SlimLlama: Large Language Models Compression via Low-Rank Feature Distillation

Yaya Sy, Christophe Cerisara, Irina Illina

SlimLlama: Large Language Models Compression via Low-Rank Feature Distillation

Yaya Sy yaya.sy@loria.fr
Christophe Cerisara christophe.cerisara@loria.fr
Irina Illina irina.illina@loria.fr

LORIA, CNRS, Nancy, France

Abstract

Current LLM structured pruning methods involve two steps: (1) compressing with calibration data and (2) continued pretraining on billions of tokens to recover the lost performance. This costly second step is needed as the first step significantly impacts performance. Previous studies have found that pretrained Transformer weights aren't inherently low-rank, unlike their activations, which may explain this performance drop. Based on this observation, we introduce a one-shot compression method that locally distills low-rank weights. We accelerate convergence by initializing the low-rank weights with SVD and using a joint loss that combines teacher and student activations. We reduce memory requirements by applying local gradient updates only. Our approach can compress Mixtral-8x7B within minutes on a single A100 GPU, removing 10 billion parameters while maintaining over 95% of the original performance. Phi-2 3B can be compressed by 40% using only 13 million calibration tokens into a small model that competes with recent models of similar size. We show our method generalizes well to non-transformer architectures: we compress Mamba-3B by 20% while maintaining 99% of its performance[1].

1 Introduction

Compressing Large Language Models is a crucial challenge that can be achieved with several complementary approaches: quantization, pruning, compression, and distillation. We propose a new one-shot compression method that locally distills low-rank weights and satisfies the three objectives:

1. The compression algorithm must be compute-efficient, e.g., it should require minimal computational resources and run in a reasonable amount of time. For instance, the iterative magnitude pruning algorithm as done in Frankle and Carbin (2019), is too costly to be applied to large-scale LLMs.

2. It must achieve fast convergence using as little training data as possible; this objective is often an issue with many model pruning methods that often require significant retraining after pruning to recover the lost performance.

3. The method should not cause a significant drop in performance; ideally, all the capabilities of the original LLM should remain intact.

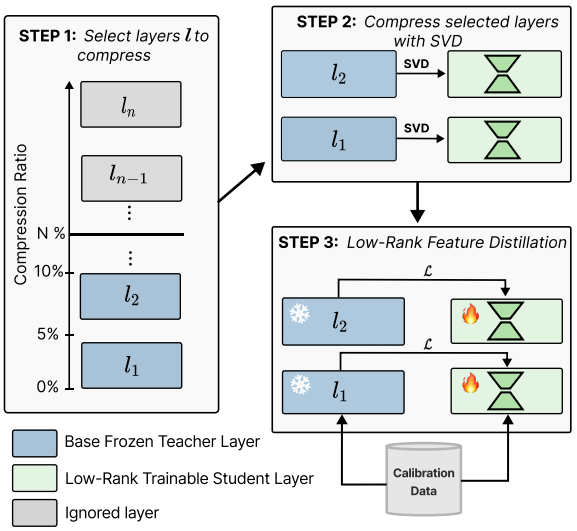

Figure 1: **Our compression approach**: **STEP 1** selects layers to compress for a target compression ratio (e.g., N%) using various strategies (see Section 5). **STEP 2** compresses and initializes the chosen parameters via SVD. **STEP 3** distills the low-rank weights with a small calibration dataset.

To address the first constraint, we compress the model with a local distillation objective, as opposed to a global objective that is more costly. We further reduce the cost by compressing and distilling only a subset of all layers. About the second constraint, we initialize the compressed layers with

[1]Code available at https://github.com/yaya-sy/SlimLlama

Method	No Custom Kernel	No Ampere-only	Memory Gain	Is One-Shot	Calib. Data
SparseGPT (Frantar and Alistarh, 2023)	✗	✗	✗	✓	-
Wanda (Sun et al., 2024)	✗	✗	✗	✓	-
Teal (Liu et al., 2024)	✗	✗	✗	✓	-
ShearedLLaMA (Xia et al., 2024)	✓	✓	✓	✗	52B
Minitron (Sreenivas et al., 2024)	✓	✓	✓	✗	94B
SlimLlama	✓	✓	✓	✓	**0.013B**

Table 1: Positioning of our approach within the compression literature. **No Custom Kernel** indicates that a custom GPU kernel is not required to observe speedup and **No Ampere-only** is whether the method requires Ampere GPU to achieve speedup. **Calib. Data** is the total number of tokens in billions used for compression.

Singular Value Decomposition (SVD), which enables reducing the required number of gradient steps as well as the calibration dataset size. We further improve convergence by combining Teacher and Student activations with a joint distillation loss. We experimentally show the robustness of our approach by compressing several state-of-the-art small and large Transformers, Mixture-of-Experts, and Mamba LLMs. Figure 1 gives an intuitive overview of the proposed method, which is detailed in Section 4 and validated in Section 6.

2 Related Works

We discuss next previous compression methods, highlight their limitations, and explain how our approach addresses them. Table 1 positions our approach to recent compression methods.

Pruning methods remove unimportant weights in the pre-trained model (LeCun et al., 1989; Han et al., 2015). Structured Pruning removes entire groups of parameters, which results in a smaller and faster model (Xia et al., 2024; Ma et al., 2023). Ma et al. (2023) propose a new gradient-based criterion to eliminate substructures in LLMs, while Xia et al. (2024) use a joint loss combining a pruning mask loss with the language modeling loss. However, optimizing these criteria can be computationally intensive. For example, the pruning step of Sheared-LLaMA (Xia et al., 2023) is 5x expensive compared to standard LM training, according to the authors. In contrast, thanks to the local gradient updates, our approach is computationally efficient, allowing us to compress a 47B model within minutes on a single A100 GPU. Regarding unstructured pruning, these methods do not provide any gains in terms of memory or speedup, at least with current algorithmic implementations. Semi-structured pruning (e.g., 2:4 and 4:8) (Sun et al., 2024; Frantar and Alistarh, 2023; Liu et al., 2024) does not lead to memory gain but can speed up processing on kernels optimized for such matrix structures. On the other hand, our method, which directly shrinks matrices, saves memory across all hardware and leads to speed up, as fewer computations are performed.

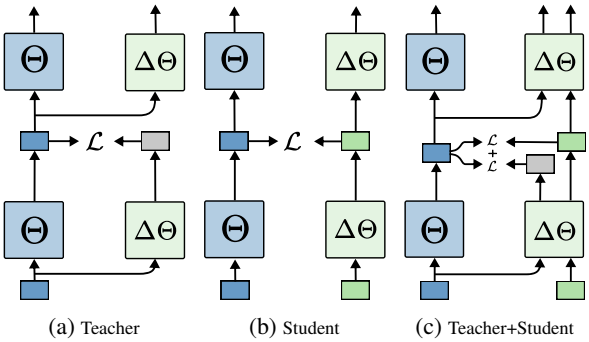

Figure 2: **Accelerating convergence by learning from Teacher and Student activations through a joint loss**. We propose to study the effect of three distillation strategies: **(a) Teacher**: the input to the compressed student layer comes from the output of the previous teacher layer; **(b) Student**: the input to the compressed student layer comes from the output of the previous student layer; **(c) Teacher+Student**: the compressed student layer receives both the output of the previous teacher layer and the output of the previous student layer, and the loss is the sum of two losses.

Low-Rank Decomposition compresses a model by approximating its pre-trained matrices with lower-dimension ones. Based on the observation by Li et al. (2018) that Neural Networks have lower intrinsic dimensions, Aghajanyan et al. (2020) show that Transformer language models also require lower intrinsic dimensions depending on the language task. However, Yu and Wu (2023); Chen et al. (2021) show that the weights in some pre-trained Transformer models are full-rank compared to their activations and propose an activation-aware compression method. Our work is related to these approaches, as we also decompose the matrices of the models. However, we apply a lightweight layer-

wise feature distillation objective to better recover from compression and propose different distillation strategies to accelerate convergence and improve performance.

Distillation. In Knowledge Distillation (KD), a small student model learns to reproduce the outputs of a larger pre-trained model. Standard KD is data-intensive when the student model's parameters are randomly initialized. Team et al. (2024) recently distilled Gemma-2 9B from a larger model using 8 trillion tokens, which is as costly as a standard pretraining stage. Another approach is layer-wise distillation, which can be more suitable in some cases, as intermediate layers may contain certain information, such as speaker identity or phonemes in pretrained speech models (Baevski et al., 2020; Pasad et al., 2022; Chang et al., 2022, 2021). For example, TinyBERT (Jiao et al., 2020) initializes the student model by removing layers from the teacher and then distills the student on task-specific knowledge, combining layer-wise and global distillation losses, which took 3.5 days to train. This is costly for LLMs. Instead of removing layers, we leverage the observation that layer activations are low-rank and approximate them with fewer parameters. Additionally, we compress the models to remain generalist rather than task-specific.

