LangCompress: Language-Aware Compression of Large Language Models

Dieu-Hien Nguyen[1*], Nguyen-Khang Le[1*], Dinh-Truong Do[1], Nguyen Le Minh[1]

[1]Japan Advanced Institute of Science and Technology

Correspondence: {ndhien, lnkhang, nguyenml}@jaist.ac.jp

Abstract

Large Language Models (LLMs) demonstrate strong multilingual capabilities but are costly to deploy due to their size and computational demands. To mitigate this, compression techniques such as pruning and quantization are widely used. However, these methods face two key limitations: (1) they assume access to high-quality instruction or calibration data, which is often unavailable for low-resource languages; and (2) they aim to preserve multilingual generality, making them inefficient for language-specific applications. We introduce **LANGCOMPRESS**, a language-aware compression framework that enhances existing compression methods for targeted deployment. LANGCOMPRESS is method-agnostic and improves state-of-the-art pruning and quantization approaches. It features two core components: an iterative self-supervised pipeline for generating instruction data in the target language, and a vocabulary simplification strategy that reduces the LM head to focus on key tokens. Experiments on perplexity, translation, and summarization tasks show that LANGCOMPRESS improves performance in the target language. The code and data are publicly available.

1 Introduction

Large Language Models (LLMs) are massive neural networks, often comprising billions of parameters and trained on trillions of tokens. Due to the immense cost of training such models from scratch, the community has largely adopted the foundation model paradigm, in which a single pre-trained LLM is reused across a wide range of downstream tasks. Despite their versatility, LLMs are computationally expensive to deploy, both in terms of memory usage and inference time. This high inference cost has driven widespread interest in model compression techniques to reduce the computational and memory requirements of LLMs without significantly degrading performance.

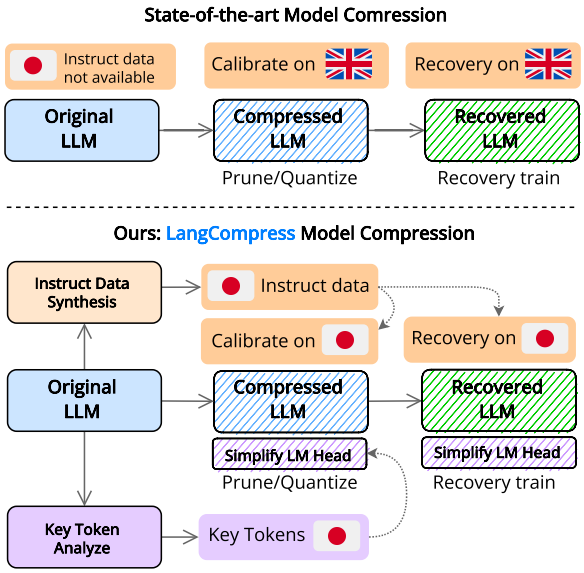


Figure 1: Model compression in a specific language (Japanese in the example) using SOTA compression approach and our LANGCOMPRESS approach.

Two of the most common model compression strategies are *pruning* (Frantar and Alistarh, 2023; Zhang et al., 2024; Sun et al., 2024) and *quantization* (Frantar et al., 2023; Lin et al., 2024). While pruning removes less important weights or neurons from the model, quantization reduces the precision of weights and activations. Pruning approaches typically require recovery fine-tuning to maintain performance, while quantization techniques often need calibration data to determine optimal quantization scales. While these approaches show great efficiency in English, they are not designed to maintain performance in other languages (Figure 1, Top). In real-world applications—especially under resource-constrained conditions—users often wish to deploy LLMs for one or a few specific languages rather than all languages supported by the LLM. Unfortunately, current compression methods are

[*]These authors contributed equally to this work

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 2068–2077

not designed to be language-adaptive. For example, abundant instruction-tuning datasets are available in English, facilitating efficient recovery training and calibration. In contrast, such data is scarce or non-existent in many low-resource languages, hindering the application of existing compression techniques.

Another source of inefficiency lies in the final layer of an LLM: the language modeling (LM) head, which maps hidden representations to a large vocabulary covering many languages. Our analysis reveals that for a given language, a very small subset of the full vocabulary is actually used. For instance, just 5% of the model's vocabulary can often cover more than 95% of the tokens used in a specific language. This insight motivates a new approach to simplify the LM head and adapt it to a specific language, both to reduce model size and improve performance.

We propose **LANGCOMPRESS**, a language-aware compression framework tailored for adapting LLMs to a specific language or a small set of target languages (Figure 1, Bottom). LANGCOMPRESS is method-agnostic and can be integrated into state-of-the-art compression methods, including structured pruning techniques such as SliceGPT and LLM-Pruner, semi-structured pruning like SparseGPT, and quantization approaches such as GPTQ and AWQ.

To address the scarcity of instruction-tuning data in many languages, LANGCOMPRESS uses the LLM itself to iteratively generate synthetic instruction datasets in the target language. Furthermore, we introduce a vocabulary simplification technique that identifies key tokens sufficient for representing the language and modifies the LM head accordingly. This dual approach reduces the model's size and improves its focus on the target language. This study has the following contributions.

- A self-supervised pipeline for generating instruction data in any language, enabling recovery training and calibration in the absence of publicly available resources.
- A method for analyzing and selecting core vocabulary tokens in a target language, and adapting the LM head of an LLM to focus on these tokens.
- Empirical evaluations demonstrating that LANGCOMPRESS can be effectively applied to state-of-the-art pruning and quantization techniques, yielding substantial performance improvements on language-specific tasks.

