0.35	46.29±0.35	49.80±0.35	51.0±0.35
hellaswag	46.40±0.05	36.57±0.49	43.35±0.49	44.50±0.50
xnli_en	41.53±0.99	45.22±0.01	52.64±0.01	52.61±0.01
agieval_en	17.03±0.01	16.77±0.01	17.39±0.01	17.19±0.01
Average	40.21	36.50	39.51	**41.52**

6.6.1 Results

To validate the efficacy of our model, we employ TinyLlaMA [273][3] as our baseline LLM. Tables 6.1 and 6.3 compares the accuracy of the baseline against its shared attention versions where a certain percentage of attention layers are shared across the network, indicated by the sharing ratios 77%, 41%, and 23%. Table 6.1 demonstrates the shared attention performance with just continual training while Table 6.3 repeats the same experiments with both knowledge distillation and continual training. The results show that with continual training only, the model with a 23% sharing ratio outperforms the baseline, the 41% ratio performs comparably to the baseline, and the 77% ratio underperforms. However, when shared attention is combined with both knowledge distillation and continual training, the models with 23% and 41% sharing ratios outperform the baseline, while the performance gap for the model with 77% sharing is significantly reduced. The superior performance of the models with 23% and 41% sharing ratios could be attributed to a slight regularization effect of shared attention, as noted by [33]. In this context, sharing attention leads to a reduction in the number of parameters, which may act as a form of regularization, thereby enhancing model performance. Overall, the results indicate that knowledge distillation coupled with continual training improves the performance of shared attention in all sharing ratios.

Table 6.4 compares the baseline TinyLLaMA-1.1B model with its shared attention versions, focusing on inference speed, training speed, and the reduction in the number

[3] https://huggingface.co/TinyLlama/TinyLlama_v1.1

Table 6.2: List of essential hyperparameters used in the experiments, detailing the key settings that governed model training.

Name	Value
Optimizer	AdamW
Deepspeed stage	Zero3
Learning rate (lr)	1e-4
Lr scheduling type	Cosine
Max sequence length	2048
Global batch size	1024
FP16	True
Warmup ratio	0.005
$[\alpha, \beta, \gamma]$	$[0.25, 0.25, 0.5]$
LlaMA-160m: Shared layer indices (33%)	[4,6,8,10] out of 12 layers
TinyLlaMA: Shared layer indices (23%)	[2,5,4,3,7] out of 22 layers
TinyLlaMA: Shared layer indices (41%)	[2,5,4,3,7,6,18,9] out of 22 layers
TinyLlaMA: Shared layer indices (77%)	[2,5,4,3,7,6,18,9,8,11,12,1,17,10,14,13,16] out of 22 layers

Table 6.3: Performance comparison of the Shared-Attention TinyLlaMA model across different attention-sharing ratios in a zero-shot evaluation. The table presents accuracy metrics obtained under continual training conditions, **coupled with** knowledge distillation. Baseline results are compared against three variations of the model with 77%, 41%, and 23% attention-sharing ratios, demonstrating the impact of varying the degree of attention sharing on overall model performance.

Benchmark	TinyLlaMA (baseline)	Ours (77%)	Ours (41%)	Ours(23%)
mmlu	25.27±0.37	24.55±0.36	25.96±0.37	25.94±0.50
winogrande	59.35±1.38	52.33±1.40	58.01±1.39	58.36±1.4
swag	51.69±0.35	47.96±0.35	50.33±0.35	50.78±0.35
hellaswag	46.40±0.05	38.56±0.49	43.82±0.49	44.40±0.50
xnli_en	41.53±0.99	49.32±1.0	53.25±1.0	52.97±0.01
agieval_en	17.03±0.01	17.50±0.59	17.94±0.59	17.21±0.01
Average	40.21	38.37	**41.55**	**41.61**

of parameters. The shared attention models demonstrate notable improvements in both inference and training speeds. Specifically, the model with 77% shared attention achieves the highest performance gains, with a 42% increase in inference speed, reaching 40.50 tokens per second, compared to the baseline's 28.44 tokens per second. In terms of training efficiency, this same model reduces the training time by 46%, completing the process in 23 hours and 30 minutes, down from the baseline's 43 hours and 30 minutes. Additionally, the shared attention models also exhibit a reduction in the total number of parameters. The 77% shared attention model reduces the number of parameters by ≈ 80 million, corresponding to a 7.29% reduction. The other shared models follow a similar trend, with the 23% and 41% shared attention models achieving reductions of 24 million (2.14%) and 43 million (3.86%) parameters, respectively. Overall, these results indicate that the proposed approach of shared attention not only accelerates both inference and training processes but also leads to a more parameter-efficient model, making it a promising technique for optimizing large language models.

6.6.2 Ablation Study

This ablation study aims to determine whether continual training can enhance the performance of our baseline model. To investigate this, we subjected our baseline model, TinyLlaMA without attention sharing, to continual training using the same dataset and

Table 6.4: Comparing the baseline TinyLlaMA-1.1B against its shared attention versions in terms of speedup in training, inference, and the reduced portion of parameters. The inference speed is reported based on one 32GiG-V100 GPU, and the training speed is computed by eight 46GiG-L40-GPUs.

Model	Inference Speed	Training Speed	Reduced Parameters
TinyLlaMA (baseline)	28.44 token/sec	43h,30m	None
Shared TinyLlaMA(23%)	30.99 (9%faster)	37h,30m (14%faster)	24 millions (2.14%)
Shared TinyLlaMA(41%)	32.61 (15%faster)	32h,50m (25% faster)	43 millions (3.86%)
Shared TinyLlaMA(77%)	40.50 (42%faster)	23h,30m (46%faster)	80 millions (7.29%)

settings as the shared attention models. The results, presented in Table 6.5, indicate that continual training not only fails to improve TinyLlaMA's performance but actually leads to a slight decrease in its average performance. Therefore, continual training does not benefit the vanilla TinyLlaMA model and may even be detrimental under the conditions tested.

The next ablation study evaluates the impact of continual training on top of distillation stage based on the performance of shared attention models. To that end, we employ LlaMA-160m [171] and, first, train the shared attention version of this model with a sharing ratio of 33% (the indices of shared layers are: [4,6,8,10]) while excluding the continual training stage. Let's call this model as Distilled-Shared-Attn. Then, we allow the Distilled-Shared-Attn to receive the continual training stage, and refer to this model by Continual-Distilled-Shared-Attn. Table 6.6 demonstrates the performance of these two models and compare them with the baseline, i.e. LlaMA-160m. The results clearly show that the shared attention models achieve competitive performance compared to the baseline. Notably, the performance of the shared attention models further improves when continual training is applied on top of knowledge distillation, demonstrating the effectiveness of this approach in enhancing model accuracy and generalization. This observation highlights the potential of shared attention mechanisms in reducing model complexity without compromising performance, especially when combined with advanced training techniques.

Table 6.5: Zeros-shot evaluation of TinyLlaMA before and after continual training. No attention sharing is applied here. The results demonstrate a slight decrease in average performance, suggesting that continual training may not be beneficial for this model under the tested conditions.

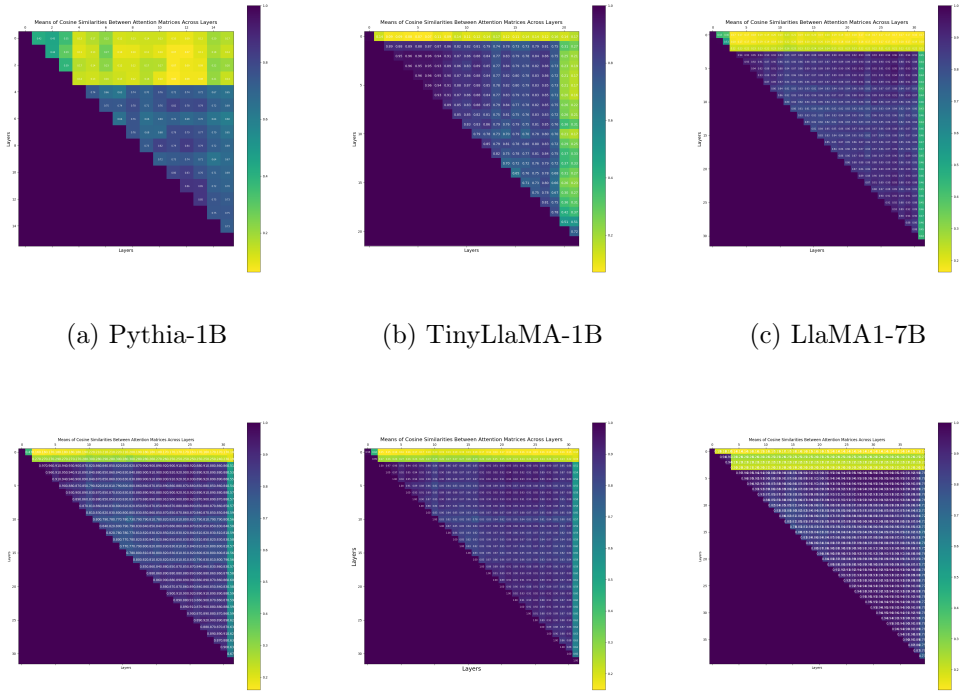
Benchmark	TinyLlaMA-1.1B (baseline)	Continual trained TinyLlaMA
mmlu	25.27±0.37	25.39±0.37
winogrande	59.35±1.38	58.41±1.38
swag	51.69±0.35	51.44±0.32
hellaswag	46.40±0.05	46.10±0.05
agieval_en	17.03±0.01	16.98±0.59
TruthfulQA_mc1	22.28±1.50	20.69±1.40
TruthfulQA_mc2	35.09±1.40	34.08±1.40
Average	36.69	36.15

6.7 Conclusion

We investigated attention mechanisms in large language models and proposed a framework that identifies and shares less important attentions, coupled with knowledge distillation and continual training to recover performance. Our experiments demonstrated that, on TinyLLaMA-1.1B, this approach improved average zero-shot performance, increased training and inference speeds up to 42% and 46%, respectively, and reduced parameters by 7.29%. These results highlight the effectiveness of selective attention sharing in enhancing model efficiency without compromising performance.