3 Background: Low-Rank Approximation of Transformer Models

3.1 Low-Rank Approximation from Weights.

Given $W \in \mathbb{R}^{d_1 \times d_2}$, a pretrained weight matrix, and an example $x \in \mathbb{R}^{d_2}$, the activations $y \in \mathbb{R}^{d_1}$ are computed as $y = Wx$. It is possible to reduce the computations in this operation by using a low-rank approximation of W: $y \simeq \Delta W x = ABx$ with $\Delta W = AB$ the low-rank decomposition of W, composed of $A \in \mathbb{R}^{d_1 \times r}$ and $B \in \mathbb{R}^{r \times d_2}$. When the rank r is small enough, the number of parameters $r(d_1 + d_2)$ in the low-rank equation is smaller than $d_1 d_2$ in the full-rank equation. Estimating the low-rank matrix $\Delta W = AB$ can be formulated as a minimization problem to find the low-rank ΔW that best approximates W:

$$\widehat{\Delta W} = \underset{\Delta W}{\operatorname{argmin}} \ \|W - \Delta W\|_F \tag{1}$$

It is well-known that this minimization problem can be approached using SVD, a low-rank approximation method that offers the optimal $r-$rank approximation of a given matrix with regard to the Frobenius norm $\|\cdot\|_F$. SVD approximates a matrix W into three matrices: $W = USV^T$, where $S \in \mathbb{R}^{d_1 \times d_2}$ is a diagonal matrix containing the singular values of W sorted in descending order, $U \in \mathbb{R}^{d_1 \times d_1}$ and $V \in \mathbb{R}^{d_2 \times d_2}$ are orthonormal matrices. The optimal low-rank approximation of $W \in \mathbb{R}^{d_1 \times d_2}$ can be obtained by keeping the first r singular values, with $r < min(d_1, d_2)$:

$$\widehat{\Delta W} = (U_{:,:r} S_{:r,:r}) V_{:r,:} = AB \tag{2}$$

with $A = U_{:,:r} S_{:r,:r}$ and $B = V_{:r,:}$ and the notation $_{:a,:b}$ refers to the slicing operation.

3.2 Low-Rank Approximation from Feature

Previous studies (Chen et al., 2021; Yu and Wu, 2023) have shown that the activations (i.e., *features*) of pretrained transformers, are more low-rank than the weights. Recently, Liu et al. (2024) show that transformer activations can be sparsified up to 60% without too much drop in performance. In Appendix A.5, we also show that the activations of the transformer layers are more low-rank than the weights. All these results suggest transformer activations can be approximated with fewer parameters. Now, let's examine how previous works (Kaushal et al., 2023; Chen et al., 2021; Yu and Wu, 2023) have approached this.

Let $\mathcal{D} = \{x_i \in \mathbb{R}^{d_2}\}_{1 \leq i \leq N}$ be a calibration dataset, and $W \in \mathbb{R}^{d_1 \times d_2}$ the parameters of a linear layer. Finding the low-rank matrix ΔW that best reproduces the activations $\{y = Wx\} \quad \forall x \in \mathcal{D}$ involves solving:

$$\widehat{\Delta W} = \underset{\Delta W}{\operatorname{argmin}} \ \frac{1}{N} \sum_{x \in \mathcal{D}} \|Wx - \Delta W x\|_F \tag{3}$$

An analytic solution to this minimization problem is the eigendecomposition of the covariance matrix of the activations. We first collect all activations $\mathcal{Y} = \{y = Wx\}_{\forall x \in \mathcal{D}}$, then the covariance matrix of the activations can be estimated as $\Sigma = \underset{y \in \mathcal{Y}}{\mathbb{E}} [yy^T] - \mathbb{E}[y]\mathbb{E}[y]^T$ with $\Sigma \in \mathbb{R}^{d_1 \times d_1}$. Since Σ is a diagonalizable matrix, we can apply its eigendecomposition: $\Sigma = USU^T$, where $U \in \mathbb{R}^{d_1 \times d_1}$ contains the eigenvectors of Σ and $S \in \mathbb{R}^{d_1 \times d_1}$ is the diagonal matrix containing the eigenvalues sorted in decreasing order. As for Eq-2, we can only keep the eigenvectors corresponding to the r largest eigenvalues, which gives us $A = U_{:,:r}$ and $B = U^T_{:,:r} W$. Compared to the low-rank matrices in Eq-2, A and B weights here learned to reproduce the activations of the base weight W, thanks to the minimization objective in Eq-3. In the next section, we highlight the limitations of this approach and

introduce our proposed method.

4 Proposed Approach

We identify and handle next three potential limitations of the approach presented in Section 3.2.

Non-linear Feature Approximation. The previous analytical solution for Eq-3 is limited to activations from linear layers. To generalize to non-linear modules, we first observe that Eq-3 can be seen as a feature distillation objective that may be optimized numerically by gradient descent rather than analytically by eigendecomposition. Given the input batch $X \in \mathbb{R}^{d \times b}$ of b examples, we denote the output activations of the i^{th} Teacher module $\mathcal{T}^{(i)}$ as

$$Y^{(i)} = \mathcal{T}^{(i)}(X; \Theta^{(i)}) \tag{4}$$

where $\Theta^{(i)}$ are the original pretrained matrices of the i^{th} Teacher, and $Y^{(i)} \in \mathbb{R}^{d \times b}$ its output activations. Similarly, we note the output activations of the i^{th} Student module $\mathcal{S}^{(i)}$ as :

$$\widehat{Y}^{(i)} = \mathcal{S}^{(i)}(X; \Delta\Theta^{(i)}) \tag{5}$$

where the Student module $\mathcal{S}^{(i)}$ is parametrized with the low-rank matrices $\Delta\Theta^{(i)}$ and $\widehat{Y}^{(i)} \in \mathbb{R}^{d \times b}$ are the output activations. We can express the distillation objective of the i^{th} Student module as:

$$\widehat{\Delta\Theta^{(i)}} = \underset{\Delta\Theta^{(i)}}{\operatorname{argmin}} \ \mathcal{L}^{(i)}(Y^{(i)}, \ \widehat{Y}^{(i)}) \tag{6}$$

where $\widehat{\Delta\Theta^{(i)}}$ are the estimated low-rank matrices of the i^{th} Student module $\mathcal{S}^{(i)}$ and $\mathcal{L}^{(i)}(Y^{(i)}, \ \widehat{Y}^{(i)})$ is the loss measuring the distance between the activations of the Teacher and the Student. We opted for the same loss as Chang et al. (2022) because we observed that the ℓ_1 loss yields instabilities:

$$\begin{aligned} \mathcal{L}^{(i)} &= \mathcal{L}^{(i)}_{\ell_1} + \mathcal{L}^{(i)}_{\cos} \\ &= \sum_{t=1}^{b} \left[\frac{1}{D} \left\| Y_t^{(i)} - \widehat{Y}_t^{(i)} \right\|_1 - \log \sigma \left(\cos \left(Y_t^{(i)}, \widehat{Y}_t^{(i)} \right) \right) \right] \end{aligned} \tag{7}$$

where D is the hidden vectors dimension, σ is the sigmoid activation and $\cos(\cdot, \cdot)$ is the cosine similarity. We propose to approximate the low-rank weights of the Students through Gradient Descent. In the linear case, this should converge towards the eigendecomposition solution described in Section 3.2. However, as we show later, we extend the distillation process to non-linear modules, which requires numerical optimization. We also show that initializing the Student's low-rank parameters with SVD, rather than randomly, improves convergence.