2 Preliminaries

LLM and Vocabulary. Let $\mathcal{M}$ denote a pre-trained LLM with vocabulary $\mathcal{V}$. The model comprises transformer decoder layers that operate on d-dimensional hidden states. The final layer's output is projected through a language modeling (LM) head with weight matrix $\mathbf{W}_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}| \times d}$, producing logits over the vocabulary.

$\mathcal{M}$ includes a vocabulary dictionary (tokenizer) that bijectively maps tokens to their IDs. At each generation step, the model outputs logits $\mathbf{l} \in \mathbb{R}^{|\mathcal{V}|}$ representing the next-token distribution. Generated token IDs are subsequently mapped back to natural language tokens via the vocabulary dictionary.

Unstructured Pruning. Pruning reduces model size and computation by removing unimportant weights or structures. It can be categorized into unstructured, semi-structured, and structured pruning. Unstructured and semi-structured pruning methods (Hassibi et al., 1993; Li and Louri, 2021; Frantar and Alistarh, 2023; Sun et al., 2024; Zhang et al., 2024; Le et al., 2025) zero out weights in the model, creating sparsity. Semi-structured pruning imposes an $N{:}M$ constraint, where at least N out of every M consecutive weights are pruned. In practice, only semi-structured pruning achieves speedup on compatible NVIDIA hardware (Mishra et al., 2021). We adopt SparseGPT (Frantar and Alistarh, 2023) for semi-structured pruning in our experiments. These methods rely on calibration data, and the data's domain or language can significantly affect pruning outcomes.

Structure Pruning. Structured pruning removes entire model components (layers, attention heads) to reduce model size (Ashkboos et al., 2024; Ma et al., 2023). This approach requires both calibration data during pruning and recovery fine-tuning on task-specific data to restore performance.

Quantization. Quantization compresses models by reducing parameter precision, typically converting 16/32-bit floats to 8/4-bit integers. For a weight matrix $\mathbf{W}$, each element w is mapped to integer values $\hat{w} = \text{round}(w/s) + z$ using scale s and zero-point z, then dequantized as $\tilde{w} = s(\hat{w} - z)$ during inference. Optimal scaling requires calibration data, posing challenges for low-resource languages where such data is scarce.

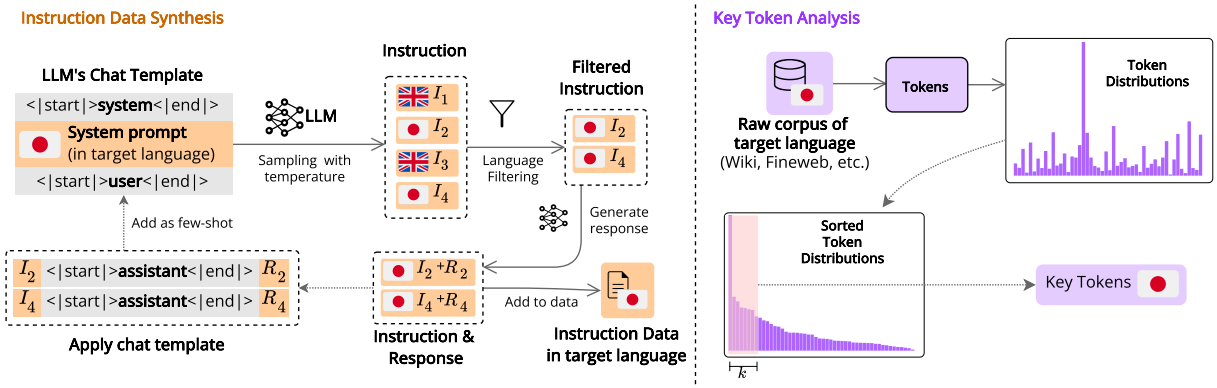


Figure 2: Instruction data synthesis pipeline in the target language, with an example of Japanese language.

3 LangCompress

3.1 Overview

LangCompress is designed to be applied on top of any existing compression technique—such as structured pruning, semi-structured pruning, or quantization—which we refer to as the *backbone method*. The LangCompress pipeline consists of two main components: **instruction data synthesis** and **vocabulary simplification**. At the beginning of the compression process, we use the original pretrained LLM to synthesize instruction data in the target language. This synthetic dataset is subsequently employed in the calibration and recovery fine-tuning stages of the backbone method, enabling effective compression even in low-resource language settings. Concurrently, we perform vocabulary analysis to identify a compact set of *key tokens* that cover the majority of the target language's usable vocabulary. After the compression step is complete, we simplify the language modeling (LM) head by retaining only the parameters corresponding to these key tokens. This modification reduces the model size and enhances its focus on the target language.

3.2 Data Synthesis for Target Language

Figure 2 (left) shows the process of instruction data synthesis for a target language with an example of Japanese. Previous study shows that because of the auto-regressive nature of LLM, with a suitable system prompt, it can generate the instruction when we input only the pre-query templates up to the position reserved for user messages. One problem is that the although the system prompt is in the target language, the LLM does not guarantee to generate the instruction in the target language. We address this using an iterative pipeline to gradually add few-shot examples to the chat template until the LLM's probability of generating instruction data in the target language is stable.

