Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Mosaic: Composite projection pruning for resource-efficient LLMs

Bailey J. Eccles [a],*, Leon Wong [b], Blesson Varghese [a]

[a] School of Computer Science, University of St Andrews, Jack Cole Building, St Andrews, KY16 9SX, Fife, Scotland, United Kingdom
[b] Autonomous Networking Research & Innovation Department, Rakuten Mobile, Inc., Rakuten Crimson House, 1-14-1 Tamagawa, Setagaya-ku, Tokyo, 158-0094, Japan

ARTICLE INFO

Keywords:
Composite projection pruning
Edge computing
Model compression
Large language models
Model pruning
Resource-efficient LLM

ABSTRACT

Extensive compute and memory requirements limit the deployment of large language models (LLMs) on any hardware. Compression methods, such as pruning, can reduce model size, which in turn reduces resource requirements. State-of-the-art pruning is based on coarse-grained methods. They are time-consuming and inherently remove critical model parameters, adversely impacting the quality of the pruned model. This paper introduces projection pruning, a novel fine-grained method for pruning LLMs. In addition, LLM projection pruning is enhanced by a new approach we refer to as composite projection pruning — the synergistic combination of unstructured pruning that retains accuracy and structured pruning that reduces model size. We develop `Mosaic`, a novel system to create and deploy pruned LLMs using composite projection pruning. `Mosaic` is evaluated using a range of performance and quality metrics on multiple hardware platforms, LLMs, and datasets. `Mosaic` is 7.19× faster in producing models than existing approaches. `Mosaic` models achieve up to 84.2% lower perplexity and 31.4% higher accuracy than models obtained from coarse-grained pruning. Up to 67% faster inference and 68% lower GPU memory use is noted for `Mosaic` models.

1. Introduction

Large language models (LLMs), such as GPT-4 [1], have found applications in chatbots [2] and generating content [3]. LLMs consist of several billion [4] to hundreds of billions [3] of parameters. Consequently, training and deploying these LLMs on even state-of-the-art hardware is challenging due to their high memory and compute demands. For example, GPT-3, a 175 billion parameter LLM, is over 350 GB in size and requires 3.14×10^{23} flops to train [3] and five 80 GB Nvidia A100 GPUs to deploy. LLMs, such as ChatGPT [1] and Gemini [5], are hosted on clusters with tens of thousands of GPUs. Hence, LLMs are trained and served from cloud data centers where their resource demands can be met.

In this context, we note the following two avenues within LLM research:

(1) **Lowering resource demands for LLM inference**. Serving LLMs on relatively resource-limited edge/mobile environments is challenging [6,7], given the substantial resource requirements of LLMs that exceed the available hardware in these environments. Running LLMs suited for such environments will reduce the need to send queries outside a user-device [8] and can offer offline capabilities so that LLMs are served even under limited network connectivity [9].

(2) **Obtaining lightweight LLMs from foundation LLMs without training from scratch**. Foundation LLMs are trained on large corpora of public data [4] and can be fine-tuned for specific tasks [10]. Modern foundation LLMs, such as LLaMa-3, rival classic LLMs on various benchmarks [11]. These LLMs can fit into a single consumer-grade GPU [11]. There is potential to compress foundation LLMs for use in resource-constrained environments while maintaining accuracy.

The motivation of this article is to *create small language models (SLMs) from foundation LLMs with similar quality while running on fewer resources*.

Research on creating SLMs is based on compression methods, such as pruning [12–14]. Pruning removes an individual or a group of parameters from the model using a ranking algorithm [12]. There are two categories of pruning: Unstructured pruning refers to setting parameter values to zero [12]; quality is maintained while model size is unaffected. Structured pruning refers to removing data structures containing parameters such as attention heads [13]; this reduces model size and inference latency but at the cost of model quality. In short, existing pruning methods do not adequately balance runtime performance and LLM quality.

Our work, `Mosaic`, is positioned to address these shortcomings. Existing pruning methods focus on coarse-grained pruning at the global and layer level of the LLM (further discussed in Section 2). They prune every LLM component uniformly. This results in removing parts of the model that are critical to quality. `Mosaic` introduces novel

* Corresponding author.
E-mail addresses: bje1@st-andrews.ac.uk (B.J. Eccles), leon.wong@rakuten.com (L. Wong), blesson@st-andrews.ac.uk (B. Varghese).

https://doi.org/10.1016/j.future.2025.108056
Received 21 January 2025; Received in revised form 17 June 2025; Accepted 27 July 2025
Available online 11 August 2025

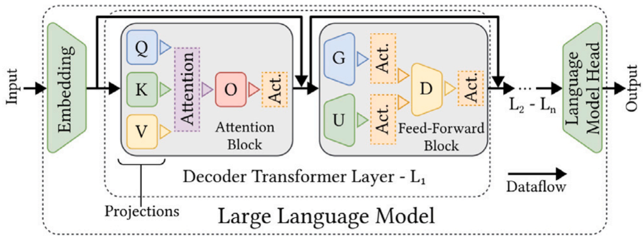

Fig. 1. Simplified overview of LLM architecture.

fine-grained pruning of LLM projections. We leverage non-uniform pruning and apply it to different components of the LLM to selectively retain critical model parts. In addition, Mosaic synergistically combines unstructured and structured pruning in LLMs for the first time to create ***'composite projection pruning'***. This pruning approach can produce compressed and resource-efficient LLMs that fit in limited memory and provide fast inference while having comparable quality to the foundation LLM. The models produced by Mosaic can be deployed on any hardware platform without requiring specific hardware/software accelerators.

Our research contributions are as follows:

(1) **Mosaic**, a novel system for compressing foundation LLMs for hardware-limited environments. Mosaic is 7.19× faster in producing compressed models than existing approaches.

(2) **LLM projection pruning**, a new method that maintains quality at higher compression levels. The method determines a projection outlier distribution to prune projections non-uniformly. Projection pruning is a performance-efficient extension of fine-grained LLM pruning explored in prior work. Mosaic models produced by projection pruning achieve up to 84.2% lower perplexity and 31.4% higher accuracy than models from uniform pruning.

(3) **Composite projection pruning**, a new approach for LLM compression that balances the benefits of unstructured pruning to maintain quality and of structured pruning to reduce model size and inference latency across various hardware platforms. For Mosaic models, up to 67% faster inference and 68% lower GPU memory usage compared to unstructured pruning while achieving up to 36× lower perplexity than structured pruning is noted.

The remainder of this article is organized as follows. Section 2 presents the background for our work. Section 3 provides an overview of the methods underpinning Mosaic and the design of the Mosaic modules. Section 4 evaluates Mosaic against relevant baselines. Section 5 presents related work. Section 6 concludes this article.

2. Background and motivation

This section considers the LLM architecture, pruning foundation LLMs to generate SLMs, the current state of LLM pruning, new opportunities in the research landscape, and the motivation for developing our approach to pruning LLMs.

2.1. LLM architecture

An LLM follows the decoder transformer architecture [15]. The architecture of foundation LLMs, such as LLaMa [4,11], and fine-tuned derivatives, such as Vicuna [16] is shown in Fig. 1. It comprises the Embedding layer, the Language Model Head, and a stack of Decoder Transformer Layers, denoted as L_1, L_2, $\cdots$ L_n. The Embedding Layer transforms natural language into tokens by representing words as numerical values. The Language Model Head translates these output tokens back into natural language.

The Decoder Transformer Layers calculate attention scores for each token and use feed-forward networks to predict output tokens. Each transformer layer comprises an Attention Block and a Feed-Forward Block. An Attention Block uses three projections — Query, Key, and Value — to calculate the relevance of each token to other tokens, generating a matrix of attention scores. The result then passes through the Output projection to the succeeding layers. A Feed-Forward Block expands the attention scores into a larger dimension. It consists of three projections — Gate, Up, and Down. The gate controls data flow to the Up projection, which expands the dimensions, and the Down projection shrinks them.

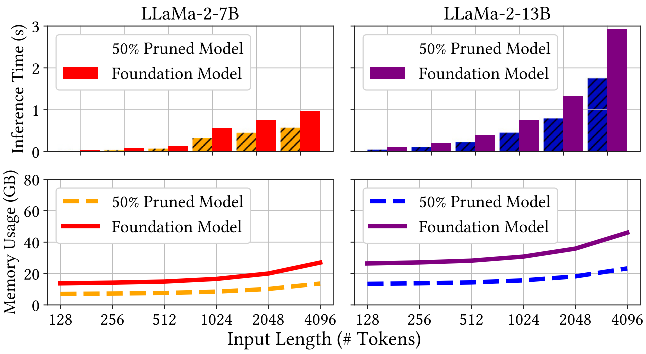

Fig. 2. GPU memory required and inference time of LLaMa-2-7B, LLaMa-2-13B, and the variants of these LLMs uniformly pruned by 50% for varying input sizes. Metrics were collected on an Nvidia A100 GPU using PyTorch 2.3.0 for inference.

Projections are the smallest units in LLMs, which contain model parameters learned during training. There are seven projections for each decoder transformer layer, which can be shortened to the set $\{Q, K, V, O, G, U, D\}$. The *parameters within the projections* are integral to the LLM. For instance, LLaMa-7B contains 7 billion parameters across 32 layers. The number of parameters determines the resources required by the LLM, which impacts model size, inference time and runtime memory use.

2.2. LLM pruning reduces resource footprint

LLMs require large volumes of input data, called tokens, to understand the context of natural language sentences [17]. Each token (t) interacts with every other token when calculating attention scores, resulting in a quadratic increase in memory use (t^2) [18]. In Fig. 2, LLaMa-2-13B requires nearly 20 GB more memory at 4096 tokens than 128 tokens, increasing memory overheads by 77% over the original model size. More tokens also increase inference time since the number of attention score calculations increases [19]. LLaMa-2-13B inference time increases 30× from 0.1 s to nearly 3 s.

LLM pruning reduces resource demand on hardware-limited devices by removing LLM parameters. Removing parameters has a *two-fold benefit*: (1) reduced model size and (2) reduced attention and activation matrix memory sizes during inference. In Fig. 2, LLM pruning of LLaMa-2-7B and LLaMa-2-13B by 50% reduces the parameter count to 3.5B and 6.5B, respectively. The dashed lines and the hatch-filled bars represent the reduced resource usage of pruned LLMs compared to foundation LLMs. Pruned LLMs are 2× smaller and 40% faster. Foundation LLMs are available in many fixed sizes, such as LLaMa-2 with a 7B, 13B, 34B and 70B variant [20]. While larger variants are targeted for multi-GPU deployments, existing LLM pruning methods are positioned to compress the smaller variants for GPUs that do not have the compute and memory resources to run them [21,22].

2.3. Opportunities in LLM pruning

Limitations of Current LLM Pruning Methods: LLM pruning requires ranking the importance of each parameter in relation to the overall model quality [23]. The lowest-ranking parameters are removed

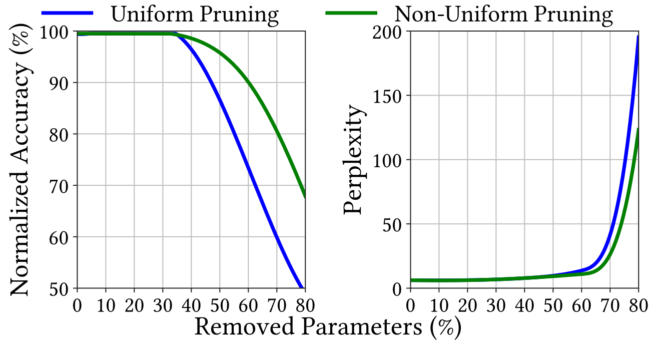

Fig. 3. Normalized accuracy (higher is better) and perplexity (lower is better) of LLaMa-3-8B as parameters are removed via uniform and non-uniform pruning.

based on a target pruning percentage [24]. For example, the lowest ranking 30% of parameters is removed if the pruning target is 30% compression.

