

Structured Pruning of Small Language Models: An Empirical Study on GQA-Aware Attention and MLP Compression with LoRA Recovery

Bakkou Adnane^{1,*}, Hirchoua Badr¹, Banane Mouad¹, Er-raji Naoufal²

¹ AIMCE Laboratory, École Nationale Supérieure d'Arts et Métiers (ENSAM), Université Hassan II de Casablanca, Casablanca, Morocco.

² Ecole Marocaine des Sciences de l'Ingénieur, Casablanca, Morocco.

* Corresponding author: adnane.bakkou-etu@etu.univh2c.ma. Other authors: hirchoua.badr@ensam-casa.ma, banane.mouad@ensam-casa.ma, na.erraji@gmail.com.

Received: 30 June 2026 Accepted: 18 September 2026 Published: 15 October 2026

DOI: 10.15849/ijasca.171 License: Open access under CC BY 4.0. Authors retain copyright.

Publisher: Al-Zaytoonah University of Jordan (ZUJ), Amman, Jordan.

Cite this article: B. Adnane, H. Badr, B. Mouad, and E. Naoufal, "Structured pruning of small language models: an empirical study on GQA-aware attention and MLP compression with LoRA recovery," *Int. J. Advance Soft Compu. Appl.*, vol. 18, no. 3, pp. 113–139, 2026, doi: <https://doi.org/10.15849/ijasca.171>.

ABSTRACT

Large language models have shown strong performance across many natural language tasks, but their size makes deployment difficult on hardware with limited memory and compute. Compression techniques aim to reduce this cost while preserving as much quality as possible. This study investigates structured pruning applied to a small transformer-based language model that uses Grouped-Query Attention (GQA), a design now common in modern LLMs but rarely addressed explicitly by existing pruning methods. The research problem is therefore how to compress such an architecture along two structural axes, attention heads and feed-forward neurons, without breaking the GQA grouping that ties query heads to their shared key-value projections. We design a two-stage pruning pipeline, guided by data-dependent importance scores computed from attention entropy and mean activation magnitude, followed by a lightweight LoRA fine-tuning step that recovers part of the lost quality without adding inference cost. The pipeline is evaluated on TinyLlama-1.1B across seven configurations covering attention-only pruning, MLP-only pruning, their combination, and LoRA recovery. For TinyLlama-1.1B on WikiText-2 at the ratios we evaluate, the results show that feed-forward pruning offers a more favorable quality-compression trade-off than head pruning, that combining both forms of pruning produces a non-linear quality drop, that LoRA recovery partially restores the lost quality at no extra inference cost, and that inference latency is unchanged by pruning at these ratios, with every configuration including the unpruned baseline falling within a narrow band on our setup. These findings suggest practical implications for practitioners deploying small GQA-based language models, although every measurement was taken on a single GPU, a Tesla P100 for quality and a Tesla T4 for latency, and no CPU, mobile or embedded target was evaluated, and they point to layer-wise adaptive pruning and pruning-quantization combinations as promising directions for further reducing deployment cost.

Keywords: Large Language Models; Structured Pruning; Grouped-Query Attention; Low-Rank Adaptation; Model Compression; TinyLlama.

1 Introduction

Large language models (LLMs) have become a central tool in modern natural language processing. Built on top of the transformer architecture [1], they reach strong performance on a wide range of tasks, from text generation and summarization to question answering and machine translation. This success has pushed both academic research and industrial applications to rely heavily on these models, even when the deployment context is constrained. The main practical issue with LLMs is their size. Recent models span several orders of magnitude, from a few hundred million parameters to well beyond a trillion in the largest systems, which translates into high memory requirements and slow inference [32]. These constraints become a real obstacle when the target environment is an edge device, a real-time pipeline, or a large-scale serving system where every millisecond and every gigabyte matter. As a result, model compression has grown into an active research area, with the goal of reducing the cost of LLMs while keeping their quality acceptable.

Several families of compression techniques have been explored. Quantization lowers the numerical precision of weights and activations. Knowledge distillation transfers the behavior of a large model into a smaller one. Pruning removes parts of the network that are considered less useful. Each family has its own strengths and limitations, and they can sometimes be combined. Among these, pruning has the appealing property of directly targeting redundancy inside the network, which makes the compression interpretable at the architectural level. Pruning itself comes in two main forms. Unstructured pruning removes individual weights and can reach high sparsity ratios, but it usually requires specialized kernels or hardware to translate sparsity into actual speedups. Structured pruning, in contrast, removes whole architectural components such as attention heads or feed-forward neurons. The resulting model is physically smaller and needs no custom kernel [31,33]. Whether it also runs faster depends on the hardware, the matrix dimensions, and the attention implementation. The trade-off is that structured pruning is more aggressive: removing the wrong component can damage the model badly.

A second factor adds complexity to the pruning of recent LLMs. Many modern architectures no longer use standard multi-head attention. Grouped-Query Attention is used in the larger LLaMA-2 models [2], in Mistral [3], in several Qwen models [4] and in TinyLlama [5], although the attention configuration varies within each family. They rely instead on Grouped-Query Attention, or GQA [6], where several query heads share a single key-value pair. This design reduces the size of the key-value cache and improves inference speed, but it also introduces a structural constraint. Attention heads are no longer independent units; they are organized in groups that must remain coherent after pruning. Most existing structured pruning works were designed for the older multi-head attention layout and do not handle this constraint explicitly.

In this work, we study structured pruning under this exact setting. We focus on a small GQA-based language model, TinyLlama-1.1B, and we investigate how its architecture can be compressed while keeping the GQA grouping valid. Our pipeline prunes along two axes. The first prunes attention heads using a hierarchical strategy that selects key-value groups first and then selects query heads inside each retained group. The second prunes neurons inside the feed-forward blocks based on their activation magnitude. We then apply a short LoRA [7] fine-tuning step to recover part of the quality loss, and we merge the LoRA weights into the base model so that no additional cost is paid at inference. The contributions of this paper are the following. First, we describe a structured pruning pipeline that is explicitly designed to respect the GQA grouping constraint, rather than ignoring it or breaking it. Second, we present an empirical analysis of seven configurations covering attention pruning, MLP pruning, their combination, and LoRA-based recovery, all evaluated on the same model and benchmark. Third, we report several practical observations: feed-forward blocks tolerate pruning more easily than attention heads, combined pruning produces non-linear quality drops, LoRA recovery offers a meaningful but partial improvement, and inference latency is unchanged by pruning at these ratios on our hardware. We hope that these observations help practitioners who deploy small language models on hardware with limited resources.

The rest of the paper is organized as follows. Section 2 reviews related work on LLM compression and structured pruning. Section 3 describes our methodology and the GQA-aware pruning pipeline, and Section 4 the experimental setup. Section 5 reports the results across all configurations, Section 6 discusses the main findings and the limitations of our study, and Section 7 concludes the paper and outlines directions for future work.

2 Related Work

This section reviews prior work that is relevant to our study. We organize it around five themes: general LLM compression, the structured versus unstructured pruning distinction, structural pruning of transformer components, the specific case of Grouped-Query Attention, and parameter-efficient fine-tuning for performance recovery. Reducing the cost of large language models has become an active research area in recent years. Three main families of techniques are commonly studied. Quantization lowers the bit-width of weights and activations, with methods like GPTQ [8], SmoothQuant [9], and AWQ [10] showing that even 4-bit representations can preserve most of the original quality. Knowledge

distillation transfers the knowledge of a large teacher model into a smaller student, following the line of work introduced for BERT-like models in DistilBERT [11] and TinyBERT [12]. Pruning, which is the focus of our work, removes parts of the model that are considered less important. These three families are not mutually exclusive, and several recent papers combine them to push compression further. A recent survey by Zhu et al. [13] provides a broader picture of the field and shows how each family contributes to the overall goal of making LLMs cheaper to run.

Pruning methods are usually divided into two categories. Unstructured pruning removes individual weights, often based on their magnitude [14] or on second-order information about the loss surface [15]. This approach can reach very high sparsity levels. SparseGPT [16] showed that LLMs as large as OPT-175B can be pruned to fifty percent sparsity in a single shot with limited quality loss, by formulating pruning as a layer-wise reconstruction problem. Wanda [17] proposed a simpler criterion that combines weight magnitude with input activation norms, without any retraining. The main limitation of unstructured pruning is practical: standard hardware does not benefit from random sparsity unless dedicated kernels or sparse matrix libraries are used. Structured pruning addresses this issue by removing entire components, such as filters in convolutional networks, neurons in feed-forward layers, or attention heads in transformers. The resulting model is physically smaller and runs faster on standard GPUs without any modification to the inference engine. The price to pay is that structured pruning is less flexible: each removed component takes away a coherent block of computation, and removing an important block can damage the model badly. Our work belongs to this second category.

Within transformers, structural pruning has been studied along two main directions: pruning attention heads and pruning feed-forward neurons. The seminal work of Michel et al. [18], titled “Are Sixteen Heads Really Better than One?”, showed that a large fraction of attention heads can be removed from trained transformers with limited impact on quality. This finding was later confirmed and extended by Voita et al. [19], who studied the role of individual heads in machine translation and observed that many of them are redundant. More recent work on LLMs followed the same intuition. LLM-Pruner [20] introduced a one-shot structured pruning framework that uses gradient-based importance scores to identify dependency groups inside the network, and applies LoRA fine-tuning afterward to recover quality. Shortened LLaMA [21] takes a different angle and prunes entire transformer blocks rather than individual heads or neurons. These works share two common assumptions that are worth noting. First, they were designed and tested mostly on models with standard multi-head attention, where each head is an independent unit. Second, they often target large models in the seven to thirteen billion parameter range. Our work moves away from both assumptions: we focus on a small model in the one billion parameter range, and we explicitly address the constraints introduced by Grouped-Query Attention.

Grouped-Query Attention was introduced by Ainslie et al. [6] as an interpolation between standard multi-head attention and multi-query attention. The idea is to share each key-value pair across several query heads, which reduces the size of the key-value cache and accelerates inference. GQA is used in many recent models, including LLaMA-2 [2], Mistral [3], Qwen [4], and TinyLlama [5]. The choice is made per model rather than per family: within LLaMA-2 it applies to the 34B and 70B variants, while the 7B and 13B variants keep standard multi-head attention, as does the original LLaMA release [22]. This trade-off between quality and inference cost explains why GQA appears in many recent model families. From a pruning perspective, GQA introduces a non-trivial constraint. Query heads are no longer independent: they belong to groups, and each group shares a single key-value projection. Removing a query head on its own does not make the attention layer invalid. It does change how many query heads share one key-value pair, and implementations that assume a uniform ratio across groups will then fail. To the best of our knowledge, only a few works have explicitly addressed this constraint, and most of them target large models or use very different pruning strategies. Recent structured pruning work can be read against this constraint. LLM-Pruner [20] groups coupled structures and removes them as units on the strength of gradient information, which in principle keeps a query head together with the projections it depends on, but the LLaMA and Vicuna models it evaluates use standard multi-head attention. Sheared LLaMA [23] learns masks over layers, heads and hidden dimensions and then continues pre-training on the pruned architecture; here again the source model is LLaMA-2-7B, where

each query head owns its key-value pair. FLAP [24] selects channels from the fluctuation of their output features and adds bias terms to compensate for what was removed, without a rule tying the surviving query heads to a shared key-value projection. SliceGPT [25] acts on the residual stream through orthogonal transforms rather than on head groups, and Shortened LLaMA [21] removes whole blocks, which avoids the question entirely. EfficientLLM [26] is the closest to our setting: it prunes during pre-training, and an appendix states the grouping rule explicitly, removing a key-value pair only once all the query heads attached to it have been removed. The difference with the present work lies in the setting rather than in the principle. EfficientLLM assumes a pre-training budget, whereas we start from a released checkpoint and a calibration set of two hundred sequences, and we make the choice of key-value group a scored decision of its own instead of a consequence of which query heads happen to survive. We therefore do not claim that GQA-aware pruning is new. What we report is a hierarchical variant that scores key-value groups first and query heads second, applied to a small released GQA model, together with a comparison against random, magnitude-based and position-based selection under one evaluation protocol.