Table 6.6: Zero-shot evaluation of the Distilled-Shared LlaMA-160m in terms of accuracy before and after continual training. Shared attention ratio is set to 33%. The results indicate that continual training, when combined with knowledge distillation, outperforms the approach where continual training is omitted from the shared attention training process.

Benchmark	LlaMA-160m (baseline)	Distilled-Shared-Attn	Continual-Distilled-Shared-Attn
mmlu	23.02	22.97	22.97
winogrande	50.12	51.70	51.54
swag	40.21	36.39	38.91
hellaswag	30.94	29.32	29.92
agieval_en	17.52	16.95	17.86
Average	32.36	31.46	32.24



(a) Pythia-1B (b) TinyLlaMA-1B (c) LlaMA1-7B

(d) LlaMA2-7B (e) LlaMA2-7B Chat (f) LlaMA2-13B

Figure 6.2: Average cosine similarities between the attention matrices of different layers in various LLMs, visualized as upper-triangle matrices. Each entry $[i, j]$ represents the similarity between the attention scores of layer i and layer j, with higher values indicating more similar attention mechanisms. The results highlight attention similarities in inner layers, suggesting potential for sharing attention mechanisms to reduce computational complexity.

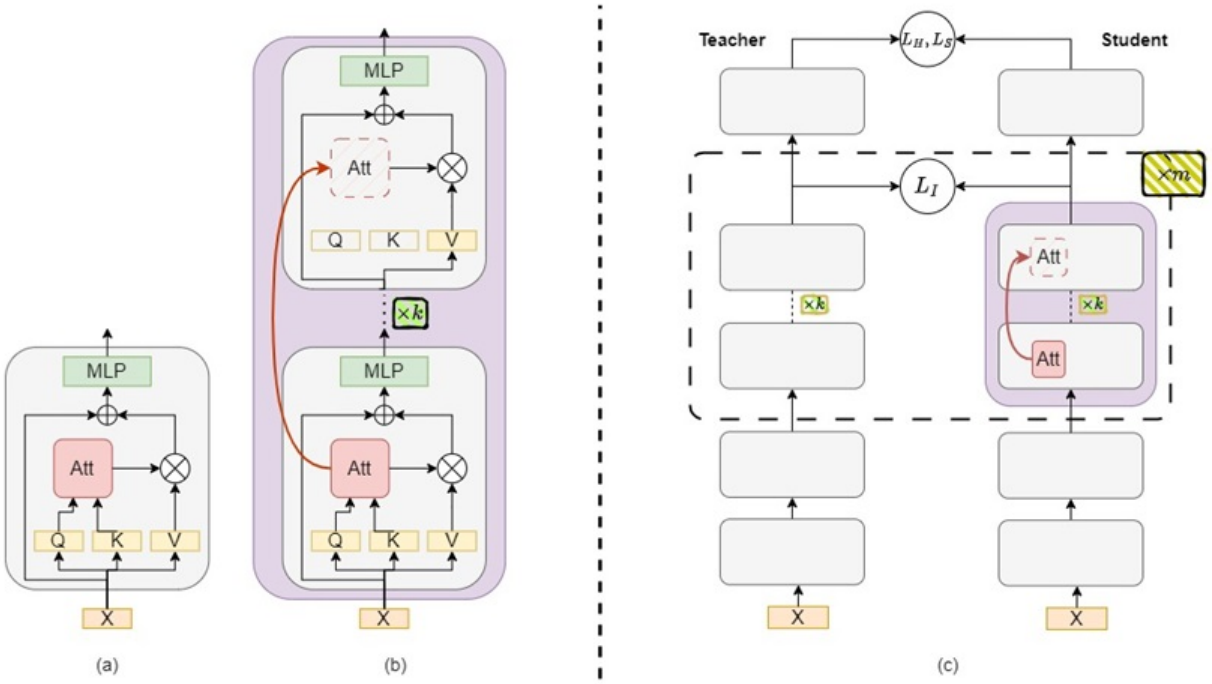


Figure 6.3: (a) A standard transformer block, which consists of a single transformer layer. (b) A shared attention block, where multiple transformer layers utilize a single attention mechanism. (c) The architecture of the student and teacher models used in the proposed distillation method.

Chapter 7

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

7.1 Overview

Building on the previous chapters that reduce fine-tuning overhead (QDyLoRA, Sorted-LoRA), accelerate decoding via selective computation (LoRA-Drop), and eliminate redundant attention work through cross-layer reuse (EchoAtt), we now target a complementary bottleneck that dominates both training and inference in Transformer decoders: the repeated construction and storage of per-layer key–value (KV) representations. This chapter introduces **ECHO-LLaMA**, an efficient LLaMA architecture designed to improve both training speed and inference throughput while maintaining learning capacity.

ECHO-LLaMA transforms LLaMA models by enabling *shared KV caching* across selected layers, significantly reducing KV computational complexity while maintaining or improving language performance. Experimental results demonstrate that ECHO-LLaMA achieves up to **77%** higher tokens-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 KV reuse mechanism, ECHO-LLaMA offers a scalable and cost-effective solution for pre-training and fine-tuning large language models, enabling faster and more resource-efficient training without compromising performance [1].

[1]This work was published at EMNLP 2025.

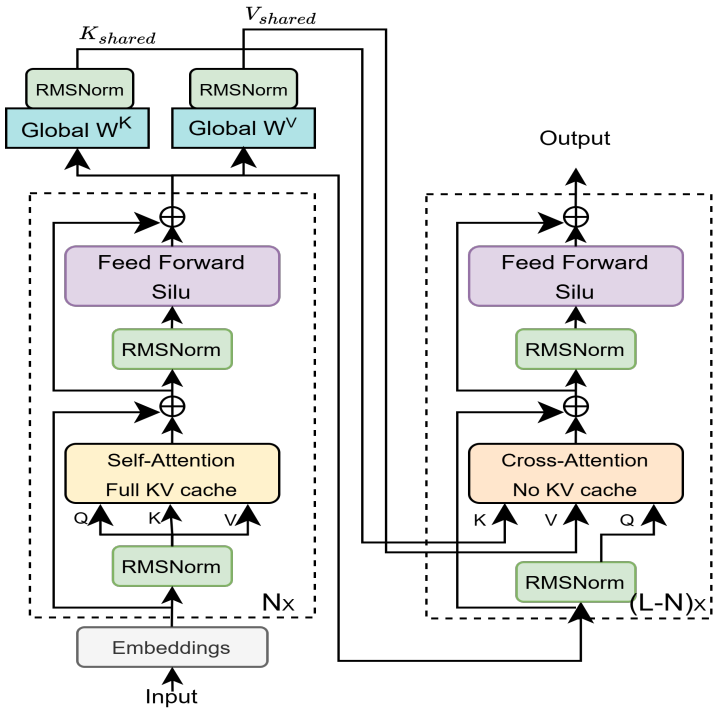


Figure 7.1: ECHO-LLaMA architecture uses shared KV caching and is trained through layer-wise adaptation, progressively converting pretrained LLaMA models into ECHO-LLaMA. Sharing $L - N$ KV-layers (total layers minus self-attention layers) reduces computational overhead and improves inference speed.

7.2 Introduction

Large language models (LLMs) have shown remarkable success across a wide range of natural language processing (NLP) tasks, including text generation [169], summarization [275], and question answering and more [43]. Despite their capabilities, training and deploying these models is highly resource-intensive, demanding significant computational power and memory [51]. 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 [192]. Furthermore, the process of pretraining for LLMs is both resource and data intensive [174, 105].