To calculate the importance of each parameter, hundreds of data samples pass through the LLM to activate each parameter and then rank them [12,21]. However, consider Fig. 2, where memory usage can go above the capacity of a single consumer GPU. Current LLM pruning methods rank parameters one layer at a time to reduce memory usage [12]. This approach results in parameters ranked by importance within individual layers rather than globally. Consequently, the pruned LLMs are of lower quality, and each layer is pruned uniformly [22]. If LLM pruning evaluates the importance of every parameter globally during ranking, then parameters could be pruned based on their importance relative to the entire LLM rather than their importance within individual layers. This allows non-uniform pruning across the model — targeting and removing redundant parameters while preserving crucial parameters required to maintain model quality.

In **uniform pruning**, every component (i.e. layer, block, and projection) of the LLM is pruned by the same amount, whereas in **non-uniform pruning**, some components are pruned more than others. Fig. 3 shows the accuracy and perplexity achieved by uniform and non-uniform pruning on language benchmarks for the foundation LLM, LLaMa-3-8B. As more parameters are removed, non-uniform pruning achieves higher accuracy and lower perplexity than uniform pruning since non-uniform pruning can more accurately rank and prune LLMs. For the same accuracy loss, non-uniform pruning can eliminate up to 25% more parameters compared to uniform pruning, removing 70% versus 55% of the parameters, respectively. When 70% of parameters are removed, non-uniform pruning achieves a 40% lower perplexity of 25 compared to uniform pruning that has 40.

Opportunity 1: Non-uniform pruning improves accuracy and perplexity compared to uniform pruning.

Defining Projection Pruning: We first consider global and layer/block pruning before defining projection pruning.

Global Pruning is synonymous with uniform pruning. It was introduced to prune LLMs with lower GPU memory overheads [12]. Pruning occurs globally for a fixed percentage [21]. Each layer, block, and projection within the model removes the same percentage of parameters without accounting for their importance between different components. While projections are pruned, they are not evaluated individually for importance; instead, they are uniformly pruned by applying the same percentage across all projections.

Layer/Block-wise Pruning is a more fine-grained pruning approach (we refer to as quasi-non-uniform layer/block pruning) in which each layer/block has a different pruning percentage [24]. For convolutional neural networks (CNNs), pruning layers by different percentages allows important parameters in sensitive layers to be untouched while aggressively pruning redundant layers [25]. This leads to better model accuracy at higher sparsities and extends to LLMs [22]. However, each projection within each layer/block is pruned to the same percentage, which does not account for the importance of parameters between the seven projections within each layer.

In this article, we define **Projection Pruning** as the finest-grained non-uniform pruning method of the projections within an LLM. Each projection is pruned to a specific percentage based on how important each parameter is against other projections of the same category. For example, all query projection parameters in a given transformer layer are ranked against all query projections across all layers. This granularity of pruning is unexplored for LLMs. Our work explores projection pruning for the first time.

Opportunity 2: The proposed projection pruning, a non-uniform pruning method, offers more control over the parameters removed when pruning.

Combining Unstructured and Structured Pruning: Uniformity determines where and how much pruning occurs within the model, while unstructured and structured pruning are categories that specify how parameters are removed.

Unstructured pruning (UP) [12,21] sets parameter weights to zero, creating model sparseness; although zeroed parameters no longer contribute to the model, the size of the model remains unchanged. Sparse models retain model quality but often require vendor-specific GPUs and acceleration libraries for limited speedup gains — for example, only 1.24× speedup at 50% sparsity for LLaMA-7B [21].

Structured pruning (SP) [13] removes entire data structures containing groups of parameters, thus significantly reducing the model size and inference latency. However, SP affects model quality, which decreases more rapidly for higher sparsities than UP. Structured pruning methods target resource-constrained devices that cannot run the original full-sized model. Typically, these devices do not have GPUs or support the libraries for sparse model inference.

In this work, we leverage the benefits of unstructured and structured pruning (i.e., retaining model quality while reducing model size and inference latency) within projection pruning. This combination is referred to as **Composite Projection Pruning**, which is unexplored for LLMs. Projection pruning has been unexamined due to the challenges in accelerating non-uniformly pruned projections [22]. The integration of unstructured and structured pruning allows for accelerating these models, even when specialized accelerators are unavailable [25,26], thereby addressing the problem of non-uniformity. Previous composite pruning methods [25,26] have focused on convolutional neural networks (CNNs), wherein each layer comprises a single type of component, for example, a collection of convolution filters or a fully connected linear layer. In contrast, layers of LLMs consist of multiple blocks featuring diverse projections of varying dimensions and purposes, which has not been investigated within the framework of composite pruning.

Opportunity 3: Combining unstructured and structured pruning for projections, referred to as *composite projection pruning*, retains model quality while reducing model size and inference latency.

Global/Layer/Projection pruning alone refers to the granularity of the pruning method, which can be applied in an unstructured or structured manner. Therefore, the potential dimensions of projection pruning that are explored in this manuscript include unstructured, structured, and composite projection pruning.

2.4. Leveraging the opportunities

We present `Mosaic`, a system that leverages the above opportunities and creates a novel approach for deriving and deploying compressed LLMs on resource-constrained hardware. `Mosaic` models bridge the performance and quality gap between UP and SP, allowing for flexible deployment of existing foundation LLMs to multiple target hardware platforms. While composite projection pruning is unique to Mosaic, the system can create pruned models using any of the three categories of model pruning depending on the available resources for the target platform. For example, a Mosaic LLM using UP will suit a GPU-rich environment to achieve high model quality. A Mosaic model derived using SP makes LLM inference possible at a minimal accuracy loss in environments that have limited resources. In addition, Mosaic models derived using composite projection pruning may be suited for weak GPUs and reducing the overall memory footprint.

3. Design of Mosaic

Mosaic adopts composite projection pruning on pre-trained foundation LLMs to create SLMs for a target device. Mosaic builds on the groundwork laid by global, layer, and block pruning while combining unstructured and structured pruning for the first time on LLMs and applies it for projections to achieve true non-uniform pruning. This section explores the proposed composite projection pruning method. When Mosaic is employed solely for unstructured or structured pruning, it utilizes the implementations outlined in Section 4.1.3.

3.1. Pruning LLMs across projections

In the simplest case, global/uniform pruning applies the same pruning target p to every component of the LLM where $p \in [0, 1)$. For example, $p = 0.3$ prunes 30% of the model parameters. For non-uniform pruning, a pruning *target* must be calculated for each component. This subsection presents the underlying concepts of projection pruning. Firstly, a ranking method is proposed to select a varying set of pruning targets for each layer from the initial p value to achieve non-uniform projection pruning. Secondly, the Projection Outlier Distribution (POD) is defined to achieve projection pruning.

From layer to projection pruning

Layer pruning creates a pruning target for each layer, which overall averages to approximately p. For a LLM with N layers,

$$p \approx \frac{\sum_{n=1}^{N} p_n}{N} \tag{1}$$

where p_n is the pruning target of the nth layer. In layer pruning, p_n is applied to all projections within the layer. Each layer has a unique pruning target, but the projections within each layer are pruned by the same amount (Fig. 1).

Projection pruning extends layer pruning by determining a pruning target for each projection. For an LLM with M projections,

$$p_n \approx \frac{\sum_{m=1}^{M} p_{n,m}}{M} \tag{2}$$

where $p_{n,m}$ is the pruning target of the mth projection in the nth layer. For example, $p_{1,1}$ is the pruning target for the Query projection in the first layer, where $p_{2,2}$ is the Key projection in the second layer (Fig. 1). Working backwards, using Eq. (1), p can be found for the entire LLM.

The number of pruning targets increases with the granularity of the pruning method. While global pruning has one pruning target, layer pruning has N pruning targets, and projection pruning has $N \cdot M$. For example, LLaMa-7B has $N = 32$ and $M = 7$ or 32 and 224 pruning targets for layer and projection pruning, respectively.

Projection outlier distribution

In global pruning, p is typically defined by the system user. However, pruning targets must be derived from an initial p value for layer and projection pruning. Existing layer pruning methods use Layer Outlier Distribution (LOD) [22] to calculate p_n for every layer. LOD creates a set of ratios based on how many outlier parameters are in each layer. The ratios are then scaled and normalized by p to calculate p_n for each layer.

Parameter outliers of a layer are defined as the set of parameters with a *weight metric* greater than the average for the layer. Existing research uses the weight metric ω_n for layer n [21,22]:

$$\omega_n = \|A_n\|_2 \cdot |\theta_n| \tag{3}$$

where $\|A_n\|_2$ is the l_2 norm of the activations of layer n and $|\theta_n|$ is the magnitude of parameter weights in layer n. LOD determines that for each parameter i in θ_n are outliers in n if:

$$IsLayerOutlier(\theta^i_n) = \omega^i_n > \alpha \cdot \overline{\omega_n} \tag{4}$$

is true, where the weight metric ω^i_n of a parameter i is greater than the threshold of a constant α, typically set to five or greater [22], multiplied by the average weight metric for the layer. The number of outliers per layer is then scaled into a ratio contrasted by the other layers to determine p_n for each layer. Layers with more outliers are important to LLM model accuracy [21]; therefore, they are pruned less than layers with fewer outliers. Important layers are assigned smaller p_n values, while less critical layers have larger p_n values to achieve the overall pruning target p (Eq. (1)).

To extend LOD for projection pruning and determining a set of $p_{n,m}$ values, Mosaic defines *Projection Outlier Distribution (POD)* by making the following fundamental changes: (1) the weight metric is calculated at the projection level and (2) outlier parameters are compared within the same projection instead of across the layer.

To achieve (1), Eq. (3) is modified as follows:

$$\omega_{n,m} = \|A_n\|_2 \cdot |\theta_{n,m}| \tag{5}$$

where the l_2 norm term remains the same; however, a weight metric for each projection m is calculated for each layer n using only the magnitude of weights in each projection m. This allows Eq. (6) to find projection outliers as the weight metric of each parameter is restricted to the set of parameters in that particular projection $\omega_{n,m}$ which achieves (2). We define POD using projection outliers with

$$IsProjectionOutlier(\theta^i_{n,m}) = \omega^i_{n,m} > \alpha \cdot \overline{\omega_{n,m}} \tag{6}$$

3.2. Composite projection pruning

In existing literature, structured pruning accelerates sparsely trained CNNs [26], such as VGG-16 [27]. This is achieved by first training a sparse model that is pruned using unstructured pruning. After training, channels that have entire zero values (as an outcome of unstructured pruning) are removed via structured pruning. This work defines composite projection pruning for the first time for LLMs, where unstructured and structured pruning are simultaneously employed across projections. This contrasts with prior work using unstructured and structured pruning in a sequence for CNNs.