Algorithm 1 Data Synthesis for Target Language

Require: Target language $\mathcal{L}$
Require: System prompt in target language $\mathcal{S}_{\mathcal{L}}$
Require: Original LLM $\mathcal{M}$
Require: Language filter function f_{filter}
Require: Chat template function f_{template}
Require: Maximum few-shot examples K
Require: Number of examples to generate N
Initialize dataset $\mathcal{D} \leftarrow \emptyset$
Initialize few-shot counter $k \leftarrow 0$
Initialize prompt $p \leftarrow f_{\text{template}}(\mathcal{S}_{\mathcal{L}})$
while $|\mathcal{D}| < N$ **do**
  Sample instructions $\mathbf{I} \leftarrow \mathcal{M}(p, \text{temp} = 1.0)$
  Filter instructions $\mathbf{I} \leftarrow f_{\text{filter}}(\mathbf{I}, \text{target} = \mathcal{L})$
  for $i = 1$ to $|\mathbf{I}|$ **do**
    $\mathcal{R}_i \leftarrow \mathcal{M}(\mathbf{I}_i)$ ▷ Generate response
    if $k < K$ **then** ▷ Append few-shot example
      $p \leftarrow p \oplus f_{\text{template}}(\mathbf{I}_i, \mathcal{R}_i)$
      $k \leftarrow k + 1$
    end if
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{I}_i, \mathcal{R}_i)\}$
  end for
end while
return $\mathcal{D}$

Algorithm 2 Key Token for Target Language

Require: Target language $\mathcal{L}$
Require: Original LLM $\mathcal{M}$
Require: Tokenizer $\mathcal{F}_{\text{token}}$ of $\mathcal{M}$
Require: Raw corpus $\mathcal{C}_{\mathcal{L}}$ in language $\mathcal{L}$
Require: Number of desired key tokens k
Tokenize corpus: $\mathbf{T} \leftarrow \mathcal{F}_{\text{token}}(\mathcal{C}_{\mathcal{L}})$
Initialize frequency map: $\mathcal{F} \leftarrow \emptyset$
for each token $t \in \mathbf{T}$ **do**
  $\mathcal{F}[t] \leftarrow \mathcal{F}[t] + 1$
end for
Sort tokens by frequency: $\mathbf{S} \leftarrow \text{SortDescending}(\mathcal{F})$
Select top-k tokens: $\mathcal{V}_{\text{simplify}} \leftarrow \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_k\}$
return $\mathcal{V}_{\text{simplify}}$

Algorithm 1 outlines the instruction data synthesis process. We begin by initializing the system

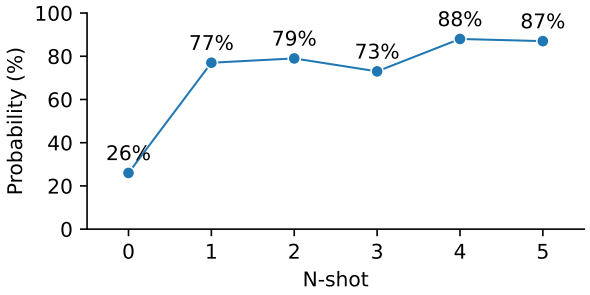


Figure 3: Few-shot Japanese count in the prompt versus probability of Japanese instruction generation on Llama3-8B-Instruct

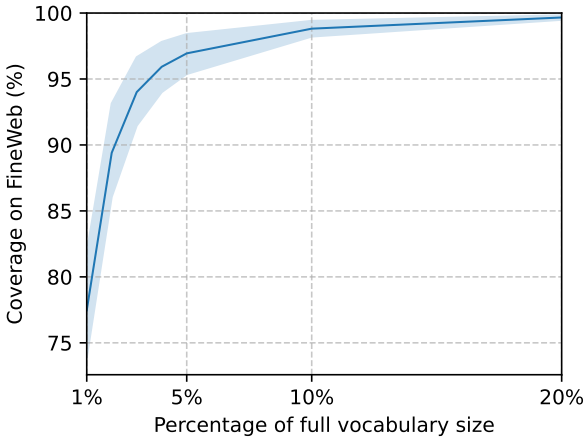


Figure 4: Coverage on FineWeb of the top 1% to 20% highest-frequency vocabulary tokens, averaged across six languages: German, Spanish, French, Japanese, Chinese, and Vietnamese.

prompt in the target language and applying the chat template. The LLM then samples a batch of instructions using a temperature-based decoding strategy. These instructions may appear in various languages, commonly English or the target language. A probabilistic N-gram language filter is applied to retain only those in the target language. The LLM subsequently generates responses for the filtered instructions, forming instruction–response pairs.

These pairs are stored as instruction data and also appended to the prompt as few-shot examples using the chat template, improving the quality of future generations. The process iteratively repeats: each round samples new instructions using the updated prompt, continuing until a desired number of samples is collected. During the early iterations, new few-shot examples are continually added until the probability of generating instructions in the target language exceeds a set threshold. Figure 3 illustrates the relationship between the number of few-shot Japanese examples and the probability of generating instructions in Japanese. In most cases, we use 10 as the maximum number of few-shot examples.

