CARVQ: Corrective Adaptor with Group Residual Vector Quantization for LLM Embedding Compression

Dayin Gou[*], **Sanghyun Byun**[*], **Nilesh Malpeddi**, **Gabrielle De Micheli**,
Prathamesh Vaste, **Bahattin Yildiz**, **Jacob Song**, **Woo Seong Chung**[†]

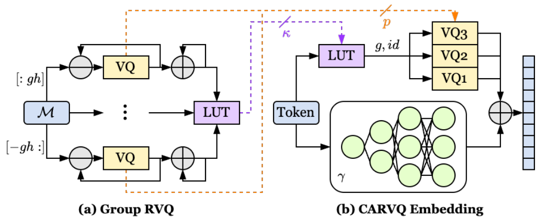
LG Electronics USA

Abstract

Large Language Models (LLMs) typically rely on a large number of parameters for token embedding, leading to substantial storage requirements and memory footprints. In particular, LLMs deployed on edge devices are memory-bound, and reducing the memory footprint by compressing the embedding layer not only frees up the memory bandwidth but also speeds up inference. To address this, we introduce CARVQ, a post-training novel Corrective Adaptor combined with group Residual Vector Quantization. CARVQ relies on the composition of both linear and non-linear maps and mimics the original model embedding to compress to approximately 1.6 bits without requiring specialized hardware to support lower-bit storage. We test our method on pre-trained LLMs such as LLaMA-3.2-1B, LLaMA-3.2-3B, LLaMA-3.2-3B-Instruct, LLaMA-3.1-8B, Qwen2.5-7B, Qwen2.5-Math-7B and Phi-4, evaluating on common generative, discriminative, math and reasoning tasks. We show that in most cases, CARVQ can achieve lower average bitwidth-per-parameter while maintaining reasonable perplexity and accuracy compared to scalar quantization. Our contributions include a novel compression technique that is compatible with state-of-the-art transformer quantization methods and can be seamlessly integrated into any hardware supporting 4-bit memory to reduce the model's memory footprint in memory-constrained devices. This work demonstrates a crucial step toward the efficient deployment of LLMs on edge devices.

1 Introduction

Transformer-based Large Language Models (LLM) are designed to handle extended contexts efficiently through processing tokens with attention mechanisms. Transformer architecture can be primarily separated into three core components: embedding layer, transformer blocks, and prediction head. Each of these plays a distinct role in the overall processing pipeline, enabling the model to handle complex tasks across diverse domains.



(a) Group RVQ (b) CARVQ Embedding

Figure 1: **Overview of the CARVQ framework.** (a) The embedding matrix $\mathcal{M}$ is partitioned into groups, and each group is compressed using Residual Vector Quantization, with the results stored in a look-up table. LUT is stored in κ-bit and VQ elements are stored in original precision p. (b) At inference time, each token retrieves its corresponding quantized vectors from the look-up table. Simultaneously, it is processed by the corrective adaptor, a lightweight MLP.

While various compression techniques have been widely studied for transformer blocks, embedding layer compression has not been investigated extensively due to its simple nature of directly mapping from vocabulary tokens to high-dimensional vector representations. As most quantization are done to 4-bit datatypes, scalar quantization is often sufficient. We show in Figure 2 that the portion of the embedding layer shrinks in larger models, but increases rapidly when post-training quantization (PTQ) operations are applied to the transformer layers. This ratio for INT4 quantized models are 52.06% for LLaMA-3.2-1B (Grattafiori et al., 2024), 13.22% for Phi-4 (14B) (Abdin et al., 2024), and 87.18% for Gemma3-270M (Team et al., 2025). Compressing the embedding layer thus has a more pronounced impact on smaller models, making them essential for deployment in resource-limited inference scenarios. Therefore, the embed-

[*] Equal Contribution
[†] Corresponding Author

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 18594–18604
November 4-9, 2025 ©2025 Association for Computational Linguistics