Fig. 4 shows that for an LLM attention block, parameters in each projection are individually pruned via unstructured pruning based on POD (Section 3.1) and simultaneously projection nodes between projections are pruned as a group via structured pruning. This combination is referred to as *composite projection pruning* and to derive SLMs from LLMs that are of high quality with low resource overheads. Section 4.2.3 will explore the benefits of composite projection pruning across a range of hardware.

Next, the modules of Mosaic are considered. It takes a foundation LLM as input, and using POD, determines $p_{n,m}$ for each projection to prune the LLM by p using unstructured, structured, and composite projection pruning.

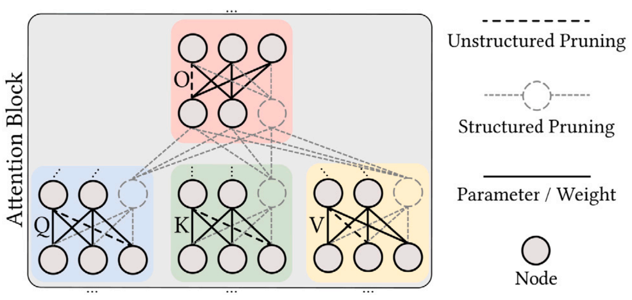

Fig. 4. Example of combining unstructured and structured pruning in composite projection pruning.

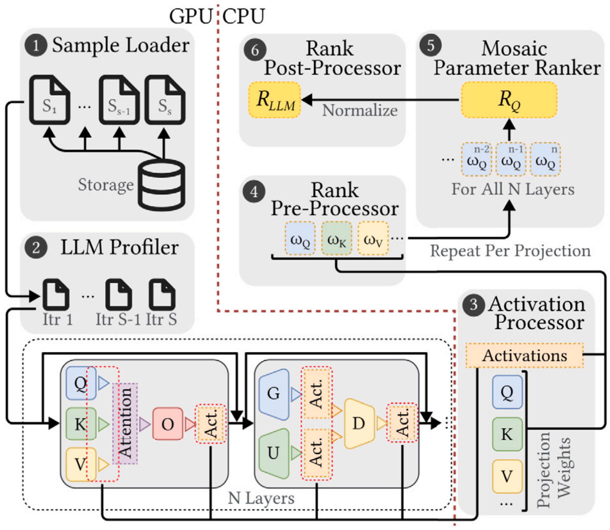

Fig. 5. Overview of the Mosaic Parameter Ranking Controller. A global rank (R_{LLM}) captures the importance of each projection in every layer which is calculated using a combination of activations and the projection weights.

3.3. Mosaic *system*

Mosaic consists of two modules — Parameter Ranking Controller (Fig. 5) and Parameter Pruning Controller (Fig. 6) that are run sequentially to deploy an LLM on a target hardware platform. The modules are detailed below.

Parameter Ranking Controller (RC) profiles the LLM to create a *global rank* (R_{LLM}) that represents the importance of each projection by identifying projection parameter outliers. Each LLM is profiled once to reuse the global rank for any pruning level p across different pruned LLM variants. As a result, the initial overhead (Section 4.2.5) in creating the global rank is offset by faster inference achieved by the pruned LLMs.

The RC comprises six components as shown in Fig. 5. After preloading the LLM into memory, the ❶ *Sample Loader* moves a small calibration set of tokens into memory (128 samples × 2048 tokens × ∼4 bytes per token =∼1 KB).

The ❷ *LLM Profiler* infers the LLM model with each sample iteratively. ❶ and ❷ are executed on the GPU. The ❸ *Activation Processor* hooks into the activation function for each projection for every sample and transfers the resulting activations to the CPU. The activations are a proxy for the l_2 norm term in Eq. (5).

The ❹ *Rank Pre-Processor* calculates the weight metric (Eq. (5)) for each projection using the captured activations and projection weights. The ❺ *Mosaic Parameter Ranker* calculates the POD (Eq. (6)) for each projection by comparing each parameter weight metric against the average weight metric, thus identifying outliers. The projections with the highest number of outliers are adjusted accordingly.

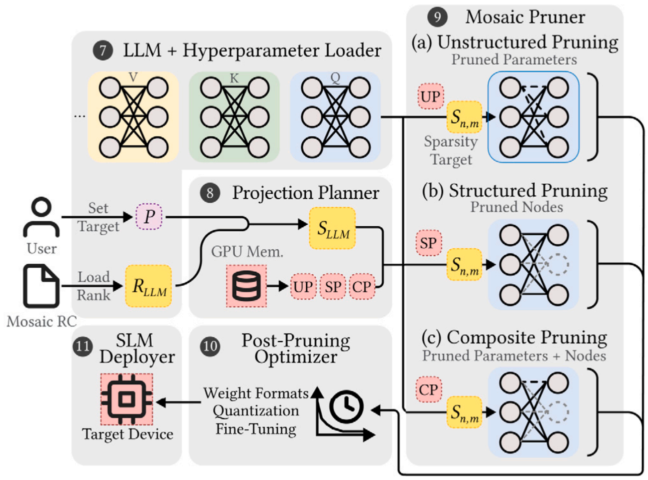

Fig. 6. Overview of the Mosaic Parameter Pruning Controller. The R_{LLM} from the Mosaic RC acts as a look-up table to scale the pruning target to a sparsity target per projection.

Finally, the ❻ *Rank Post-Processor* normalizes all projection ranks into a global rank, which captures the importance of each projection against every other projection in the LLM. The global rank is the output of the RC module and serves as the input to the next module.

Parameter Pruning Controller (PC) prunes the LLM for deployment as shown in Fig. 6 using five components executed on the CPU.

The ❼ *LLM + Hyperparameter Loader* loads the LLM into GPU memory and the global rank from the RC and a pruning target defined by the user where $p \in [0, 1)$.

The ❽ *Projection Planner* scales the global rank by the pruning target, creating a *sparsity target* for each projection in the LLM. The average sparsity target across all projections equals p (Eq. (1) and Eq. (2)). In addition, the available GPU memory of the target deployment platform is used to determine the pruning category.

The ❾ *Mosaic Pruner* prunes each projection by the sparsity target using the identified pruning category, which considers factors such as whether the target platform has enough memory to load the LLM after pruning. Three pruning categories are considered as follows:

(a) Unstructured Projection Pruning suitable for cloud-tier hardware such as server-grade GPUs where memory capacity and bandwidth are not necessarily a bottleneck, and the availability of sparsity accelerators such as NVIDIA CUTLASS can leverage the sparsity from unstructured pruning.

(b) Structured Projection Pruning is chosen for low-end edge devices where GPUs may be unavailable and are typically memory-limited.

(c) Composite Projection Pruning combines (a) and (b) categories and prunes in that order. Composite pruning is reserved for platforms with mobile or older-generation GPUs.

The LLM is then pruned using the determined pruning category, where each projection is reduced by the sparsity target, creating an overall pruned LLM.

The ❿ *Post-Pruning Optimizer* prepares the pruned LLM for deployment, including steps to improve downstream task performance such as fine-tuning low-rank adaptors (LoRA) [28], further compression such as quantization [29], or converting the model weights into different inference formats for specific accelerators. For example, ONNX weights for DeepSparse CPU accelerators [30].

The ⓫ *SLM Deployer* deploys the pruned LLM, which is a small language model (SLM) [7], to the target device.

Algorithm 1 Mosaic Parameter Ranking Controller (RC)

```
Input: Calibration samples $S$, Foundation large language model $LLM$,
    Projection Outlier Constant $\alpha$
Output: Global rank $R_{LLM}$
 1: $S$.GPU(); $LLM$.GPU(); // Move $S$ and $LLM$ to GPU mem.
 2: $N \leftarrow$ length($LLM.layers$); // # LLM Layers
 3: $M \leftarrow$ length($LLM.layers[0].projs$); // # LLM Projections
 4: $R_{LLM} \leftarrow [N][M]$; // Empty list of lists of global rank for each projection
    for each layer
 5: for $n$ in $LLM.layers$ do
 6:     $A \leftarrow []$; // Empty list of layer activations
 7:     for $s$ in $S$ do
 8:         $A[s] = LLM.layers[n]$.infer($s$); // Capture activations of $n$
 9:     end for
10:     for $m$ in $LLM.layers[n].projs$ do
11:         $\omega_{n,m} \leftarrow \|A_n\|_2 \cdot |\theta_{n,m}|$; // Weight metric, Equation 5
12:         $\omega_{n,m} \leftarrow$ cat(flatten($\omega_{n,m}$)); // Reshape $\omega_{n,m}$
13:         $C_{n,m} \leftarrow$ numel($m$); // # Parameters in $m$
14:         $O_{n,m} \leftarrow \sum_{i=1}^{C_{n,m}} 1(\omega^{i}_{n,m} > \alpha \cdot \overline{\omega_{n,m}})$; // # Outliers, Equation 6
15:         $R_{n,m} \leftarrow \frac{O_{n,m}}{C_{n,m}} \cdot 100$; // Projection rank of $m$
16:         $R_{LLM}[n][m] \leftarrow R_{n,m}$; // Update global rank $R_{LLM}$
17:     end for
18: end for
19: return normalize($R_{LLM}$);
```

3.4. Implementation

Mosaic is implemented using Python 3.8.10, PyTorch 2.3.0, Transformers 4.43.1, and CUDA 11.7.

Algorithm 1 shows the procedure of the Mosaic Parameter Ranking Controller (Fig. 5). The RC generates the global rank using Eq. (5) and Eq. (6) to generate pruning targets for projections in the PC. First, the calibration samples S, the LLM to prune, and a constant α for Eq. (6) are loaded into system memory. The samples and LLM are then moved to the GPU memory (Line 1). Metadata, such as the number of layers N and projections M, are calculated by measuring the LLM dimensions (Lines 2–3). Lastly, an empty list of lists to store the global rank is initialized using N and M (Line 4). The CPU portion of the algorithm is then activated layer by layer as the samples are passed through the LLM (Lines 6–9). A captures the activations (Line 8). For each projection, the weight metric is calculated by multiplying the l_2 norm of the activations by the magnitude of the projection weights (Eq. (5), Line 11). The weight metric vector is flattened and concatenated across samples (Line 12). The mean of the weight metric is used with α (Line 14) to calculate the ratio of projection outliers (Eq. (6), Line 15) by dividing the number of outliers by the total parameters in the projection (Line 13). The global rank is then updated for the projection in the current layer (Line 16). The algorithm calculates the global rank for every projection across every layer R_{LLM}. Finally, the normalized global rank is returned (Line 19).

4. Experiments

This section presents the experimental setup in Section 4.1 and the results from evaluation in Section 4.2.

4.1. Experimental setup

4.1.1. Testbed

Five hardware platforms as shown in Table 1 are used as the evaluation testbed: (P1) A 2× A100 GPU system akin to a Google Cloud a2-ultragpu-2g instance running Ubuntu 20.04.06 LTS with kernel version 5.4.0-193-generic is used for all experiments unless stated otherwise. (P2) A 2× A6000 GPU system akin to cloud GPU servers with lower GPU memory/bandwidth than P1. (P3) A consumer desktop GPU. (P4) A small form factor SoC GPU. (P5) A Raspberry Pi 5 with limited GPU memory and bandwidth.

Table 1
Hardware platforms used in the evaluation.