3.3 Vocabulary Simplification

Figure 2 (right) and Algorithm 2 illustrate the vocabulary analysis process used to select key tokens for simplifying the LM head. Starting from a raw target-language corpus (e.g., Wikipedia, multilingual C4, FineWeb), we sample and tokenize the data to compute token frequencies. Sorting these, we select the top k most frequent tokens as key tokens. Figure 4 shows that the top 5% of tokens cover over 95% of tokens in FineWeb. We then reshape the LM Head to retain only rows for these key tokens.

3.4 LangCompress's Compression

With both the instruction data $\mathcal{D}$ and key tokens $\mathcal{V}_{\text{simplify}}$ in the target language, we compress the LLM using a chosen backbone method. To simplify the LM head, we follow prior work on vocabulary reduction (Zhao et al., 2025) by constructing a new matrix $\widetilde{\mathbf{W}}_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}_{\text{simplify}}| \times d}$ from the original LM head $\mathbf{W}_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}| \times d}$:

$$\widetilde{\mathbf{W}}_{\text{LM}}[i,:] = \mathbf{W}_{\text{LM}}[\mathcal{V}_{\text{simplify}}[i],:]$$

where $i = 1, \ldots, |\mathcal{V}_{\text{simplify}}|$. We then replace the original LM head with $\widetilde{\mathbf{W}}_{\text{LM}}$.

Finally, we perform compression using the backbone method and the instruction dataset $\mathcal{D}$. For pruning methods, $\mathcal{D}$ is used for recovery training; for quantization methods, it serves as the calibration set.

4 Experiments

4.1 Models

We evaluate the following model families:

Llama 3 Llama-3-8B, Llama-3-8B-Instruct, and Llama-3.1-8B (Grattafiori et al., 2024), each with a 128K-token vocabulary.

Llama 2 Llama-2-7B with a 32K vocabulary.

Qwen 2.5 Qwen-2.5-7B (Bai et al., 2023) with a vocabulary of approximately 152K tokens.

Phi 3 Phi-3-mini-4k-instruct (Abdin et al., 2024) with a 32K-token vocabulary.

4.2 Tasks and Datasets

Perplexity. We measure the perplexity of LLMs using Wikipedia in the target language. Lower

Method	DE	ES	FR	JA	ZH	VI
Llama3-8B						
Original	5.08	5.13	5.40	6.34	8.46	6.44
GPTQ	49.25	64.52	781.84	14.79	36.60	67.48
GPTQ-LC	30.22	6.41	71.61	9.56	14.45	47.15
AWQ	5.62	5.64	5.90	7.22	9.52	7.34
AWQ-LC	5.55	5.52	5.82	7.16	9.47	7.22
SparseGPT	32.60	22.85	25.01	130.19	136.78	61.87
SparseGPT-LC	14.81	12.60	16.55	22.04	29.28	17.87
SliceGPT	156.19	162.34	86.33	65K	41K	2K
SliceGPT-LC	17.57	14.80	15.25	37.41	87.13	22.48
LLM-Pruner	8.06	7.62	8.03	10.18	14.90	11.25
LLM-Pruner-LC	7.84	7.53	7.87	9.78	13.52	10.17
Llama3.1-8B						
Original	5.03	5.09	5.37	6.34	8.39	6.36
LLM-Pruner	7.87	7.40	7.81	10.10	14.26	10.99
LLM-Pruner-LC	7.62	7.26	7.65	9.78	13.03	9.59
Qwen2.5-7B						
Original	6.22	5.75	6.02	7.31	10.15	6.32
GPTQ	6.68	6.13	6.33	8.36	11.88	7.07
GPTQ-LC	6.47	5.98	6.27	7.68	10.76	6.58
AWQ	6.62	6.05	6.35	7.82	10.73	6.69
AWQ-LC	6.61	6.04	6.05	7.80	10.72	6.68
SparseGPT	17.97	13.39	14.28	45.22	40.11	29.12
SparseGPT-LC	9.97	8.63	10.24	12.51	19.32	9.65
Llama2-7B						
Original	5.67	5.06	5.32	3.43	4.26	2.53
SliceGPT	233.12	329.72	171.28	5K	8K	12.17
SliceGPT-LC	16.20	16.32	15.50	11.20	15.45	6.14
LLM-Pruner	8.74	7.50	7.60	5.18	6.70	3.69
LLM-Pruner-LC	8.29	7.26	7.52	4.86	6.70	3.32
Llama3-8B-Instruct						
Original	6.71	6.95	7.18	9.16	12.39	9.22
SliceGPT	171.88	160.23	488.79	58K	29K	3K
SliceGPT-LC	20.92	15.23	15.67	86.33	103.96	28.62
LLM-Pruner	9.82	9.18	9.66	12.97	19.81	15.20
LLM-Pruner-LC	9.18	8.68	9.10	11.53	17.11	12.32
Phi3-Instruct						
Original	5.83	5.15	5.49	6.63	7.80	4.77
SliceGPT	196.46	181.20	397.88	5K	3K	16.00
SliceGPT-LC	20.19	14.53	14.52	14.73	27.02	9.47

Table 1: Perplexity (lower is better) on target-language Wikitext. 'LC' = LANGCOMPRESS applied. Original models in gray, baselines in red, and LANGCOMPRESS results in blue.

perplexity indicates better language modeling performance.

Summarization. For summarization, we use the MLSum dataset (Scialom et al., 2020) and evaluate performance with ROUGE scores.

Translation. For translation, we use the FLORES dataset (Goyal et al., 2022), translating from English to the target language. Performance is evaluated using BLEU scores.