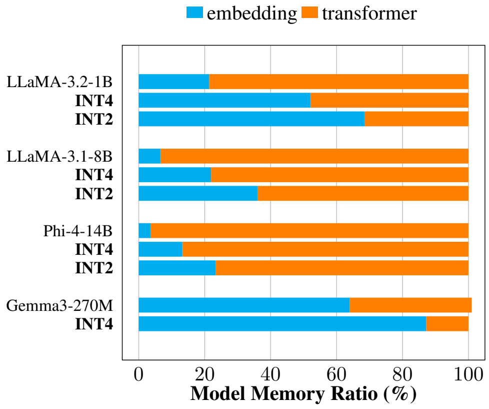


Figure 2: With transformer layer quantization, the memory ratio of embedding layer increases compared to embedding layers left in FP16.

ding layer often becomes a bottleneck in resource-constrained environments as its relative memory footprint expands in compressed models, prompting further investigation into embedding layer compression.

In this work, we specifically focus on the deployment of LLMs on memory-constrained edge devices, where available memory is limited to a few GB. Such situations usually require the use of smaller (less than 8B), quantized models where the relative contribution of the full-precision (FP16) embedding layer to the total model memory is greater compared to larger models. Saving of a mere 0.5GB can grant 2B additional parameters at 4-bit precision or longer context-length, which would greatly improve the output performance.

Quantization can be applied either during training in what is called quantization-aware training (QAT) or after during post-training quantization (PTQ). While QAT remains the most effective for preserving the model's accuracy, it also presents some practical limitations. It requires access to the original training data, which is often private for LLMs. Secondly, QAT is computationally intensive as it requires retraining and fine-tuning of the model, often requiring a repeat of the specific training procedure of the model (e.g., instruction-tuning with RLHF). These constraints make QAT difficult to apply generally across diverse pre-trained models. On the other hand, PTQ offers a more generalizable and data-independent solution, as it can directly be applied to a frozen model. This is the solution adopted in this work.

To effectively target the embedding layer, we propose a novel compression method called CARVQ, a post-training Corrective Adaptor with Group RVQ. The proposed Corrective Adaptor (γ) relies on the composition of both linear and non-linear maps. More precisely, we define it as the composed map $\gamma = \sigma_1 \circ \sigma_0$, where σ_0 first embeds the tokens into a very small dimension m and σ_1 expands the resulting vectors back to the embedding dimension n through a multi-layer perceptron with small hidden dimensions. This Corrective Adaptor compensates the loss from group RVQ operation, which we use as an inexpensive strategy to retain the essential knowledge in the original embedding matrix. Through a careful decision of centroid bitwidth κ, CARVQ is orthogonal to existing transformer-layer quantization approaches such as activation-aware weight quantization (AWQ) without requiring any additional datatype support.

To study the impact of CARVQ on pre-trained LLMs, we apply the proposed method on variations of three architectures, namely LLaMA-3.2-1B (Grattafiori et al., 2024), LLaMA-3.2-3B (Grattafiori et al., 2024), LLaMA-3.2-3B-Instruct (Grattafiori et al., 2024), LLaMA-3.1-8B (Grattafiori et al., 2024), Qwen2.5-7B (Yang et al., 2024a), Qwen2.5-Math-7B (Yang et al., 2024b), and Phi-4 (14B) (Abdin et al., 2024). We evaluate these models on four common types of NLP tasks: generative, discriminative, math, and reasoning. In most cases, we observe that the model performance drop to be near-lossless at 2.4-bit bitwidth-per-parameter (bpp) and reasonable (perplexity < 18) at 1.6-bit bitwidth-per-parameter on average.

The main contributions are summarized below:

1. We introduce CARVQ, a novel post-training method for LLM embedding layer compression without requiring specialized hardware to support lower-bit storage. CARVQ combines group RVQ with Corrective Adaptor to maximize information retention in low bits.