Platform	CPU (Arch.)	GPU (Per GPU: Mem., Bandwidth)
P1	AMD EPYC 7713 (x86)	2×Nvidia A100 (80 GB, 1935 GB/s)
P2	AMD EPYC 7713P (x86)	2×Nvidia A6000 (48 GB, 768 GB/s)
P3	Intel i9-13900KS (x86)	Nvidia RTX 3080 (10 GB, 760 GB/s)
P4	Cortex-A78AE (ARM)	Nvidia AGX Orin (64 GB*, 205 GB/s)
P5	Cortex-A76 (ARM)	Broadcom VideoCore VII (4 GB†, 15 GB/s)

*Combined system and GPU memory capacity.
†Maximum GPU mem. assigned from a shared 8 GB system mem. pool.

4.1.2. Evaluated models

Five LLMs with varying characteristics shown in Table 2 are considered. Each model has seven projections per layer. All pruning methods are evaluated on each model for quality and performance metrics for a range of sparsity values. The LLMs have the following characteristics:

(1) *Parameter Count* - four different sizes: 6.74 billion, 7.37 billion, 8.03 billion, and 13.02 billion parameters.

(2) *Model Depth* - model depth of 32 and 40 layers.

(3) *Block Parameter Distribution* - ranging from 1:2.7 to 1:3.5 attention block to feed-forward parameter ratio ($^{Attention\ Dim.}/_{Feed-Forward\ Dim.}$).

(4) *Extent of Training* - each LLM is trained with different data sizes, ranging from 1.4 trillion to over 15 trillion tokens.

(5) *Fine-tuned Parameters* - Vicuna-7B v1.5 is a fine-tuned LLM for conversational tasks, whereas LLaMa models are foundation models without fine-tuning.

(6) *Context Length* - from 2 K to 128 K token context length.

4.1.3. Pruning methods

Three *pruning uniformity methods* that use SparseGPT [12] to prune the lowest ranking parameters using the inverse Hessian matrix and a subsequent weight update in a one-shot manner are considered:

(1) **Global Pruning -** each component is pruned uniformly by p as presented in Section 3.1.

(2) **Layer Pruning -** implemented by OWL [22] and uses the Wanda [21] weight metric (Eq. (3)) to identify LOD (Eq. (4)) across layers. Using LOD and p; OWL derives p_n for each layer (Eq. (1)). Each layer is pruned by p_n.

(3) **Projection Pruning -** implemented by Mosaic that uses the weight metric (Eq. (5)) to identify POD presented in Eq. (6) across projections. Using POD and p, Mosaic derives $p_{n,m}$ for each projection (Eq. (2)). Each projection is pruned by $p_{n,m}$.

Three *pruning category methods* are implemented as:

(1) **Unstructured Projection Pruning -** pruned parameters are masked by setting their weights to zero. A zeroed parameter is considered pruned.

(2) **Structured Projection Pruning -** parameters are pruned by removing attention and feed-forward heads and channels using LLM-Pruner [13] as illustrated in Fig. 4.

(3) **Composite Projection Pruning -** Mosaic first applies unstructured pruning to individual parameters, followed by structured pruning to remove the lowest-magnitude attention and feed-forward heads. This approach is analogous to prior composite pruning methods developed for CNNs [26].

4.1.4. Dataset metrics and evaluation

11 natural language datasets across four tasks as shown in Table 3 are considered. These datasets cover a wide range of dynamic inputs and are used to evaluate model performance across diverse domains and task settings. They are summarized as follows:

(1) *Common-sense reasoning task accuracy* - seven datasets used to assess language comprehension for a variety of reasoning tasks commonly

Table 2
LLM architecture and training details. The model size is for parameters in FP16 half precision. Attention Dimensions and Feed-Forward Dimensions are the number of channels in the innermost dimension of each projection.

Model	Params.	Layers	Attention Dim.	Feed-Forward Dim.	Model size	Training size	Context Len.
LLaMa-3.1-8B [31]	8.03B	32	4,096	14,336	16.07 GB	>15T	128K
LLaMa-3-8B [11]	8.03B	32	4,096	14,336	16.07 GB	≳15T	8K
LLaMa-2-13B [20]	13.02B	40	5,120	13,824	26.03 GB	2T	4K
LLaMa-7B [4]	6.74B	32	4,096	11,008	13.48 GB	1.4T	2K
Vicuna-7B v1.5 [16]	6.74B	32	4,096	11,008	13.48 GB	2T + 0.37B*	4K

*Foundation model training size plus fine-tuning dataset size.

Table 3
Datasets used in the evaluation. Batch size (BS) is fixed per dataset.

Task	Dataset	Metric	BS
Common-sense Reasoning Accuracy	ARC-e [32]	Norm. Accuracy	32*
	ARC-c [32]	Norm. Accuracy	
	BoolQ [33]	Accuracy	
	HellaSwag [34]	Norm. Accuracy	
	OBQA [35]	Norm. Accuracy	
	RTE [36]	Accuracy	
	WinoGrande [37]	Accuracy	
Perplexity	PTB [38]	PPL	1
	WikiText-2 [39]	PPL	
Calibration	C4 [40]	–	128
Fine-tuning	Alpaca [41]	Accuracy	64

*BS of 24 for LLaMa-2-13B.

used for evaluating LLM accuracy [9]. In the results presented in this article, model accuracy is reported as the equal-weighted mean across all seven datasets. Note that while some datasets present accuracy as a percentage of correct answers, others use normalized accuracy, where accuracy is adjusted according to the difficulty of the answer [9]. Accuracy is obtained using LM Evaluation Harness v0.4.3. A higher value indicates better performance for accuracy metrics.

(2) *Perplexity* - two datasets are used to evaluate perplexity, which measures the average entropy of next-token predictions. A lower perplexity value denotes better performance, reflecting low entropy.

(3) *Calibration* - As considered in the existing pruning literature, 128 samples from the C4 [40] dataset are used to calculate the activations in Eq. (3) and Eq. (5), and to calculate the inverse Hessian in SparseGPT.

(4) *Fine-tuning* - a synthetic dataset Alpaca [41] is used to fine-tune a low-rank adapter (LoRA) of pruned models to recover reasoning accuracy post-pruning.

4.1.5. Other metrics

Accuracy is assessed based on zero-shot performance. Time and inference latency is the average from five trials, and one standard deviation is also presented. Memory use of the GPU is the allocated amount in `nvidia-smi` and for CPU is obtained from `/proc/meminfo`.

4.2. Results

The experiments evaluate the following:

E1 - Non-uniform pruning underpinning `Mosaic` outperforms uniform pruning in terms of accuracy and perplexity.

E2 - Projection pruning enables fine-grained control, leading to more optimally pruned LLM components.

E3 - Composite projection pruning leverages the higher accuracy of unstructured pruning with memory efficiency and faster inference of structured pruning.

E4 - Fine-tuning LLMs after projection pruning regains accuracy faster and more effectively than uniform pruning.

E5 - `Mosaic` achieves better overall performance with lower overheads than other methods.

Table 4
Mean zero-shot accuracy of two LLMs as parameters are removed using global, layer and projection pruning.

		Removed Parameters				
Model	Method	0%	20%	40%	60%	80%
LLaMa-3.1-8B	Global	**69.35**	68.88	65.98	55.53	37.29
	Layer	**69.35**	68.93	66.24	57.68	38.79
	Projection	**69.35**	**69.02**	**66.41**	**60.86**	**42.89**
LLaMa-2-13B	Global	**67.07**	66.12	64.71	59.75	36.90
	Layer	**67.07**	66.37	65.13	60.95	39.98
	Projection	**67.07**	**66.43**	**65.31**	**62.48**	**48.48**

Additional experiments, such as evaluating `Mosaic` with model quantization and reduced calibration sample size, and a full breakdown of the accuracy on individual datasets and models is provided in Appendix.

The results presented in this article are based on analyzing over 5 TB of data produced by the experiments.

4.2.1. Projection pruning performance (E1)

Global, layer and projection pruning is evaluated on all models in Table 2 on the perplexity and accuracy datasets in Table 3. Each model is evaluated for varying sparsities up to 80% as the model collapses beyond this range [12,42]. Perplexity is reported per dataset, whereas accuracy is the mean across the seven datasets shown in Table 3.

> **Observation 1:** Projection pruning achieves up to 84.2% lower perplexity and up to 31.4% higher accuracy than global and layer pruning for the different models, datasets, and sparsities considered.

Perplexity: Fig. 7 shows the WikiText-2 and PTB dataset perplexity achieved on the five LLMs for global, layer and projection pruning up to 80% sparsity (removed parameters). As sparsity increases, perplexity naturally increases as there are fewer parameters in the LLM. Projection pruning has the lowest perplexity for all models, sparsities, and datasets. At 80% sparsity, projection pruning ranges from 18.9% to 84.2% lower perplexity on WikiText-2 than global pruning and 16.8% to 82.1% on PTB. Layer pruning is positioned in-between projection and global pruning on older LLaMa models, such as LLaMa-7B and LLaMa-2-13B, but with newer LLaMa-3-8B and LLaMa-3.1-8B perform much closer to global pruning such as only 5.4% lower perplexity than global on LLaMa-3-8B at 80% sparsity on PTB.

Accuracy: Table 4 shows the mean zero-shot accuracy of LLaMa-3.1-8B and LLaMa-2-13B for each pruning method at 20%, 40%, 60%, and 80% sparsity. 0% sparsity is the accuracy of the original LLM with no pruning. Projection pruning maintains the highest accuracy for both models and all sparsity values. At 40% sparsity, projection pruning improves accuracy by less than 1% for LLaMa-2-13B against other methods. However, at 80% sparsity, this difference increases to 31.4% higher accuracy. Similarly, LLaMa-3.1-8B achieves 13.2% higher accuracy at the same sparsity level.

4.2.2. Projection pruning control (E2)

This experiment examines the empirical differences between global, layer, and projection pruning, highlighting how the choice of granu-

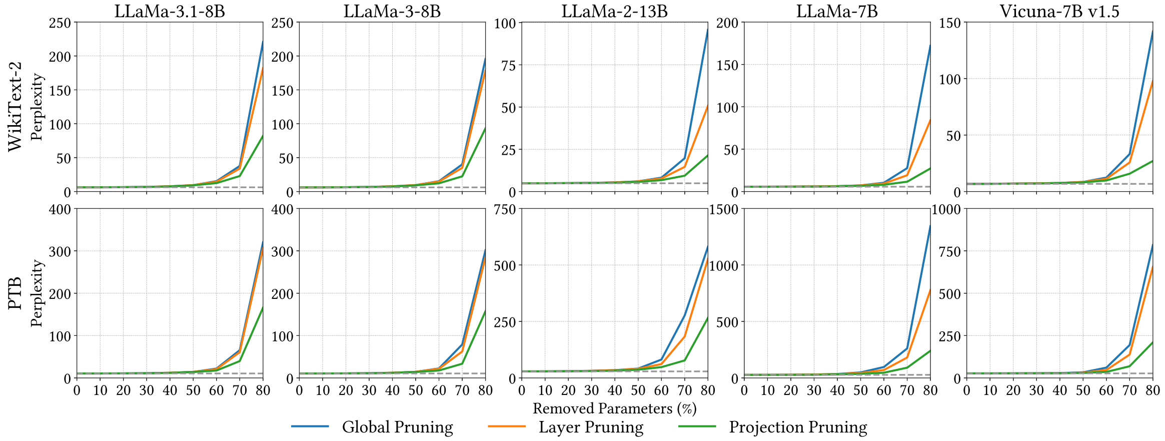

Fig. 7. Perplexity (lower is better) on WikiText-2 and PTB for LLMs as parameters are removed using different pruning methods. Dashed line is perplexity of the foundation model.

larity significantly affects pruning outcomes. Fig. 8 shows the pruning target of each layer of LLaMa-3.1-8B pruned by 80% for each method. On WikiText-2, global, layer and projection pruning achieve a perplexity of 221, 182 and 82, respectively. On PTB, 320, 306 and 166, respectively.