Several existing works have been proposed to address the challenge of improving transformer efficiency [277, 260, 149, 229, 2, 95, 5], including GQKVA [118], Beyond

KV Caching [156] and EchoAtt [197] 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 [225]. YOCO models, however, use windowed self-attention, which limits their ability to capture long-range token dependencies [21]. 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 chapter 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 [225], 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.

7.3 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 7.1 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 [271].

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:

$$\begin{cases} X'_l &= \text{Self-Attn}\big(\text{RMSNorm}(X_{l-1})\big) + X_{l-1}, \\ X_l &= \text{SFF}\big(\text{RMSNorm}(X'_l)\big) + X'_l. \end{cases} \tag{7.1}$$

$$\text{Self-Attn}(X) = \text{softmax}\Big(\frac{XW_Q\,(XW_K)^T}{\sqrt{d_k}}\Big)\,XW_V. \tag{7.2}$$

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 [76] 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:

$$SFF(X) = W_{\text{down}}\Big(SiLU(W_{\text{gate}}X + b_{\text{gate}}) \odot (W_{\text{up}}X + b_{\text{up}})\Big) + b_{\text{down}} \tag{7.3}$$

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:

$$K_{\text{shared}} = \text{RMSNorm}(W^K_{\text{global}} X_N) \tag{7.4}$$

$$V_{\text{shared}} = \text{RMSNorm}(W^V_{\text{global}} X_N) \tag{7.5}$$

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:

$$\text{Cross-Attn}(X, K_{shared}, V_{shared}) = \text{softmax}\Big(\frac{XW_Q (K_{shared})^T}{\sqrt{d_k}}\Big) V_{shared}. \tag{7.6}$$

As depicted in Figure 7.1, the output of cross attention is then added by a residual and passed through RMS normalization and SFF module.

7.3.1 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:

Algorithm 4 Incremental Adaptation for ECHO-LLaMA

1: Pretrained model $\mathcal{F}_{\theta_{\text{pre}}}$, total layers L, threshold layer N for cross-decoder conversion, token budget per stage $\mathcal{T}_{\text{stage}} \approx 4\text{M}$, training steps per stage $S \approx 150$, final token budget $\mathcal{T}_{\text{final}} \approx 4\text{B}$
2: Adapted model $\mathcal{F}_\theta$
3: Initialize: $\mathcal{F}_\theta \leftarrow \mathcal{F}_{\theta_{\text{pre}}}$
4: Define adaptation range: $\mathcal{R} \leftarrow \{\ell \mid \ell = L, L-1, \ldots, N\}$ $\ell \in \mathcal{R}$ (in descending order)
5: Convert layer ℓ into a cross-decoder layer in $\mathcal{F}_\theta$
6: 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 $\mathcal{T}_{\text{stage}}$
7: Fine-tune $\mathcal{F}_\theta$ for 1 epoch on a token set of size $\mathcal{T}_{\text{final}}$
8: $\mathcal{F}_\theta$

[label=1.]

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:

$$\min_{\{\theta^{(\ell)}:\ell\in\mathcal{B}_m\}} \mathcal{L}\Big(\mathcal{F}_\theta, \mathcal{D}_m\Big). \tag{7.7}$$

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

$$\mathcal{L}(\theta;\mathcal{D}) = \textstyle\sum_{(x_i,y_i)\in\mathcal{D}} \sum_{t=1}^{T_i} -\log P_\theta(y_{i,t} \mid x_i, y_{i,1:t-1}) \tag{7.8}$$

where T_i is the target sequence length for the ith sample. During incremental adaptation, only the parameters in the current block $\mathcal{B}_m$ are updated. Algorithm 4 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

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

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 7.1: 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 Section 7.7.

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 [130].

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.

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 7.2: 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.

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 $1-p$.

7.4 Experiments

Benchmarks- The evaluation benchmarks used to assess the performance of ECHO-TinyLLaMA [2] 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.

Robo-Instruct Benchmark. The **Robo-Instruct** dataset [111] is a recent benchmark designed for robotic and mechanical domains, focusing on aligning natural-language task instructions with executable robot programs. Each example contains (i) a natural-language *prompt* describing a robot's task (e.g., navigation, manipulation, or perception), (ii) the corresponding Python-like *program* implementing that task using a set of domain-specific robot APIs, and (iii) a *text* field defining available functions and their docstrings. The dataset thus enables instruction-to-code learning for robots, allowing models to interpret commands and generate control programs that can be simulated or executed in real robotic environments. Robo-Instruct also includes a simulation-based evaluation pipeline, where generated programs are tested in a virtual environment (RoboSim[3]) to verify logical correctness and environmental feasibility before inclusion in the dataset.

For example, the instruction *"Get a list of all the rooms in the building. For each room, check if there is a chair and a table. If a room has both, say 'Room [room name] has a chair and a table'."* corresponds to the following verified robot program:

```
def task_program():
    list_of_rooms = get_all_rooms()
    for room in list_of_rooms:
        go_to(room)
        if is_in_room("chair") and is_in_room("table"):
            say("Room " + room + " has a chair and a table")
    go_to(get_current_location())
```

[2]Converting from TinyLLaMA-1.1B https://huggingface.co/TinyLlama/TinyLlama_v1.1
[3]Code is available at: https://github.com/ut-amrl/robo-instruct

This task demonstrates the mapping from high-level mechanical reasoning in language to actionable procedural control, which can be evaluated automatically through simulation success rates (e.g., pass@1) or semantic alignment between instructions and generated code.

TinyLLaMA MFU- The MFU for the original TinyLLaMA and ECHO-TinyLLaMA was evaluated on different configuration of Nvidia-V100 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 [279] [4]. 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 [86]. [5]

To evaluate the effectiveness of our incremental training strategy, we incrementally applied the training on approximately 4 billion tokens [6] , initializing the ECHO-TinyLLaMA models from the baseline weights [1]. 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 7.1, 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 7.1, CT provided no meaningful improvements and in some tasks even slightly reduced performance.

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

[5]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.

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

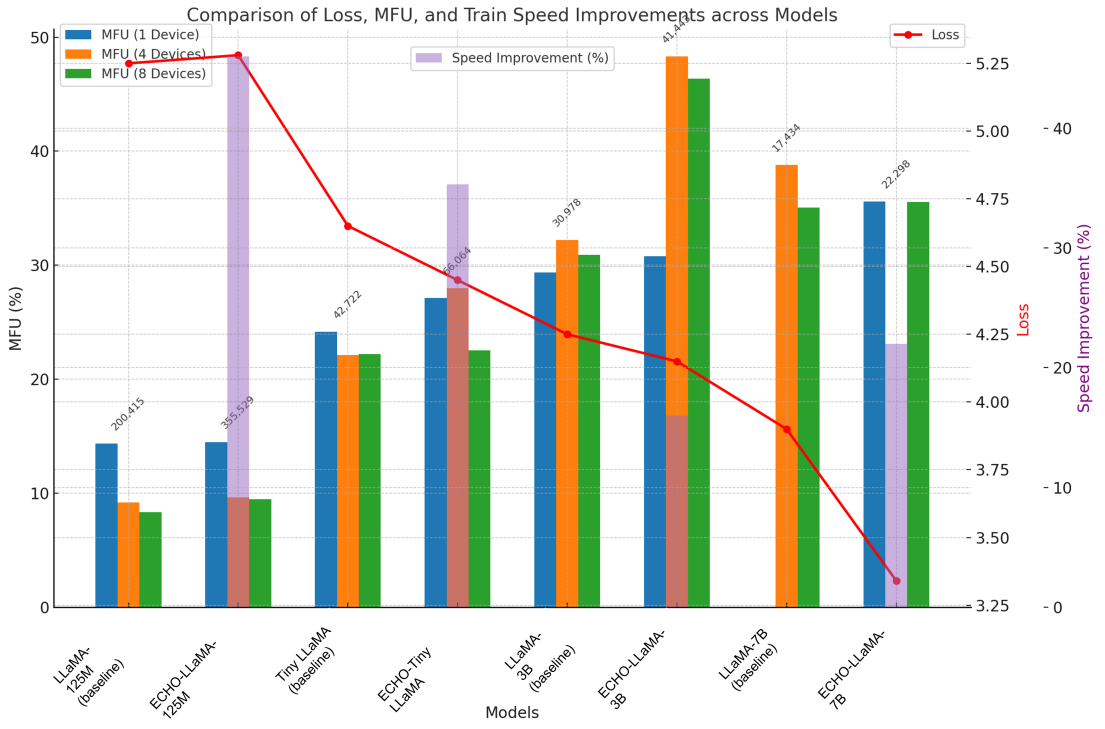


Figure 7.2: Comparison between of training throughput (Tokens/sec) and final loss for ECHO-LLaMA models and their baselines. ECHO versions consistently achieve lower loss with higher or comparable Tokens/sec speed. Each model is pretrained from scratch on 4B tokens through 1000 steps. Values on top each set of bars shows train token/second throughput.