4.3 Model Compression Baselines

LANGCOMPRESS can be integrated with various model compression methods to enhance performance in a target language. We evaluate the following techniques:

Structured Pruning. We use LLM-Pruner (Ma et al., 2023) (20%–50% sparsity) and SliceGPT (Ashkboos et al., 2024) (10%–50% sparsity).

Semi-Structured Pruning. We adopt SparseGPT (Frantar and Alistarh, 2023) with a 2:4 sparsity ratio—the only scheme known to yield actual speedups (Mishra et al., 2021).

Quantization. We use GPTQ (Frantar et al., 2023) and AWQ (Lin et al., 2024), quantize weights to 4 bits while keeping activations at 16 bits, balancing efficiency and accuracy.

4.4 Languages

We conduct experiments on Latin-based scripts—German (DE), Spanish (ES), French (FR), and Vietnamese (VI)—as well as logographic scripts—Japanese (JA) and Chinese (ZH).

4.5 Experimental Settings

Instruction Data Synthesis. For LANGCOMPRESS, we use the Alpaca instruction template (Taori et al., 2023) for foundation models (e.g., LLaMA-3-8B, Qwen2.5-7B), and the default chat templates for instruction-tuned models (e.g., LLaMA-3-8B-Instruct, Phi3-Instruct). We apply `lingua-py`* as a probabilistic N-gram language filter. We set the few-shot maximum to $K = 10$. For fair comparison, we generate the same amount of instruction data as used in the recovery settings of each baseline compression method.

Vocabulary Simplification. We use the FineWeb2 corpus (Penedo et al., 2025) as the raw data source. For LLaMA-3 and Qwen2.5 models, we set the number of key tokens to $k = 32{,}000$, and for LLaMA-2 and Phi3 models, we use $k = 16{,}000$.

5 Results

5.1 Main Results

Perplexity. Table 1 reports the perplexity results. LANGCOMPRESS consistently improves the perplexity of compressed models across various languages and architectures. For each compression

*https://github.com/pemistahl/lingua-py

Method	DE	ES	FR	JA	VI
Llama3-8B					
Original	17.42	17.92	25.01	21.73	27.70
GPTQ	23.11	16.13	31.91	6.24	25.65
GPTQ - LC	23.55	18.57	32.98	17.93	35.31
AWQ	10.14	15.67	11.71	4.12	29.17
AWQ - LC	13.65	15.93	25.98	4.64	30.59
SparseGPT	0.60	0.56	0.62	0.00	0.26
SparseGPT - LC	1.26	13.34	1.06	0.00	1.23
SliceGPT	0.78	0.67	0.88	0.0	0.20
SliceGPT - LC	3.60	5.57	5.93	8.08	5.63
LLM-Pruner	13.89	18.67	25.74	24.12	23.09
LLM-Pruner - LC	17.88	19.81	26.53	32.45	30.83
Llama3.1-8B					
Original	18.26	21.38	32.31	37.30	25.39
LLM-Pruner	8.87	16.87	15.58	24.17	11.47
LLM-Pruner - LC	15.51	21.87	29.18	31.73	21.16
Qwen2.5-7B					
Original	19.88	15.61	12.69	39.56	13.63
GPTQ	10.98	19.60	36.35	16.86	13.29
GPTQ - LC	22.59	22.01	38.28	19.06	17.95
AWQ	3.15	16.70	11.14	6.94	11.38
AWQ - LC	3.94	31.04	36.11	15.80	13.13
SparseGPT	7.27	10.24	17.85	4.15	6.18
SparseGPT - LC	18.50	18.95	25.41	22.63	25.02
Llama2-7B					
Original	23.01	25.57	38.03	21.84	34.01
SliceGPT	0.42	0.19	0.00	0.05	0.00
SliceGPT - LC	4.32	13.02	5.34	9.03	12.88
LLM-Pruner	4.06	4.46	6.21	6.82	1.76
LLM-Pruner - LC	5.64	6.65	11.26	10.47	2.36
Llama3-8B-Instruct					
Original	8.56	3.97	15.35	27.46	19.04
SliceGPT	2.94	0.59	0.37	0.62	2.04
SliceGPT - LC	13.18	4.36	4.13	9.40	0.00
LLM-Pruner	1.71	5.41	7.39	7.54	9.74
LLM-Pruner - LC	20.13	17.67	12.47	20.95	17.50
Phi3-Instruct					
Original	27.25	25.35	43.09	34.15	12.08
SliceGPT	2.60	1.70	2.57	1.13	0.58
SliceGPT - LC	3.26	4.95	6.02	5.18	0.00

Table 2: Translation performance (BLEU) on FLORES from English to target languages. 'LC' indicates LANGCOMPRESS applied to the respective compression method. Original model results are in gray, baselines in red, and LANGCOMPRESS results in blue.