Global Pruning: Each layer is uniformly pruned as shown by the horizontal blue line in Fig. 8. No distinction is made between the importance of each layer in the LLM; consequently, this method over-prunes Layers 0–18, while other methods choose to prune less. Conversely, global pruning under-prunes Layers 19–31, while other methods choose to prune more. Critical layers are over-pruned as the importance of specific parameters and layers is not considered, adversely impacting perplexity and accuracy.

Layer Pruning: Each layer is pruned with a different target to allow critical and redundant layers to be identified. Layer pruning identifies that layers 0–18 should be pruned less, whereas layers 19–31 can be pruned more. As a result, for an average target of 80%, layers are pruned in the range of 66% to 82.8%. Although layer pruning improves perplexity by 17.6% and 4.4% over global pruning for WikiText-2 and PTB, respectively, it fails to identify critical layers (1–10) that more fine-grained (projection) pruning prunes less.

Projection Pruning: Each projection in a layer is pruned individually, resulting in a non-uniform set of pruning targets. In Fig. 8, each projection in every layer is pruned by a different target. Within the attention block (green in Fig. 8), the Key projection has the most redundant parameters, whereas the Output projection is pruned the least, suggesting it has more critical parameters. Within the feed-forward block (purple in Fig. 8), the Gate projection is pruned more, whereas the Down projection is pruned the least. For an average target of 80%, projections can be pruned anywhere in the range of 57.4% to 87.5%, improving perplexity by 55% and 45.8% for WikiText-2 and PTB, respectively.

Fig. 8 how projection-level pruning, although each pruning method ranks parameters individually, the collective importance of parameters within a single projection can lead to varying scores across different projections. Consequently, projections within the same layer may see up to a 10% difference in pruning to accommodate their importance. This fine-grained approach contrasts global and layer-wise pruning methods, which aggressively remove crucial parameters by pruning coarse data structures, such as entire layers, into a fixed pruning target, resulting in less accurate pruned models.

Observation 2: Projection pruning enables non-uniform pruning to identify ideal pruning targets, achieving up to 63% lower perplexity than uniform pruning.

4.2.3. Composite projection pruning performance (E3)

This experiment compares composite projection pruning against unstructured and structured baselines across a range of performance and accuracy metrics. Fig. 9 shows inference latency and memory usage of LLaMa-7B on the five hardware platforms shown in Table 1 for varying pruning targets. Inference is carried out with 2048 tokens input and an output of 128 tokens per MLPerf standardized benchmark [43] with a batch size of 12. On P5, input tokens of 128 and output tokens of 16 with a batch size of 1 are used due to memory and compute constraints. Inference latency is the wall clock time for processing the entire input string and generating the output string. GPU memory used on each platform includes model weights, activations, attention, the software libraries and the `Mosaic` framework. The size of software libraries, such as Python packages compiled for different architectures, varies across platforms, affecting the size of memory allocated. On P3 and P5 with low GPU memory, the model weights are offloaded to device storage using swap memory when exceeding GPU memory capacity (dashed gray lines in Fig. 9). Table 5 shows the perplexity achieved on LLaMa-7B for different pruning methods.

System Performance: Composite and structured pruning reduce inference time and GPU memory used as more parameters are removed. Unstructured pruning does not offer any performance benefits. Although structured pruning has better runtime performance, model quality measured by perplexity (Table 5) is impacted. When 80% of the parameters are removed, composite pruning has 30%–67% lower inference latency than unstructured pruning while reducing memory use by 60%–68%.

Low Memory Platforms: The GPU memory available on P3 and P5 is lower than the foundation model of LLaMa-7B. Therefore, model layers are offloaded to storage, and only a subset is loaded into memory. During inference, layers are transferred in and out of memory and storage, resulting in a data transfer bottleneck, thus increasing inference latency. For example, on P3, inference latency is 420 s when using offloading but once a model is pruned enough to use less than 10 GB of GPU memory, inference latency is reduced by up to 30× (14 s for 80% pruned `Mosaic` model). The foundation model and the pruned models obtained from unstructured pruning cannot be run on P5. When the memory required is reduced from 10 GB to 7.9 GB using `Mosaic`, the inference latency reduces from 400 s to 110 s. Composite pruning of `Mosaic` enables model deployment on memory-constrained hardware by structurally reducing the memory footprint, overcoming the limitations of unstructured pruning, which alone does not reduce inference latency or enable execution under tight memory budgets. This approach allows Mosaic to adapt the sparsification strategy to hardware constraints by applying structured pruning where unstructured sparsity alone cannot be exploited due to a lack of hardware accelerators or limited memory capacity.

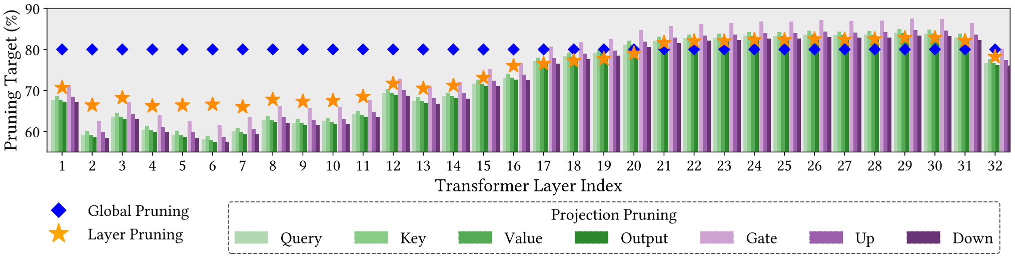

Fig. 8. Pruning targets across all layers and projections for LLaMa-3.1-8B pruned by 80%. For projection pruning, green bars represent the attention block projections, whereas purple bars represent the feed-forward block projections.

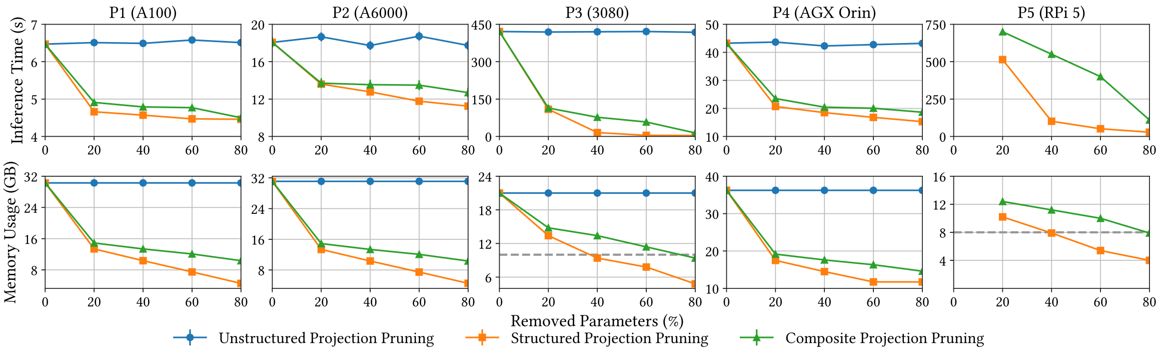

Fig. 9. Inference latency and GPU memory use of pruned LLaMa-7B for varying pruning targets (measured as % of removed parameters) on different hardware platforms shown in Table 1. The dashed gray line is GPU memory capacity (for P3 and P5), and the foundation model and unstructured pruned models cannot be run on P5 due to limited resources.

Table 5
Perplexity of LLaMa-7B for unstructured, composite, and structured projection pruning for five pruning targets.

		Removed Parameters				
Perplexity	Method	0%	20%	40%	60%	80%
WikiText-2	Unstructured	5.68	5.76	6.14	7.98	27.24
	Composite	5.68	15.28	20.76	80.06	938.00
	Structured	5.68	18.61	647.20	4,074.49	33,586.00
PTB	Unstructured	27.34	27.61	30.87	46.41	240.43
	Composite	27.34	67.68	100.03	253.45	1282.09
	Structured	27.34	90.02	538.65	2,330.66	23,447.05

Model Quality: Compared to structured pruning, models derived from composite pruning have a lower perplexity as more parameters are removed. At pruning targets higher than 40%, the models produced by structured pruning collapse (or are rendered unusable) due to very high perplexity. Models from composite pruning have up to 36× lower perplexity than structured pruning.

> **Observation 3:** Composite projection pruning combines the merits of unstructured and structured pruning. It makes inference 67% faster and reduces GPU memory use by 68% compared to unstructured pruning while achieving up to 36× lower perplexity than structured pruning.

4.2.4. Fine-tuning performance (E4)

The SLM quality is regained after pruning using parameter-efficient fine-tuning (PEFT) approaches, such as low-rank adaptation [28] (LoRA). Alpaca [41] (Table 3) is used to train a LoRA adapter for LLaMa-3.1-8B for two epochs [13]. LoRA creates an 84 MB adapter which merges into the original pruned model weights at runtime. Fig. 10 and Table 6 summarize the findings.

> **Observation 4:** Fine-tuning offers more performance and quality gains on LLMs compressed using projection pruning. For the same amount of fine-tuning, similar quality as global and layer pruning is achieved up to 7.5× faster, with up to 34.4% lower perplexity and 15.4% higher accuracy.

Training and Evaluation: Fig. 10 shows the reduction in training and evaluation loss on the Alpaca dataset for each pruning method during LoRA fine-tuning. Global and layer pruning reach a final training loss of 1.76 and 1.72, whereas projection is 1.53. Projection pruning achieves a loss of 1.72 within 250 training steps in 30 min. This is a 6.2× speedup over other methods that need 1550 training steps and three hours to complete. Similarly, global and layer pruning reach a final evaluation loss of 1.85 and 1.80, whereas projection is 1.62. Projection reaches a loss of 1.8 at 200 evaluation steps — 7.5× faster than the other methods.

Full Dataset Evaluation: Table 6 demonstrates that by fine-tuning on one dataset, namely Alpaca, the perplexity and accuracy improves on other datasets listed in Table 3. All methods show better perplexity and accuracy following fine-tuning. Projection pruning starts with higher accuracy and gains more from fine-tuning than other methods. This indicates that projection pruning retains an optimal set of parameters that can be effectively retrained.

4.2.5. End-to-end overheads (E5)

The combination of pruning and fine-tuning overheads is referred to as end-to-end overhead. These overheads are considered for global, layer and projection pruning in Fig. 11.

> **Observation 5:** The end-to-end overhead of `Mosaic` that includes the time for projection pruning and fine-tuning to produce deployment-ready models is up to 7.19× lower than existing methods.