Section 7.7 includes more results on larger models and more diverse datasets. Moreover, Section 7.7.3 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 7.2 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 [7] 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

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

sizes. ECHO-LLaMA models achieve lower loss values and up to 15% improvements in MFU, as 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 7.4, 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 7.3. 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 7.6 and Table 7.7 in the Section.

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

In another experiment, ECHO-LLaMA is evaluated on Robo-Instruct Benchmark. The results in Table 7.5 clearly demonstrate that the proposed **ECHO-LLaMA** framework maintains or improves task performance while substantially reducing key–value (KV) memory usage, thus achieving a favorable trade-off between accuracy and efficiency in robotic reasoning tasks. Across all evaluated models, the 25% KV-sharing configuration consistently outperforms or matches the baseline LLMs under both zero-shot and few-shot

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

settings, confirming that partial KV reuse preserves representational fidelity while reducing inference-time computation. For instance, ECHO-CodeLlama-7B (25% KV) achieves an average pass@1 score of 63.01—an improvement over the baseline 62.15—despite operating with only one quarter of the original cache size. A similar pattern is observed for Llama-3-8B and Qwen-7B, where 25% KV sharing yields 0.7–0.8 percentage-point gains on average. In contrast, excessive sharing (50% KV) slightly degrades accuracy, suggesting that optimal sharing ratios depend on model capacity and domain complexity.

From a mechanical-domain perspective, these findings underscore the practicality of **ECHO-LLaMA** for embodied and robotic systems, where compute and memory budgets are constrained by onboard hardware. By enabling efficient context reuse without significant accuracy loss, ECHO-LLaMA enhances the viability of large language models as real-time task planners and controllers in robotic manipulation, navigation, and multi-step reasoning environments. The method thus bridges the gap between instruction-following language models and resource-efficient robotic intelligence.

7.5 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

Table 7.5: Performance comparison of **Robo-Instruct** fine-tuned LLMs with their ECHO version on the RoboEval benchmark (pass@1). The ECHO results are reported only at 25% and 50% of KV sharing. The generation temperature is set at zero. The dataset and evaluation strategy are based on the instruction reported at Ref. [111].

Model	**Zero-shot**	**5-shot**	**Average**
CodeLlama-7B	61.15	63.14	62.15
ECHO-CodeLlama-7B (25% KV)	61.78	64.23	**63.01**
ECHO-CodeLlama-7B (50% KV)	60.21	61.43	60.82
Llama-3-8B	58.65	60.19	59.42
ECHO-Llama-3-8B (25% KV)	59.11	61.21	**60.16**
ECHO-Llama-3-8B (50% KV)	57.17	58.35	57.76
Qwen-7B	61.25	63.37	62.31
ECHO-Qwen-7B (25% KV)	61.54	64.05	**62.80**
ECHO-Qwen-7B (50% KV)	59.76	60.44	60.10

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 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 7.2 and Section 7.8.

7.6 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.

7.7 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 7.6 presents a detailed comparison of baseline models and their ECHO-enhanced counterparts across a broad range of standard evaluation benchmarks, including MMLU [101, 99], C-Eval [113], GSM8K [57], MATH [102], HumanEval [46], and MBPP [16, 233].

We examine LLaMA2-7B [233]. 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 [19] 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.

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 7.6: 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 7.7: Evaluating the generalizability of the ECHO mechanism across several LLaMA3.2 models on diverse benchmark. The results are reported in terms of accuracy.

7.7.1 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 7.3 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.



Figure 7.3: 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.

7.7.2 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 7.4, 7.5, 7.6, and 7.7 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 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.

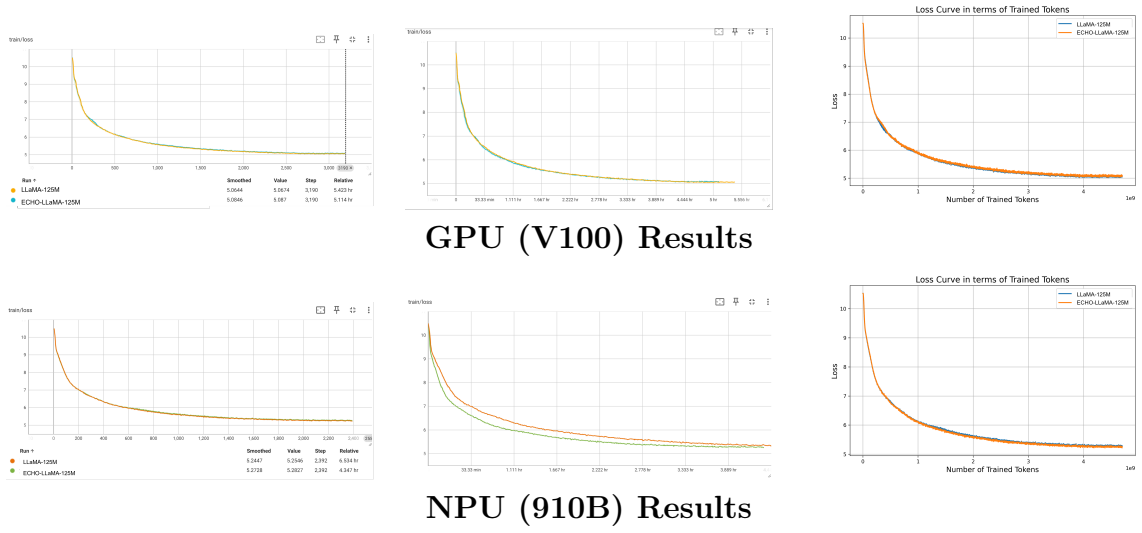


Figure 7.4: Training results for the LLaMA-125M LLM on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

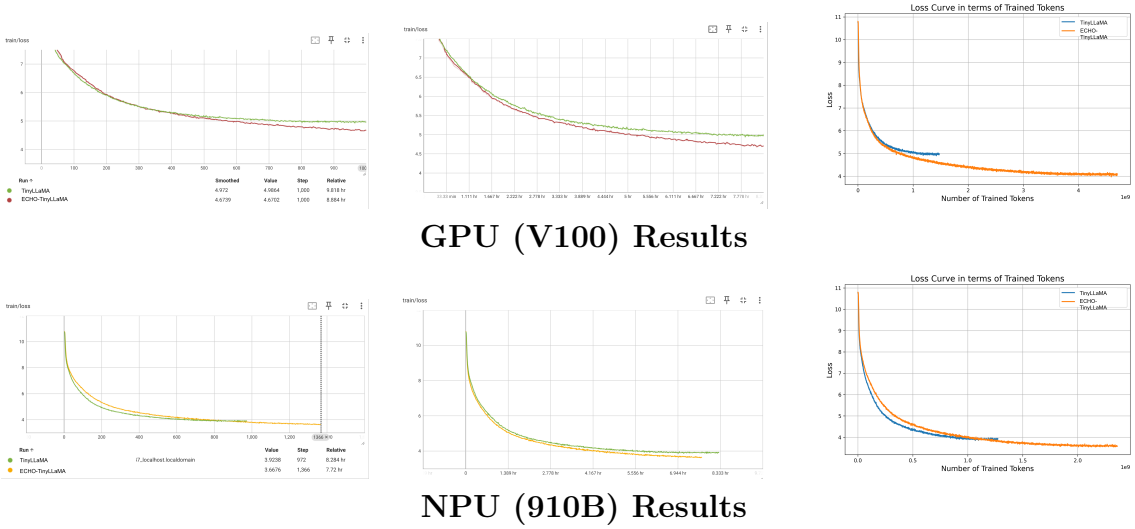


Figure 7.5: Training results for TinyLLaMA-1.1B LLM on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

7.7.3 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 7.8 and 7.9 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%.

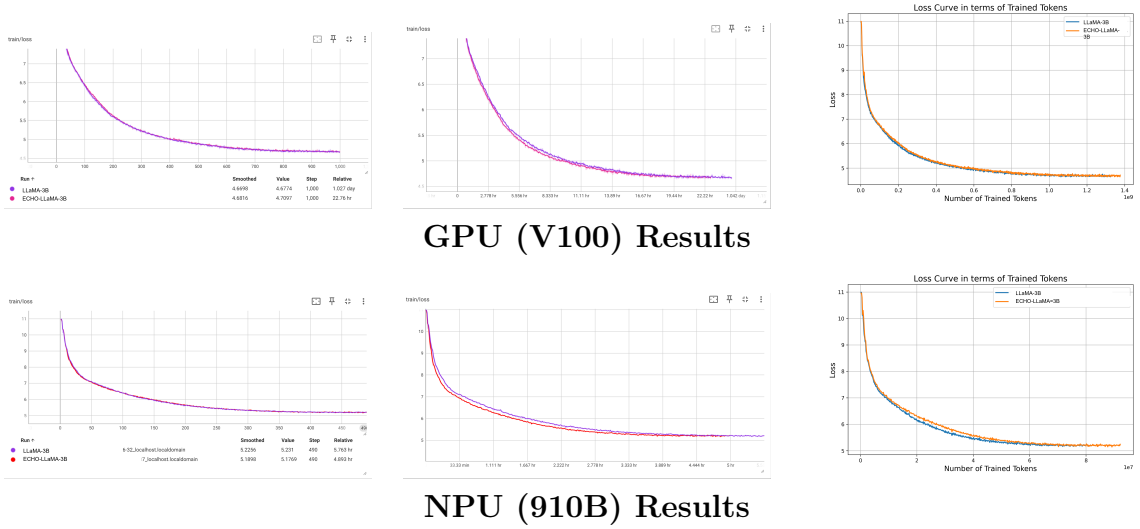


Figure 7.6: Training results for LLaMA-3B on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

7.7.4 MFU-Loss-Speed

Figure 7.3 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.