Method	DE	ES	FR
Llama3-8B			
Original	11.36	11.18	11.08
GPTQ	12.15	10.64	14.02
GPTQ - LC	13.36	10.82	14.27
AWQ	11.80	10.78	13.85
AWQ - LC	12.62	10.62	13.31
SparseGPT	11.27	9.26	12.44
SparseGPT - LC	13.62	10.51	13.25
SliceGPT	3.38	2.36	3.49
SliceGPT - LC	10.87	11.13	11.14
LLM-Pruner	12.00	10.54	11.72
LLM-Pruner - LC	12.19	10.59	13.30
Llama3.1-8B			
Original	11.15	10.91	14.91
LLM-Pruner	11.70	10.44	11.63
LLM-Pruner - LC	11.78	10.73	13.98
Llama2-7B			
Original	12.56	11.82	13.97
SliceGPT	3.31	2.51	2.47
SliceGPT - LC	9.58	10.97	11.54
LLM-Pruner	8.19	10.44	10.78
LLM-Pruner - LC	8.19	10.64	10.99
Llama3-8B-Instruct			
Original	16.09	13.53	14.97
SliceGPT	4.09	3.12	3.65
SliceGPT - LC	14.58	10.94	12.18
LLM-Pruner	14.52	11.87	14.82
LLM-Pruner - Ours	14.32	12.57	15.97
Phi3-Instruct			
Original	14.30	12.02	13.30
SliceGPT	2.07	1.97	2.16
SliceGPT - LC	8.20	10.59	11.63

Table 3: Summarization performance (ROUGE-Lsum) on DE, ES, and FR. 'LC' denotes LANGCOMPRESS applied to the corresponding compression method. Original models are highlighted in gray, baselines in red, and LANGCOMPRESS results in blue.

method—GPTQ, AWQ, SparseGPT, SliceGPT, and LLM-Pruner—integrating LANGCOMPRESS yields significant gains, especially for non-English languages. Improvements span multiple language families, including European (e.g., German, Spanish, French), East Asian (e.g., Japanese, Chinese), and Southeast Asian (e.g., Vietnamese), demonstrating the method's language-agnostic effectiveness. These gains hold across diverse base models such as LLaMA variants, Qwen2.5, and Phi3, confirming the robustness and general applicability of LANGCOMPRESS under compression.

Downstream Tasks. Table 2 presents translation performance (BLEU) on the FLORES benchmark

from English to various target languages. LANGCOMPRESS consistently enhances the performance of existing compression methods across languages. These improvements are observed across different techniques—pruning and quantization—and remain robust across diverse model architectures, including LLaMA-2, LLaMA-3, Qwen2.5, and Phi-3. Similarly, Table 3 reports summarization performance (ROUGE-Lsum). LANGCOMPRESS again achieves consistent gains over baseline compression methods across languages, with improvements that persist across both pruning and quantization, as well as across multiple architectures.

5.2 Analysis

Perplexity Across Sparsity. We evaluate LANGCOMPRESS under varying sparsity levels using two structured pruning methods—SliceGPT and LLM-Pruner on LLaMA-3-8B (Figure 5). LANGCOMPRESS consistently reduces perplexity across all sparsity settings. Notably, improvements with LLM-Pruner become more significant at higher sparsity, indicating that LANGCOMPRESS is especially effective in high-sparsity regimes. For SliceGPT, gains are substantial and stable across all levels.

Comparison with Raw Text Calibration. Quantization methods such as GPTQ and AWQ can use raw text (e.g., C4) for calibration. However, our results show that instruction-formatted data yields better calibration. As shown in Figure 6a, LANGCOMPRESS-generated instruction data consistently outperforms raw text, leading to improved perplexity and demonstrating its effectiveness for quantization.

Ablation Study. We examine the individual contributions of instruction data synthesis and vocabulary simplification in LANGCOMPRESS. Figure 6b shows the perplexity results using LLM-Pruner and SliceGPT. Both components contribute to performance gains, with the combination yielding the best results.

Other Analysis. We evaluate the performance and runtime across different LM head sizes after simplification, with detailed results provided in Appendix A.

6 Related Work

Unstructured and Semi-Structured Pruning. Model pruning techniques include unstructured, semi-structured, and structured pruning. Unstructured and semi-structured pruning (Hassibi et al., 1993; Li and Louri, 2021; Frantar and Alistarh, 2023; Sun et al., 2024; Zhang et al., 2024; Le et al., 2025) introduce sparsity by zeroing out weights in the model. Semi-structured pruning imposes an N:M constraint, requiring N zeros in every M consecutive elements. In practice, only semi-structured pruning with hardware support (e.g., NVIDIA GPUs) provides real speedup (Mishra et al., 2021).

Structured Pruning. Structured pruning removes entire components (e.g., layers, attention heads) from the model, reducing both size and inference cost. LLM-Pruner (Ma et al., 2023) prunes based on gradient importance, while SliceGPT (Ashkboos et al., 2024) replaces full weight matrices with smaller dense matrices. Unlike unstructured methods, structured pruning physically removes parameters, reducing memory and computation. We experiment with both LLM-Pruner and SliceGPT.