Beyond Teacher-only Activations. The second potential limitation of the approach described in Section 3.2 and equations 4 and 5 is that the Student and the Teacher modules take the same input X, which is the output of the precedent layer of the Teacher. This means that the Student is trained from the activations of the Teacher module only, which are not available at inference time. Yu and Wu (2023) use this approach to approximate the Linear Layers of Transformer models, referring to it as *Atomic Feature Mimicking*. Let $\widehat{Y}_T^{(i)} = \mathcal{S}^{(i)}(Y_T^{(i-1)}; \Delta\Theta^{(i)})$ be the output activations of the Student module when fed $Y_T^{(i-1)}$ the output activations of the Teacher T at the previous layer $i-1$. The loss can then be written as:

$$\mathcal{L}_T^{(i)} = \mathcal{L}^{(i)}(Y_T^{(i)}, \ \widehat{Y}_T^{(i)}) \tag{8}$$

where Y_i^T are the gold activations of the current Teacher. This distillation procedure is illustrated in Figure 2a. This approach can lead to a fast convergence as the Student modules learn from the gold and high-quality activations produced by the Teacher modules. However, it can cause a performance drop during inference as the Teacher activations will not be available. An alternative approach is to have the Student module take as input the activations of the previous Student modules rather than those of the Teacher modules:

$$\mathcal{L}_S^{(i)} = \mathcal{L}^{(i)}(Y_T^{(i)}, \ \widehat{Y}_S^{(i)}) \tag{9}$$

where $\widehat{Y}_S^{(i)} = \mathcal{S}^{(i)}(\widehat{Y}_S^{(i-1)}; \Delta\Theta^{(i)})$ is the output activations of the current Student module S when taking as input the output activations $\widehat{Y}_S^{(i-1)}$ of the Student module of the previous layer, and $Y_T^{(i)}$ are the gold activations of the corresponding Teacher module. This is the standard distillation and is illustrated in Figure 2b. However, although we found this loss performs generally better than the previous one, it leads to a slow convergence due to the errors in the activations of the Student modules. To address both issues, we propose to combine the two losses:

$$\mathcal{L}_{T+S}^{(i)} = \mathcal{L}_T^{(i)} + \mathcal{L}_S^{(i)} \tag{10}$$

This distillation approach is illustrated in Figure 2c. We don't introduce any hyperparameter for controlling the importance of one loss over another in the joint loss. We leave this for future work.

Module-wise Distillation. Finally, we observe that the functions $\mathcal{S}$ can be any other module in the neural network than linear layers: any sub-part of the model that outputs low-rank activations can be compressed with our method. In our experiments, we distill at the Transformer layer level and leave comparisons with other distillation levels for future work. We also experiment with other models than

pretrained Transformer-based language models.

5 Fast and Memory-Efficient Compression

Computing the optimal compression rank of each matrix individually may be costly: while search algorithms can be a solution for smaller models, they can be difficult to scale to LLMs with billions of parameters. Assuming a target compression rate N, we propose to consider three simple strategies: **uniform**, **top**, and **bottom** first compression strategies, which are illustrated in Appendix A.9.

Uniform. This approach removes $N\%$ parameters to each weight matrix in the model. This strategy involves training all LLM weights.

Bottom. This approach removes $N\%$ parameters from the model by prioritizing lower layers as detailed in Algorithm 1. The core of the algorithm is line 7, where the layers are ordered in bottom-top order and ranks are sorted in decreasing order. The argument k controls the *spread* of compression: the lower its value the more weights in the bottom layer will be compressed first. The algorithm is also illustrated in Steps 1 and 2 in Figure 1. This strategy is efficient as only a subset of weights in the bottom layers will be trained. Consequently, since there is no need to forward through the whole model, only a subset of the weights needs to be loaded into GPU memory, making this approach scalable to larger models.

Top. This approach removes $N\%$ parameters from the model by prioritizing top layers. This consists roughly of reversing the order of layers in the line 7 of Algorithm 1. However, this strategy requires loading all the model's parameters into GPU memory. It also involves forwarding through the entire model, even if only the top layer weights are being trained.

Algorithm 1: Bottom Layers First Compression Algorithm

Input: $\mathcal{M}$ is the base model and $\mathcal{M}'$ its copy; S is the target size of the compressed model; k is the minimum possible rank to set; m the increment when generating ranks.
Result: The low-rank model $\mathcal{M}'$.

1 $\mathcal{R} \leftarrow \varnothing$; `// Stack that will contain ranks and weight matrices.`
2 **for** *each weight matrix* $W \in \mathbb{R}^{d_1 \times d_2}$ *in* $\mathcal{M}$ **do**
3 $b \leftarrow \min(d_1, d_2)$
4 **for** $r = k$; $r \leq b$; $r = r + m$ **do**
5 **if** $r \times (d_1 + d_2) < d_1 \times d_2$ **then**
6 $\mathcal{R} \leftarrow \mathcal{R} \cup \{(r, W)\}$
7 **sort** $\mathcal{R}$ primarily by layer index in increasing order and secondarily by rank in decreasing order.
8 **while** $|\mathcal{M}'| > S$ **do**
9 **get** the next (r, W) from the stack $\mathcal{R}$
10 **compute** A, B from W using Eq. 2 with the rank r
11 **replace** W by A, B
12 **return** $\mathcal{M}'$

6 Experiments on Transformers

6.1 Setup

Models. We first evaluate our method on various transformer-based LLMs: a 47B mixture-of experts language model (Mixtral-v0.1 8x7B (Jiang et al., 2024)), medium-sized LLMs (Mistral-v0.1 7B (Jiang et al., 2023) and Phi-3 14B [2]), and a small language model (Phi-2 3B[3]).

Data. For calibration data, we use 13 million tokens randomly sampled from Slim-Orca (Lian et al., 2023)[4], an open-source replication of the Orca instruction dataset (Mukherjee et al., 2023). In preliminary experiments, we also tested other datasets, such as RedPajama (Computer, 2023), but found the results were better with instruction data.

Hyperparameters. When compressing models using Algorithm 1, we set a minimum rank of $k = 1024$ (except for Phi-3 14B, where $k = 1536$ yielded better results). For all models, the rank increment m is set to 256. We use the Teacher+Student loss (Figure 2c). Section 6.2 ablates these choices.

Evaluation. We evaluate the speed by measuring the time to forward a batch of 4 prompts of varying sequence lengths. Using lm-evaluation-harness (Gao et al., 2023), we evaluate the zero-shot performance of the base and compressed models on 9 downstream tasks: TruthfulQA (**TruQA**; (Lin et al., 2022)), Social IQa (**SIQA**; (Sap et al., 2019)), **LogiQA** (Liu et al., 2020), WinoGrande (**WinoG**; (Sakaguchi et al., 2019)), Arc Easy (**ARC-E**; (Clark et al., 2018)), Arc Challenge (**ARC-C**; (Clark et al., 2018)), **BoolQ** (Clark et al., 2019), **PIQA** (Bisk et al., 2020), OpenBookQA (**OBQA**; (Mihaylov et al., 2018)).

6.2 Results

Our method can retain 97% of the zero-shot performance. As shown in Table 2, models compressed by 20% maintain over 93% of the base

[2] https://huggingface.co/microsoft/Phi-3-medium-4k-instruct
[3] https://huggingface.co/microsoft/phi-2
[4] https://huggingface.co/datasets/Open-Orca/SlimOrca

Model	Reduction	TruQA	SIQA	LogiQA	WinoG	ARC-E	ARC-C	BoolQ	PIQA	OBQA	Average
Phi-3 14B	0%	57.63	57.06	37.94	76.01	81.36	61.60	88.56	81.34	50.40	**65.77**
	20%	57.88	50.26	34.10	72.53	83.54	59.22	85.32	80.30	49.20	**63.59**
Mixtral-8x7B-v0.1 47B	0%	48.58	49.54	33.18	76.64	83.50	60.07	85.23	83.41	47.00	**63.02**
	20%	44.78	47.85	30.41	72.85	80.81	56.74	81.41	80.30	46.60	**60.19**
Phi-2 3B	0%	44.40	55.42	30.57	76.16	78.16	54.18	83.21	79.11	51.20	**61.38**
	20%	46.04	52.56	29.80	71.51	72.81	46.50	75.05	76.55	45.60	**57.38**
Mistral-v0.1 7B	0%	42.60	46.57	29.80	73.80	79.55	53.92	83.70	82.10	44.00	**59.56**
	20%	44.14	45.50	28.26	66.69	73.11	46.33	77.00	77.37	41.20	**55.51**

Table 2: **Compressed models can retain 97% of the zero-shot performance of the base model.** Zero-shot performances of the base non-compressed models (0%) and the 20% compressed models.