7.7.5 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.

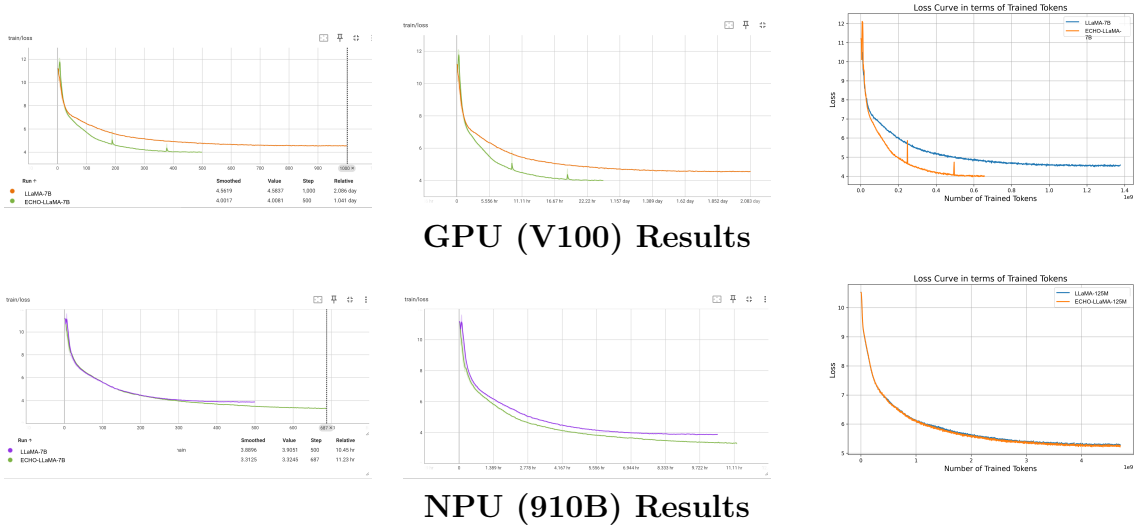


Figure 7.7: Training results for LLaMA-7B on GPU (V100) and NPU (910B). The columns represent (1) Train Loss vs. Steps, (2) Train Loss vs. Train Time, and (3) Train Loss vs. Number of Train Tokens.

As shown in Figure 7.11, 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.

7.7.6 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 7.10, 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.

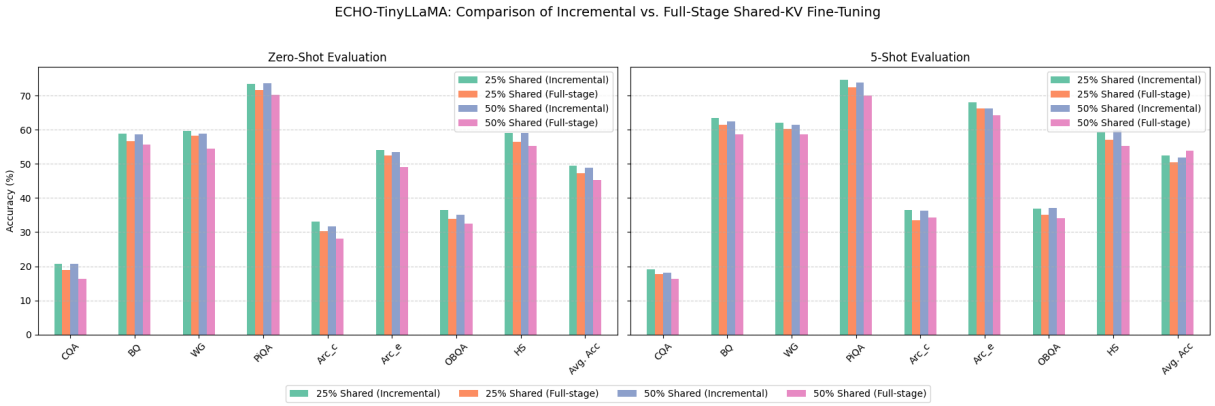


Figure 7.8: **LLaMA-7B**: Comparing Incremental Sharing over Full-Stage Sharing for shared-KV adaptation.

7.7.7 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 7.12 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.

7.7.8 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 7.8, a smaller block size leads to better performance. Specifically, setting the block

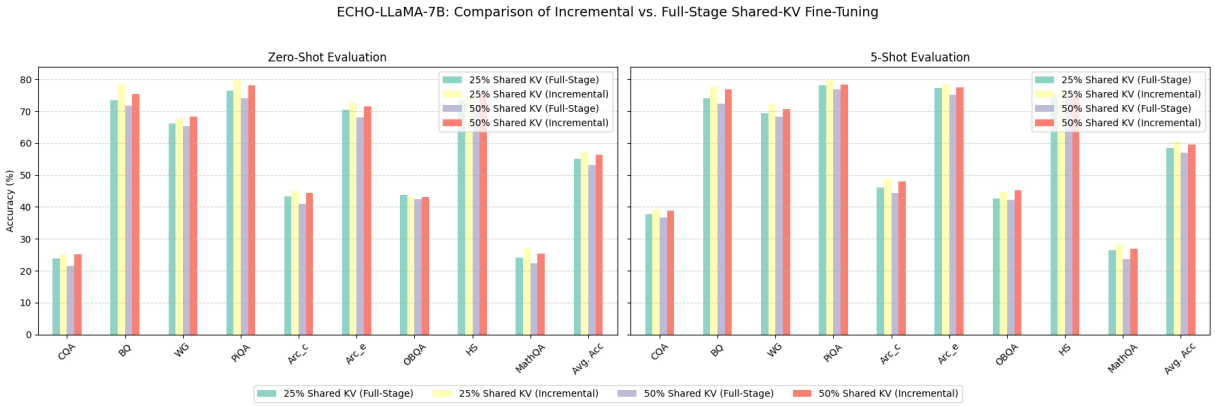


Figure 7.9: **LLaMA-7B**: Comparing Incremental Sharing over Full-Stage Sharing for shared-KV adaptation.

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.

7.8 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 7.9, 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.

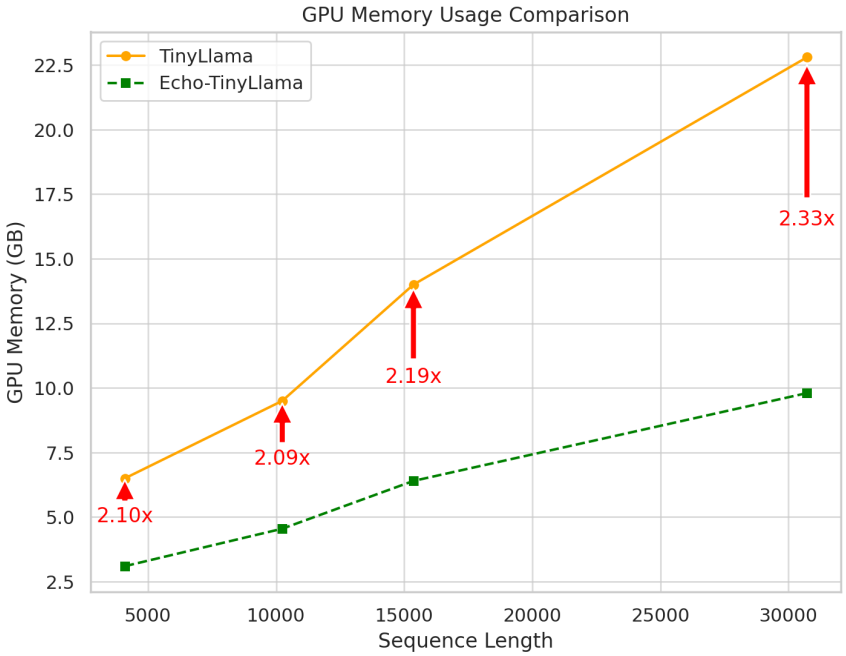


Figure 7.10: GPU memory usage comparison between the baseline TinyLLaMA and ECHO-TinyLLaMA across different sequence lengths. ECHO-TinyLLaMA consistently consumes nearly half the memory compared to TinyLLaMA, as indicated by the annotated reduction ratios.