After pruning, the model usually loses some quality. Recovering this loss through full fine-tuning is expensive, especially for LLMs. Parameter-efficient fine-tuning methods provide a cheaper alternative by updating only a small fraction of the parameters. Among these methods, Low-Rank Adaptation, or LoRA [7], has become one of the most popular. LoRA inserts pairs of low-rank matrices into existing linear layers and trains only these matrices, while keeping the original weights frozen. The number of trainable parameters is typically below one percent of the total model size, which makes the method very efficient. LoRA has been used successfully as a recovery step after structured pruning. LLM-Pruner [20] applies LoRA fine-tuning on a small calibration set after pruning, and reports a meaningful recovery of the lost quality. A useful property of LoRA is that the trained low-rank matrices can be merged into the base weights after fine-tuning, which means that the final model has the same shape and the same inference cost as a standard pruned model. We adopt the same recovery strategy in our pipeline. LoRA is chosen because its updates can be merged into the base weights, which keeps the recovered model identical in shape and inference cost to the pruned one. Our experiments show that it recovers part of the lost quality; they do not establish that it is the best available recovery method. To make the positioning of our work clearer, Table 1 summarizes how a few representative pruning approaches relate to our setting. The comparison focuses on four practical criteria: the type of pruning, whether the method explicitly handles the GQA grouping constraint, whether a recovery step is included, and the typical scale of the models studied.

Table 1. Comparison of representative LLM pruning approaches with our work.

Method	Pruning Type	GQA Handling	Recovery Step	Target Scale
SparseGPT [16]	Unstructured	Not explicitly addressed	None	Large (up to 175B)
Wanda [17]	Unstructured	Not explicitly addressed	None	Large (7B–70B)
LLM-Pruner [20]	Structured (groups)	Not explicitly addressed	LoRA	Large (7B)
Shortened LLaMA [21]	Structured (blocks)	Not explicitly addressed	LoRA	Large (7B–13B)
Ours	Structured	Explicit	LoRA	Small (1.1B)

The table highlights two aspects that distinguish our study from prior work. The first is the explicit treatment of the GQA grouping constraint, which is left implicit in most existing approaches because they target older or larger models with standard multi-head attention. The second is the focus on small language models in the one billion parameter range, a regime that has received less attention in the structured pruning literature than the seven billion parameter range and above. The works reviewed in this section cover the main techniques and ideas that are relevant to our study. The next section presents our methodology in detail and explains how we combine these ideas to build a structured pruning pipeline that respects the GQA grouping constraint.

3 Method

3.1 Problem Formulation

We consider a pre-trained transformer-based language model f with parameter set θ . The model is built from L stacked decoder layers. Each layer combines a multi-head attention module with a position-wise feed-forward block. The hidden dimension is denoted d_{model} , and the intermediate dimension of the feed-forward block is denoted d_{ff} . The attention module follows the Grouped-Query Attention design [6]. It uses H_q query heads and H_{kv} key-value heads, with the constraint that H_q is a multiple of H_{kv} . Each key-value head is shared across $g = H_q / H_{kv}$ query heads, which we call a query group. We write g for this ratio throughout, and reserve r for the LoRA rank introduced later in this section. The dimension of a single head is denoted d_h , and we have $H_q \cdot d_h = d_{model}$.

Our goal is to obtain a smaller model f' such that the number of parameters of f' is strictly smaller than that of f , while keeping the language modeling quality close to the original. Quality is measured by perplexity on a held-out evaluation set. We also track inference latency and the total parameter count, since both reflect the practical cost of running the model. The compression operates along two structural axes. The first axis removes a fraction of attention heads under the constraint that the GQA grouping remains valid after pruning. The second axis removes a fraction of neurons inside the feed-forward block. Both operations modify the actual shape of the weight matrices, so the resulting model is physically smaller. After pruning, a short LoRA fine-tuning step is applied to recover part of the lost quality. The LoRA weights are merged into the base model so that no additional module is required at inference time. This formulation reflects three deliberate choices. We work at the architectural level rather than at the level of individual weights, so that the compression translates into actual gains on standard hardware. We treat the GQA grouping as a hard constraint, which means that pruning cannot break the relationship between query heads and their shared key-value pair. And we keep the recovery step intentionally lightweight, so that pruning itself remains the main source of compression rather than a long retraining phase.

3.2 Pipeline Overview

The pipeline is organized as a sequence of independent stages, each producing an artifact that is reused by the next stage. This modular design makes the experiments easier to manage and allows isolated evaluation of each pruning component. The first stage operates on the original pre-trained model and produces two outputs. It collects an importance score for every attention head and an importance score for every neuron inside the feed-forward blocks. These scores are computed by passing a small calibration subset of the training data through the model and recording the relevant statistics. The scores are saved on disk so that the next stages can reuse them without re-running the calibration. The second stage applies structured pruning to the attention heads. Using the head importance scores from the first stage, it selects the heads to keep in each layer under the GQA grouping constraint, then physically resizes the query, key, value, and output projection matrices accordingly. The result is a smaller model with fewer attention heads but the same number of feed-forward neurons. The third stage applies structured pruning to the feed-forward blocks. It uses the neuron importance scores from the first stage to select the neurons to keep in each layer, then physically resizes the gate, up, and down projection matrices of every feed-forward block. This stage can be applied either to the original model or to the head-pruned model from the previous stage, which allows us to evaluate both attention-only pruning, MLP-only pruning, and their combination of both. The fourth stage performs a short LoRA fine-tuning step on the pruned model. Only a small number of low-rank parameters are updated, while the original weights remain frozen. After fine-tuning, the LoRA weights are merged back into the base model. The final model has the same architecture as the pruned model from the previous stage, with no additional inference cost. The fifth and last stage evaluates all pruned and recovered configurations on the same benchmark, using a consistent protocol for perplexity and latency measurement. Results from this stage are used in Section 5.

Figure 1 gives a visual summary of the pipeline. The arrows represent the flow of artifacts from one stage to the next.

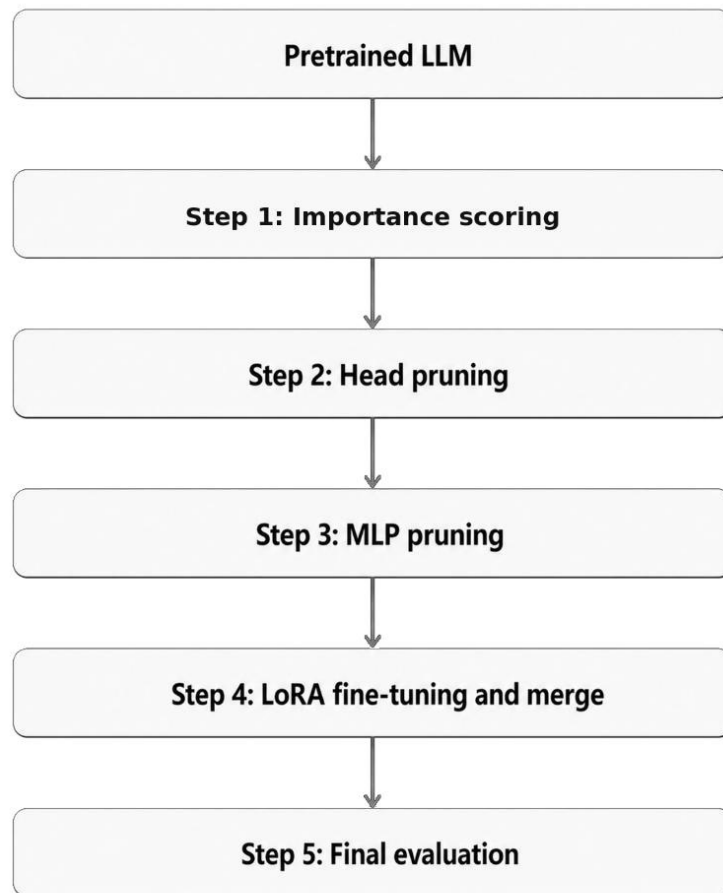


Fig. 1. Overview of the structured pruning pipeline. The pipeline operates in five stages: importance scoring, head pruning, MLP pruning, LoRA recovery and merging, and final evaluation. Source: created by the authors.

3.3 Data-Dependent Importance Scoring

The first stage of the pipeline computes an importance score for each prunable component of the model. These scores are later used to decide which heads and which neurons should be removed. Rather than relying on weight magnitude alone, we use signals that depend on how the model actually processes input data. We therefore call these scores data-dependent: they summarise how the model behaves on real inputs rather than the static values of its parameters. They are behavioural statistics and we do not claim that they measure semantic content directly. Both scoring procedures share the same setup. We use a small calibration set drawn from the WikiText-2 training split, containing N token sequences of fixed length T , and the model is run in inference mode (no gradient computation) on each sequence. During this forward pass, we record the statistics needed to compute the scores. The calibration set is intentionally small, so that the importance scoring step remains cheap compared to the full training of the model.

For attention heads, we use the entropy of the attention distribution as the importance signal. The intuition is that a focused head, which concentrates its attention on a few tokens, encodes a specific relational pattern and is therefore informative. A diffuse head, which spreads its attention almost uniformly across all tokens, makes few clear choices and is more likely to be redundant. Let $A(l, h)_{b,t}$ be the attention distribution produced by head h in layer l , for token position t in calibration sample b . By construction, this distribution is non-negative and sums to one. The Shannon entropy of this distribution is given by Equation (1):

$$H(A(l, h)_{b,t}) = - \sum_{i=1}^T A(l, h)_{b,t,i} \cdot \log(A(l, h)_{b,t,i} + \epsilon) \quad (1)$$

where ϵ is a small constant added for numerical stability. The importance score of head h in layer l is then defined as the average entropy across all token positions and all calibration samples, as shown in Equation (2):

$$S_{head}(l, h) = \frac{1}{N \cdot T} \sum_{b=1}^N \sum_{t=1}^T H(A(l, h)_{b,t}) \quad (2)$$

A lower value of $S_{head}(l, h)$ indicates a more focused head, which we treat as more important. When the head pruning step needs to remove heads, it removes the heads with the highest entropy first, since they correspond to the most diffuse and least specialized attention patterns. This use of attention entropy as an importance signal is consistent with earlier observations on transformer interpretability [19], which showed that some heads specialize in specific syntactic or positional patterns while others remain unfocused. Our scoring procedure does not require gradients or labeled data, which keeps it cheap to compute.

For neurons inside the feed-forward block, we use a different signal. The feed-forward block in TinyLlama follows the SwiGLU design [27], which contains three linear projections: a gate projection, an up projection, and a down projection. The intermediate representation has dimension d_{ff} and is the point where the actual neuron-level activations are exposed. Each neuron in this intermediate space corresponds to one column of the down projection matrix and one column of both the gate and up projection matrices. For each neuron j in layer l , we record the absolute value of its activation just before the down projection, averaged across all tokens and all calibration samples. Formally, if $a(l)_{b,t,j}$ denotes this activation, the importance score of neuron j in layer l is given by Equation (3):

$$S_{mlp}(l, j) = \frac{1}{N \cdot T} \sum_{b=1}^N \sum_{t=1}^T |a(l)_{b,t,j}| \quad (3)$$

The motivation is straightforward: a neuron whose activation is consistently close to zero contributes very little to the output of the feed-forward block, regardless of the magnitude of its weights. Removing such a neuron has a limited effect on the overall computation. A neuron whose activation has a large average magnitude, on the other hand, plays an active role in shaping the output and should be kept. Unlike the attention head case, here a higher score indicates a more important neuron. When the MLP pruning step needs to remove neurons, it keeps the top k neurons with the highest scores in each layer and removes the rest. In practice, the activation $a(l)_{b,t,j}$ is captured using a forward hook attached to the input of the down projection of each layer. This hook is registered before the calibration pass starts and removed afterward, so that it does not interfere with later forward passes. The collected scores are saved on disk as a tensor of shape (L, d_{ff}) and reused by the MLP pruning stage. Both scoring procedures use the same calibration set, run in a single pass through the model. This keeps the cost of the importance estimation step very low compared to the cost of training or fine-tuning the model.