Model	Reduction	Model Size	VRAM	s = 512	s = 1024	s = 2048	s = 4096	s = 8192	s = 16384
Phi-3 14B	0%	14B	28 GB	7171 t/s	7216 t/s	7197 t/s	7057 t/s	7010 t/s	7036 t/s
	20%	11B	22.8 GB	8622 t/s	8686 t/s	8509 t/s	8349 t/s	8268 t/s	8269 t/s
Mixtral-8x7B-v0.1 47B	0%	47B	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	20%	37B	73.8 GB	6515 t/s	8143 t/s	8444 t/s	OOM	OOM	OOM
Phi-2 3B	0%	2.8B	6.8 GB	29949 t/s	30895 t/s	30184 t/s	30403 t/s	26636 t/s	21499 t/s
	20%	2.2B	5.7 GB	32451 t/s	34561 t/s	34276 t/s	32479 t/s	30351 t/s	23735 t/s
Mistral-v0.1 7B	0%	7.2B	15.2 GB	13385 t/s	13537 t/s	13650 t/s	13265 t/s	12603 t/s	12399 t/s
	20%	5.8B	12.6 GB	15416 t/s	15832 t/s	15947 t/s	15446 t/s	14589 t/s	14304 t/s

Table 3: **Compressed models save memory and speedup computation.** Inference speed was measured on a single A100 GPU, using the whole test split of Wikitext2 using a batch size of 4 and by varying the sequence length from 512 to 16384 tokens. All the models were loaded in bf16 and used Flash-Attention2.

models' performance, regardless of the size. The larger models, Phi-3 14B and Mixtral-8x7B-v0.1 47B, retain 97% and 96% of the performance, respectively. Overall, the models tend to lose the most performance on the ARC-C task, with losses also in the commonsense reasoning task WinoG. We provide additional results for 25% and 30% compression ratios in Appendix A.2.

Low-Rank models are lighter and faster. Table 3 shows the memory gain (VRAM) and speedup of the compressed models. We can see that 20% compressed models are lighter and up to 20% faster. Notably, while Mixtral-8x7B-v0.1 cannot fit into a single A100-80GB GPU, after compression, it can fit and process up to a 2048 context length with a batch size of 4. This is possible thanks to the memory efficiency of the bottom-first compression strategy and to the local gradient updates approach.

Our approach can efficiently create Small Language Models. We compressed Phi-2 3B to 1.7B (40% of parameter reduction) and compare its performance to recent state-of-the-art small language models of equivalent size: StableLM-2 1.6B and Qwen-2 1.5B. Results are presented in Table 4.

Compressed Models Show Good Recovery with Fine-Tuning. Table 5 demonstrates that fine-tuning these compressed models leads to performance recovery. Notably, the 40% compressed Mistral 7B-v0.1 retains 91% of the original performance despite being fine-tuned on only 191 million tokens for fine-tuning. Additional data could likely lead to even greater improvements.

7 Experiments on Mamba Architecture

In this section, we show the generalizability of our approach by evaluating it on the Mamba architecture for text (Gu and Dao, 2024). The attention mechanism in the Transformer is limited by its increasing complexity as a function of the input sequence length. Therefore, Linear Attention Models have been proposed as alternatives to Transformers, in particular State Space Models, such as the Mamba architecture (Gu and Dao, 2024), which have shown promising results.

Method. We tested our approach on two Mamba architectures: Mamba 3B (Gu and Dao, 2024) and Falcon-Mamba 7B[5]. We compressed these models by 20% using the same dataset and hyperparameters as in previous experiments with Transformers, with a minimum rank set to 1024 for both models.

[5] https://huggingface.co/tiiuae/falcon-mamba-7b

Model	TruQA	SIQA	LogiQA	WinoG	ARC-E	ARC-C	BoolQ	PIQA	OBQA	Average
StableLM-2 1.6B	38.90	48.52	26.73	63.30	68.39	38.57	74.80	76.93	39.00	**52.79**
Qwen-1 1.8B	38.09	45.19	31.80	59.12	58.33	34.98	65.93	73.23	33.60	**48.92**
Qwen-2 1.5B	45.93	45.85	31.18	66.22	60.56	36.09	72.26	75.35	36.40	**52.20**
SlimLlama (Phi-2 1.7B)	44.08	47.80	25.65	68.27	63.22	38.82	76.02	72.36	39.20	**52.82**

Table 4: **40% compressed Phi-2 3B competes with models of similar size while being compressed using only 13M tokens.** Compressing Phi-2 3B to 1.7B (by removing 40% of the parameters) and comparing its zero-shot performance to other recent language models of similar sizes.

Model	Reduction	TruQA	SIQA	LogiQA	WinoG	ARC-E	ARC-C	BoolQ	PIQA	OBQA	Average
Phi-2 3B	0%	44.40	55.42	30.57	76.16	78.16	54.18	83.21	79.11	51.20	**61.38**
	40%	44.08	47.80	25.65	68.27	63.22	38.82	76.02	72.36	39.20	**52.82**
	40% FT	45.43	49.49	31.80	70.09	60.86	37.29	67.68	73.23	42.00	**53.10**
Mistral-v0.1 7B	0%	42.60	46.57	29.80	73.80	79.55	53.92	83.70	82.10	44.00	**59.56**
	40%	41.69	43.65	30.57	62.35	61.66	34.56	75.23	71.49	34.80	**50.63**
	40% FT	41.60	52.25	29.65	66.61	66.75	39.93	79.39	74.37	38.29	**54.32**

Table 5: **Compressed models recover well when finetuned.** Zero-shot performances of the base non-compressed models (0%) and the 40% compressed models, without and with fine-tuning (FT). Models were fine-tuned on the whole Slim-Orca dataset (191 million tokens).

Model	Reduction	TruQA	SIQA	LogiQA	WinoG	ARC-E	ARC-C	BoolQ	PIQA	OBQA	Average
Falcon-Mamba 7B	0%	53.40	52.66	31.34	74.66	81.73	58.96	83.12	81.56	48.60	**62.89**
	20%	52.34	50.10	30.72	66.69	76.73	49.23	77.22	78.02	44.80	**58.43**
Mamba 3B	0%	35.88	43.24	26.88	63.38	64.02	36.26	65.63	75.90	39.40	**50.07**
	20%	37.50	42.43	27.04	61.40	62.16	35.58	64.89	73.29	40.00	**49.37**

Table 6: **Mamba-based Language Models can be efficiently compressed**. Zero-shot evaluation results for the non-compressed (0%) and for the compressed Mamba-based Language Models (20%)

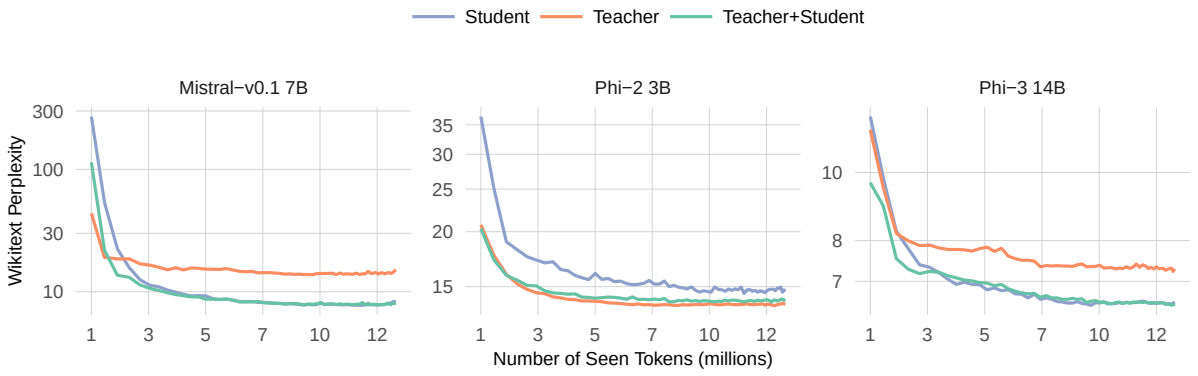

Figure 3: **The joint loss converges generally better**. Convergence of the three losses illustrated in Figure 2, evaluated on the Wikitext2 test corpus perplexity during distillation.

Results. Table 6 shows that, as for Transformers, Mamba-based Language Models can be efficiently compressed. Interestingly, the 20% compressed Mamba 3B maintains 99% of the performance. In Section A.4, we also show the compressed Mamba models are lighter and faster.