Figure 7.11: Scaling law diagram comparing training loss for ECHO-LLaMA of different sizes against the baselines.

7.9 Questions / Answers

This section addresses several questions relevant to the research.

7.9.1 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-

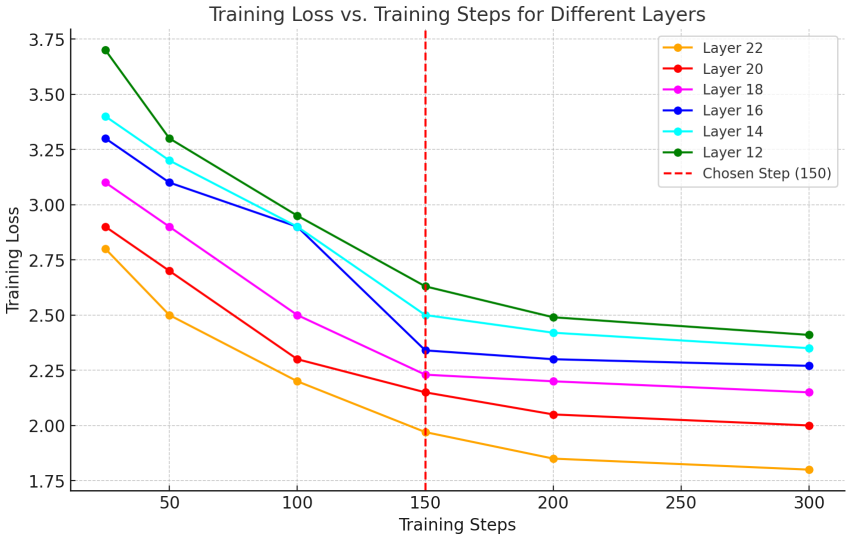


Figure 7.12: Training Loss vs. Training Steps for Different Layers in ECHO-TinyLLaMA. The loss decreases significantly up to 150 steps, after which the rate of improvement diminishes. This justifies our selection of $S \approx 150$ training steps as the optimal point for balancing computational cost and performance.

LLAMA proposes a lightweight fine-tuning approach that transforms existing pre-trained models into ECHO architectures. For these reasons, we did not include a direct comparison with YOCO in our evaluation.

7.9.2 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.

Table 7.8: 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

7.10 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 chapter, 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 7.10 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.

Table 7.9: 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

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 7.10: 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.

7.11 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.

7.11.1 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
}
```


7.11.2 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",
```

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 7.11: Notation used throughout this chapter.

```
  "pad_token_id": None,
  "bos_token_id": 1,
  "eos_token_id": 2
}
```

7.11.3 Notation

Table 7.11 summarizes the key notations used throughout the chapter. These notations are consistent with standard transformer-based LLM literature and are used to describe our cross-layer KV-sharing architecture and training procedure.

Chapter 8

Conclusion and Future Work

This thesis addressed the critical challenges of computational inefficiency, scalability, and resource optimization in **Large Language Models (LLMs)**. Driven by the observation that the immense scale which grants LLMs their versatility also introduces significant bottlenecks during pre-training, fine-tuning, and inference, this work developed a suite of five novel, modular strategies. These contributions collectively advance the efficiency and accessibility of LLMs across the entire development lifecycle, enabling their practical deployment in resource-constrained environments from cloud to edge devices.

8.1 Summary of Contributions

This dissertation's core objective was achieved through the design, implementation, and rigorous validation of five distinct, yet complementary, methods. Each targets a specific bottleneck in the LLM pipeline, as detailed in the preceding chapters.

8.1.1 Enhancing Fine-Tuning Efficiency

- **QDyLoRA (Quantized Dynamic Low-Rank Adaptation):** To mitigate the substantial memory demands of fine-tuning, QDyLoRA introduced an innovation over standard QLoRA. This method allows for **efficient fine-tuning across multiple LoRA ranks in a single training pass**, significantly reducing memory consumption while enhancing model adaptability. QDyLoRA ensures that a single training run yields models optimized for varying hardware constraints, making LLM adaptation more flexible and cost-effective.

- **Sorted-LoRA (Dynamic Model Adaptation):** Addressing the inefficiency of training separate models for different deployment hardware, Sorted-LoRA introduced a dynamic adaptation method. By integrating LoRA modules and employing a **sorted nested training strategy**, it enables the simultaneous learning of **multiple sub-models of varying sizes** within a single training cycle. This innovation provides a scalable, cost-effective solution for multi-size deployment.

8.1.2 Accelerating Inference and Computational Efficiency

- **ECHO-LLaMA (Cross-Layer KV Sharing):** Focused on improving inference throughput in autoregressive models, ECHO-LLaMA reduced redundant Key-Value (KV) computations. By reusing **shared global KV representations in the upper layers**, this method substantially lowers both the computational and memory overhead associated with the KV cache. This achieved up to **7% higher test-time throughput** and **16% higher Model FLOPs Utilization (MFU)** compared to the baseline.

- **LoRA-Drop (Adaptive Layer Deactivation):** Addressing computational redundancy at the token level, LoRA-Drop accelerates inference by selectively deactivating the main transformer layers for simpler tokens. By relying on lightweight integrated **LoRA modules for sufficient contextual adjustment**, it bypasses heavy matrix multiplications, resulting in a significant reduction in inference latency and energy cost without compromising generation quality.

- **EchoAtt (Attend, Copy, then Adjust):** This framework tackled the quadratic complexity of self-attention by leveraging the empirical discovery of **highly similar attention matrices across inner layers**. EchoAtt constructed a shared attention student model via Knowledge Distillation and continual training, selectively sharing attention matrices in less critical layers. This resulted in a **15**% inference speedup, **25**% training speedup, and a **4**% parameter reduction on TinyLLaMA-1.1B, while surprisingly **improving zero-shot performance**.

8.2 Conclusion

The contributions of this thesis collectively provide several frameworks for optimizing LLMs across memory, speed, and architectural efficiency. From the dynamic fine-tuning of QDyLoRA and Sorted-LoRA to the targeted inference acceleration of ECHO-LLaMA and LoRA-Drop, and the foundational attention optimization of EchoAtt, this work offers a powerful suite of solutions. The core finding is that by systematically identifying and exploiting various forms of redundancy—be it in parameter space, token-level computation, or attention patterns—it is possible to significantly reduce the computational footprint of LLMs without sacrificing their powerful generalization

capabilities. This research successfully strengthens the feasibility of deploying large-scale AI systems in real-world environments, making them more adaptable, accessible, and practical for applications in domains like **healthcare, biomedicine, education, and robotics**.

8.3 Future Work

Building upon the success of these specialized strategies, future research can focus on their synergistic potential and broader application:

1. **Unified Framework Integration:** Explore combining compatible methods, such as integrating EchoAtt's attention sharing into an ECHO-LLaMA architecture, to achieve stacked efficiency gains in a single model structure.
2. **Dynamic and Adaptive Optimization:** Develop a system that dynamically selects the optimal configuration (LoRA rank via QDyLoRA, layer depth via Sorted-LoRA, and layer deactivation via LoRA-Drop) based on real-time task complexity and hardware availability during inference.
3. **Generalization to Multimodal Models:** Extend the efficiency principles, particularly attention sharing (EchoAtt) and KV reuse (ECHO-LLaMA), to Vision-Language Models (VLMs) and other multimodal foundation models to tackle their even greater resource demands.
4. **Theoretical Foundation of Regularization:** Conduct deeper theoretical analysis into why methods like EchoAtt and LoRA-Drop not only maintain but, in some cases, improve zero-shot performance, formalizing the relationship between structured compression and model generalization.

References

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

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

[3] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[4] Orevaoghene Ahia, Julia Kreutzer, and Sara Hooker. The low-resource double bind: An empirical study of pruning for low-resource machine translation. *arXiv preprint arXiv:2110.03036*, 2021.

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

[6] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[7] AI@Meta. Llama 3 model card. 2024.

[8] Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, et al. Colt5: Faster long-range transformers with conditional computation. *arXiv preprint arXiv:2303.09752*, 2023.

[9] American Psychological Association. *Publications Manual*. American Psychological Association, Washington, DC, 1983.

[10] Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of*

the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 2357–2367, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[11] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, December 2005.

[12] Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40, 2007.

[13] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[14] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024.

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

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

[17] Muhammad Awais, Muzammal Naseer, Salman Khan, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Foundation models defining a new era in vision: a survey and outlook. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

[18] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[19] 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, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[20] Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

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

[22] Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Smollm-corpus, 2024.

[23] Jonathan Berant, Andrew K. Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Conference on Empirical Methods in Natural Language Processing*, 2013.

[24] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2(3):8, 2023.

[25] Lucas Beyer, Pavel Izmailov, Alexander Kolesnikov, Mathilde Caron, Simon Kornblith, Xiaohua Zhai, Matthias Minderer, Michael Tschannen, Ibrahim Alabdulmohsin, and Filip Pavetic. Flexivit: One model for all patch sizes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14496–14506, 2023.