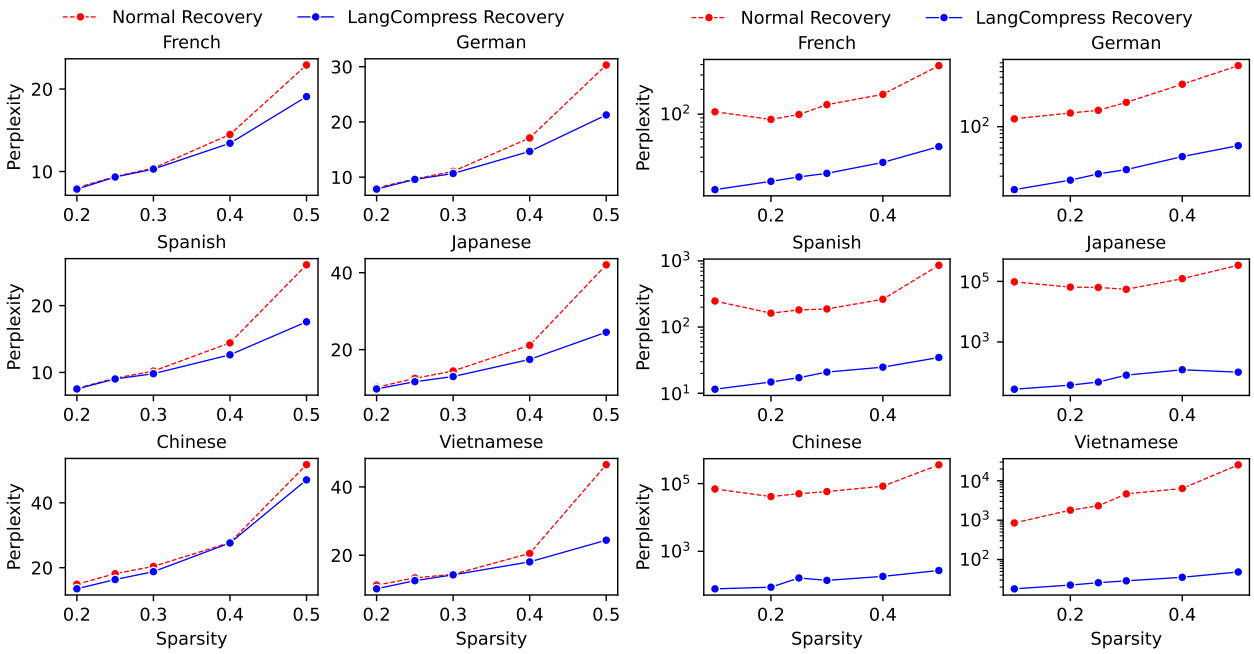
Quantization. Quantization reduces model size and computation by lowering the precision of weights. GPTQ (Frantar et al., 2023) is a post-training method using approximate second-order information to preserve accuracy with 3–4 bit weights. AWQ (Lin et al., 2024) introduces an activation-aware approach, selecting salient weights based on activation statistics. We experiment with both GPTQ and AWQ.

7 Conclusions

We presented LANGCOMPRESS, a language-aware compression framework that improves the efficiency and performance of LLMs in language-specific settings. By integrating self-supervised instruction data generation with vocabulary simplification, LANGCOMPRESS overcomes key limitations of existing compression methods, especially in low-resource scenarios. It is compatible with various pruning and quantization techniques and consistently enhances performance on target languages while reducing model size. These results highlight its potential for practical, multilingual, and domain-specific LLM deployment.

8 Limitations

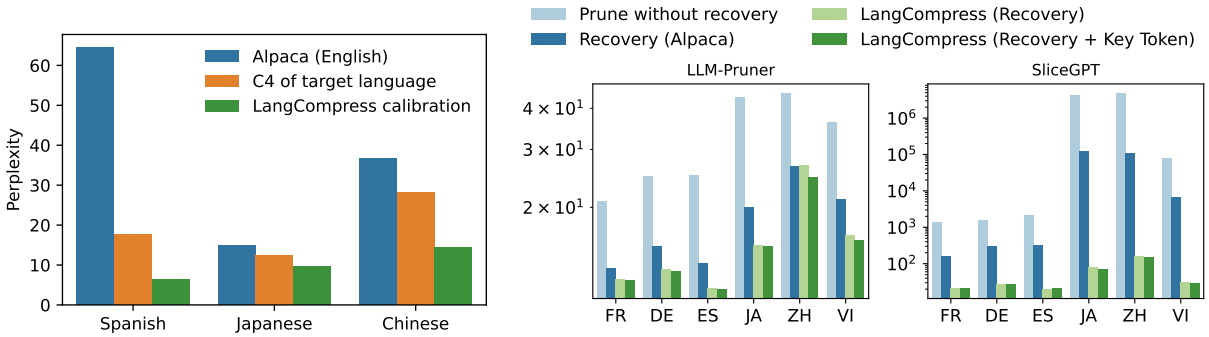
While LANGCOMPRESS demonstrates strong potential for language-specific compression, some



(a) LLM-Pruner with Sparisty from 20% to 50%

(b) SliceGPT with Sparsity from 10% to 50%

Figure 5: Perplexity (lower is better) of pruning methods using normal recovery and LANGCOMPRESS recovery, measured with Llama3-8B on target-language Wikitext.



(a) Perplexity performance of Llama3-8B using GPTQ Quantization with different calibration data.

(b) Perplexity performance of Llama3-8B using pruning methods LLM-Pruner and SliceGPT.

Figure 6: Ablation study of LANGCOMPRESS on Quantization and Pruning. (a) GPTQ with different calibration data; (b) Pruning methods with recovery schemes: no recovery, English data recovery (Alpaca), and LANGCOMPRESS with/without vocabulary simplification.

limitations remain, primarily related to the trade-off introduced by the language-specific LM head and the preprocessing overhead in the compression steps, as detailed below.