2. We evaluate the proposed compression method on various task types, achieving better model perplexity and evaluation scores than the common approach of scalar quantization. CARVQ achieves 1.6 average-bitwidth-per-parameter compression on all models while scalar quantization does not hold model performance below 3 bits.

3. We demonstrate that CARVQ is compatible with transformer-layer quantization methods without requiring special datatype sup-

ports, unlike scalar quantization. This allows CARVQ to be readily fitted to most deployed LLMs today. We evaluate CARVQ on INT4-AWQ-quantized LLMs.

2 Related Works

The rapid evolution of transformer-based architectures has significantly enhanced performance across natural language processing (NLP), computer vision (CV), and multimodal (MMMU) tasks. However, the massive computational and memory demands of these models remain a significant barrier to their deployment in resource-constrained environments such as edge devices and real-time applications. To address this issue, various compression techniques have been investigated.

Architecture Preserving Compression (Xiao et al., 2023; Lin et al., 2024; Fang et al., 2023; Ma et al., 2023) aims to reduce the size of transformer models while preserving their overall structure and operational flow. These methods focus on minimizing redundant computations or parameters without altering the architecture itself. Quantization (Xiao et al., 2023; Lin et al., 2024; Huijben et al., 2024; Egiazarian et al., 2024; van Baalen et al., 2024; Dettmers et al., 2024) converts high-precision weights and activations into lower-precision representations to reduce memory and computational costs. By carefully balancing precision loss and performance, quantization methods are particularly effective in deploying models on hardware with constrained resources, such as system-on-chip (SoC). Another complementary approach, pruning (Fang et al., 2023; Ma et al., 2023; Ashkboos et al., 2024), eliminates weights or attention heads deemed less impactful based on predefined criteria. For example, head pruning and structured pruning have shown that many weights in transformer blocks contribute minimally to the overall accuracy. These methods allow significant reductions in size and computational demand while retaining the architectural integrity of the model.

Architecture Adaptive Compression (Hu et al., 2022; Oseledets, 2011) involves reconfiguring the model structure to achieve compression by replacing or simplifying specific layers. These methods embrace the notion that certain architectural modifications can provide significant efficiency gains while maintaining task accuracy. Low-Rank Adaptation (LoRA) (Hu et al., 2022) introduces low-rank parameter updates to large pre-trained weights, reducing memory footprint during fine-tuning on downstream tasks without the need to retrain. Another innovative method, Tensor Train Decomposition (Oseledets, 2011), decomposes large tensors into smaller low-rank tensors, significantly reducing memory footprint while maintaining accuracy.

Embedding Layer Compression (Xu et al., 2023; Vincenti et al., 2024) specifically targets the reduction of parameters associated with embedding layers, which often constitute a substantial proportion of the overall model size, especially in multilingual architectures. In transformer models, embedding layers map input vocabulary and image-patch tokens to high-dimensional vectors, and their size grows proportionally with the vocabulary size. TensorGPT (Xu et al., 2023) leverages tensor train decomposition to represent these high-dimensional embeddings in a more compact manner, achieving parameter reduction with minimal degradation to model accuracy. However, the linear nature of TensorGPT results in accuracy drops in high-compression. Similarly, Dynamic Vocabulary Pruning (Vincenti et al., 2024) adjusts the vocabulary size adaptively based on the task or data requirements, pruning infrequently used tokens to reduce the embedding matrix dimensions. However, these methods are limited by the requirement of fine-tuning, lacking generalization.

3 Background

3.1 Input embedding matrix $\mathcal{M}$

We denote by $\mathcal{W}$ the set of all tokens, known as the vocabulary, for a given alphabet.

Definition 3.1. An embedding is a mapping from $\mathcal{W}$ to $\mathbb{R}^n$ for $n \geq 1$. It takes as input a token T and returns an n-dimensional real vector, that is $\sigma : \mathcal{W} \to \mathbb{R}^n$ sends T to $(v_1, v_2, \cdots, v_n)$, where $v_i \in \mathbb{R}$.

The embedding map is learned during training where weights of the following matrix are given.