3.4 GQA-Aware Structured Head Pruning

This subsection describes how we prune attention heads in a model that uses Grouped-Query Attention. We first explain the structural constraint that GQA imposes on any pruning strategy. We then present a hierarchical selection procedure that respects this constraint by construction. Finally, we describe how the selected heads are physically removed from the four projection matrices of the attention block.

In standard multi-head attention, every query head has its own key and value projections. The heads are independent, and pruning any subset of them is straightforward: the corresponding rows and columns of the four projection matrices (W_Q, W_K, W_V, W_O) can simply be removed. The remaining heads still form a valid attention layer. In Grouped-Query Attention, this independence no longer holds. The H_q query heads are partitioned into H_{kv} groups, and all the query heads inside the same group share the same key-value pair. A query group of size $g = H_q / H_{kv}$ contains g query heads attached to a single key projection and a single value projection. This sharing is what reduces the size of the key-value cache and gives GQA its inference advantage. From a pruning perspective, this design creates a coupling between query heads and their key-value pair. Removing a query head is allowed, since each

query head still has its own row in W_Q . But removing the key-value pair of a group while keeping some of its query heads would leave those queries without anything to attend to. Conversely, removing some query heads from a group while keeping the key-value pair is fine, but it changes the effective ratio g inside that group and may break implementations that assume a uniform ratio across all groups. A clean solution is to enforce a stronger invariant: after pruning, the new set of query groups should still have a uniform structure. That is, the new model should have H'_q query heads and H'_{kv} key-value heads such that H'_q is a multiple of H'_{kv} , with the same ratio $g' = H'_q / H'_{kv}$ inside each group. This keeps the grouping structure valid without requiring g' to equal the original ratio g , and a change in that ratio may affect the model independently of the pruning itself. This is the constraint we adopt in our pipeline.

To respect the constraint described above, we use a two-level selection procedure. The first level selects which key-value groups to keep. The second level selects which query heads to keep inside each retained group. This hierarchical structure is what makes the resulting model GQA-coherent by construction. Given a target query-head count H'_q and the original number of key-value heads H_{kv} , we first determine the new number of key-value heads H'_{kv} . We first take the proportional estimate $H'_{kv} = \max(1, \text{round}(H_{kv} \cdot H'_q / H_q))$, then decrement it while H'_q is not divisible by H'_{kv} and H'_{kv} is greater than one. The loop always terminates, since $H'_{kv} = 1$ divides any H'_q , so the procedure returns a valid pair for every target. When the proportional estimate already divides H'_q , no adjustment takes place. For TinyLlama-1.1B with $H_q = 32$, $H_{kv} = 4$ and a target of $H'_q = 24$, the estimate gives $H'_{kv} = 3$, which divides 24 and is kept unchanged. This guarantees that the new ratio $g' = H'_q / H'_{kv}$ is an integer. The final H'_{kv} is at least one, so that the attention layer remains functional. For each layer l , we then compute a score for every key-value group. The score of group g in layer l is the average of the head importance scores over the g query heads attached to that group, as shown in Equation (4):

$$S_{kv}(l, g) = \frac{1}{g} \sum_{h \in Q_g} S_{head}(l, h) \quad (4)$$

where Q_g denotes the set of query heads belonging to group g . Since lower head entropy corresponds to higher head importance, lower group scores indicate more important groups. We sort the groups by ascending score and keep the top H'_{kv} groups in each layer. Averaging inside a group has a cost that should be stated. A group is kept or removed as a whole, so a head with a very low entropy score can be discarded because the other heads sharing its key-value pair score poorly. With $H_{kv} = 4$ and $g = 8$, the granularity is coarse: dropping one group removes eight query heads at once on the strength of a single averaged number. Other aggregation functions, such as the minimum score within the group or an average weighted by the spread of the scores, would protect such heads, and a finer variant could keep all four groups and prune query heads only, at the price of a ratio that is no longer the original one. We did not evaluate these alternatives, and we report the choice of the mean as a limitation of the present selection rule. Within each retained group, we then perform a second selection step. The original group contains g query heads, and the new ratio g' tells us how many of them must be kept. We sort the g query heads of the group by ascending head score and keep the top g' heads. The heads that are not selected are removed from the model. The remaining heads in each group are still attached to a single key-value pair, so the GQA structure is preserved. The full selection procedure is described in Algorithm 1. It runs once per layer and produces, for each layer, the index sets of the query heads and key-value heads that should be kept.

Algorithm 1. Hierarchical GQA-Aware Head Selection.

Require: Number of layers L ; original (H_q, H_{kv}) ; target query-head count H'_q ; head importance scores $S_{head}[l, h]$
Ensure: For each layer l : index sets $Q_{keep}[l]$ and $K_{keep}[l]$
1: $H'_{kv} \leftarrow \max(1, \text{round}(H_{kv} \cdot H'_q / H_q))$
2: while $H'_q \bmod H'_{kv} \neq 0$ and $H'_{kv} > 1$ do $H'_{kv} \leftarrow H'_{kv} - 1$
3: $g' \leftarrow H'_q / H'_{kv}$
4: for $l = 1$ to L do
5: Compute group scores: $S_{kv}[l, g] \leftarrow \text{mean over } h \text{ in } Q_g \text{ of } S_{head}[l, h]$

```

6: Sort groups  $g$  by  $S_{kv}[1, g]$  in ascending order
7:  $K\_keep[1] \leftarrow$  first  $H'_{kv}$  groups in the sorted order
8:  $Q\_keep[1] \leftarrow \emptyset$ 
9: for  $g$  in  $K\_keep[1]$  do
    10: Sort heads  $h$  in  $Q_g$  by  $S_{head}[1, h]$  in ascending order
    11: Add the first  $g'$  heads to  $Q\_keep[1]$ 
12: end for
13: end for
14: return ( $Q\_keep, K\_keep$ )

```

Once the index sets $Q_keep[l]$ and $K_keep[l]$ are known for every layer, the actual pruning is performed at the matrix level. The attention block of layer l contains four linear projections that we need to modify: the query projection W_Q , the key projection W_K , the value projection W_V , and the output projection W_O . The query projection W_Q has shape $(H_q \cdot d_h, d_{model})$. Each query head occupies a contiguous block of d_h rows. To prune the query heads that are not in $Q_keep[l]$, we keep only the rows corresponding to the retained heads. The new matrix has shape $(H'_q \cdot d_h, d_{model})$. The key and value projections W_K and W_V have shape $(H_{kv} \cdot d_h, d_{model})$. Each key-value head occupies a contiguous block of d_h rows. We keep only the rows corresponding to the retained key-value heads in $K_keep[l]$. The new matrices have shape $(H'_{kv} \cdot d_h, d_{model})$. The output projection W_O has shape $(d_{model}, H_q \cdot d_h)$. It receives the concatenation of the per-head outputs as its input. Since some query heads are removed, the input dimension of W_O must shrink accordingly. We keep only the columns corresponding to the retained query heads in $Q_keep[l]$. The new matrix has shape $(d_{model}, H'_q \cdot d_h)$. After these four operations, we update the configuration of the attention layer to reflect the new values of H'_q , H'_{kv} , and g' . The pruned model can then be used directly for inference: the attention computation flows through the resized matrices without requiring custom sparse kernels or specialised inference operators, since the layer is still a valid GQA attention layer with smaller projections. The total reduction in attention parameters depends on the chosen target. In our experiments, we target a 25% reduction in the number of query heads, which translates into $32 \rightarrow 24$ query heads and $4 \rightarrow 3$ key-value heads for TinyLlama-1.1B. The detailed parameter counts for all configurations are reported in Section 5.

3.5 Structured MLP Neuron Pruning

This subsection describes how we prune neurons inside the feed-forward blocks. The procedure is simpler than the head pruning described above, because the neurons of an MLP layer are independent units and do not share any structural constraint similar to the GQA grouping.

The MLP block in TinyLlama follows the SwiGLU design. It contains d_{ff} neurons in its intermediate dimension, where $d_{ff} = 5632$ for TinyLlama-1.1B. Each neuron corresponds to one column of the gate projection matrix W_{gate} , one column of the up projection matrix W_{up} , and one row of the down projection matrix W_{down} . For a target reduction ratio ρ_{mlp} , we keep the top $k = \lfloor d_{ff} \cdot (1 - \rho_{mlp}) \rfloor$ neurons in each layer, where ρ_{mlp} is the fraction of neurons to remove. The selected neurons are the ones with the highest mean absolute activation scores, as defined by the importance metric in Equation (3). Concretely, for each layer l , we sort the d_{ff} scores in descending order and we record the indices of the top k values. These indices form the keep set K_l for that layer. The selection is performed independently for every layer, but the same target ratio ρ_{mlp} is applied across all layers. Activation magnitudes vary between layers, so the scores are only compared within a layer and never across layers. We acknowledge that a layer-wise adaptive ratio could improve the results further, since the importance distribution is unlikely to be uniform across the network. The scoring procedure itself supports layer-dependent ratios. We use a uniform ratio in this study to keep the experimental setup simple and to isolate the effect of the pruning ratio itself.

Once the keep set K_l is known for every layer, the actual pruning of the MLP block is performed by resizing its three projection matrices. The gate and up projections W_{gate} and W_{up} both have shape (d_{ff}, d_{model}) . Each row corresponds to one neuron in the intermediate dimension. To prune the neurons that are not in K_l , we keep only the rows whose indices are in K_l . After this operation, both

matrices have shape (k, d_{model}) . The down projection W_{down} has shape (d_{model}, d_{ff}) . Its columns correspond to the neurons in the intermediate dimension. We keep only the columns whose indices are in K_l . After this operation, the matrix has shape (d_{model}, k) . These three resizings must use the same index set K_l , since the gate, up, and down projections are connected by the SwiGLU computation. In SwiGLU, the output of the gate projection is multiplied element-wise with the activated up projection, and the result is then passed through the down projection. Removing a neuron means removing its contribution from all three matrices in a consistent way; otherwise the dimensions would not match. After all layers have been processed, we update the configuration of the model to reflect the new value of d_{ff} . The pruned model can then be used directly for inference, without any custom kernel, since the MLP block is still a standard SwiGLU layer with smaller projections. The MLP pruning step can be applied to two starting points. It can be applied to the original pre-trained model, in which case it produces an MLP-only pruned variant. Or it can be applied to the model already pruned at the attention level in the previous subsection, in which case it produces a model that has been pruned along both axes. Both starting points are used in our experiments, which allows the ablation study presented in Section 5.

3.6 LoRA-Based Performance Recovery

After structural pruning, the model usually loses some quality. Recovering this loss through full fine-tuning is expensive, since it would require updating all the remaining parameters of the model on a large dataset. We instead use a parameter-efficient fine-tuning step based on Low-Rank Adaptation [7], which updates only a small fraction of the parameters and keeps the rest frozen. We choose LoRA for two reasons. It fits the compute budget of this study, and it merges back into the base model, so the recovered checkpoint keeps the architecture and the inference cost of the pruned one. We do not claim that it is the best recovery method. A short round of full fine-tuning, distillation from the unpruned model, or a longer schedule on in-domain text could each recover more, and we did not compare against them. The results below therefore show that LoRA recovers part of the loss, not that it recovers as much as can be recovered. LoRA replaces the update of a linear layer by a low-rank decomposition. Let W be the weight matrix of a linear layer in the pruned model, with m rows and n columns. During LoRA fine-tuning, this matrix is kept frozen, and a learnable update ΔW is added on top. The update ΔW is parameterized as the product of two low-rank matrices, as shown in Equation (5):

$$\Delta W = B \cdot A \quad (5)$$

where A has shape (r, n) and B has shape (m, r) , with r much smaller than both m and n . The forward pass becomes, as given by Equation (6):

$$y = Wx + (\alpha/r) \cdot BAx \quad (6)$$

where α is a scaling constant that controls the magnitude of the LoRA contribution. Only A and B are updated during training; the original matrix W never changes. For a typical choice of $r = 8$, the number of parameters added by a single LoRA module is $r \cdot (m + n)$, which is much smaller than the $m \cdot n$ parameters of the original matrix. The total number of trainable parameters in our setup is below one percent of the total model size. This makes the recovery step cheap to run on the same hardware that was used for inference, without any need for distributed training or large memory budgets. LoRA can in principle be applied to any linear layer in the model. In our experiments, we apply it to all the linear projections inside the transformer blocks. For the attention block, this includes the query, key, value, and output projections. For the feed-forward block, this includes the gate, up, and down projections. Other parts of the model, such as the embedding layer and the language modeling head, are not modified. The LoRA rank is set to $r = 8$, the scaling factor to $\alpha = 16$, and a small dropout of 0.05 is applied to the LoRA path during training. These values follow common defaults in the literature and were not tuned further. We did not search for optimal hyperparameters, since the goal of this step is recovery rather than full retraining. The fine-tuning data is a subset of the Alpaca instruction-tuning dataset [28], from which we sample 5,000 examples. Each example is formatted as an instruction followed by an optional input and a target response. The model is trained with the standard causal language modeling objective on this formatted text. The choice of an instruction dataset, rather than the WikiText-2 training split

itself, is motivated by the diversity of the prompts in Alpaca, which exposes the pruned model to a broad range of linguistic contexts. We discuss the implications of this choice in Section 6. Training is performed for one epoch, with a per-device batch size of one and a gradient accumulation of sixteen steps, which gives an effective batch size of sixteen. The learning rate is set to 2×10^{-4} , the optimizer is AdamW, and mixed-precision training in float16 is used to fit the run on a single 16 gigabyte GPU. The maximum sequence length during fine-tuning is five hundred and twelve tokens.