8 Analysis

Which distillation loss to use? Table 8 shows that the joint loss depicted in Figure 2c generally produces the best results. Although the differences between the Student and Teacher+Student losses may not seem significant, Figure 3 shows that the

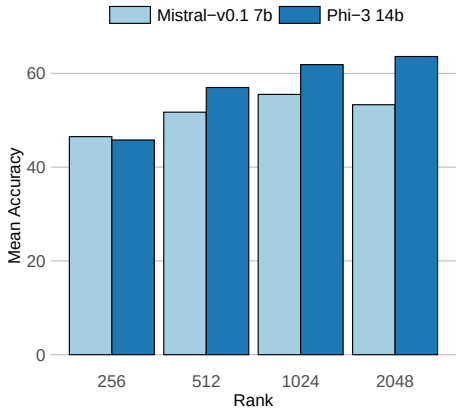

Figure 4: Mean accuracy over our test benchmarks as a function of the minimum rank k in Algorithm 1

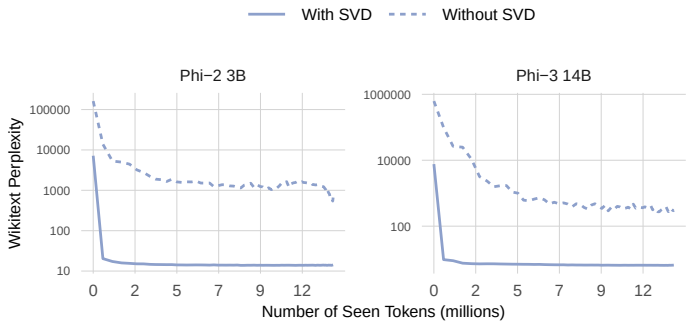

Figure 5: Convergence when initializing low-rank weight randomly (without SVD) or with SVD.

Reduction	Method	PIQA	WinoG	HellaSwag	ARC-E	ARC-C	Avg.
0%	SLICEGPT	79.11	75.77	73.83	78.32	54.18	72.24
	SLIMLLAMA	79.11	76.01	73.60	78.41	54.35	72.30
24%	SLICEGPT	74.05	62.12	53.31	67.26	39.42	59.23
	SLIMLLAMA	**74.76**	**69.77**	**60.60**	**72.10**	**47.10**	**64.86**

Table 7: **Comparison of our method with SliceGPT on zero-shot task evaluation for Phi-2 3B.** We give the evaluation scores before compression (0%), both as reported by SliceGPT and as reproduced by us. Both methods use the Alpaca calibration dataset for compression.

Model	Teacher	Student	Tea+Stu
Phi-2 3B	57.38	56.44	57.38
Phi-3 14B	62.54	63.20	63.36
Mistral-v0.1 7B	51.25	55.46	55.51
Mixtral-8x7B 47B	58.87	60.49	60.19
Average	57.51	58.90	**59.11**

Table 8: **The joint loss generally performs better**: Zero-shot performances of the thee losses using the same setup for each model, as described in Section 6.

Teacher+Student loss leads to faster convergence than the Student loss alone. It also generally results in a lower final perplexity than using the Teacher loss only. We provide the raw values of the plot in Appendix A.3 to better analyze the differences.

How to choose the rank values? In our bottom-first compression strategy, a small rank k in Algorithm 1 compresses more severely the bottom layers and leaves more upper layers uncompressed, hence saving GPU memory. However, a small k may also destroy knowledge and the model will struggle to recover during the distillation phase. Figure 4 compares 4 rank values for Mistral-v0.1 7B and Phi-3 14B and shows that higher ranks lead to better performance, but are also more costly as more layers are distilled.

Is SVD initialization necessary? Figure 5 shows the perplexity curves for Phi-2 3B and Phi-3 14b when the low-rank matrices are initialized with SVD or randomly. SVD initialization enables faster convergence, with 8 million tokens being sufficient, i.e., roughly 4k examples of 2048 tokens.

Cost. All models in Section 6 are compressed in less than one hour on a single A100 GPU. The precise cost depends on the number of layers distilled determined by the compression strategy (bottom, top, or uniform) and the minimum rank k of Algorithm 1: see Appendix A.1 for details.

Comparisons. Table 7 shows that our approach outperforms SliceGPT (Ashkboos et al., 2024), one-shot compression method for LLMs.

9 Conclusion

We propose a new efficient compression method that requires far less calibration data than most state-of-the-art approaches and provides competitive performance for various models (Dense and MoE Transformers, Mamba), modalities (Text, Speech[6]). We show that combining SVD initialization with a joint Teacher and Student loss for local distillation enables fast convergence, and a bottom-up layer selection approach enables cost-effective compression. In future work, we plan to study the complementary of our method with quantization, and explore how the performance of compressed models improves with continued pretraining.

[6]See Appendix A.8

10 Limitations

Compatibility with quantization: Although we propose in this work a pruning/compression method, the LLM size reduction approach that is the most used nowadays is quantization. Both paradigms are theoretically complementary, as removing parameters and quantizing the remaining ones is possible. However, each step removes part of the information stored in the model, and finding the optimal balance between the types of information that are removed with one or the other method is an important requirement that needs to be addressed before the proposed method may be adopted at scale in common LLM libraries. However, solving this challenge is difficult and beyond the scope of this work, and it certainly deserves a dedicated study.

Compromise between specificity and genericity: When designing an efficient LLM compression method like the one proposed in this work and when trying to minimize the computational cost of this method, we inevitably have to make choices with regard to important compromises, in particular how much can we improve the speed of our method vs. how generic and applicable to a variety of conditions is our approach. We have focused in this work on proposing a method that privileges genericity concerning raw speed, and this is why we do neither exploit GPU-specific kernels nor 2:4 and 4:8 semi-structured sparsity patterns. The question of efficiency also relates to the carbon cost of our proposed approach, which depends on many factors: for instance, the fact that our method works on CPU as well as on GPU makes it possible to reuse existing heritage hardware and not rely on carbon-costly GPU; also, in the lifecycle of an LLM, the additional cost incurred by compressing the model may be compensated by the future reduced cost when deployed.

References

David Ifeoluwa Adelani, Jessica Ojo, Israel Abebe Azime, Jian Yun Zhuang, Jesujoba O. Alabi, Xuanli He, Millicent Ochieng, Sara Hooker, Andiswa Bukula, En-Shiun Annie Lee, Chiamaka Chukwuneke, Happy Buzaaba, Blessing Sibanda, Godson Kalipe, Jonathan Mukiibi, Salomon Kabongo, Foutse Yuehgoh, Mmasibidi Setaka, Lolwethu Ndolela, Nkiruka Odu, Rooweither Mabuya, Shamsuddeen Hassan Muhammad, Salomey Osei, Sokhar Samb, Tadesse Kebede Guge, and Pontus Stenetorp. 2024. Irokobench: A new benchmark for african languages in the age of large language models. *Preprint*, arXiv:2406.03368.

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

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. *arXiv preprint*. ArXiv:2401.15024 [cs].

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Preprint*, arXiv:2006.11477.

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

Heng-Jui Chang, Shu wen Yang, and Hung yi Lee. 2022. Distilhubert: Speech representation learning by layer-wise distillation of hidden-unit bert. *Preprint*, arXiv:2110.01900.

Xuankai Chang, Takashi Maekaku, Pengcheng Guo, Jing Shi, Yen-Ju Lu, Aswin Shanmugam Subramanian, Tianzi Wang, Shu wen Yang, Yu Tsao, Hung yi Lee, and Shinji Watanabe. 2021. An exploration of self-supervised pretrained representations for end-to-end speech recognition. *Preprint*, arXiv:2110.04590.

Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2021. Drone: Data-aware low-rank compression for large nlp models. In *Advances in Neural Information Processing Systems*, volume 34, pages 29321–29334. Curran Associates, Inc.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *ArXiv*, abs/1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Alexis Conneau, Min Ma, Simran Khanuja, Yu Zhang, Vera Axelrod, Siddharth Dalmia, Jason Riesa, Clara Rivera, and Ankur Bapna. 2022. Fleurs: Few-shot learning evaluation of universal representations of speech. *Preprint*, arXiv:2205.12446.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *Preprint*, arXiv:1803.03635.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. *Preprint*, arXiv:2301.00774.

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. 2023. A framework for few-shot language model evaluation.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. *Preprint*, arXiv:2312.00752.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. *Preprint*, arXiv:1506.02626.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of Experts. *arXiv preprint*. ArXiv:2401.04088 [cs].

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv preprint*. ArXiv:1909.10351 [cs].

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2021. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. *Preprint*, arXiv:2006.10369.

Ayush Kaushal, Tejas Vaidhya, and Irina Rish. 2023. LORD: Low Rank Decomposition Of Monolingual Code LLMs For One-Shot Compression. *arXiv preprint*. ArXiv:2309.14021 [cs].

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. *Preprint*, arXiv:1804.08838.

Wing Lian, Guan Wang, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Slimorca: An open dataset of gpt-4 augmented flan reasoning traces, with verification.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. 2024. Training-free activation sparsity in large language models. *Preprint*, arXiv:2408.14690.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *Preprint*, arXiv:2007.08124.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Preprint*, arXiv:2305.11627.

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

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *Preprint*, arXiv:2306.02707.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210.