Definition 3.2. Let V denote the number of tokens in a given vocabulary, *i.e.,* we have $V = |\mathcal{W}|$. The embedding weight matrix is the matrix $\mathcal{M} \in \mathbb{R}^{V \times n}$ where each row $\mathcal{M}_i$, for $i = 1, \cdots, V$ of the matrix corresponds to $\mathcal{M}_i = \sigma(T_i)$ for $T_i \in \mathcal{W}$.

The number of coefficients $V \times n$ in the embedding weight matrix corresponds to the number of parameters in the model. Initially, the embedding coefficients in the matrix $\mathcal{M}$ are set to random real values. During the training process, these coefficients are updated through backpropagation as the model analyzes more and more data. Token embeddings can be pre-trained using traditional algorithms such as skip gram (Guthrie et al., 2006) or CBOW (Mikolov, 2013). However, for task-specific foundational LLM model fine-tuning, it is usually preferable to adapt the token embedding to the data distribution considered and the semantics of the specific downstream tasks.

3.2 Group Residual Vector Quantization

Residual Vector Quantization (Chen et al., 2010) (RVQ) can be used to compress the embedding layer in an LLM by representing the n-dimensional embedding vectors using a sequence of lower-dimensional quantized residuals. More specifically, RVQ works by iteratively applying vector quantization (Gray, 1984) (VQ) and encoding the residuals.

Vector quantization represents a high-dimensional vector with the closest centroid from a pre-trained codebook $\mathcal{C} \in \mathbb{R}^{K \times n}$ defined as a set of K centroids, *i.e.,* n-dimensional vectors. First, the codebook is trained using for example K-mean clustering algorithm on the entire embedding matrix. For each embedding vector $\sigma(T_i) \in \mathbb{R}^n$, one can compute the distance $\hat{j} = \arg\min_j ||\sigma(T_i) - c_i||_2$ from it to each centroid $c_i \in \mathcal{C}$ and identify the closest one. The embedding vector $\sigma(T_i)$ is now represented by the index $\hat{j}$ of the closest centroid $c_i \in \mathcal{C}$. During inference, the embedding vector $\sigma(T_i)$ is reconstructed as an approximation of the original vector with some quantization error, called residual, defined by the quantity $r_i = \sigma(T_i) - c_i$, computed for all $T_i \in V$. This process can be repeated iteratively, say L times, and one can then apply vector quantization on the residuals obtained from the previous step using a new codebook $\mathcal{C}'$, resulting in a new set of residuals $r'_i = r_i - c'_i$. If one repeats this process where at each iteration a different codebook is used, each embedding vector $\sigma(T_i)$ is now represented by a vector of length L which represents the sequence of L quantized indices $(j_{i,1}, \cdots, j_{i,L})$ where $j_{i,j}$ is the index of the closest centroid in the codebook $\mathcal{C}_j$ at layer j for the embedding vector $\sigma(T_i)$. In order to reconstruct the embedding vector $\sigma(T_i)$ one can sum the centroids selected by the indices $(j_{i,1}, \cdots, j_{i,L})$. The goal of RVQ is to minimize the reconstruction error while still keeping the storage cost as small as possible.

Group RVQ. Group RVQ was introduced in (Yang et al., 2023) and offers a variant of RVQ where the set of embedding vectors is divided into groups and RVQ is applied to each group separately as described above. The group RVQ outputs are then combined to obtain the final quantization results. In this work, we will consider this group RVQ method as it is expected to have smaller reconstruction error than standard RVQ since it operates over smaller sets.