After fine-tuning, the LoRA weights A and B are merged back into the base weight matrix W of each modified layer. The merge replaces W as shown in Equation (7):

$$W' = W + (\alpha/r) \cdot BA \quad (7)$$

which is computed once and stored. After this operation, the modified layers no longer contain any LoRA module, and they look exactly like standard linear layers with the new weight matrix W' . The model can then be loaded and used for inference without the LoRA library and without any additional computational cost compared to the pruned model obtained before recovery. This merging step is what makes the recovery free of additional inference-time architectural overhead. The total parameter count of the recovered model is identical to the parameter count of the corresponding pruned model, and the latency is also identical, since the matrix shapes are unchanged. The only difference is in the values of the weights, which now contain the recovery information learned by LoRA.

4 Experimental Setup

This subsection describes the experimental setup used to evaluate the pruning pipeline introduced above. We first present the base model, then the datasets used for calibration, recovery, and evaluation. We list the pruning configurations that we compare, give the training hyperparameters, and define the metrics that we report. The hardware and software environment used to run all experiments is described at the end.

4.1 Base Model

All experiments are performed on TinyLlama-1.1B [5], a small open-source language model that follows the LLaMA architecture. We use the intermediate checkpoint at training step 1.43M, which corresponds to approximately three trillion training tokens. The model has 22 transformer decoder layers and a hidden dimension of $d_{model} = 2048$. Each attention block uses Grouped-Query Attention with $H_q = 32$ query heads and $H_{kv} = 4$ key-value heads, which gives a query-to-key-value ratio of $g = 8$. The feed-forward block uses the SwiGLU design with an intermediate dimension of $d_{ff} = 5632$. The total number of parameters is approximately 1.1 billion. We chose TinyLlama for three reasons. First, it uses Grouped-Query Attention, which is the architectural feature that motivates our study. Second, its size sits in the regime of small language models, which are particularly relevant for deployment on hardware with limited memory and compute. Third, the model is fully open-source, with weights and training details available to the public, which makes the entire experimental pipeline reproducible.

4.2 Datasets

We use two datasets in our experiments, each playing a different role. The WikiText-2 raw dataset [29] is used both for calibration and for final evaluation. WikiText-2 is a standard benchmark for language modeling, built from verified Wikipedia articles. Its training split is used to compute the importance scores described earlier, by sampling a small subset of the data and running it through the model in inference mode. The test split is used to compute the perplexity of every pruned and recovered configuration. For the calibration step, we use 200 sequences of 256 tokens each, sampled without replacement from the training split with the fixed random seed 0. We chose this budget to keep the calibration cheap relative to the rest of the pipeline. We did not vary the size of the calibration set, so we do not claim that 200 sequences is a stability threshold. We did repeat the calibration with three independent samples of that size. The resulting rankings agree closely: the mean Spearman correlation across the 22 layers is 0.999 for attention heads and 0.988 for feed-forward neurons, with minima of

0.996 and 0.970. The scores are therefore stable with respect to the sample, at this budget. For the evaluation step, we use the full test split with a sliding-window protocol, described below. The recovery step uses a subset of the Alpaca instruction-tuning dataset [28]. Alpaca contains around fifty-two thousand instruction-response pairs generated using a teacher model, covering a wide range of topics and task formulations. From this collection, we sample 5,000 examples with the fixed random seed 42 for reproducibility. Each example is formatted as an instruction, an optional input, and a target response. The text is then tokenized with a maximum sequence length of 512 tokens, and the model is fine-tuned with the standard causal language modeling objective on this formatted data. The reasons for using an instruction dataset rather than the WikiText-2 training split for recovery are discussed in Section 6.

4.3 Pruning Configurations

We compare seven pruned and recovered configurations against the unpruned baseline, summarized in Table 2. The first configuration is the unpruned baseline. Three configurations apply pruning along a single axis: one for head pruning only, and two for MLP neuron pruning at two different ratios. Two configurations apply both forms of pruning together, by applying MLP pruning on top of an already head-pruned model. The last two configurations correspond to the LoRA recovery applied to the two combined-pruning variants.

The head pruning ratio is fixed at twenty-five percent, which translates into a reduction from 32 to 24 query heads and from 4 to 3 key-value heads in each layer of TinyLlama. The MLP pruning ratios are ten and twenty percent, applied uniformly across all layers. We do not search over a finer grid of ratios in this study, since the goal is to characterize the trade-offs of representative configurations rather than to find the optimal point.

Table 2. Pruning configurations evaluated in this work.

Tag	Head Pruning	MLP Pruning	LoRA Recovery	Description
Baseline	–	–	–	Original TinyLlama-1.1B
Heads25	20%	–	–	Heads only
MLP10	–	10%	–	MLP only
MLP20	–	20%	–	MLP only
Heads25+MLP10	20%	10%	–	Combined
Heads25+MLP20	20%	20%	–	Combined
Heads25+MLP10+LoRA	20%	10%	✓	Combined + recovery
Heads25+MLP20+LoRA	20%	20%	✓	Combined + recovery

4.4 Training Hyperparameters

The LoRA recovery step uses the hyperparameters described above. We summarize them here for reference. The LoRA rank is $r = 8$, the scaling factor is $\alpha = 16$, and the dropout applied to the LoRA path is 0.05. LoRA modules are inserted into all the linear projections of the transformer blocks, including the four projections of the attention block and the three projections of the feed-forward block. Training runs for one epoch, with a per-device batch size of one and a gradient accumulation of sixteen, which gives an effective batch size of sixteen. The optimizer is AdamW with a learning rate of 2×10^{-4} . Mixed-precision training in float16 is used throughout, both to fit the model on the available GPU memory and to speed up training. The maximum sequence length during fine-tuning is 512 tokens. We do not perform any hyperparameter search. The values are taken from common LoRA defaults reported in the literature, which is consistent with our intent of using LoRA as a recovery step rather than as a full retraining stage.

Table 3 reports the parameter efficiency of the LoRA recovery step for the Heads25+MLP10 configuration. The total reported here counts the LoRA adapters while they are still separate from the base model. It is therefore larger than the 971.9 million of Table 8, which reports the merged checkpoint: 971,909,120 base parameters plus 5,807,296 adapter parameters give the 977,716,416 below. Merging removes the difference and leaves the pruned model at its original size.

Table 3. Parameter efficiency of LoRA fine-tuning for Run 1, Heads25+MLP10.

Metric	Value
Total parameters	977,716,416
Trainable parameters (LoRA)	5,807,296
Trainable fraction	0.594%

Table 4 reports the parameter efficiency of the LoRA recovery step for the Heads25+MLP20 configuration. The same convention applies: 895,809,536 base parameters plus 5,510,032 adapter parameters give the 901,319,568 below, and merging returns the model to 895.8 million.

Table 4. Parameter efficiency of LoRA fine-tuning for Run 2, Heads25+MLP20.

Metric	Value
Total parameters	901,319,568
Trainable parameters (LoRA)	5,510,032
Trainable fraction	0.611%

4.5 Evaluation Metrics

We report three metrics for each configuration: language modeling quality, parameter count, and inference latency. Quality is measured by perplexity on the test split of WikiText-2. We use a sliding-window evaluation protocol with a context length of 1024 tokens and a stride of 512 tokens. This protocol is the standard one used in prior LLM evaluation work [16], [17]. For each window, the loss is computed only on the tokens that were not seen in the previous window, which avoids double-counting the overlap. The final perplexity is the exponential of the average per-token loss. A lower perplexity indicates a better fit on the held-out text. The perplexity of the original TinyLlama baseline establishes the reference point, and we report the perplexity of each pruned and recovered configuration alongside the increase relative to this baseline. The parameter count of each configuration is computed by summing the size of all the weight tensors in the model. We report both the absolute count, in millions of parameters, and the parameter reduction relative to the baseline, defined as the percentage of parameters removed relative to the baseline. We use this convention throughout the paper, including in Figure 4. This is a parameter count and not a measurement of memory use, which also depends on activations, framework overhead, and the key-value cache. The compression ratio reflects the actual reduction in model size, since all the pruning operations described above modify the shapes of the weight matrices physically. There is no masking or simulation: the smaller model is structurally smaller. We also report measured memory rather than parameter counts alone. On our setup the weights occupy 2098 MiB for the baseline, 1953 MiB for MLP10 and 1808 MiB for MLP20, and peak inference memory follows the same ordering. The key-value cache is a separate matter: it requires 22.0 KiB per token for the baseline and for both MLP configurations, since feed-forward pruning leaves the number of key-value heads untouched. Only attention pruning reduces it, by one quarter when the key-value heads go from four to three. The two pruning axes therefore act on different parts of the memory budget, which matters when the context is long or the batch is large. Latency is measured in milliseconds per generated token. For each configuration, we run the model in greedy decoding mode on the fixed prompt “Write a short paragraph about pruning.”, generate 64 new tokens, and record the wall-clock time. Generation uses a batch size of one with the key-value cache enabled, and tokenisation is performed outside the timed region. Each configuration is measured with three prompt lengths of 6, 45 and 138 tokens, and with two generation lengths of 64 and 256 tokens, giving six operating points. Each point is timed 20 times for

the shorter generation and 10 times for the longer one, after three warm-up runs. Calls to `torch.cuda.synchronize()` bracket the timed region, since CUDA kernels are asynchronous and wall-clock timing without them would measure queue submission rather than computation. Model loading is excluded. The configurations are measured in randomised order within a single session, and the unpruned baseline is measured both first and last as a control on ordering effects. We report the mean, the standard deviation and a 95% confidence interval for every point. The variability across runs is small for the configurations we tested, but a larger number of runs and a wider variety of prompts would be needed to obtain tight confidence intervals.

4.6 Hardware and Software Environment

All experiments are hosted on the Kaggle platform, on two GPU types. Importance scoring, pruning, LoRA recovery, perplexity and zero-shot evaluation run on a single NVIDIA Tesla P100 with 16 gigabytes of memory, under Python 3.12.13, PyTorch 2.5.1 with CUDA 12.1, Transformers 5.0.0, PEFT 0.19.1, Accelerate 1.13.0 and Datasets 5.0.0. Latency is measured separately on a single NVIDIA Tesla T4 with 16 gigabytes of memory, under PyTorch 2.10.0 with CUDA 12.8 and the same library versions otherwise. We separate the two because latency is the only reported metric that depends on the device; perplexity and zero-shot accuracy are device-independent up to floating-point reordering. The base model is the public checkpoint TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T. The code, the selection indices and the raw measurement files are available at <https://github.com/AdnaneBakkou/structured-pruning-tinyllama>. The model is loaded in float16 precision throughout, both for the calibration step, the pruning operations, and the LoRA fine-tuning. The attention layers use the eager implementation provided by the Hugging Face Transformers library, rather than Flash Attention or other fused kernels, to keep the experimental conditions consistent across all configurations. The implementation relies on PyTorch, the Hugging Face Transformers library, and the PEFT library for LoRA. The full pipeline is organized into five Jupyter notebooks, which makes the experiments easy to reproduce and to inspect. The artifacts produced by each stage, such as importance scores and pruned model checkpoints, are saved on disk and reused by the following stages.