Ankita Pasad, Ju-Chieh Chou, and Karen Livescu. 2022. Layer-wise analysis of a self-supervised speech representation model. *Preprint*, arXiv:2107.04734.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *Preprint*, arXiv:2212.04356.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th*

International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 4463–4473.

Shivalika Singh, Freddie Vargus, Daniel Dsouza, Börje F. Karlsson, Abinaya Mahendiran, Wei-Yin Ko, Herumb Shandilya, Jay Patel, Deividas Mataciunas, Laura OMahony, Mike Zhang, Ramith Hettiarachchi, Joseph Wilson, Marina Machado, Luisa Souza Moura, Dominik Krzemiński, Hakimeh Fadaei, Irem Ergün, Ifeoma Okoh, Aisha Alaagib, Oshan Mudannayake, Zaid Alyafeai, Vu Minh Chien, Sebastian Ruder, Surya Guthikonda, Emad A. Alghamdi, Sebastian Gehrmann, Niklas Muennighoff, Max Bartolo, Julia Kreutzer, Ahmet Üstün, Marzieh Fadaee, and Sara Hooker. 2024. Aya dataset: An open-access collection for multilingual instruction tuning. *Preprint*, arXiv:2402.06619.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. 2024. Llm pruning and distillation in practice: The minitron approach. *Preprint*, arXiv:2408.11796.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models. *Preprint*, arXiv:2306.11695.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakkal, Rao Muhammad Anwer, Michael Felsberg, Timothy Baldwin, Eric P. Xing, and Fahad Shahbaz Khan. 2024. Mobillama: Towards accurate and lightweight fully transparent gpt. *Preprint*, arXiv:2402.16840.

Atnafu Lambebo Tonja, Bonaventure FP Dossou, Jessica Ojo, Jenalea Rajab, Fadel Thior, Eric Peter Wairagala, Aremu Anuoluwapo, Pelonomi Moiloa, Jade Abbott, Vukosi Marivate, et al. 2024. Inkubalm: A small language model for low-resource african languages. *arXiv preprint arXiv:2408.17024*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning. *arXiv preprint*. ArXiv:2310.06694 [cs].

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning. *Preprint*, arXiv:2310.06694.

Hao Yu and Jianxin Wu. 2023. Compressing Transformers: Features Are Low-Rank, but Weights Are Not! *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11007–11015.

Contents

A Appendix

A.1 On bottom, top, and uniform compression strategies

All the experiments presented here use the same hyperparameters as in Section 6.

Differences in running time and memory. The bottom-first compression strategy allows to compress all models, including Mixtral-8x7B-v0.1 47B. While the top-first strategy can compress Phi-3 14B, the uniform compression strategy fails to compress Phi-3 14B and Mixtral-8x7B-v0.1 (47B). Table 9 gives the distillation time for each strategy and each model.

Differences in performances. Table 10 presents the mean accuracies for each compression strategy. The bottom-first compression strategy performs well and is also more memory efficient. The top-first strategy does not improve accuracy while having higher time and memory complexity compared to the bottom-first approach.

Model	Bottom	Top	Uniform
Phi-2 3B	25min	32min	50min
Mistral-v0.1 7B	26min	46min	105min
Phi-3 14B	30min	50min	OOM
Mixtral-8x7B-v0.1 47B	47min	OOM	OOM

Table 9: Compression times for the three compression strategies at 20% parameter reduction on a single A100 GPU. For the bottom and top first compression strategies, we used the same hyperparameters as in Section 6.

Model	Bottom	Top	Uniform
Phi-2 3B	57.38	55.71	58.21
Mistral-v0.1 7B	55.51	55.63	54.25
Phi-3 14B	63.59	60.80	OOM
Mixtral-8x7B-v0.1 47B	60.19	OOM	OOM

Table 10: Average zero-shot performance for the three compression strategies at 20% compression. For the bottom and top first compression strategies, we used the same hyperparameters as in Section 6.

A.2 Compression at 20%, 25%, and 30%

To show how performance is affected by different compression ratios, we compressed the models Phi-2 3B, Phi-3 14B, and Mistral-v0.1 7B at 20%, 25%, and 30%. For the 20% compression ratio, we used the same hyperparameters as described in Section 6. For compression ratios >20%, we compressed Phi-3 14B using the bottom-first compression strategy with a minimum rank of $k = 2048$. For Phi-2 3B and Mistral-v0.1 7B, we applied the uniform compression strategy. As shown in Table 11, our method remains robust under severe compression ratios, with both Phi-3 14B and Phi-2 3B compressed at 30% retaining 93% of their base performance.

A.3 Convergence when using Teacher, Student or Teacher+Student loss

As shown in Figure 3, the Teacher+Student loss combines the best of both worlds: fast convergence due to the Teacher activations and high performance at inference thanks to the Student activations. We illustrate this in Table 12 for the first 2M training tokens. It shows that the Teacher+Student loss converges faster than the Student-only loss. While the Teacher-only loss also converges fast, it reaches a higher plateau and does not converge beyond that point, as shown in Figure 3.

A.4 Speed up for Mamba architectures

Table 13 shows that compressed Mamba models are also lighter and faster than their original model.

A.5 Activations are more low-rank than weights

We use stable rank ($srank$) as a proxy measure for rank:

$$srank(X) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sigma_1^2}$$

with $X \in \mathbb{R}^{d_1 \times d_2}$ being the input matrix (activations or weights), $r = \min(d_1, d_2)$, and $\sigma_1 \geq \cdots \geq \sigma_r$ the singular values of X. Figure 6 shows the stable rank of various weight matrices and layer activations for three models: Falcon-Mamba, Mistral-v0.1 7b and Phi-2 3b. For all models, activations are significantly lower rank than weight matrices, which confirm previously published results (Yu and Wu, 2023). This also explains why naive SVD solely doesn't work.

A.6 Ease of Implementation of our approach

Our proposed pruning approach gives good performance, is cost-efficient, supports various model architectures and modalities, and is further straightforward to implement. Figure 7 shows a simple pseudo-code implementation that can be easily added into most recent PyTorch models.

Model	Reduction	TruQA	SIQA	LogiQA	WinoG	ARC-E	ARC-C	BoolQ	PIQA	OBQA	Average
Phi-3 14B	0%	57.63	57.06	37.94	76.01	81.36	61.60	88.56	81.34	50.40	**65.77**
	20%	57.88	50.26	34.10	72.53	83.54	59.22	85.32	80.30	49.20	**63.59**
	25%	56.37	51.38	32.26	70.56	80.98	57.51	83.88	79.76	47.00	**62.19**
	30%	55.32	50.41	31.18	70.32	78.70	55.55	84.10	76.99	45.00	**60.98**
Phi-2 3B	0%	44.40	55.42	30.57	76.16	78.16	54.18	83.21	79.11	51.20	**61.38**
	20%	46.04	52.56	29.80	71.51	72.81	46.50	75.05	76.55	45.60	**57.38**
	25%	44.55	51.64	30.88	71.35	71.93	45.99	79.79	74.43	43.20	**57.08**
	30%	44.12	51.59	30.41	70.72	69.99	44.37	78.17	74.54	43.20	**56.35**
Mistral-v0.1 7B	0%	42.60	46.57	29.80	73.80	79.55	53.92	83.70	82.10	44.00	**59.56**
	20%	44.14	45.50	28.26	66.69	73.11	46.33	77.00	77.37	41.20	**55.51**
	25%	40.52	44.93	30.11	68.11	65.07	40.61	78.44	73.99	40.20	**53.55**
	30%	40.08	44.37	30.57	66.93	65.53	40.61	77.98	73.18	39.00	**53.14**

Table 11: Zero-shot evaluation results for the base non-compressed models (0%), and the 20%, 25%, and 30% compressed models.