Compression with group RVQ. Residual Vector Quantization is used to reduce the storage size of the embedding matrix but does not effectively reduce the number of parameters in the embedding layer. Let us analyze how the storage size is compressed when using group RVQ with group size g with sub-vector dimension h. We initially have n-dimensional embedding vectors for each vocabulary in V resulting in an embedding matrix of size $n \times V$. We start by splitting the input set of embedding vectors into nV/gh groups such that each group contains g embedding vectors of dimension h for any $g, h > 0$ such as $gh|nV$. We then consider the bit-compression ratio for *a single group*. We will compute the following ratio:

$$\frac{\text{storage emb. matrix}}{\text{storage comp. model}} = \frac{gh \times p}{\text{storage comp. model}}$$

where p is the precision of the original embedding coefficients. Let us now count the storage size in the compression model. There are two elements that need to be stored. The quantized indices and the codebooks. The total number of centroids is 2^κ, where κ is the number of bits necessary to index a centroid, and each centroid is of length h. The resulting codebook storage for L iterations is then equal to $Lh2^\kappa \times p$ bits. For the quantization indices, each of the g embedding vectors goes through L iterations. For each iteration, an index is stored into a 2^κ-sized codebook, requiring κ bits. Finally, we have the compression ratio

$$\text{compression-ratio}_{\text{bits}} = \frac{gh \times p}{Lh2^\kappa \times p + gL\kappa}.$$

In our work, we will focus on the average bitwidth-per-parameter representing how many bits per original parameter we are now effectively using after compression. This quantity is computed as

$$B_{\text{RVQ}} = \frac{p}{\text{comp-ratio}_{\text{bits}}} = p \times \frac{Lh2^\kappa \times p + gL\kappa}{gh \times p}.$$

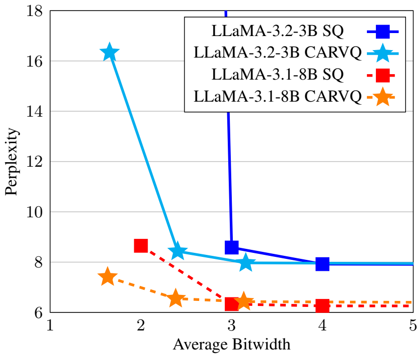


Figure 3: Wikitext-2 Perplexity for LLaMA-3.2-3B and LlaMA-3.1-8B with scalar quantization and CARVQ.

4 CARVQ Embedding Compression

In this section, we introduce a novel embedding CARVQ, defined as the composition of two methods: Corrective Adaptor (4.1) and Group RVQ (4.2). Corrective Adaptor works as non-linear intermediate maps σ_1 expanding the small dimension output of linear map σ_0 to the original embedding dimension. Through this design we can efficiently compensate the precision loss from Group RVQ operation. Figure 1 illustrates CARVQ's framework.

4.1 Corrective Adaptor

Corrective Adaptor (CA) performs contraction-expansion strategy to significantly reduce the number of parameters required to map a token to its embedding. First, let us define the following map to map a token $\mathcal{W}$ to a small-dimension vector $\sigma_0 \colon \mathcal{W} \to \mathbb{R}^m$ sending $T \mapsto (v_1, v_2, \cdots, v_m)$ where $m \ll n$. We will call index the output of σ_0. We will refer to the dimension m as the *corrective width*. Then, we define $\sigma_1 \colon \mathbb{R}^m \to \mathbb{R}^n$ sending $(v_1, v_2, \cdots, v_m) \mapsto (\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_n)$. This map expands the m-dimensional vectors to the embedding dimension n. Note that the number of parameters in the model is then equal to $mV + nm$, and as V and n are usually fixed by the dataset and model considered, the corrective width m is the hyper-parameter introduced by our new method that will change the average bitwidth-per-parameter.

Defining the map σ_0. Initially, for a given corrective width m, the coefficients of the vectors $\sigma_0(T) = (v_1, \cdots, v_m)$ are randomly assigned for each token T in the vocabulary. These coefficients are then updated via training against the original embedding matrix $\mathcal{M}$.