5 Results and Discussion

This section reports the results of all the configurations described in Section 4. We start with the unpruned baseline, which serves as the reference for all subsequent comparisons. We then present the effect of head pruning alone, the effect of MLP pruning alone, and the effect of combining both forms of pruning. We then report the results obtained after LoRA recovery and put all configurations together in a global trade-off analysis. The second half of the section discusses the main findings and the limitations of the study.

All perplexity values are computed on the test split of WikiText-2, using the sliding-window protocol described in Section 4. All latency values are measured on the Tesla T4, and all quality values on the Tesla P100, as described in Section 4. Parameter counts are reported in millions, rounded to one decimal place, and as a percentage of the baseline. The underlying values are exact tensor counts. The baseline holds 1,100,048,384 parameters, and the 1100.0 M of the tables is that number rounded. All percentages are computed from the exact counts rather than from the rounded ones, so they can be reproduced from the checkpoints.

5.1 Baseline Evaluation

The unpruned TinyLlama-1.1B model establishes the reference point for our experiments. Its evaluation results are reported in Table 5.

Table 5. Baseline evaluation of TinyLlama-1.1B on WikiText-2.

Configuration	Parameters (M)	% of baseline	Perplexity	Latency (ms/token)
Baseline (TinyLlama-1.1B)	1100.0	100.0%	7.40	27.43 ± 0.46

The baseline reaches a perplexity of 7.40 on WikiText-2, which is consistent with the values reported for the same checkpoint. As a second check on the evaluation protocol, we ran the zero-shot benchmarks published on the model card and reproduced them closely: HellaSwag 59.15 against 59.20 reported, PIQA 73.0 against 73.29, and ARC-easy 55.7 against 55.25. Perplexity depends on context length, stride, tokenizer and checkpoint, so we state ours explicitly: a sliding window of 1024 tokens with a stride of 512 over the full test split. The inference latency is approximately 37.8 milliseconds per generated token under our setup. These two values define the quality and speed targets that the pruned configurations are compared against.

5.2 Head Pruning

Table 6 reports the results of pruning twenty-five percent of the query heads while preserving the GQA structure, as described in Section 3. The configuration is referred to as Heads25.

Table 6. Effect of head pruning on TinyLlama-1.1B.

Configuration	Parameters (M)	% of baseline	Perplexity	Latency (ms/token)
Baseline	1100.0	100.0%	7.40	27.43 ± 0.46
Heads25	1048.1	95.3%	15.24	29.80 ± 0.93

Two observations stand out from this table. First, the parameter reduction is modest. Pruning twenty-five percent of the query heads removes only about 4.7% of the total model parameters, since the attention block represents only a fraction of the overall model size. Second, the impact on perplexity is large. The perplexity more than doubles, going from 7.40 to 15.24. The degradation is large. It reflects both the selection rule we use and the structural change we apply, so it does not on its own establish that every attention head is irreplaceable. The latency does not improve. The baseline runs at 27.4 milliseconds per token and the head-pruned model at 28.3 to 29.8 depending on the operating point, so removing heads makes generation slightly slower. One plausible reason is that dropping one key-value group out of four leaves an attention layer whose dimensions align less well with the underlying kernels, and the arithmetic saved is smaller than the regularity lost. We note that the quadratic cost of attention in sequence length applies to the prefill phase, not to cached autoregressive decoding, where each new token attends over a linearly growing context. These two observations together suggest that head pruning costs quality without buying latency at this scale. The trade-off becomes more visible when we compare it with MLP pruning in the next subsection.

5.3 MLP Pruning

Table 7 reports the results of pruning the feed-forward blocks alone, with two ratios: ten percent (MLP10) and twenty percent (MLP20). These configurations apply MLP pruning directly on the unpruned baseline, without touching the attention heads.

Table 7. Effect of MLP pruning on TinyLlama-1.1B.

Configuration	Parameters (M)	% of baseline	Perplexity	Latency (ms/token)
Baseline	1100.0	100.0%	7.40	27.43 ± 0.46
MLP10	1023.8	93.1%	8.06	28.23 ± 0.42
MLP20	947.7	86.2%	9.21	26.93 ± 0.42

The picture changes considerably compared to head pruning. With ten percent MLP pruning, the perplexity rises only slightly, from 7.40 to 8.06, while removing about 6.9% of the total parameters. With twenty percent MLP pruning, the perplexity reaches 9.21 for a parameter reduction of 13.8%. In both cases, the perplexity increase is much smaller than the increase observed for the Heads25 configuration, even though more parameters are being removed. This contrast reveals an important property of the architecture: feed-forward blocks tolerate pruning more easily than attention heads. The intermediate dimension of the MLP, which is $d_{ff} = 5632$ in TinyLlama, is nearly three times the hidden dimension $d_{model} = 2048$. This over-parameterization gives the feed-forward blocks a natural redundancy that pruning can exploit without large quality losses. Latency is again unchanged. The MLP10 configuration runs at 27.4 to 28.2 milliseconds per token and MLP20 at 26.9 to 27.5, against 27.4 for the baseline. Every difference falls within the confidence intervals of the measurements. We return to this point later in this section.

5.4 Combined Pruning

When both forms of pruning are applied together, the resulting model is smaller than any of the single-axis variants, but the impact on quality is more severe than the sum of the individual effects. Table 8 reports the results of the two combined configurations.

Table 8. Effect of combined head and MLP pruning on TinyLlama-1.1B.

Configuration	Parameters (M)	% of baseline	Perplexity	Latency (ms/token)
Baseline	1100.0	100.0%	7.40	27.43 ± 0.46
Heads25+MLP10	971.9	88.4%	17.13	27.47 ± 0.39
Heads25+MLP20	895.8	81.4%	20.26	27.69 ± 0.49

The Heads25+MLP10 configuration removes about 11.6% of the parameters and reaches a perplexity of 17.13. The Heads25+MLP20 configuration removes about 18.6% of the parameters and reaches a perplexity of 20.26. Both values are worse than what a simple addition of the individual effects would predict. To make this point concrete, consider the Heads25+MLP10 case. Heads25 alone increases the perplexity from 7.40 to 15.24, a delta of 7.85 points. MLP10 alone increases it from 7.40 to 8.06, a delta of 0.66 points. A purely additive model would therefore predict a combined perplexity of about 15.90, while we observe 17.13. The same holds for Heads25+MLP20, where the additive prediction is 17.06 against an observed 20.26. This non-linearity, which we discuss later in this section, suggests that the two pruning operations interact in a way that is not captured by treating them independently. The latency of the combined configurations is similar to the latency of the single-axis configurations, around 27 milliseconds per token. This is consistent with the observation that the speedup gained from structural pruning saturates quickly, and that further parameter reduction does not translate into a proportional latency reduction.

5.5 LoRA-Based Recovery

The two combined configurations from the previous subsection lose a significant amount of language modeling quality compared to the baseline. We apply the LoRA recovery procedure described in Section 3 to both of them and report the results in Table 9.

Table 9. Effect of LoRA recovery on the combined-pruning configurations.

Configuration	Parameters (M)	% of baseline	Perplexity	Latency (ms/token)
Baseline	1100.0	100.0%	7.40	27.43 ± 0.46

Configuration	Parameters (M)	% of baseline	Perplexity	Latency (ms/token)
Heads25+MLP10	971.9	88.4%	17.13	27.47 ± 0.39
Heads25+MLP10+LoRA	971.9	88.4%	11.44	27.47 ± 0.39
Heads25+MLP20	895.8	81.4%	20.26	27.69 ± 0.49
Heads25+MLP20+LoRA	895.8	81.4%	12.80	27.69 ± 0.49

The LoRA fine-tuning step reduces the perplexity of both configurations by a meaningful margin. The Heads25+MLP10 model improves from 17.13 to 11.44, a reduction of 5.69 points. The Heads25+MLP20 model improves from 20.26 to 12.80, a reduction of 7.46 points. The recovery is larger for the more aggressively pruned configuration, which is consistent with the intuition that a more degraded starting point leaves more room for improvement. It is worth noting that the recovery is not complete. Even after LoRA, the Heads25+MLP10+LoRA configuration reaches a perplexity of 11.44, which is still 55% above the baseline value of 7.40. The Heads25+MLP20+LoRA configuration reaches 12.80, which is 73% above the baseline. These remaining gaps confirm that LoRA acts as a partial compensation rather than as a full restoration of the pruned model. We discuss the possible reasons for this incomplete recovery later in this section. The latency of the recovered models is essentially identical to the latency of the corresponding pruned models. This is expected, since the LoRA weights are merged into the base weights after fine-tuning, as described in Section 3, and the resulting model has the same matrix shapes as before. The merge operation does not add any computational cost at inference time, which means that the quality improvement from LoRA comes essentially for free in terms of inference speed. The training dynamics of the LoRA fine-tuning are illustrated in Figures 2 and 3. Both runs show an overall decreasing trend in training loss over the first epoch, with small increases between individual logging points. The loss starts higher for the Heads25+MLP20 configuration, at 1.98, compared with 1.85 for Heads25+MLP10, which reflects the larger initial degradation of the more aggressively pruned model. The two runs do not end at the same level. Over 280 optimisation steps the first settles near 1.42 in training loss and 1.384 in validation loss, and the second near 1.50 and 1.470, so the more aggressively pruned configuration stays above the other throughout training. Figures 2 and 3 show the loss curves of the two runs.

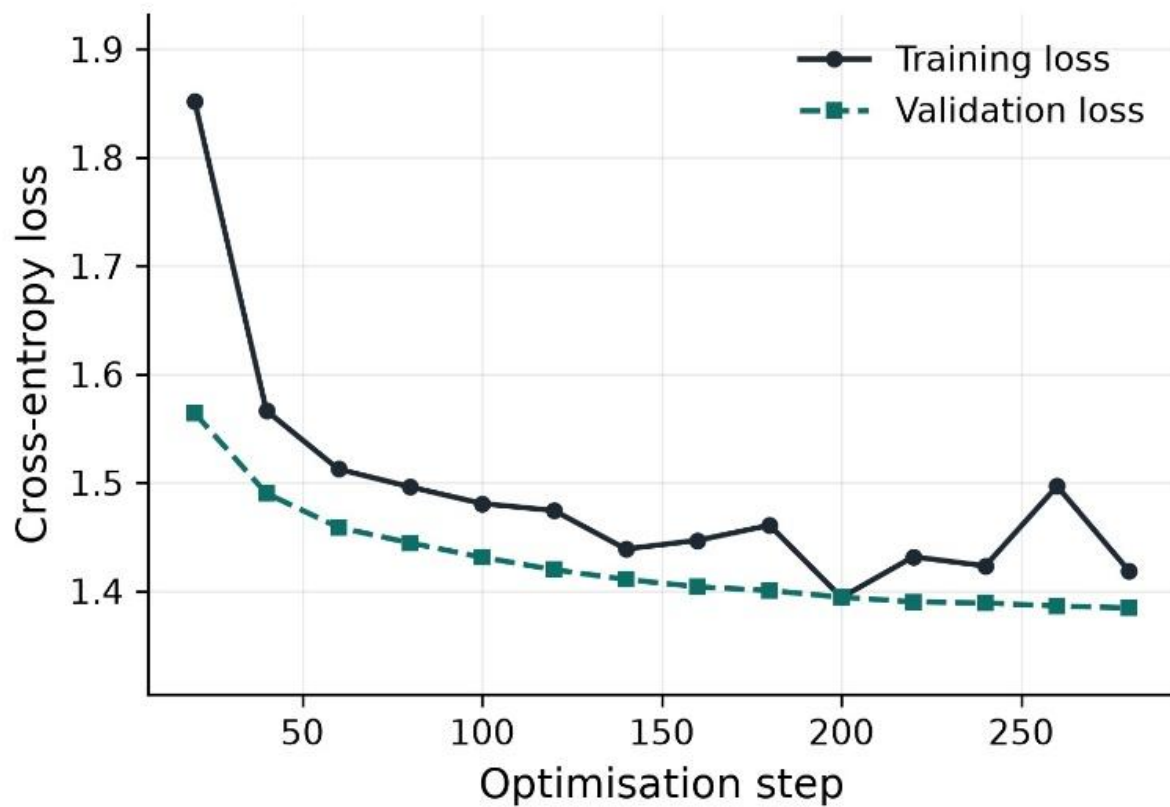


Fig. 2. LoRA training and validation loss for the Heads25+MLP10 configuration. The training runs for one epoch over 5,000 Alpaca examples, with about 0.59% of trainable parameters. The dashed curve is the loss on a held-out five percent of the fine-tuning data, which falls from 1.564 to 1.384 without turning upward. Source: created by the authors.