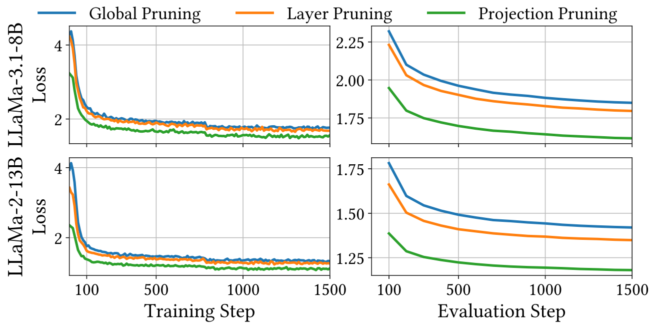

Fig. 10. Training and evaluation loss of fine-tuning an 80% pruned LLaMa-3.1-8B and LLaMa-2-13B.

Table 6
Perplexity and accuracy achieved before and after fine-tuning an 80% pruned LLaMa-3.1-8B on Alpaca.

	Before Fine-tuning		After Fine-tuning	
Method	PPL	Accuracy	PPL	Accuracy
Global	220.53	37.29	41.96 (↓81.0%)	43.33 (↑16.2%)
Layer	181.79	38.79	37.08 (↓79.6%)	44.46 (↑14.6%)
Projection	**82.08**	**42.89**	**27.54** (↓66.4%)	**50.01** (↑16.6%)

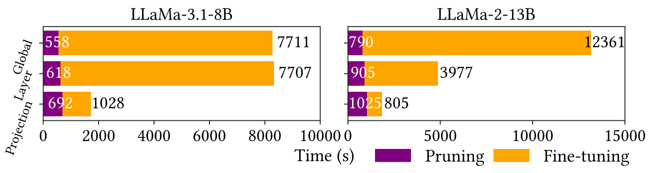

Fig. 11. End-to-end overhead (pruning time in white font and fine-tuning in black font) for LLaMa-3.1-8B and LLaMa-2-13B pruned by 80% and fine-tuned.

The *pruning overhead* (purple bars) of layer and projection pruning is higher than global pruning. This is because a weight metric is calculated for every parameter to obtain their pruning targets (Section 3.1), then prune the LLM based on target percentages. Global pruning of LLaMa-3.1-8B and LLaMa-2-13B requires 23.31 GB and 33.34 GB of memory, respectively, for the model parameters in FP16 precision, software libraries, 128 samples from the calibration dataset, and model activations from calibration. Layer and projection pruning requires additional memory due to the weight metrics — 24.22 GB and 35.81 GB for LLaMa-3.1-8B and LLaMa-2-13B, respectively.

As explored in Section 4.2.4, LLMs are typically fine-tuned for deployment to regain accuracy lost during pruning. The *fine-tuning overhead* is shown in Fig. 11 (orange bars). The models pruned using layer and projection pruning are fine-tuned to match the same accuracy achieved by global pruning after it is fine-tuned for two epochs. For LLaMa-3.1-8B, projection pruning produces a deployment ready model over 4.8× faster than global and layer pruning. In the case of LLaMa-2-13B, projection pruning is 2.67× and 7.19× faster than layer and global pruning, respectively.

5. Related work

The three main LLM compression methods to improve resource efficiency are model quantization, knowledge distillation, and pruning.

5.1. Model quantization

State-of-the-art quantization methods, such as GPTQ [29] and AWQ [44] reduce FP16 weights of the LLM into lower precision, such as INT3, INT4, or INT8, which requires less memory. Although the model size is reduced by more than half with minimal accuracy loss, the overall memory required remains high because model activations are not quantized and still use the higher FP16 precision. As memory demands grow due to larger activations from longer input lengths (more tokens), the quantized weights account for only a small fraction of the total memory usage, which reduces the overall effectiveness of this approach. Alternative methods that quantize model activations have lower accuracies and rely on a decomposition scheme to keep outlier weights at FP16 [45]. Although all quantization methods reduce memory requirements, orthogonal approaches are required to lower inference latency. For example, GPTQ relies on custom CUDA kernels [29], and AWQ utilizes QKV kernel fusion [44]. They are, therefore, suited to GPU-based systems for performance gains, which may not be available in resource-constrained environments.

5.2. Knowledge distillation

Smaller student models are trained to replicate the output of larger teacher models; they are used in older transformer architectures, such as BERT [2] with DistilBERT [46]. Minitron [14] is created using neural architecture search from Nemotron [47] using pruning and knowledge distillation. Although Minitron requires 40× fewer training tokens than other similar-sized LLMs to reach the same accuracy, the computational requirements are extensive (128× Nvidia A100 80 GB GPUs [14]).

5.3. Model pruning

Unstructured Pruning: SparseGPT [12] and Wanda [21] are *unstructured pruning* methods. SparseGPT implements Optimal Brain Surgeon (OBS) [48] and prunes models with hundreds of billions of parameters in a few hours. In contrast, Wanda employs a simpler pruning metric based solely on model weights and activations, making it two orders of magnitude faster than SparseGPT. Unstructured pruning does not reduce model size since parameters are set to zero. These models require specialized sparse acceleration libraries, such as NVIDIA CUTLASS, which is limited to models that are 50% sparsity pruned using a specific *semi-structured* format [12,21,49].

Structured Pruning: LLM-Pruner [13] and Sheared-LLaMa [50] are *structured pruning* methods and produce smaller and faster LLMs. LLM-Pruner establishes pruning groups to efficiently prune LLM neurons in a one-shot manner, followed by fine-tuning to regain accuracy. In contrast, Sheared-LLaMa combines pruning and training in an iterative process. The pruned models are resource-efficient, but the overall model accuracy is lower than the original model. The above methods reduce layer size by removing neurons. BlockPruner [51] and ShortGPT [52] remove entire blocks and transformer layers, respectively.

Non-uniform Pruning: Unlike uniform pruning that applies the same pruning ratio to all layers [12,13], non-uniform pruning adjusts the pruning ratio for each layer. This has better accuracy than uniform pruning for CNNs [24,25,53]. OWL [22] demonstrates that this observation extends to LLMs. However, LLMs contain components within layers (projections), which in previous research have been pruned uniformly [54]. Therefore, the opportunity for non-uniform projection pruning of LLMs is explored in this article.

Combining Unstructured and Structured Pruning for LLMs: Structured pruning of sparse models created by unstructured pruning has been explored for CNNs to make sparse models run on general-purpose hardware without the need for accelerators [25,26]. They create small and fast sparse models. `Mosaic` is proposed to synergistically combine both pruning methods for LLMs and apply them at the granularity of projections rather than layers or blocks for the first time, referred to as *composite projection pruning*. While existing work has explored composite pruning primarily in the context of CNNs [25,26], `Mosaic` extends this approach to LLMs by targeting projection components unique to their architecture.

6. Conclusion

Deploying large language models (LLMs) on hardware-limited (compute and memory) resources remains a challenge [12,21,22]. Model pruning [12,13] is one method that compresses a large foundation model to create a small language model (SLM). However, existing pruning methods negatively impact model quality [13,50] or rely on vendor-specific hardware and software [12,44]. This is because existing methods prune LLMs using coarse-grained approaches, such as uniform pruning, that inherently remove critical parameters since all layers of the LLM are pruned uniformly. Consequently, the SLMs produced are unusable when a large number of model parameters are removed [13] or require specialized software and hardware to demonstrate any performance gain when fewer parameters are removed [12,21,22].

This article explores a new paradigm for pruning LLMs, referred to as projection pruning. Projections are the smallest LLM components within a layer that capture intrinsic learning properties during training. Uniform pruning removes parameters from every projection equally, thereby removing parameters critical to model quality and under-pruning less important projections. While existing methods identify that specific layers should be pruned non-uniformly [22], this article investigates the optimization of pruning targets for every projection to maximize model quality. To this end, a novel projection-based pruning system for LLMs, `Mosaic`, is developed to reduce the resource requirements for producing SLMs while improving model accuracy compared to other pruning methods. `Mosaic` proposes *composite projection pruning*, which combines unstructured pruning with structured pruning to produce high-quality SLMs that can be deployed on a range of hardware platforms. `Mosaic` produces faster models than existing methods. `Mosaic` models have lower perplexity and higher accuracy than coarse-grained pruning. They achieve faster inference and have a lower GPU memory use than structure pruning.

This work has considered dense LLM architectures. Applying projection pruning to Mixture-of-Experts (MoE) models, such as Mixtral or DeepSeek, presents a promising direction for future research. MoE architectures introduce unique challenges, such as dynamic expert routing and load balancing, that may require new pruning strategies beyond those explored here. Extending projection pruning to MoEs remains an open area for continued investigation. Additionally, many LLM pruning methods rely on calibration datasets; future work will explore developing ranking methods that remove this dependency.

CRediT authorship contribution statement

Bailey J. Eccles: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Leon Wong:** Writing – original draft, Supervision. **Blesson Varghese:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Resources, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Bailey Eccles has patent PROJECTION-BASED LANGUAGE MODEL PRUNING pending to Rakuten Mobile, Inc. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is funded by Rakuten Mobile, Inc., Japan, and supported by funding from UKRI EP/Y028813/1.

Appendix A. Hardware platforms

Tables G.7 and G.8 provide complete hardware (CPU, GPU, RAM) and OS-level breakdown of the platforms used in the experiments.

Appendix B. Model sources

Table G.9 provides the Hugging Face model sources for use in the Transformers Python library for each LLM used in the experiments presented in the article. We provide the exact source as the same LLM can come in different formats, such as serialized Python objects (`.bin`) or Safetensors (`.safetensors`) depending on the Transformers library version used. In addition, the same LLM may come in FP16 or FP32 bit precision, depending on the source.

Appendix C. Detailed accuracy tables

Tables G.10 and G.11 provide the full zero-shot accuracy breakdown for LLaMa-3.1-8B and LLaMa-2-13B results for global, layer, and projection pruning. Note that at lower pruning targets, projection pruning does not outperform all other methods; however, on average, projection pruning achieves a higher accuracy. At higher pruning targets, projection pruning demonstrates superior performance.

Appendix D. Calibration sample size

Fig. G.12 shows the perplexity (on the WikiText-2 and PTB dataset) and pruning time for LLaMa-3.1-8B pruned by 80% using global, layer, and projection pruning for calibration sample sizes 2^n, where $n = 0, 1, 2, \dots, 8$. Most pruning methods presented in the literature choose 128 calibration samples as the default value as it optimally balances model quality against pruning time. Perplexity tends to improve until 128 samples, after which there is diminishing gains in perplexity with respect to pruning time. While projection pruning has a higher pruning time across all sample sizes, projection pruning achieves lower perplexity than global and layer pruning for all sample sizes. Notably, for half the sample size (64), projection pruning achieves a lower perplexity of 121 (WikiText-2) and 195 (PTB) than global and layer pruning at 128 samples, reaching 221 (WikiText-2, Global), 182 (WikiText-2, Global) and 320 (PTB, Global), 306 (PTB, Layer), respectively. In this case, projection pruning takes 515 s, whereas global and layer pruning takes 558 s and 618 s, respectively.

Appendix E. Comparison to older pruning methods

Table G.12 shows LLaMa-7B pruned by 70% zero-shot accuracy for all seven datasets compared to older LLM pruning methods. This model and pruning target are commonly used in the literature as most methods tend to collapse beyond the 70% pruning target, and LLaMa-7B was popular then. While more modern and better-trained LLMs are available now and were evaluated in the experiments, these tables provide a reference to compare older pruning methods.