Defining the map σ_1. The question now remains as to how one defines the map σ_1. We define σ_1 as a multi-layer perceptron, a composition of linear and non-linear functions h_L and h_{NL_i}, respectively. This means we have the following composed embedding $\sigma_1 = h_L \circ h_{NL_k} \circ \cdots \circ h_{NL_1}$, where the non-linear maps h_{NL_i} for $i = 1, \cdots, k$, are defined as $h_{NL_i} \colon \mathbb{R}^{m_i} \to \mathbb{R}^{m_{i+1}}$ sending $x \mapsto \text{Relu}(W_i \cdot x + b_i)$ with $W_i \in \mathbb{R}^{m_{i+1} \times m_i}$ and $b_i \in \mathbb{R}^{m_{i+1}}$. Note that $m_1 = m$. The last function h_L is linear and corresponds to a weighted summation $h_L \colon \mathbb{R}^{m_{k+1}} \to \mathbb{R}^n$ sending $x \mapsto W_L \cdot x + b_L$ with $W_L \in \mathbb{R}^{n \times m_{k+1}}$ and $b_L \in \mathbb{R}^n$. The last embedding h_L must be linear so that the token embedding can have both negative and positive values in its vector representation. We add layer normalization between dense layers with ReLU activation to facilitate training. The dimensions m_i for $i = 1, \cdots, k+1$ are experimentally chosen and fine-tuned taking into account downstream tasks. Moreover, the number k of non-linear maps to apply can also vary. The number of model parameters in this case is equal to

$$N_P = mV + \sum_{i=1}^{k} m_i m_{i+1} + \sum_{i=1}^{k} m_{i+1} + m_{k+1} n + n$$

where the values $m_i m_{i+1}$ are the number of parameters in the matrices W_i, the m_{i+1} are the number of parameters in the b_i, mV is the size of $\mathcal{M}_{\sigma_0}$ as before, nm_{k+1} and n are the sizes of W_L and b_L.

Compression ratio. By introducing an intermediate mapping that operates on much smaller vectors of dimension m, we were able to compress our model by the following ratio

$$\frac{nV}{mV + \sum_{i=1}^{k} m_i m_{i+1} + \sum_{i=1}^{k} m_{i+1} + m_{k+1} n + n}$$

Parameters corresponding to biases are nearly negligible compared to weight parameter counts.

4.2 Combining Corrective Adaptor with group RVQ

For the final compression of the embedding matrix, we combine our Corrective Adaptor method with group RVQ explained in Section 3.2. Group RVQ retains knowledge in the original embedding matrix of the pre-trained model with a minimal number of bits without requiring any special datatype support in hardware. As the original datasets for most models are not available during compression, this

Prec.	Method	LLaMA-3.2-1B		LLaMA-3.2-3B		LLaMA-3.2-3B-Inst		LLaMA-3.1-8B		Qwen2.5-7B		Mean	
		Prec.	PPL	Prec.	PPL	Prec.	PPL	Prec.	PPL	Prec.	PPL	Prec.	Δ PPL
16	FP16	16	9.75	16	7.81	16	11.05	16	6.24	16	6.85	16	0
4 ± 0.5	INT4	4	9.98	4	7.92	4	11.18	4	6.26	4	6.85	4	0.098
3 ± 0.5	INT3	3	11.42	3	8.58	3	12.26	3	**6.33**	3	**6.86**	3	0.750
	CARVQ-4	3.201	**10.15**	3.155	**7.97**	3.155	**11.48**	3.133	6.43	3.131	**6.86**	3.155	**0.238**
2 ± 0.5	INT2	2	181	2	154	2	110	2	8.65	2	7.44	2	83.88
	CARVQ-3	2.451	**10.81**	2.405	**8.43**	2.405	**11.70**	2.383	**6.55**	2.381	**6.87**	2.405	**0.532**
	CARVQ-2	1.701	14.27	1.655	16.34	1.655	14.49	1.633	7.41	1.631	6.91	1.655	3.544

Table 1: CARVQ Results on Text Generation for Wikitext-2. LLaMA-3.2-1B (128256x2048), LLaMA-3.2-3B (128256x3072), LLaMA-3.1-8B (128256x4096), Qwen2.5-7B (152064x3584). CARVQ-L represents L-iteration RVQ. For CARVQ, the centroid count K is fixed to 16 (4-bit), the sub-vector dimension h to 8, and the group size g to 1024. Precision in the table above corresponds to average bitwidth-per-parameter B.