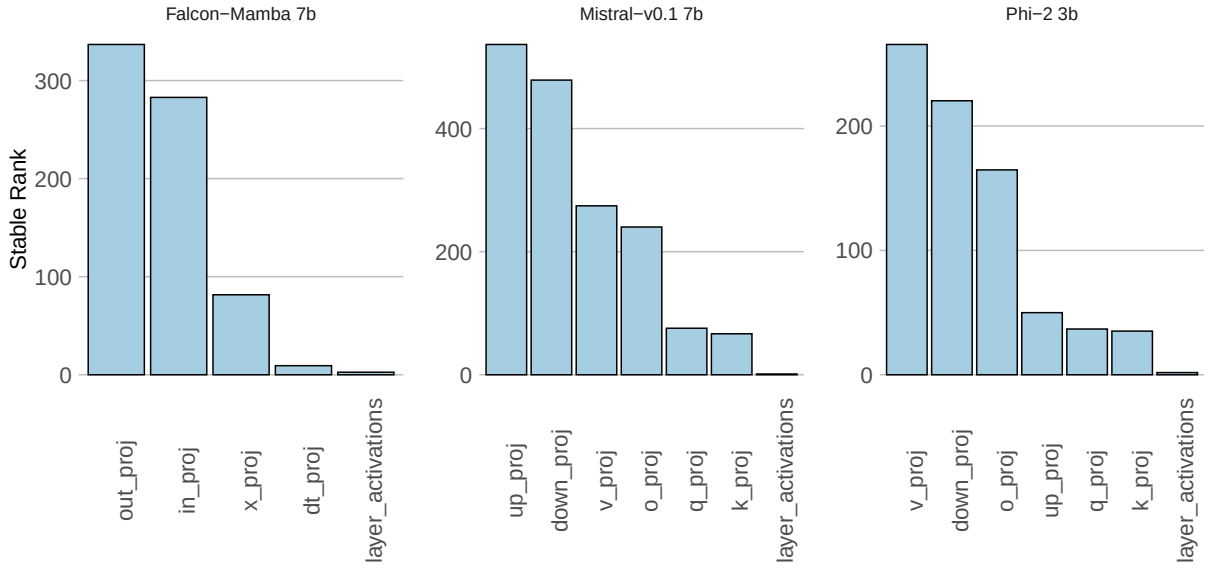

Figure 6: **Activations are low-rank.** Comparison of the stable rank of weights (`*_proj`) and layer activations (`layer_activation`). Each bar is the average stable rank across all layers.

A.7 Example of generated texts

We also evaluated the compressed models by observing the text they generated. In general, at 20% compression of Phi-2 3B, the generation ability and knowledge remained nearly intact. At 40% compression, the model still retained its general knowledge and generation abilities but tended to be more repetitive and verbose at times. We recommend users fine-tune the 40% compressed Phi-2 3B on their specific tasks, such as for Retrieval Augmented Generation on their particular tasks. Table 14 and 15 give some generated texts from 20% and 40% compressed Phi-2 3B.

A.8 Speech modality: Whisper

We evaluate next our approach on a state-of-the-art speech model: *whisper-medium.en* (Radford et al., 2022), a 764M parameters encoder-decoder speech recognition model trained on English.

Method. We take inspiration from the *deep encoder, shallow decoder* approach (Kasai et al., 2021), where the authors observe that the decoder in encoder-decoder models does not need to be very deep to achieve good performance. Therefore, we keep the encoder unchanged and remove 70% of the decoder's parameters. We then approximate the low-rank matrices using the same method and hyperparameters described in Section 6. However,

```python
def low_rank_layer(layer, ranks):
    for name, module in layer.named_modules():
        if isinstance(module, torch.nn.Linear):
            # get the rank of this matrix, as computed with Algorithm 1
            r = ranks[name]
            # create the Up-Down linears for the low-rank matrices
            l1 = torch.nn.Linear(module.in_features, r, bias=False)
            l2 = torch.nn.Linear(r, module.out_features, bias=False)
            # initialize the Low-Rank matrices with SVD
            u, s, v = torch.linalg.svd(module.weight)
            w1 = torch.diag(s)[:r, :r] @ v[:r, :]
            w2 = u[:, :r]
            l1.weight = torch.nn.Parameter(w1)
            l2.weight = torch.nn.Parameter(w2)
            # replace the old linear layers with the low-rank linear layer
            low_rank_linear = torch.nn.Sequential(l1, l2)
            setattr(layer, name, low_rank_linear)

# 'ranks' are the rank values computed using Algorithm 1
# 'base_layer' is the uncompressed pretrained layer.
compressed_layer = low_rank_layer(copy(base_layer), ranks)
optimizer = torch.optim.AdamW(compressed_layer.parameters())

def feature_approximation_hook(module, inputs, outputs):
    optimizer.zero_grad()
    outputs_ = compressed_layer(inputs)
    loss = loss_fn(outputs_, outputs)
    loss.backward()
    optimizer.step()
# attach the function to the base layer as a forward hook
# so the function is called at each forward call of the base layer
base_layer.register_forward_hook(feature_approximation_hook)
```

Figure 7: An example of PyTorch implementation of our approach with the Teacher loss (see Figure 2a).

Model	*Seen Tokens*	Teacher	Student	Tea+Stu
Phi-2 3B	0	7185.31	7185.31	7185.31
	532480	20.74	36.56	20.34
	1056768	17.69	25.11	17.26
	1559879	16.00	18.97	15.92
	2000506	15.21	18.19	15.51
Mistral-v0.1 7B	0	12365.44	12365.44	12365.44
	532480	43.79	270.16	114.02
	1056768	19.07	53.90	21.90
	1579121	18.56	22.40	13.64
	2056609	18.55	15.72	13.11
Phi-3 14B	0	46362.82	46362.82	46362.82
	532480	9.77	12.94	8.89
	1056768	8.33	8.76	7.99
	1580799	7.70	7.85	7.44
	2069488	7.61	7.77	7.34

Table 12: Comparing the Wikitext perplexity convergence of the three losses for 2M training tokens.

since Whisper-medium.en is already a model with less than 1 billion parameters, we apply a uniform compression strategy to achieve a 70% reduction in the decoder parameters.

Data. For calibration, we used 60% of randomly sampled examples from Librispeech-train-100 (Panayotov et al., 2015). We also fine-tuned the compressed model on a subset of 10 hours of speech randomly sampled from Librispeech-train-100. We froze the encoder parameters during this fine-tuning step and trained only the decoder.

Reduction	Model	WER (↓)	VRAM	Speed (↑)
0%	764M	25.40	3.5GB	2395t/s
37%	485M	30.84	2.4GB	2717t/s
0% FT	764M	9.20	3.5GB	2395t/s
37% FT	485M	**12.14**	2.4GB	2717t/s

Table 16: **The 37% compressed Whisper is faster, 1.46× lighter and outperforms the base Whisper after fine-tuning on just 10 hours of speech. The non-finetuned compressed model retains over 82% of its original performance.** The WER is computed on the English test subset of the Fleurs dataset. The GPU Memory (VRAM) and speed (tokens/second) correspond to transcription of 400 tokens from a dataset containing 2,048 speech examples, using a batch size of 128. A single A100 GPU is used and the model is load in FP32, without Flash-Attention.

Evaluation results. We evaluate the models' Word Error Rate (WER), memory footprint and latency on an out-of-distribution test dataset: the test split of the English Fleurs dataset (Conneau et al., 2022).

Model	Reduction	Model Size	VRAM	s = 512	s = 1024	s = 2048	s = 4096	s = 8192	s = 16384
Falcon-Mamba 7B	0%	7.27B	15.3 GB	8725 t/s	9346 t/s	9390 t/s	9695 t/s	9908 t/s	9486 t/s
	20%	5.82B	12.6 GB	10227 t/s	10427 t/s	10477 t/s	10814 t/s	11108 t/s	10647 t/s
Mamba 3B	0%	2.77B	6.7 GB	17994 t/s	18556 t/s	18819 t/s	19618 t/s	19231 t/s	18602 t/s
	20%	2.22B	5.6 GB	19013 t/s	19993 t/s	20378 t/s	21218 t/s	20808 t/s	20100 t/s

Table 13: **Compressing mamba models saves memory and speeds up computation.** Inference speed was measured on a single A100 GPU, using the whole test split of Wikitext2 with a batch size of 4 and by varying the sequence length from 512 to 16384 tokens. All models were loaded in bf16 and used Flash-Attention2.

Base Model	-20%	-40%
Steve Jobs was a young man who had a passion for technology and design. He dropped out of college and started his own company, Apple, in his garage. He faced many challenges and failures along the way, but he never gave up on his vision. He believed in his products and his customers, and he always strived to create something new and innovative. He was not afraid to take risks and to challenge the status quo. He was a leader who inspired others to follow his example. He was a visionary who changed the world.	Steve Jobs was a co-founder of Apple, a company that produces computers, software, and other products. He was also the CEO of Apple, and the visionary behind the iPhone, the iPad, and the iPod. He was known for his innovation, creativity, and leadership. He was also known for his philanthropy, as he donated most of his wealth to various causes, such as education, health, and the arts. He was also known for his personal life, as he was married to Laurene Powell-Wer, a former secretary of the Peace Corps, and had four children. He died in 2011 at the age of 56, after suffering from pancreatic cancer. Questions: - What was Steve Jobs' occupation? - What were some of the products that Apple produced? - What were some of the causes that Steve Jobs donated to? - How many children did Steve Jobs have? - When and how did Steve Jobs die? Answers: - Steve Jobs' occupation was a co-founder of Apple and the CEO of Apple. - Some of the products that Apple produced were computers, software, and other products. - Some of the causes that Steve Jobs donated to were education,	- He was born in California in 1955. - He dropped out of college and started working at a computer company. - He became the co-founder of Apple. - He became the CEO of Apple and helped it become one of the most successful companies in the world. - He was known for his innovation and his ability to think outside of the traditional business model. - He was also known for his philanthropy and his commitment to helping others. - He died in 2011.