Appendix F. Model quantization

Model quantization is another compression method for LLMs that reduces the bit precision of the model parameters. This reduces the memory footprint of the model. Table G.13 compares GPTQ, a quantization method used for LLMs, for four different bit targets using the group hyperparameter of 128 for LLaMa-3.1-8B compared against `Mosaic`. Zero-shot accuracy, as well as speedup and compression, is presented. Speedup is the improvement to LLM inference. While GPTQ can accelerate inference with custom kernels, the results presented are for hardware (P1) without the software accelerator. `Mosaic` does not require custom kernels to accelerate inference. Compression is the file size compression of the LLM weights. While GPTQ sets weights to the target bit size, activations are still 16 bits during inference, which has the same memory usage as the unquantized/unpruned (dense) model. For `Mosaic`, the pruning target proportionally decreases inference memory usage.

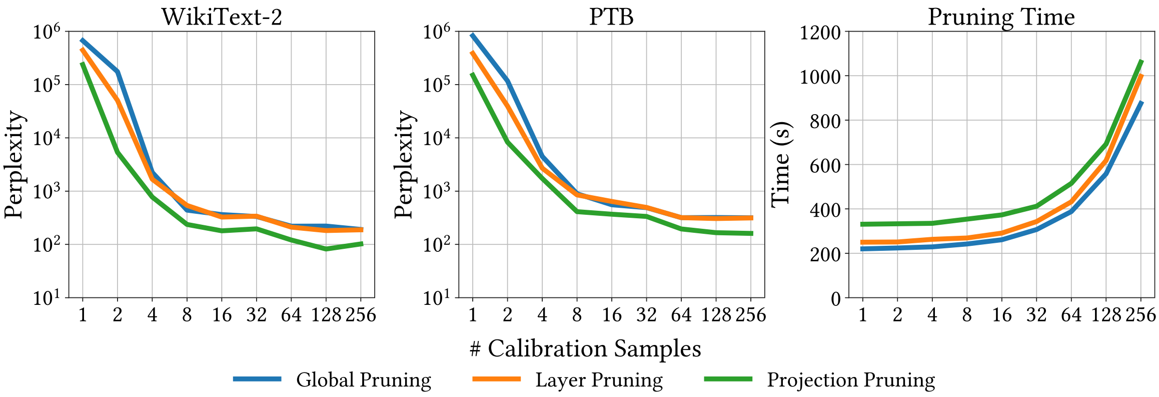

Fig. G.12. Perplexity (on the WikiText-2 and PTB dataset) and pruning time for LLaMa-3.1-8B pruned by 80% using global, layer, and projection pruning for calibration sample sizes in powers of two from 1 to 256.

Table G.7
CPU and system level details of hardware used.

Platform	CPU	Cores (Threads)	CPU Arch.	Operating system	Kernel	Memory
P1	AMD EPYC 7713	64 (128) up to 3.67 GHz	Zen 3 (x86)	Ubuntu 20.04.06 LTS	5.4.0-193-generic	192 GB
P2	AMD EPYC 7713P	64 (128) up to 3.67 GHz	Zen 3 (x86)	Ubuntu 20.04.06 LTS	5.15.0-117-generic	256 GB
P3	Intel i9-13900KS	24 (32) up to 6.00 GHz	Raptor Lake (x86)	Ubuntu 20.04.06 LTS	5.15.153.1-WSL2	64 GB
P4	Cortex-A78AE	12 (12) up to 2.20 GHz	ARMv8.2-A (ARM)	Ubuntu 20.04.06 LTS	5.10.104-tegra	64 GB
P5	Cortex-A76	4 (4) up to 2.40 GHz	ARMv8.2-A (ARM)	Debian 12 (bookworm)	6.6.51+rpt-rpi-2712	8 GB

Table G.8
GPU hardware details.

Platform	# GPUs	GPU	GPU memory	Bandwidth	Cores	GPU Arch.	TDP
P1	2	Nvidia A100	80 GB	1935 GB/s	6,912	Ampere	400 W
P2	2	Nvidia RTX A6000	48 GB	768 GB/s	10,752	Ampere	300 W
P3	1	Nvidia RTX 3080	10 GB	760 GB/s	8,704	Ampere	320 W
P4	1	Jetson AGX Orin SoC	64 GB	205 GB/s	2,048	Ampere	60 W
P5	1	Broadcom BCM2712	4 GB	15 GB/s	12	VideoCore VII	27 W

Table G.9
Hugging Face source for each LLM used in the experiments.

Model	Source
LLaMa-3.1-8B	`meta-llama/Llama-3.1-8B`
LLaMa-3-8B	`meta-llama/Meta-Llama-3-8B`
LLaMa-2-13B	`TheBloke/Llama-2-13B-fp16`
LLaMa-7B	`huggyllama/llama-7b`
Vicuna-7B v1.5	`lmsys/vicuna-7b-v1.5`

Appendix G. Additional calibration datasets and models

Activation-based pruning and quantization methods typically use the English subset of the C4 dataset (c4-en) and the LLaMA family of LLMs for evaluation. While C4 provides broad linguistic coverage and has been shown to generalize well across a variety of downstream tasks, it may not fully capture the needs of models trained for non-English languages. For instance, a Mistral-7B-v0.1 extended with a tokenizer designed to support Japanese characters, and fine-tuned for Japanese-language benchmarks [55]. Intuitively, using a calibration dataset composed primarily of English samples may not yield a pruned model that performs optimally on Japanese reasoning tasks such as JSQuAD, JCS, JNLI, and MARC-ja [55]. To evaluate this, Table G.14 compares the performance of Mistral-7B-v0.1 pruned with `Mosaic` using both the English (c4-en) and Japanese (c4-ja) subsets of C4. The model is evaluated on WikiText-2 (English) and four Japanese-language benchmarks covering a range of task types: reading comprehension (JSQuAD), multiple choice (JCS, JNLI), and text

Table G.10
LLaMa-3.1-8B zero-shot accuracy on the seven datasets using global, layer, and projection pruning for different pruning targets. Bold values denote the highest accuracy for each pruning target.

Pruning target	Method	ARC-c	ARC-e	BoolQ	HellaSwag	OBQA	RTE	WinoGrande	Mean
0%	–	53.50	81.19	82.02	78.85	44.60	71.84	73.48	69.35
20%	Global	52.65	**80.85**	81.83	79.00	44.80	69.68	73.32	68.88
	Layer	53.07	80.22	81.87	**79.18**	44.40	**70.04**	**73.72**	68.93
	Projection	**53.16**	80.39	**82.14**	79.00	**45.20**	69.68	73.56	**69.02**
40%	Global	**50.43**	**77.02**	78.23	**77.38**	**43.80**	62.09	72.93	65.98
	Layer	50.09	75.67	79.88	76.65	42.80	65.34	**73.24**	66.24
	Projection	50.34	75.34	**79.94**	76.77	42.60	**67.87**	71.98	**66.41**
60%	Global	34.73	57.32	75.29	62.12	37.40	54.15	67.72	55.53
	Layer	37.54	61.99	77.25	64.35	37.60	56.68	68.35	57.68
	Projection	**42.23**	**67.80**	**78.92**	**68.72**	**38.80**	**59.21**	**70.32**	**60.86**
80%	Global	20.82	29.42	52.11	28.43	27.00	52.35	50.91	37.29
	Layer	20.14	30.35	61.47	30.20	26.80	**52.71**	49.88	38.79
	Projection	**24.49**	**36.66**	**64.68**	**38.59**	**27.80**	**52.71**	**55.33**	**42.89**

Table G.11
LLaMa-2-13B zero-shot accuracy on the seven datasets using global, layer, and projection pruning for different pruning targets. Bold values denote the highest accuracy for each pruning target.

Pruning target	Method	ARC-c	ARC-e	BoolQ	HellaSwag	OBQA	RTE	WinoGrande	Mean
0%	–	49.15	77.53	80.55	79.39	45.20	65.34	72.30	67.07
20%	Global	49.74	77.02	81.10	76.69	**46.20**	**59.57**	72.53	66.12
	Layer	**49.83**	**77.06**	**81.22**	**79.86**	45.40	59.21	71.98	66.37
	Projection	**49.83**	76.81	80.70	79.64	45.80	**59.57**	**72.69**	**66.43**
40%	Global	48.04	74.07	81.28	77.59	45.20	54.51	**72.30**	64.71
	Layer	**48.72**	**75.67**	**80.61**	**78.24**	44.60	56.32	71.74	**65.13**
	Projection	47.44	73.70	81.13	77.65	**46.20**	**59.21**	71.82	65.31
60%	Global	40.78	64.90	79.48	67.72	40.60	53.79	70.96	59.75
	Layer	40.53	64.94	81.16	68.68	41.00	**58.84**	**71.51**	60.95
	Projection	**43.60**	**69.74**	**81.93**	**72.80**	**42.00**	56.32	70.96	**62.48**
80%	Global	22.95	28.91	51.25	29.64	24.60	52.71	48.22	36.90
	Layer	21.76	32.53	62.17	31.39	27.20	52.71	52.09	39.98
	Projection	**27.90**	**46.09**	**68.38**	**47.47**	**32.60**	**53.79**	**63.14**	**48.48**

Table G.12
LLaMa-7B pruned by 70% zero-shot accuracy on seven datasets for different pruning methods.

Pruning target	Method	ARC-c	ARC-e	BoolQ	HellaSwag	OBQA	RTE	WinoGrande	Mean
0%	–	41.38	67.67	75.14	74.80	41.40	66.43	70.01	62.40
70%	Magnitude	22.35	26.98	38.29	24.68	25.80	52.71	51.46	34.61
	Wanda	19.80	34.22	55.11	31.83	26.00	57.40	51.38	39.39
	SparseGPT	24.57	43.06	64.53	42.11	27.80	53.79	58.64	44.93
	OWL	27.65	45.41	67.13	48.56	32.00	53.43	62.03	48.03
	Mosaic	**30.63**	**49.45**	**69.72**	**54.72**	**35.80**	**58.84**	**64.33**	**51.93**

Table G.13
Model quantization and pruning zero-shot accuracy, speedup, and compression (comp.) of LLaMa-3.1-8B for different bit and pruning targets.

Category	Target	ARC-c	ARC-e	BoolQ	HellaSwag	OBQA	RTE	WinoGrande	Mean	Speedup	Comp.
Dense	16 bit/100%	53.50	81.19	82.02	78.85	44.60	71.84	73.48	69.35	1.00×	1.00×
Quantization (GPTQ)	8 bit	53.24	77.61	81.28	79.12	44.80	68.95	72.85	68.26	0.48×	1.74×
	4 bit	38.40	64.81	79.39	76.24	41.80	66.43	71.59	62.67	0.47×	2.80×
	3 bit	23.89	36.32	55.05	39.29	31.20	51.99	55.56	41.90	0.44×	3.31×
	2 bit	26.45	24.83	49.69	26.14	27.40	48.38	48.22	35.87	0.33×	4.04×
Pruning (Mosaic)	20%	53.16	80.39	82.14	79.00	45.20	69.68	73.56	69.02	1.32×	1.24×
	40%	50.34	75.34	79.94	76.77	42.60	67.87	71.98	66.41	1.35×	1.59×
	60%	42.23	67.80	78.92	68.72	38.80	59.21	70.32	60.86	1.36×	2.33×
	80%	24.49	36.66	64.68	38.59	27.80	52.71	55.33	42.89	1.44×	4.20×

Table G.14
Mistral-7B-v0.1 pruned by 50% zero-shot accuracy and perplexity on four Japanese datasets for two different C4 calibration datasets.