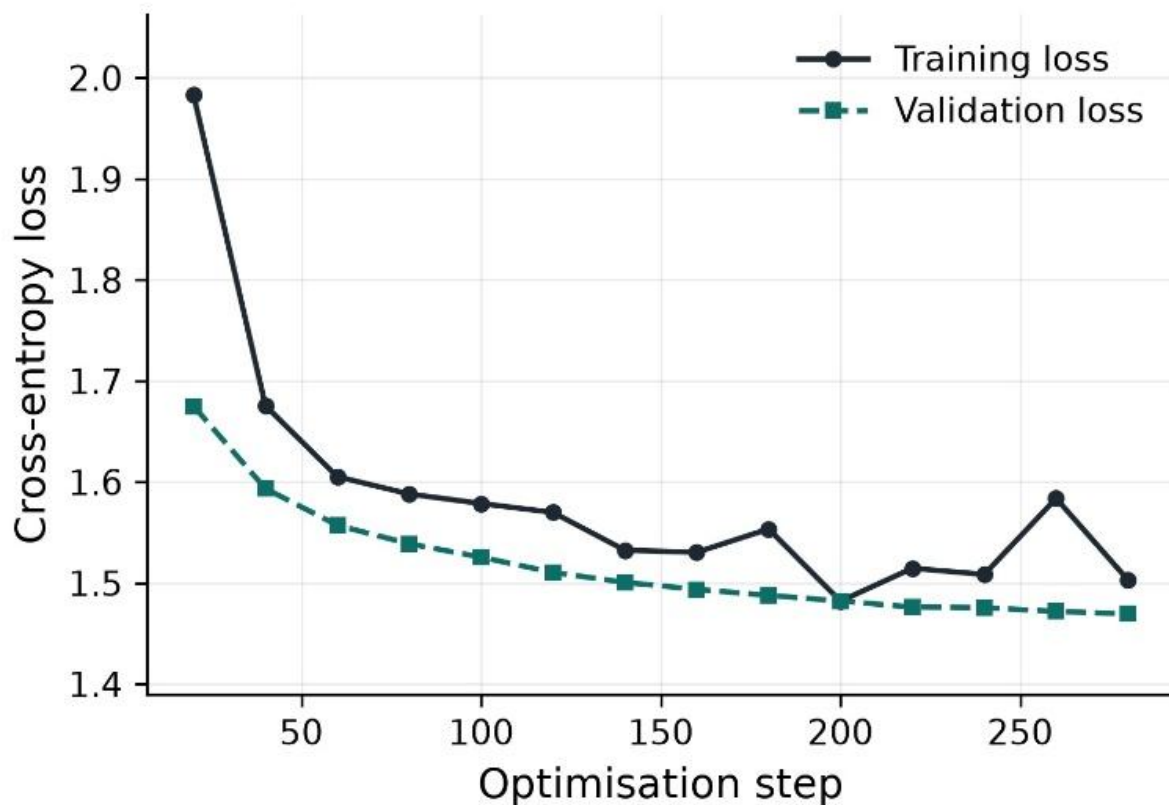


Fig. 3. LoRA training and validation loss for the Heads25+MLP20 configuration. The starting loss is higher, reflecting the larger initial degradation of this more aggressively pruned model. The dashed curve is the validation loss on the same held-out split, which falls from 1.675 to 1.470. Source: created by the authors.

Fine-tuning updated only 0.594% of total parameters (5.8M of 977.7M) via LoRA adapters, with the training loss falling from 1.852 at step 20 to 1.419 at step 280, a single epoch of about 30 minutes of wall-clock time, with the training loss falling from 1.983 to 1.503 over the same number of steps for the Heads25+MLP20 run. Neither curve is monotone: both contain small increases between logging points, and the lowest training value of the second run, 1.482, is recorded at step 200 rather than at the end.

The absence of large fluctuations in the loss curves indicates that the LoRA optimization is stable in our setup. We also tracked the loss on a held-out five percent of the fine-tuning data. It decreased from 1.564 to 1.384 for Heads25+MLP10 and from 1.675 to 1.470 for Heads25+MLP20, without turning upward at any point, so we find no sign of overfitting on this budget. A longer fine-tuning schedule, with more epochs or a larger calibration set, could potentially improve the recovery further. We leave this exploration to future work.

5.6 Comparison of Selection Rules

The results reported so far use the importance scores described in Section 3. To establish that those scores carry information, and not merely that pruning a smaller model degrades it, we compared seven selection rules at identical compression ratios, with the same GQA constraint, the same resizing code and the same evaluation protocol. Only the rule that decides what to remove changes. Table 10 reports the outcome. Alongside our own criteria we implemented a gradient-based rule in the spirit of LLM-Pruner [20], which scores a group by the first-order Taylor term $|dL/dW \cdot W|$ accumulated over the same calibration set. We did not run the authors' code, whose dependency handling targets standard multi-head attention; we implemented their criterion inside our own GQA-aware pipeline so that the comparison isolates the scoring rule.

Table 10. Selection rules compared at identical compression ratios under a single evaluation protocol.

Configuration	Selection rule	Parameters (M)	Perplexity	Increase vs baseline
Heads25	Attention entropy (proposed)	1048.1	15.24	+7.84
Heads25	Gradient-based Taylor	1048.1	10.53	+3.13
Heads25	Causal-normalised entropy	1048.1	15.24	+7.84
Heads25	Inverted entropy	1048.1	26.34	+18.94
Heads25	Weight magnitude	1048.1	27.29	+19.89
Heads25	Random, 3 seeds	1048.1	36.87 $\pm$ 13.29	+29.47
Heads25	Position-based	1048.1	13.55	+6.15
Heads25	Best fixed group choice	1048.1	10.73	+3.33
Heads25	Worst fixed group choice	1048.1	21.26	+13.86
MLP10	Mean activation (proposed)	1023.8	8.06	+0.66
MLP10	Gradient-based Taylor	1023.8	8.04	+0.64
MLP10	Random, 3 seeds	1023.8	9.32 $\pm$ 0.11	+1.92
MLP10	Weight magnitude	1023.8	65.53	+58.13
MLP20	Mean activation (proposed)	947.7	9.21	+1.81
MLP20	Gradient-based Taylor	947.7	9.01	+1.61
MLP20	Random, 3 seeds	947.7	118.96 $\pm$ 178.82	+111.56
MLP20	Weight magnitude	947.7	199.21	+191.81

For the feed-forward blocks the activation criterion is informative. Mean activation reaches 8.06 at ten percent and 9.21 at twenty percent, against 9.32 and 118.96 for random selection averaged over three seeds, and it beats every individual draw. Weight magnitude collapses on this axis, at 65.53 and 199.21. We attribute that failure to the SwiGLU gating: the output of a neuron is a product of two projections, so the norm of either one on its own is a poor proxy for the contribution of the pair. The gradient-based rule is marginally ahead of mean activation, at 8.04 and 9.01, a gap of 0.02 and 0.20 perplexity points for the cost of a backward pass over the calibration set.

For attention heads the picture is different, and we report it as we found it. Attention entropy reaches 15.24, ahead of every random draw, which spans 26.07 to 51.71, and ahead of weight magnitude at 27.29. Reversing the criterion and keeping the highest-entropy heads instead gives 26.34, so the ordering carries real signal rather than being an arbitrary permutation. Normalising the entropy by the number of valid positions under the causal mask leaves the selection unchanged. But the entropy criterion is beaten by three other rules: a rule that ignores the scores and keeps the lowest indices reaches 13.55, the best of the four uniform key-value group choices reaches 10.73, and the gradient-based Taylor rule reaches 10.53. At this granularity, removing one key-value group discards eight query heads at once, and which group is discarded matters more than how the heads inside it are ranked: the spread across the four uniform choices, 10.73 to 21.26, is wider than the gap between our criterion and the best of them.

We therefore state the scope of our claim precisely. The activation criterion is competitive for feed-forward pruning, where it matches a gradient-based rule at a lower cost. The entropy criterion is better than chance for attention heads and its ordering is meaningful, but it is not the best rule available: a gradient-based criterion of the kind used by established structured pruning methods selects better key-value groups on this model, and we do not claim otherwise. The contribution of this paper is the GQA-aware pipeline and the empirical comparison, not the superiority of the entropy score.

5.7 Global Trade-off Analysis

To put all configurations into a single picture, we analyze the trade-offs between quality, compression, and latency across the seven configurations. Two complementary views are useful: a quality versus compression view, which highlights how much perplexity is lost as parameters are removed, and a latency versus parameter count view, which shows how much speedup is actually obtained from the compression. Figure 4 reports the perplexity of every configuration against the corresponding

compression ratio. The baseline appears in the bottom-left corner with no compression and the lowest perplexity. The MLP-only configurations (MLP10, MLP20) follow a smooth trajectory upward and to the right, with a small slope, which means that perplexity rises slowly as more MLP neurons are removed. The Heads25 configuration, in contrast, sits much higher on the perplexity axis, despite having a smaller compression ratio. The combined configurations appear even higher, in the upper-right region of the plot. The two LoRA-recovered configurations are visibly lower than their non-recovered counterparts, which shows the recovery effect graphically.

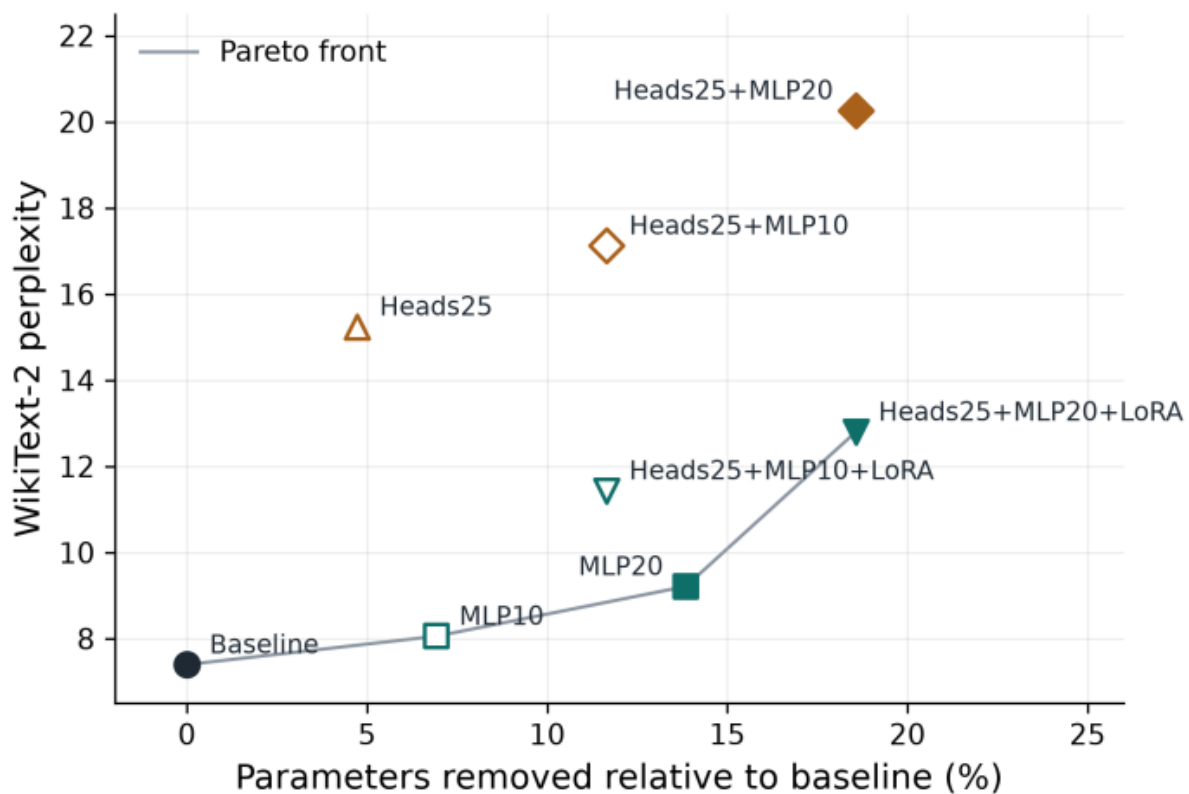


Fig. 4. Perplexity versus parameter compression for all configurations. The two objectives conflict, since perplexity should be low and compression high, so the preferable direction is the lower right. The grey line joins the non-dominated configurations, that is, those that no other configuration beats on both axes at once. LoRA-recovered configurations sit visibly below their non-recovered counterparts. Source: created by the authors.