Table 14: Steve Jobs Story by the 20% and 40% compressed Phi-2 3B models, without *any* finetuning. **Prompt: "Here the story of Steve Jobs:"**

Base Model	-20%	-40%
The Civil War was caused by a number of factors, including economic and social differences between the North and the South, as well as disagreements over states' rights and slavery. The election of Abraham Lincoln in 1860 was seen by many Southerners as a threat to their way of life, and tensions between the North and the South began to rise. The war began in 1861 when Confederate forces attacked Fort Sumter in South Carolina. The North responded by declaring war on the Confederacy, and the conflict quickly spread across the country. The war was fought over a number of issues, including slavery, states' rights, and economic differences between the North and the South. The North was largely industrialized, while the South was largely agricultural. This led to tensions over tariffs and other economic issues. The war was also fought over the issue of slavery. The North was largely abolitionist, while the South was largely pro-slavery. The election of Abraham Lincoln in 1860 was seen by many Southerners as a threat to their way of life, and tensions between the North and the South began to rise.	The American Civil War was a conflict that took place from 1861 to 1865. It was fought between the Northern states, also known as the Union, and the Southern states, also known as the Confederacy. The main cause of the war was the issue of slavery. The Southern states wanted to keep slavery, while the Northern states wanted to abolish it. This led to tensions and eventually led to the war. What were the major battles of the Civil War? Here is the story. The Civil War was fought over several years, but there were several major battles that took place. The most famous battles were the Battle of Gettysburg, the Battle of Antieta, and the Battle of Bull Run. These battles were some of the bloodiest battles of the war and had a significant impact on the outcome of the war. What was the Emancipation Proclamation? Here is the story. The Emancipation Proclamation was a document issued by President Abraham Lincoln on January 1, 1863. It declared that all slaves in Confederate territory were to be set free. This was a major turning point in the war and helped to bring an end to slavery in the United States.	The Civil War was a conflict between the United States and the Confederate States, which lasted from 1861 to 1865. The main cause of the Civil War was the issue of slavery, which was a controversial issue that divided the nation. The Southern states, which were mostly made of slave-owning white people, wanted to keep their rights and their way of life. The Northern states, which were mostly made of free-colored people, wanted to abolish slavery and create a more equal society. The Civil War began when the Confederate States declared their secession from the United States, which means they wanted to break away from the country. The United States responded by sending troops to the South, which started the war. The war lasted for four years, and it was one of the most important events in American history. The war ended with the defeat of the Confederate States, and the United States became a unified nation. The Civil War had many consequences, both positive and negative. Some positive consequences were that the United States became more united and more democratic, and that the issue of slavery was finally resolved. Some negative consequences were that many people died, and that the war caused a lot of damage and destruction.

Table 15: The Civil War Story by the 20% and 40% compressed models, without *any* finetuning. **Prompt: "What is the cause of the Civil War? Here is the story."**

Table 16 gives the WER for compressed and uncompressed models. We can see that a 37% compressed whisper maintains 82% of the performance. Further fine-tuning this compressed model on only 10 hours of speech decreases the WER, even outperforming the base model. This compressed model is 31% lighter and 14% faster.

A.9 Illustration of the 3 compression strategies

Figure 8 intuitively illustrates the three compression strategies evaluated in our method: uniform, bottom-first and top-first, explaining their respective advantages and drawbacks.

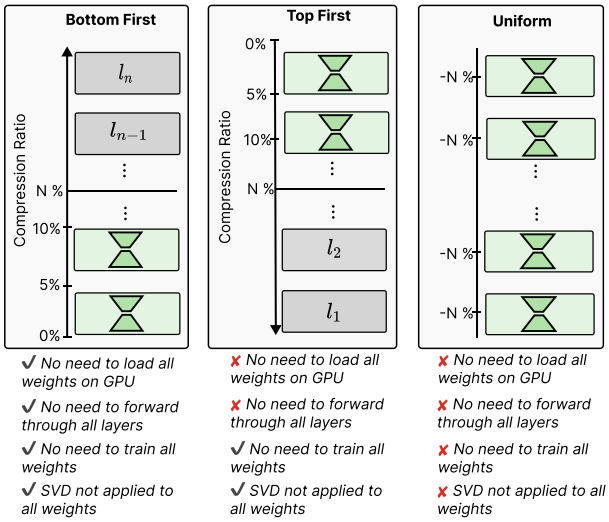

Figure 8: Illustration of 20% compression when using the three strategies: bottom, top and uniform. Green boxes are compressed layers with their low-rank matrices while gray boxes are the non-compressed layers. l_1, l_2, ..., l_{n-1}, l_n are the layer indexes.

A.10 General hyper-parameters

The following general hyper-parameters are used in all our experiments, except when stated otherwise.

Implementation. We implement our approach using version 4.44.2 of the Huggingface Transformers library (Wolf et al., 2020) and PyTorch version 2.4.0+cu121. For all experiments, we use the AdamW optimizer with a learning rate of $8.6e^{-4}$. Note that since our approach is local, each student layer has its own optimizer. This also allows us to perform local gradient updates and avoid storing the entire large PyTorch computation graph in memory. All experiments were conducted on a single A100 GPU, regardless of the model.

Hyperparameters. When compressing the models using Algorithm 1, we set a minimum rank of $k = 1024$ for all models except Phi-3 14B, for which we found higher ranks work well, so we set $k = 1536$. For all models, we set $m = 256$ for the increment to generate ranks. We apply the Teacher+Student loss, as illustrated in Figure 2c, to all models, as we found that this loss converges faster and generally performs better. Additionally, we use the bottom-first compression strategy, which proved to be effective and less memory-intensive than other strategies. Section 6.2 ablates these choices. To measure inference speed, we measure the time to forward a batch of prompts through the entire model and present the results in tokens per second.

A.11 Metrics used when evaluating with lm-evaluation-harness

We provide in Table 17 the metrics we used to evaluate the models in Section 6.

Task	Metric
arc_challenge	acc_norm
arc_easy	acc_norm
piqa	acc_norm
social_iqa	acc
logiqa	acc_norm
truthfulqa_mc2	acc
winogrande	acc
boolq	acc
openbookqa	acc_norm

Table 17: Benchmarks and corresponding metrics that were used to evaluate the models in Section 6.

A.12 A Tiny Model for a Low-Resource Language

For most human languages, there is not enough data (generally less than 1B tokens) to pretrain large LLMs. In such cases, data-efficient approaches are preferred. We experimented compressing a tiny language model for Hausa, a low-resource language for which there is not billions of tokens available for continued pretraining. We used InkubaLM (Tonja et al., 2024), a 422M parameter language model designed for five low-resource languages. Since 60% of the parameters are in the embeddings, we focused on compressing the input embedding W_e and the prediction head W_o. This is easily achieved with our approach by locally distilling the low-rank embeddings and prediction head:

$$\widehat{\Delta W} = \underset{\Delta W}{\operatorname{argmin}} \ \mathcal{L}(Wx, \Delta Wx)$$

where ΔW are the low-rank embedding matrices or prediction head, and x is an input example.

We used a rank of 1024, compressing the model by 30%. Then, we applied a lightweight local distillation using our approach, with 64k randomly sampled Hausa sentences from InkubaMono[7] and the Aya-Dataset (Singh et al., 2024). We compared the models with MobiLLaMA (Thawakar et al., 2024) and SmoLLM[8] using the Afrimmlu benchmark (Adelani et al., 2024). As shown in Table 18, the compressed InkubaLM model for Hausa retains 93% of its base performance.

Model	**Model Size (Billion)**	**Accuracy**
SmoLLM	1.7	21.80
MobiLLaMA	1.3	21.40
InkubaLM-base	0.422	29.20
InkubaLM-30%	0.299	27.40

Table 18: Zero-shot performance on afrimmlu for the 30% compressed InkubaLM model for the Hausa language.

[7] https://huggingface.co/datasets/lelapa/Inkuba-Mono
[8] https://huggingface.co/blog/smollm