[26] Srinadh Bhojanapalli, Ayan Chakrabarti, Andreas Veit, Michal Lukasik, Himanshu Jain, Frederick Liu, Yin-Wen Chang, and Sanjiv Kumar. Leveraging redundancy in attention with reuse transformers. *arXiv preprint arXiv:2110.06821*, 2021.

[27] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[28] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[29] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019.

[30] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[31] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[32] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[33] Yelysei Bondarenko, Riccardo Del Chiaro, and Markus Nagel. Low-rank quantization-aware training for llms. *arXiv preprint arXiv:2406.06385*, 2024.

[34] Neal R Brodnik, Samuel Carton, Caelin Muir, Satanu Ghosh, Doug Downey, McLean P Echlin, Tresa M Pollock, and Samantha Daly. Perspective: large language models in applied mechanics. *Journal of Applied Mechanics*, 90(10):101008, 2023.

[35] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[36] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[37] Han Cai, Chuang Gan, Ji Lin, and Song Han. Network augmentation for tiny deep learning. *arXiv preprint arXiv:2110.08890*, 2021.

[38] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment, 2020.

[39] Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. *arXiv preprint arXiv:2406.10260*, 2024.

[40] Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model, 2024.

[41] Ruisi Cai, Zhenyu Zhang, and Zhangyang Wang. Robust weight signatures: gaining robustness as easy as patching weights? In *International Conference on Machine Learning*, pages 3495–3506. PMLR, 2023.

[42] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

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

[44] Arnav Chavan, Zhiqiang Shen, Zhuang Liu, Zechun Liu, Kwang-Ting Cheng, and Eric P Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4931–4941, 2022.

[45] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[46] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[47] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[48] Tianlong Chen, Zhenyu Zhang, Ajay Jaiswal, Shiwei Liu, and Zhangyang Wang. Sparse moe as the new dropout: Scaling dense and self-slimmable transformers. *arXiv preprint arXiv:2303.01610*, 2023.

[49] Wei Chen, Yu Liu, Weiping Wang, Erwin M Bakker, Theodoros Georgiou, Paul Fieguth, Li Liu, and Michael S Lew. Deep learning for instance retrieval: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[50] Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, et al. On the representation collapse of sparse mixture of experts. *Advances in Neural Information Processing Systems*, 35:34600–34613, 2022.

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

[52] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, et al. Palm: Scaling language modeling with pathways, 2022.

[53] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[54] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[55] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

[56] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[57] 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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[58] Tri Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[59] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[60] Jeff Dean. Introducing Pathways: A next-generation AI architecture. *Google Blog*, 366, 2021.

[61] DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, A. X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. Deepseek llm: Scaling open-source language models with longtermism, 2024.

[62] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[63] Mostafa Dehghani, Alexey Gritsenko, Anurag Arnab, Matthias Minderer, and Yi Tay. Scenic: A jax library for computer vision research and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21393–21398, 2022.

[64] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[65] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[66] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pages 7750–7774. PMLR, 2023.

[67] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[68] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

[69] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

[70] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[71] David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A Saurous, Jascha Sohl-Dickstein, et al. Language model cascades. *arXiv preprint arXiv:2207.10342*, 2022.

[72] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[73] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts, 2022.

[74] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

[75] Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022.

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

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

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

[79] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

[80] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

[81] Wikimedia Foundation. Wikimedia downloads.

[82] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR, 2023.

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

[84] Daocheng Fu, Xin Li, Licheng Wen, Min Dou, Pinlong Cai, Botian Shi, and Yu Qiao. Drive like a human: Rethinking autonomous driving with large language models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 910–919, 2024.

[85] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[86] 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, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024.

[87] Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023.

[88] John W Goodell, Satish Kumar, Weng Marc Lim, and Debidutta Pattnaik. Artificial intelligence and machine learning in finance: Identifying foundations, themes, and research clusters from bibliometric analysis. *Journal of Behavioral and Experimental Finance*, 32:100577, 2021.

[89] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[90] Matteo Grimaldi, Luca Mocerino, Antonio Cipolletta, and Andrea Calimera. Dynamic convnets on tiny devices via nested sparsity. *IEEE Internet of Things Journal*, 10(6):5073–5082, 2022.

[91] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[92] Jianyuan Guo, Kai Han, Yunhe Wang, Han Wu, Xinghao Chen, Chunjing Xu, and Chang Xu. Distilling object detectors via decoupled features. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2154–2164, 2021.

[93] Song Guo, Jiahang Xu, Li Lyna Zhang, and Mao Yang. Compresso: Structured pruning with collaborative prompting learns compact large language models. *arXiv preprint arXiv:2310.05015*, 2023.

[94] Dan Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK, 1997.

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

[96] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[97] Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. What matters in transformers? not all attention is needed. *arXiv preprint arXiv:2406.15786*, 2024.

[98] Yunhao He, Rui Zhang, Qingyang Li, and Wei Liu. AdaSkip: Adaptive Sublayer Skipping for Accelerating Long-Context LLM Inference. *arXiv preprint arXiv:2501.02336*, 2025.

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

[100] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

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

[103] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[104] Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

[106] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. Characterising bias in compressed models. *arXiv preprint arXiv:2010.03058*, 2020.

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

[108] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[109] Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[110] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[111] Zichao Hu, Junyi Jessy Li, Arjun Guha, and Joydeep Biswas. Robo-instruct: Simulator-augmented instruction alignment for finetuning code llms. *arXiv preprint arXiv:2405.20179*, 2024.

[112] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2(2), 2017.

[113] 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. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*, 2023.

[114] Aref Jafari, Mehdi Rezagholizadeh, Pranav Sharma, and Ali Ghodsi. Annealing knowledge distillation. *arXiv preprint arXiv:2104.07163*, 2021.

[115] Ashutosh Jain, Abhinav Vyas, Ankit Rao, and Partha Mazumder. FiRST: Fine-Tuned Router-Selective Transformers for Efficient LLM Inference. *arXiv preprint arXiv:2410.12513*, 2024.

[116] Ajay Jaiswal, Zhe Gan, Xianzhi Du, Bowen Zhang, Zhangyang Wang, and Yinfei Yang. Compressing llms: The truth is rarely pure and never simple, 2023.

[117] Benyamin Jamialahmadi, Parsa Kavehzadeh, Mehdi Rezagholizadeh, Parsa Farinneya, Hossein Rajabzadeh, Aref Jafari, Boxing Chen, and Marzieh S Tahaei. Balcony: A lightweight approach to dynamic inference of generative language models. *arXiv preprint arXiv:2503.05005*, 2025.

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

[119] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[120] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[121] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[122] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[123] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[124] Simarjeet Kaur, Jimmy Singla, Lewis Nkenyereye, Sudan Jha, Deepak Prashar, Gyanendra Prasad Joshi, Shaker El-Sappagh, Md Saiful Islam, and SM Riazul Islam. Medical diagnostic systems using artificial intelligence (ai) algorithms: Principles and perspectives. *IEEE Access*, 8:228049–228069, 2020.

[125] Parsa Kavehzadeh, Mojtaba Valipour, Marzieh Tahaei, Ali Ghodsi, Boxing Chen, and Mehdi Rezagholizadeh. Sorted llama: Unlocking the potential of intermediate layers of large language models for dynamic inference. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2129–2145, 2024.

[126] Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, and Bin Liu. Continual pre-training of language models. In *International Conference on Learning Representations*, 2023.

[127] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[128] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*, 2023.

[129] Sehoon Kim, Karttikeya Mangalam, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Big little transformer decoder. *arXiv preprint arXiv:2302.07863*, 2023.

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

[131] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[132] Jongwoo Ko, Seungjoon Park, Yujin Kim, Sumyeong Ahn, Du-Seong Chang, Euijai Ahn, and Se-Young Yun. NASH: A simple unified framework of structured pruning for accelerating encoder-decoder language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[133] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations–democratizing large language model alignment. *arXiv preprint arXiv:2304.07327*, 2023.

[134] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

[135] Alex Krizhevsky. Convolutional neural networks for object classification in cuda. *University of Toronto, EECE1742S: Programming Massively Parallel Multiprocessors Using CUDA*, 2009.

[136] Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, et al. Nemo: a toolkit for building ai applications using neural modules. *arXiv preprint arXiv:1909.09577*, 2019.

[137] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[138] Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. *arXiv preprint arXiv:2110.03742*, 2021.

[139] Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, Prateek Jain, et al. Matformer: Nested transformer for elastic inference. *arXiv preprint arXiv:2310.07707*, 2023.

[140] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

[141] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pages 5544–5555. PMLR, 2020.

[142] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

[143] Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. *arXiv preprint arXiv:2210.03858*, 2022.

[144] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[145] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

[146] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[147] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations, 2017.

[148] Leslie Lamport. *LaTeX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[149] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {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, 2024.

[150] Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 552–561. AAAI Press, 2012.