Calibration dataset	WikiText-2	JSQuAD	JCS	JNLI	MARC-ja	Mean
- (Base Model)	5.47	78.97	84.18	56.82	96.39	79.09
c4-en	**7.21**	**72.11**	68.01	**29.87**	89.12	**64.78**
c4-ja	7.26	70.08	**69.52**	27.65	**91.50**	64.69

classification (MARC-ja). The results show that both calibration sets produce comparable perplexities and accuracies across all evaluation datasets. On average, c4-en slightly outperforms c4-ja in terms of mean accuracy and perplexity, particularly on JSQuAD and JNLI. However, c4-ja achieves marginally better performance on JCS and MARC-ja. These differences may be attributed to the significantly smaller size and narrower linguistic diversity of c4-ja (0.8 TB), compared to the larger and more heterogeneous c4-en dataset (10 TB), allowing for this dataset to perform well in multilingual settings.

Data availability

The data underpinning this research are available from https://github.com/blessonvar/Mosaic.

References

[1] OpenAI, GPT-4 technical report, 2023, arXiv:2303.08774.

[2] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: North American Chapter of the Association for Computational Linguistics, 2019.

[3] OpenAI, Language models are few-shot learners, in: Advances in Neural Information Processing Systems, 2020.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and efficient foundation language models, 2023, arXiv:2302.13971.

[5] G. Team, Gemini: A family of highly capable multimodal models, 2024, arXiv:2312.11805.

[6] Z. Lin, G. Qu, Q. Chen, X. Chen, Z. Chen, K. Huang, Pushing large language models to the 6G edge: Vision, challenges, and opportunities, 2024, arXiv:2309.16739.

[7] Microsoft, Phi-3 technical report: A highly capable language model locally on your phone, 2024, arXiv:2404.14219.

[8] G. Team, Gemma: Open models based on gemini research and technology, 2024, arXiv:2403.08295.

[9] S. Mehta, M.H. Sekhavat, Q. Cao, M. Horton, Y. Jin, C. Sun, I. Mirzadeh, M. Najibi, D. Belenko, P. Zatloukal, M. Rastegari, OpenELM: An efficient language

model family with open training and inference framework, 2024, arXiv:2404.14619.

[10] X. Chu, L. Qiao, X. Lin, S. Xu, Y. Yang, Y. Hu, F. Wei, X. Zhang, B. Zhang, X. Wei, C. Shen, MobileVLM : A fast, strong and open vision language assistant for mobile devices, 2023, arXiv:2312.16886.

[11] AI@Meta, Llama 3 model card, 2024, URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[12] E. Frantar, D. Alistarh, SparseGPT: Massive language models can be accurately pruned in one-shot, in: International Conference on Machine Learning, 2024.

[13] X. Ma, G. Fang, X. Wang, LLM-pruner: On the structural pruning of large language models, in: Advances in Neural Information Processing Systems, 2023.

[14] S. Muralidharan, S.T. Sreenivas, R. Joshi, M. Chochowski, M. Patwary, M. Shoeybi, B. Catanzaro, J. Kautz, P. Molchanov, Compact language models via pruning and knowledge distillation, 2024, arXiv:2407.14679.

[15] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training, in: OpenAI Blog, 2018.

[16] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J.E. Gonzalez, I. Stoica, E.P. Xing, Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality, 2023, URL https://lmsys.org/blog/2023-03-30-vicuna/.

[17] O. Press, N. Smith, M. Lewis, Train short, test long: Attention with linear biases enables input length extrapolation, in: International Conference on Learning Representations, 2022.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017.

[19] T. Dao, D.Y. Fu, S. Ermon, A. Rudra, C. Ré, FlashAttention: Fast and memory-efficient exact attention with IO-awareness, in: Advances in Neural Information Processing Systems, 2022.

[20] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C.C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P.S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E.M. Smith, R. Subramanian, X.E. Tan, B. Tang, R. Taylor, A. Williams, J.X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, T. Scialom, Llama 2: Open foundation and fine-tuned chat models, 2023, arXiv:2307.09288.

[21] M. Sun, Z. Liu, A. Bair, J.Z. Kolter, A simple and effective pruning approach for large language models, in: International Conference on Learning Representations, 2024.

[22] L. Yin, Y. Wu, Z. Zhang, C.-Y. Hsieh, Y. Wang, Y. Jia, M. Pechenizkiy, Y. Liang, Z. Wang, S. Liu, Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity, in: International Conference on Machine Learning, 2024.

[23] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, in: Advances in Neural Information Processing Systems, 1989.

[24] D. Blalock, J.J. Gonzalez Ortiz, J. Frankle, J. Guttag, What is the state of neural network pruning? in: Machine Learning and Systems, 2020.

[25] B.J. Eccles, L. Wong, B. Varghese, Rapid deployment of DNNs for edge computing via structured pruning at initialization, in: IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, 2024.

[26] B.J. Eccles, P. Rodgers, P. Kilpatrick, I. Spence, B. Varghese, DNNShifter: An Efficient DNN Pruning System for Edge Computing, Future Gener. Comput. Syst. 152 (2024).

[27] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, 2015.

[28] E.J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, LoRA: Low-rank adaptation of large language models, in: International Conference on Learning Representations, 2022.

[29] E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh, GPTQ: Accurate quantization for generative pre-trained transformers, in: International Conference on Learning Representations, 2023.

[30] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, W. Leiserson, S. Moore, B. Nell, N. Shavit, D. Alistarh, Inducing and exploiting activation sparsity for fast inference on deep neural networks, in: International Conference on Machine Learning, 2020.

[31] A. Llama Team, The llama 3 herd of models, 2024, arXiv:2407.21783.

[32] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, O. Tafjord, Think you have solved question answering? Try arc, the AI2 reasoning challenge, 2018, arXiv:1803.05457.

[33] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, K. Toutanova, Boolq: Exploring the surprising difficulty of natural yes/no questions, in: ACL: Human Language Technologies, vol. 1, 2019.

[34] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, Y. Choi, HellaSwag: Can a machine really finish your sentence? in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019.

[35] T. Mihaylov, P. Clark, T. Khot, A. Sabharwal, Can a suit of armor conduct electricity? A new dataset for open book question answering, in: Conference on Empirical Methods in Natural Language Processing, 2018.

[36] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S.R. Bowman, GLUE: A multi-task benchmark and analysis platform for natural language understanding, in: International Conference on Learning Representations, 2019.

[37] K. Sakaguchi, R.L. Bras, C. Bhagavatula, Y. Choi, WinoGrande: An adversarial winograd schema challenge at scale, Commun. ACM 64 (9) (2021).

[38] M.P. Marcus, M.A. Marcinkiewicz, B. Santorini, Building a large annotated corpus of english: The penn treebank, Comput. Linguist. (1993).

[39] S. Merity, C. Xiong, J. Bradbury, R. Socher, Pointer sentinel mixture models, in: International Conference on Learning Representations, 2017.

[40] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P.J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, J. Mach. Learn. Res. (2020).

[41] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, T.B. Hashimoto, Stanford alpaca: An instruction-following LLaMA model, 2023, https://github.com/tatsu-lab/stanford_alpaca.

[42] H. Tanaka, D. Kunin, D.L. Yamins, S. Ganguli, Pruning neural networks without any data by iteratively conserving synaptic flow, Adv. Neural Inf. Process. Syst. (2020).

[43] V.J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J.S. Gardner, I. Hubara, S. Idgunji, T.B. Jablin, J. Jiao, T.S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A.T.R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, Y. Zhou, MLPerf inference benchmark, 2019, arXiv:1911.02549.

[44] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, S. Han, AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration, in: Proceedings of Machine Learning and Systems, 2024.

[45] T. Dettmers, M. Lewis, Y. Belkada, L. Zettlemoyer, LLM.int8(): 8-bit matrix multiplication for transformers at scale, in: Advances in Neural Information Processing Systems, 2022.

[46] V. Sanh, L. Debut, J. Chaumond, T. Wolf, DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter, 2020.

[47] Nvidia, Nemotron-4 340b technical report, 2024, arXiv:2406.11704.

[48] B. Hassibi, D.G. Stork, G. Wolff, Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, 1993.

[49] N. Zheng, B. Lin, Q. Zhang, L. Ma, Y. Yang, F. Yang, Y. Wang, M. Yang, L. Zhou, SparTA: Deep-learning model sparsity via tensor-with-sparsity-attribute, in: 16th USENIX Symposium on Operating Systems Design and Implementation, 2022.

[50] M. Xia, T. Gao, Z. Zeng, D. Chen, Sheared LLaMa: Accelerating language model pre-training via structured pruning, in: International Conference on Learning Representations, 2024.

[51] L. Zhong, F. Wan, R. Chen, X. Quan, L. Li, BlockPruner: Fine-grained pruning for large language models, 2024, arXiv:2406.10594.

[52] X. Men, M. Xu, Q. Zhang, B. Wang, H. Lin, Y. Lu, X. Han, W. Chen, ShortGPT: Layers in large language models are more redundant than you expect, 2024, arXiv:2403.03853.

[53] Y. Cai, W. Hua, H. Chen, G.E. Suh, C.D. Sa, Z. Zhang, Structured pruning is all you need for pruning CNNs at initialization, 2022, arXiv:2203.02549.

[54] M.A. Gordon, K. Duh, N. Andrews, Compressing BERT: Studying the effects of weight pruning on transfer learning, 2020, arXiv:2002.08307.

[55] A. Levine, C. Huang, C. Wang, E. Batista, E. Szymanska, H. Ding, H.W. Chou, J.-F. Pessiot, J. Effendi, J. Chiu, K.T. Ohlhus, K. Chopra, K. Shinzato, K. Murakami, L. Xiong, L. Chen, M. Kubota, M. Tkachenko, M. Lee, N. Takahashi, P. Jwalapuram, R. Tatsushima, S. Jain, S.K. Yadav, T. Cai, W.-T. Chen, Y. Xia, Y. Nakayama, Y. Higashiyama, RakutenAI-7B: Extending large language models for Japanese, 2024, arXiv:2403.15484.

Bailey J. Eccles received the M.Eng degree in Computer Science from Queens University Belfast, UK in 2021. He is currently pursuing a Ph.D. degree in Computer Science at the University of St Andrews, UK. His major interests are in the areas of machine learning, edge computing, model compression, and optimization.

Leon Wong is the Research Collaboration and Engineering Lead for Autonomous Networks Research & Innovation in Rakuten Mobile, Inc. He is currently serving as chairman of ITU-T Focus Group of Autonomous Networks (FG-AN), established under ITU-T Study Group 13 - Future networks and emerging network technologies. He is also the co-chair of FG-AN Ad hoc group for Japan's Telecommunication Technology Committee (TTC).

Blesson Varghese received the Ph.D. degree in computer science from the University of Reading, UK. He is a Reader in computer science at the University of St Andrews, UK, and the Principal Investigator of the Edge Computing Hub funded by Rakuten Mobile, Inc., Japan. He serves as a research theme leader in the UKs flagship National Edge AI Hub. He has held a Royal Society fellowship to British Telecommunications plc and was the recipient of the IEEE Rising Star Award 2021 from the Technical Committee on the Internet. His interests include distributed systems that span the cloud-edge-device continuum.