- **Language-Specific Trade-off.** Vocabulary simplification enhances performance in the target language but reduces multilingual capabilities. This makes LANGCOMPRESS most suitable for deployment in resource-constrained, language-specific scenarios.
- **Preprocessing Overhead.** Although inference remains lightweight, compression involves additional steps such as instruction data synthesis and vocabulary analysis, requiring moderate computational resources and preprocessing time.
- **Limited Evaluation Scale.** Our experiments focus on smaller models (7B–8B) due to resource constraints. Future work will explore scalability to larger models and evaluate performance across more tasks and domains.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, and 110 others. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. _eprint: 2404.14219.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. In *The Twelfth International Conference on Learning Representations*.

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

Elias Frantar and Dan Alistarh. 2023. SparseGPT: massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*. JMLR.org.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The Flores-101 Evaluation Benchmark for Low-Resource and Multilingual Machine Translation. *Transactions of the Association for Computational Linguistics*, 10:522–538. Place: Cambridge, MA Publisher: MIT Press.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 Herd of Models. _eprint: 2407.21783.

Babak Hassibi, David G Stork, and Gregory J Wolff. 1993. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE.

Khang Nguyen Le, Ryo Sato, Dai Nakashima, Takeshi Suzuki, and Minh Le Nguyen. 2025. OptiPrune: Effective Pruning Approach for Every Target Sparsity. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3600–3612, Abu Dhabi, UAE. Association for Computational Linguistics.

Jiajun Li and Ahmed Louri. 2021. Adaprune: An accelerator-aware pruning technique for sustainable CNN accelerators. In *IEEE Transactions on Sustainable Computing*, volume 7, pages 47–60. Issue: 1.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pages 87–100.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating Sparse Deep Neural Networks. _eprint: 2104.08378.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. 2025. FineWeb2: One Pipeline to Scale Them All – Adapting Pre-Training Data Processing to Every Language. _eprint: 2506.20920.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. 2020. MLSUM: The Multilingual Summarization Corpus. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8051–8067, Online. Association for Computational Linguistics.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A Simple and Effective Pruning Approach for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. Publication Title: GitHub repository.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. 2024. Plug-and-Play: An Efficient Post-training Pruning Method for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Sun Ao, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jie Zhou, Hao Zhou, Jianyong Wang, Maosong Sun, and Zhiyuan Liu. 2025. FR-Spec: Accelerating Large-Vocabulary Language Models via Frequency-Ranked Speculative Sampling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3909–3921, Vienna, Austria. Association for Computational Linguistics.

A Vocabulary Simplification Sizes

During the vocabulary simplification step in the LM head, we can control the vocabulary size, i.e., the number of key tokens retained. We experiment with different vocabulary sizes and report the corresponding performance in Table 4 and the runtime results in Table 5.

Performance of different vocabulary sizes . We conducted an ablation study on vocabulary size and its effect on the perplexity of LLAMA3 (128K full vocabulary). As shown in Table 4, reducing the vocabulary size can occasionally improve perplexity; however, setting it too low leads to performance degradation. Based on these results, a size of 32K offers a balanced trade-off between compression and performance.

Runtime in LM head. The primary objective of the LM head simplification module is to guide the LLM to focus on the target language vocabulary while maintaining performance after pruning or quantization. As a secondary benefit, it reduces the effective size of the LM head, leading to lower latency and memory usage during inference. We further measured the LM head runtime using LLAMA3-8B (hidden size 4096, sequence length 2048, vocabulary size 128K) on 1,000 examples. The results, shown in Table 5, demonstrate the efficiency gains achieved through simplification.

Method	Vocabulary Size	JA	ZH	VI
Original Model	-	6.34	8.46	6.44
SliceGPT	Full	39.46	91.59	22.53
	32K	37.41	87.13	22.48
	16K	38.14	88.96	26.61
	8K	39.85	101.44	30.47
GPTQ	Full	9.92	18.43	79.88
	32K	9.56	14.45	47.15
	16K	10.21	18.52	89.52
	8K	11.82	22.45	102.94
AWQ	Full	7.36	9.59	8.21
	32K	7.16	9.47	7.22
	16K	7.57	10.26	9.18
	8K	8.08	12.63	10.84

Table 4: Perplexity of models across languages (JA, ZH, VI) under varying vocabulary sizes. Lower values indicate better performance. Original models are highlighted in gray, full vocabulary as baselines in red, and LANGCOMPRESS results with 32K vocabulary in blue, representing the best perplexity within each method.

Vocab	Time	Params	Speedup	LM Head Mem Saved
NVIDIA A100-PCIE-40GB				
16K	1.50	65M	7.41×	88%
32K	2.88	131M	3.87×	75%
64K	5.54	262M	2.01×	50%
128K	11.13	525M	1.00×	0%
NVIDIA A40				
16K	2.38	65M	7.70×	88%
32K	4.54	131M	4.04×	75%
64K	9.14	262M	2.01×	50%
128K	18.35	525M	1.00×	0%
NVIDIA RTX A6000				
16K	2.34	65M	7.68×	88%
32K	4.43	131M	4.06×	75%
64K	8.89	262M	2.02×	50%
128K	17.95	525M	1.00×	0%
NVIDIA A100-80GB-PCIe				
16K	1.27	65M	7.41×	88%
32K	2.52	131M	3.75×	75%
64K	4.73	262M	1.99×	50%
128K	9.43	525M	1.00×	0%

Table 5: LM head runtime and efficiency comparison across different GPUs and vocabulary sizes using Llama3-8B. Full vocabulary as baselines in red.