[151] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. 2023.

[152] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 8607–8617, 2021.

[153] Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

[154] Shengrui Li, Junzhe Chen, Xueting Han, and Jing Bai. Nuteprune: Efficient progressive pruning with numerous teachers for large language models. *arXiv preprint arXiv:2402.09773*, 2024.

[155] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

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

[157] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[158] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

[159] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[160] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

[161] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024.

[162] Meng Liu, Fan Meng, and Yu Zhou. Unified Layer Skipping: Towards Stable and Practical Acceleration of Large Language Model Inference. *arXiv preprint arXiv:2404.06954*, 2024.

[163] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[164] Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. Webglm: Towards an efficient web-enhanced question answering system with human preferences. *arXiv preprint arXiv:2306.07906*, 2023.

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

[166] Xuan Luo, Zhihong Tang, Jingtao Han, and Liwei Wang. FlexiDepth: Dynamic Depth Allocation for Efficient Large Language Model Inference. *arXiv preprint arXiv:2502.01584*, 2025.

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

[168] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

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

[170] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*, 2021.

[171] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification, 2023.

[172] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018.

[173] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

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

[175] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5191–5198, 2020.

[176] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[177] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics.

[178] Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial nli: A new benchmark for natural language understanding, 2020.

[179] nostalgebraist. Interpreting gpt: The logit lens. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens, August 2020. Accessed: 2025-02-22.

[180] NVIDIA. Tensorrt-llm: A tensorrt toolbox for optimized large language model inference, 2023.

[181] Salome Oniani, Gonçalo Marques, Sophio Barnovi, Ivan Miguel Pires, and Akash Kumar Bhoi. Artificial intelligence for internet of things and enhanced medical systems. *Bio-inspired neurocomputing*, pages 43–59, 2021.

[182] OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie

Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2023.

[183] R OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5):1, 2023.

[184] Koyena Pal, Jiuding Sun, Andrew Yuan, Byron C Wallace, and David Bau. Future lens: Anticipating subsequent tokens from a single hidden state. *arXiv preprint arXiv:2311.04897*, 2023.

[185] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Nghia The Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, 2016.

[186] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics.

[187] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016.

[188] Jupinder Parmar, Shrimai Prabhumoye, Joseph Jennings, Mostofa Patwary, Sandeep Subramanian, Dan Su, Chen Zhu, Deepak Narayanan, Aastha Jhunjhunwala, Ayush Dattagupta, et al. Nemotron-4 15b technical report. *arXiv preprint arXiv:2402.16819*, 2024.

[189] Amit R Patel, Kashyap K Ramaiya, Chandrakant V Bhatia, Hetalkumar N Shah, and Sanket N Bhavsar. Artificial intelligence: Prospect in mechanical engineering field—a review. *Data Science and Intelligent Applications: Proceedings of ICDSIA 2020*, pages 267–282, 2021.

[190] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

[191] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

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

[193] Alec Radford. Improving language understanding by generative pre-training. 2018.

[194] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[195] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

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

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

[198] Hossein Rajabzadeh, Mojtaba Valipour, Tianshu Zhu, Marzieh Tahaei, Hyock Ju Kwon, Ali Ghodsi, Boxing Chen, and Mehdi Rezagholizadeh. Qdylora: Quantized dynamic low-rank adaptation for efficient large language model tuning. *arXiv preprint arXiv:2402.10462*, 2024.

[199] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[200] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, 2021.

[201] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

[202] Mohammad Sadegh Rasooli and Joel R. Tetreault. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733, 2015. version 2.

[203] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search, 2019.

[204] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[205] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

[206] Daniel Rotem, Michael Hassid, Jonathan Mamou, and Roy Schwartz. Finding the sweet spot: Analysis and improvement of adaptive inference in low resource settings. *arXiv preprint arXiv:2306.02307*, 2023.

[207] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[208] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[209] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019.

[210] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

[211] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[212] Mohammadreza Salehi, Sachin Mehta, Aditya Kusupati, Ali Farhadi, and Hanna Hajishirzi. Sharcs: Efficient transformers through routing with dynamic width sub-networks. *Findings of Empirical Methods in Natural Language Processing*, 2023.

[213] V Sanh. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[214] Marc Schmitt. Artificial intelligence in business analytics, capturing value with machine learning applications in financial services. 2020.

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

[216] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, pages 492–504. PMLR, 2023.

[217] Shihao Shao, Kaifeng Chen, Arjun Karpur, Qinghua Cui, Andre Araujo, and Bingyi Cao. Global features are all you need for image retrieval and reranking. *arXiv preprint arXiv:2308.06954*, 2023.

[218] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[219] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[220] David R So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668*, 2021.

[221] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023.

[222] Benjamin Spector and Chris Re. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*, 2023.

[223] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497. Springer, 2019.

[224] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

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

[226] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35:12991–13005, 2022.

[227] Annamalai Suresh, R Udendhran, and S Vimal. *Deep neural networks for multimodal imaging and biomedical applications*. IGI Global, 2020.

[228] Marzieh Tahaei, Aref Jafari, Ahmad Rashid, David Alfonso-Hermelo, Khalil Bibi, Yimeng Wu, Ali Ghodsi, Boxing Chen, and Mehdi Rezagholizadeh. Efficient citer: Tuning large language models for enhanced answer quality and verification. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4443–4450, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

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

[230] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

[231] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[232] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[233] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[234] Georgy Tyukin, Gbetondji JS Dovonon, Jean Kaddour, and Pasquale Minervini. Attention is all you need but you don't need all of it for inference of large language models. *arXiv preprint arXiv:2407.15516*, 2024.

[235] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.

[236] Mojtaba Valipour, Mehdi Rezagholizadeh, Hossein Rajabzadeh, Marzieh Tahaei, Boxing Chen, and Ali Ghodsi. Sortednet, a place for every network and every network in its place: Towards a generalized solution for training many-in-one neural networks. *arXiv preprint arXiv:2309.00255*, 2023.

[237] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[238] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2023.

[239] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.

[240] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

[241] Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to grow pretrained models for efficient transformer training. *arXiv preprint arXiv:2303.00980*, 2023.

[242] Peihao Wang, Rameswar Panda, and Zhangyang Wang. Data efficient neural scaling law via model reusing. In *International Conference on Machine Learning*, pages 36193–36204. PMLR, 2023.

[243] Xiaofang Wang, Dan Kondratyuk, Kris M Kitani, Yair Movshovitz-Attias, and Elad Eban. Multiple networks are more efficient than one: Fast and accurate models via ensembles and cascades. *arXiv preprint arXiv:2012.01988*, 2020.

[244] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[245] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*, 2022.

[246] Zhenyu Wang. Logitlens4llms: Extending logit lens analysis to modern large language models. *arXiv preprint arXiv:2503.11667*, 2025.

[247] Juyun Wee, Minjae Park, and Jaeho Lee. Prompt-based depth pruning of large language models, 2025.

[248] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[249] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[250] Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao Ma, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. Dilu: A knowledge-driven approach to autonomous driving with large language models. *arXiv preprint arXiv:2309.16292*, 2023.

[251] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[252] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

[253] Ge Zhang Yao Fu Wenhao Huang Huan Sun Yu Su Wenhu Chen Xiang Yue, Xingwei Qu. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

[254] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR, 2023.

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

[256] An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. Qwen2. 5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025.

[257] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[258] Huanrui Yang, Hongxu Yin, Maying Shen, Pavlo Molchanov, Hai Li, and Jan Kautz. Global vision transformer pruning with hessian-aware saliency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18547–18557, 2023.

[259] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6):1–32, 2024.

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

[261] Yifei Yang, Zouying Cao, and Hai Zhao. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*, 2024.

[262] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10809–10818, 2022.

[263] Chengxuan Ying, Guolin Ke, Di He, and Tie-Yan Liu. Lazyformer: Self attention with lazy update. *arXiv preprint arXiv:2102.12702*, 2021.

[264] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[265] Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width amp; depth pruning for vision transformers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(3):3143–3151, Jun. 2022.

[266] Jiahui Yu and Thomas Huang. Universally slimmable networks and improved training techniques, 2019.

[267] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019.

[268] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[269] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[270] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019.

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

[272] Cha Zhang and Yunqian Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

[273] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[274] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

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

[276] Yihua Zhang, Ruisi Cai, Tianlong Chen, Guanhua Zhang, Huan Zhang, Pin-Yu Chen, Shiyu Chang, Zhangyang Wang, and Sijia Liu. Robust mixture-of-expert training for convolutional neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 90–101, 2023.

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

[278] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[279] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 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, 2024. Association for Computational Linguistics.

[280] Zhenyi Zheng, Xinyu Zhang, Zhiqiang Li, Tao Lin, and Hanrui Wu. RingAttention: Efficient Attention for Long Sequences via Ring-Topology Parallelism. *arXiv preprint arXiv:2403.09345*, 2024.

[281] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

[282] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017.