Reading the same points as a Pareto front sharpens the conclusion. Four configurations are non-dominated: the baseline at 7.40, MLP10 at 8.06 with 6.9% of the parameters removed, MLP20 at 9.21 with 13.8% removed, and Heads25+MLP20+LoRA at 12.80 with 18.6% removed. Every other configuration is dominated, meaning that another configuration is at once smaller and more accurate. Heads25 is dominated by MLP10, which removes more parameters and loses far less quality. Heads25+MLP10 and its recovered version are both dominated by MLP20. Latency does not alter this ordering, since all eight configurations fall inside a band of three milliseconds per token. On this model and this benchmark, head pruning reaches the front only at the highest compression level we tested, and only after LoRA recovery. The shape of the cloud in Figure 4 confirms the observations made in the previous subsections. MLP pruning lies on a favorable region of the trade-off curve. Head pruning lies on a much less favorable region. LoRA recovery shifts the combined points downward, which means that the recovered models offer a better trade-off than the pruned-only models at the same compression ratio. Figure 5 reports the inference latency of every configuration against the absolute parameter count. The baseline is not set apart on the latency axis: at 27.4 milliseconds per token it falls inside the same band as the pruned models. All the pruned configurations cluster in a much narrower region around 27 milliseconds per token, despite spanning a range of parameter counts from about 895 million to about 1050 million. The relationship between parameter count and latency is therefore far from linear in this regime.

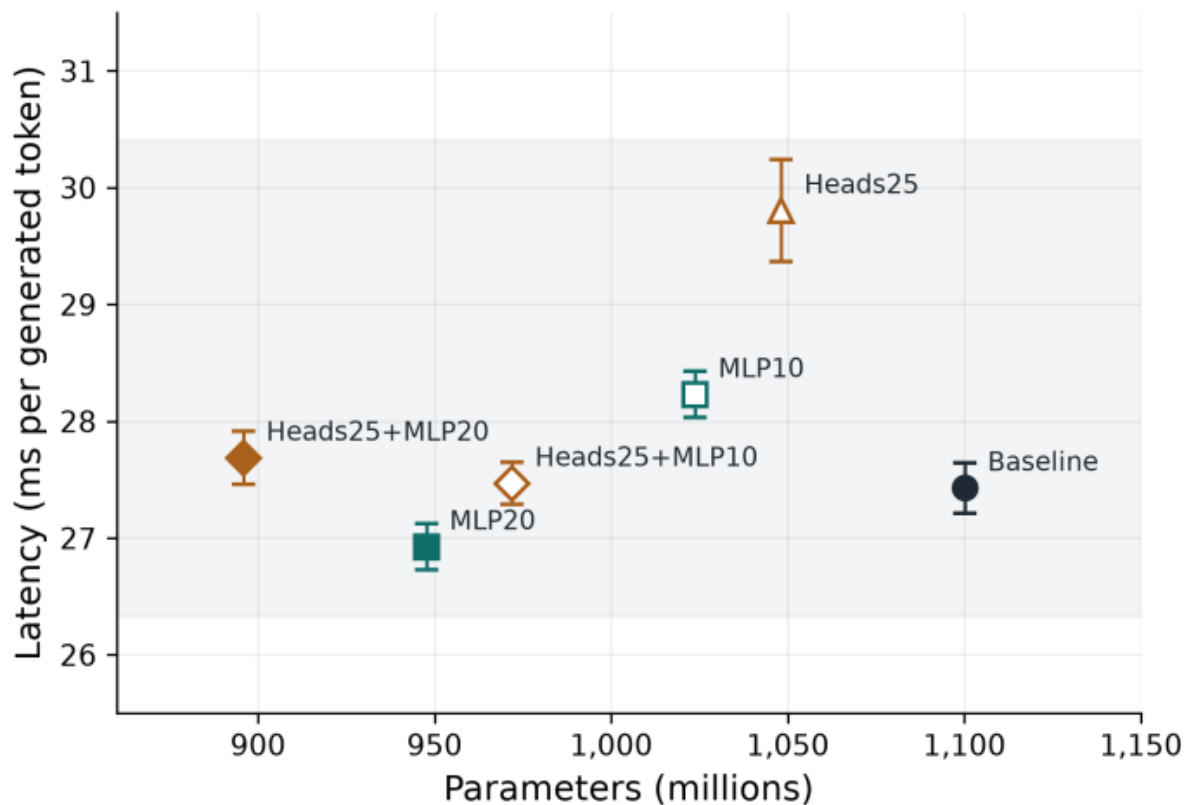


Fig. 5. Inference latency versus parameter count for all configurations. Latency is flat across the range: every configuration, including the unpruned baseline, falls between 26.9 and 29.8 ms/token regardless of parameter count. Error bars show the 95% confidence interval over 20 repetitions. Source: created by the authors.

This second figure shows that latency is essentially flat across the whole range. Every configuration, including the unpruned baseline, sits between 26.9 and 29.8 milliseconds per token, and the differences are of the same order as the standard deviations. Removing 18.6% of the parameters buys no measurable speedup. The control on ordering supports this reading: the baseline measured at the start of the session and at the end differ by 0.04 milliseconds per token, so the flatness is a property of the models and not an artefact of the measurement sequence. A plausible reading is that inference at this scale is dominated by costs that do not scale with the parameter count, such as memory access patterns, kernel launch overhead and the Python-side generation loop. We did not profile the decoding step, so we report this as a hypothesis rather than a measured explanation. Together, Figures 4 and 5 separate what pruning does buy from what it does not. Quality degrades according to the type of component being pruned, with attention heads being far more sensitive than feed-forward neurons. Latency, on the other hand, is largely determined by whether any structural pruning has been applied at all, and is much less sensitive to how much pruning is done after the initial step. These two regimes lead to different design recommendations for practitioners, which we discuss next.

6 Discussion

This subsection interprets the results presented above and discusses what they mean for the design of structured pruning pipelines on small GQA-based language models. We focus on five points: the difference in sensitivity between attention and MLP components, the non-linear behavior of combined pruning, the effectiveness and limits of LoRA recovery, the insensitivity of latency to pruning at these ratios, and the limitations of the present study. The clearest pattern in our results is the gap in pruning sensitivity between the attention block and the feed-forward block. Removing twenty-five percent of the query heads more than doubles the perplexity, while removing ten percent of the MLP neurons causes only a modest increase. Even when twenty percent of the MLP neurons are removed, the perplexity stays well below the value reached by the head-only pruning configuration. This contrast can

be explained by two architectural facts. First, the feed-forward block in TinyLlama has an intermediate dimension of 5632, while the hidden dimension is 2048. This ratio of about 2.75 gives the feed-forward block more internal capacity than the rest of the layer. A larger intermediate dimension is an architectural choice, and on its own it does not prove that the block is redundant. A large fraction of its neurons can be removed without losing essential information, and one possible reason is that the remaining neurons can represent similar signals through different combinations, although we did not test this directly. Second, attention heads are not interchangeable in the same way. Each head can specialize in a particular type of relational pattern, and removing a head removes a specific function rather than a redundant copy of an existing one. This finding is consistent with earlier work on attention head specialization in transformers [19], which observed that some heads encode positional information, others syntactic dependencies, and others lexical co-occurrence. The practical lesson is that the order in which one prunes these two components matters. If the goal is to achieve a moderate compression with minimal quality loss, MLP pruning should be the first choice. Head pruning should be reserved for cases where additional speedup is critical and a measurable quality degradation is acceptable.

The combined-pruning configurations reveal a non-trivial interaction between the two pruning operations. As shown above, the perplexity of the Heads25+MLP10 configuration is 17.13, which is higher than what a purely additive model would predict. The same observation holds for Heads25+MLP20. The combined effect is therefore larger than the sum of the individual effects. The combined model nevertheless remains worse than either single-axis pruning applied alone. The non-linearity does not remove the cumulative cost of pruning two components; it only makes that cost differ from what a purely additive model would predict. A possible explanation for this super-additive behavior is that the model has less spare capacity left after the first pruning step, so the second one removes components that were compensating for it. Under this hypothesis, removing attention heads would cost the model some ability to capture fine-grained contextual interactions, and the feed-forward blocks would absorb part of that loss by reorganizing their internal representations. Pruning the MLP afterwards would then reduce a compensation whose capacity has already been partly consumed, which would leave the overall degradation below the additive prediction. This interpretation is speculative and would need a more detailed analysis to be confirmed, but it offers a plausible direction for understanding the observed behavior.

LoRA fine-tuning recovers a meaningful portion of the quality lost during pruning. The Heads25+MLP10 configuration improves from 17.13 to 11.44, and the Heads25+MLP20 configuration improves from 20.26 to 12.80. These reductions are substantial, especially given that the recovery uses less than one percent of the total model parameters and runs for only one epoch on a small instruction dataset. The recovery is nonetheless only partial. Both LoRA-recovered configurations still have perplexity values that are much higher than the baseline. Several factors contribute to this incomplete recovery, and the control run reported below shows that they do not carry equal weight. The first factor is the structural nature of the loss. When entire attention heads or neuron blocks are removed, the functions encoded by those components cannot be restored by adjusting the weights that remain. LoRA can adjust the surviving weights to reduce the resulting damage, but it cannot recreate the missing structure. The capacity of the recovery is therefore bounded by the capacity of the low-rank update. The second factor is the choice of fine-tuning data. We used a subset of the Alpaca instruction-tuning dataset rather than the WikiText-2 training split. The Alpaca data exposes the model to a broad range of instruction-following contexts, but it does not match the distribution of the WikiText-2 evaluation set. We ran that comparison directly. Repeating the recovery on the WikiText-2 training split, with the same rank, the same hyperparameters and the same budget of 5,000 sequences, brings Heads25+MLP10 to 8.67 and Heads25+MLP20 to 9.47, against 11.44 and 12.80 with Alpaca. Measured as a fraction of the gap between the pruned model and the baseline, the instruction data recovers 58% and 58%, and the matched-domain data recovers 87% and 84%. Most of what Alpaca leaves behind is therefore distribution mismatch and not damage that the low-rank update is unable to repair. Two qualifications belong with this result. Recovering on WikiText-2 train and evaluating on WikiText-2 test is an in-domain setting, so these figures are an upper bound rather than a general-purpose comparison, and we did not measure whether the matched-domain recovery preserves instruction-following behaviour. We

keep Alpaca as the recovery set reported throughout the paper, since it reflects how LoRA is normally used, and we report the control so that the residual gap is not read as structural damage alone. The third factor is the limited training budget. Our LoRA fine-tuning runs for one epoch, which is a deliberately short schedule. A longer schedule with more epochs and a larger calibration set could produce a stronger recovery. The training loss curves shown in Figures 2 and 3 suggest that the loss had not fully plateaued at the end of one epoch, which means there is still room for improvement. Despite these limitations, the recovery step is valuable. The improvement in perplexity comes essentially for free at inference time, since the LoRA weights are merged into the base model after fine-tuning and the resulting model has the same shape and the same latency as the pruned model without LoRA. In a practical deployment scenario, applying LoRA recovery is therefore a low-risk decision: it improved the quality of the pruned model in both configurations we evaluated and does not add any cost at runtime.

One of the clearest observations from the global trade-off analysis above is that inference latency is insensitive to pruning at these ratios. The unpruned baseline runs at 27.4 milliseconds per token, and every pruned configuration falls between 26.9 and 29.8 depending on the operating point, with differences of the same order as the standard deviations. This is visible in Figure 5, where the configurations span more than 150 million parameters along the horizontal axis but only two or three milliseconds along the vertical axis. A plausible explanation is that latency at this scale is dominated by costs that do not follow the parameter count. These include the launch overhead of GPU kernels, memory transfers between the CPU and the GPU, the management of the key-value cache during generation, and the per-token Python overhead in the generation loop. Removing parameters shortens the matrix multiplications, but leaves those fixed costs untouched, and at 1.1 billion parameters they appear to dominate. No profiling was carried out at the kernel level, so this remains an interpretation rather than a measurement. This observation has one practical consequence for practitioners working on small LLMs. If per-token latency is the objective, structural pruning at these ratios is the wrong tool on this hardware, since it costs quality and returns nothing measurable. Its benefits lie elsewhere: in the parameter count, in the memory footprint of the weights, and for attention pruning in the size of the key-value cache, which matters once the context is long or the batch is large.

We close this section with an honest discussion of the limitations of our study. First, all experiments are performed on a single model: TinyLlama-1.1B. This choice is motivated by the model's open-source availability, its use of Grouped-Query Attention, and its small size, which make it a natural fit for our research question. However, the conclusions drawn from a single model may not generalize to other architectures or other parameter scales. The relative sensitivity of attention versus MLP, the non-linear behavior of combined pruning, and the insensitivity of latency could behave differently on larger GQA-based models such as Mistral-7B [3], Qwen-7B [4], or LLaMA-2-13B [2]. Extending the study to a broader set of models is an obvious direction for future work. Second, quality is measured only by perplexity on the WikiText-2 test set. This is a standard benchmark for language modeling, but it does not capture the full range of capabilities of a language model. Downstream task performance, such as question answering, common-sense reasoning, or summarization, can be affected differently by pruning. A more complete evaluation would include benchmarks such as LAMBADA, HellaSwag, ARC, or MMLU. This extension would also help test whether the observed sensitivity differences between attention and MLP pruning hold beyond the language modeling objective. Third, the pruning ratios used in this study are applied uniformly across all layers. Earlier work on transformer pruning has shown that the importance of individual layers varies, with some layers being more sensitive to compression than others. A layer-wise adaptive pruning ratio could likely improve the trade-off between compression and quality. We chose the uniform setting to keep the experimental design simple and to focus on the comparison between configurations, but the uniform assumption is a real limitation. Fourth, the LoRA recovery step is intentionally short, with one epoch of fine-tuning on a small instruction dataset. As discussed above, a longer schedule and a different choice of fine-tuning data could produce a stronger recovery. The limited budget reflects a deliberate choice to keep pruning as the main source of compression, but it also means that the reported recovery numbers should be interpreted as a lower bound on what LoRA could achieve. Finally, latency values are averaged over only three runs on a fixed prompt. We acknowledged this limitation earlier in this section, and we treat the reported latency values

as indicative trends rather than precise benchmarks. A more rigorous protocol would average over a larger number of runs, use prompts of varying lengths, and report confidence intervals. The latency conclusions in this paper should be read with this caveat in mind. These limitations do not invalidate the observations made in the previous subsections, but they define the scope within which those observations should be interpreted. We believe that addressing them in future work would strengthen the empirical understanding of structured pruning on small GQA-based language models.

7 Conclusion

This paper has presented an empirical study of structured pruning on a small GQA-based language model. We described a pipeline that operates in five stages: importance scoring of attention heads and MLP neurons, GQA-aware pruning of attention heads through a hierarchical selection procedure, top-k pruning of MLP neurons, LoRA-based fine-tuning to recover part of the lost quality, and a final evaluation stage. The pruning operations physically resize the weight matrices, so the resulting models are structurally smaller and need no custom kernel. They were not faster in our measurements: on a Tesla T4 with the eager attention implementation, every configuration including the unpruned baseline fell inside a band of three milliseconds per generated token. Gains under other kernels, other GPUs, or optimized serving frameworks may differ. The LoRA weights are merged into the base model after fine-tuning, which means that the recovery step adds no cost at inference time. We evaluated seven configurations on TinyLlama-1.1B and the WikiText-2 benchmark. The results show that MLP pruning offers a more favorable quality-compression trade-off than head pruning, since the over-parameterized feed-forward blocks can absorb a meaningful reduction in their intermediate dimension without large perplexity increases. Head pruning, in contrast, causes a large quality drop even at modest reduction levels, which suggests that attention heads encode functions that are harder to replace through compensation. Combined pruning degrades quality by more than the sum of the two single-axis effects, and LoRA recovery partially restores the lost quality without adding any inference overhead. We also found that latency is unchanged by pruning at these ratios: the unpruned baseline and every pruned configuration fall within a narrow band, so the compression buys parameters and memory rather than speed on this hardware. The contributions of this work can be summarized in three points. First, we described a structured pruning approach that explicitly preserves the GQA grouping constraint, which most prior work either ignores or treats as a side issue. Second, we presented an ablation study covering attention-only pruning, MLP-only pruning, combined pruning, and LoRA recovery on the same model and benchmark, which makes the comparison between configurations clean and consistent. Third, we reported a set of practical observations that can inform the design of compression pipelines for small language models running on hardware with limited resources.

Several directions can extend the present study. The first direction is a layer-wise adaptive pruning strategy. The uniform pruning ratios used in this work do not exploit the variation in importance across layers. A method that allocates more pruning budget to layers with lower average importance, and less to critical layers, would likely improve the quality-compression trade-off. Designing such a method requires a principled way to estimate per-layer sensitivity, and several approaches from the literature could be adapted for this purpose. The second direction is the combination of pruning with quantization. The two compression families are complementary: pruning reduces the number of parameters, while quantization reduces the bit-width of each parameter. Applying a low-bit quantization scheme such as GPTQ [8] or AWQ [10] to a pruned model, or combining pruning with a quantized low-rank recovery step in the manner of QLoRA [30], could yield models that are both smaller and faster, with a moderate additional quality cost. This combined pipeline is particularly attractive for deployment on edge devices, where memory budgets are tight.

DECLARATIONS

Funding. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflicts of interest. The authors declare that there is no conflict of interest regarding the publication of this article.

Ethics approval. [Not applicable — the study involves no human or animal subjects; it uses a publicly released language model checkpoint (TinyLlama-1.1B) and publicly available benchmark datasets (WikiText-2, Alpaca); authors to confirm.]

Informed consent. [Not applicable — no human subjects were involved; authors to confirm.]

Data and code availability. The code supporting this study is publicly available at <https://github.com/AdnaneBakkou/structured-pruning-tinyllama>. The repository contains the pruning pipeline, the importance scoring procedure, the evaluation scripts, the head and neuron selection indices for every configuration, the raw measurement files behind Tables 5 to 10 and Figures 2 to 5, and the exact package versions and random seeds used. The base model is the public checkpoint TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T. The WikiText-2 and Alpaca datasets are obtained from their original public sources. The pruned and recovered checkpoints are not distributed directly because of their size, but they can be regenerated from the released code and the published seeds.

Author contributions. Bakkou Adnane designed the pruning pipeline, implemented the experiments, prepared the datasets, performed the experimental evaluation, analyzed the results, and wrote the manuscript. Hirchoua Badr supervised the work, contributed to the experimental design and methodological development, assisted with the implementation and preparation of the experimental setup, participated in data preparation and analysis, contributed to the interpretation and validation of the results, contributed to the development and refinement of the pruning methodology, prepared the figures and tables, and critically revised the manuscript. Banane Mouad and Er-raji Naoufal contributed to the analysis and interpretation of the results and revised the manuscript. All authors read and approved the final version.

Use of artificial intelligence. During the preparation of this work the authors used a generative AI assistant in order to improve the language and readability of the manuscript, and to support the implementation of the experimental scripts and the generation of the figures. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgements. The authors would like to thank the AIMCE Laboratory at the École Nationale Supérieure d'Arts et Métiers (ENSAM), Université Hassan II de Casablanca, for providing the research environment in which this work was carried out.

REFERENCES

- [1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [2] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023b). LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- [3] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7B. *arXiv preprint arXiv:2310.06825*.
- [4] Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. (2023). Qwen technical report. *arXiv preprint arXiv:2309.16609*.
- [5] Zhang, P., Zeng, G., Wang, T., & Lu, W. (2024). TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.
- [6] Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., & Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- [7] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.
- [8] Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*.
- [9] Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., & Han, S. (2023). SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*.
- [10] Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., & Han, S. (2024). AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Conference on Machine Learning and Systems (MLSys)*.
- [11] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
- [12] Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., & Liu, Q. (2020). TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.
- [13] Zhu, X., Li, J., Liu, Y., Ma, C., & Wang, W. (2024). A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12, 1556-1577. https://doi.org/10.1162/tacl_a_00704
- [14] Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

- [15] LeCun, Y., Denker, J., & Solla, S. (1990). Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [16] Frantar, E., & Alistarh, D. (2023). SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning (ICML)*.
- [17] Sun, M., Liu, Z., Bair, A., & Kolter, J. Z. (2024). A simple and effective pruning approach for large language models. In *International Conference on Learning Representations (ICLR)*.
- [18] Michel, P., Levy, O., & Neubig, G. (2019). Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [19] Voita, E., Talbot, D., Moiseev, F., Sennrich, R., & Titov, I. (2019). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.
- [20] Ma, X., Fang, G., & Wang, X. (2023). LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [21] Kim, B.-K., Kim, G., Kim, T.-H., Castells, T., Choi, S., Shin, J., & Song, H.-K. (2024). Shortened LLaMA: Depth pruning for large language models with comparison of retraining methods. In *Workshop on Efficient Systems for Foundation Models at ICML*.
- [22] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023a). LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- [23] Xia, M., Gao, T., Zeng, Z., & Chen, D. (2024). Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *International Conference on Learning Representations (ICLR)*.
- [24] An, Y., Zhao, X., Yu, T., Tang, M., & Wang, J. (2024). Fluctuation-based adaptive structured pruning for large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(10), 10865-10873.
- [25] Ashkboos, S., Croci, M. L., Nascimento, M. G. D., Hoeffler, T., & Hensman, J. (2024). SliceGPT: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations (ICLR)*.
- [26] Xing, X., Liu, Z., Xiao, S., Gao, B., Liang, Y., Zhang, W., Lin, H., Li, G., & Zhang, J. (2025). EfficientLLM: Scalable pruning-aware pretraining for architecture-agnostic edge language models. *arXiv preprint arXiv:2502.06663*.
- [27] Shazeer, N. (2020). GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*.
- [28] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., & Hashimoto, T. B. (2023). Stanford Alpaca: An instruction-following LLaMA model [Computer software]. GitHub. Retrieved 3 September 2026, from https://github.com/tatsu-lab/stanford_alpaca
- [29] Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2017). Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*.
- [30] Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [31] Hirchoua, B., Ouhbi, B., Frikh, B. (2019). Topic Hierarchies for Knowledge Capitalization using Hierarchical Dirichlet Processes in Big Data Context. In: Ezziyyani, M. (eds) *Advanced Intelligent Systems for Sustainable Development (AI2SD'2018)*. AI2SD 2018. *Advances in Intelligent Systems and Computing*, vol 915. Springer, Cham. https://doi.org/10.1007/978-3-030-11928-7_54
- [32] Badr Hirchoua, Brahim Ouhbi, and Bouchra Frikh. 2017. A new knowledge capitalization framework in big data context. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services (iiWAS '17)*. Association for Computing Machinery, New York, NY, USA, 40–48. <https://doi.org/10.1145/3151759.3151780>
- [33] El Motaki, S., Hirchoua, B. (2022). A Novel Deep Learning Architecture Based IoT Time-Series for Energy Consumption Forecasting in Smart Households. In: Boulouard, Z., Ouassia, M., Ouassia, M., El Himer, S. (eds) *AI and IoT for Sustainable Development in Emerging Countries. Lecture Notes on Data Engineering and Communications Technologies*, vol 105. Springer, Cham. https://doi.org/10.1007/978-3-030-90618-4_6