Method	B_{RVQ}	B_{CA}	mem. gain (GB)
CARVQ-4	3.000	0.155	0.756
CARVQ-3	2.250	0.155	0.831
CARVQ-2	1.500	0.155	0.877

Table 2: Memory overhead of the corrective adaptor is sufficiently small compared to the group RVQ for the overall compression ratio to be dictated by the group RVQ compression.

Model	Model (GFLOPS)	CA (MFLOPS)	CA (MB)
LLaMA-3.2-1B	0.62	0.63	2.52
LLaMA-3.2-3B	1.61	0.89	3.56
LLaMA-3.1-8B	3.75	1.15	4.62
Qwen-2.5-7B	3.53	1.02	4.08
Phi-4 (14B)	7.08	1.41	5.66

Table 3: Corrective adaptor computational overhead is at most 0.1% of the original model with configurations described in Section 5.1.

PTQ approach allows retention of knowledge, especially when leveraging Corrective Adaptor to minimize the precision loss between the output of original embedding $\mathcal{M}$ and CARVQ. Without the Corrective Adaptor, group RVQ would be too destructive, significantly impacting the performance of the model. Their combination allows an inexpensive method minimizing the reconstruction loss without any fine-tuning.

We now concretely explain how we combine these methods. We start by reshaping the embedding matrix. We divide each embedding vector (*i.e.,* each row of $\mathcal{M}$) into j sub-vectors each of dimension h, where naturally $jh = n$. The embedding matrix $\mathcal{M}$ is then reshaped into matrix $\mathcal{M}'$ of size $nV/h \times h$ where each row now corresponds to an h-dimensional sub-vector. The reason we reshape the matrix is to consider RVQ on smaller vectors of dimension $h \ll n$ for which similarity search (e.g., nearest neighbor search) is more effective. In addition to this reshaping, as we consider group RVQ, we split $\mathcal{M}'$ into nV/gh groups such that each group corresponds to a matrix of size $g \times h$ which will be compressed using RVQ.

Recall that L is the number of iterations considered in RVQ, 2^{κ} is the number of centroids, and for each group of size g, each sub-vector of dimension h, after applying RVQ, is represented as an L-dimensional vector $(j_1, j_2, \cdots, j_L)$ of indices. From Section 3.2, we already have the average bitwidth-per-parameter after applying group RVQ referred to as B_{RVQ}. Let us now analyze the average bitwidth-per-parameter when combining group RVQ with our Corrective Adaptor. We have already described in Section 4.1 the parameter count N_p corresponding to our embeddings σ_0 and σ_1. Therefore, we define $B_{\text{CA}} = p \times \frac{N_p}{nV}$, to be the average bitwidth-per-parameter resulting from our Corrective Adaptor compression. The average bitwidth-per-parameter considering group RVQ and Corrective Adaptor is equal to $B_{\text{CARVQ}} = B_{\text{CA}} + B_{\text{RVQ}}$. In most scenarios, the number of parameters in RVQ is much greater than the corrective layer, meaning $B_{CA} \ll B_{RVQ}$. We show this in Table 2. This shows that, as long as the corrective width m is kept relatively low, the hidden dimension of the intermediate map σ_1 is not too relevant. We also find the computational overhead to be at most 0.1% of the entire model, as shown in Table 3.

5 Experimental results

5.1 Implementation Details

LLM Evaluation. We evaluate our method on different architectures: LLaMA-3.2-1B (Grattafiori et al., 2024), LLaMA-3.2-3B (Grattafiori et al., 2024), LLaMA-3.2-3B-Instruct (Grattafiori et al., 2024), LLaMA-3.1-8B (Grattafiori et al., 2024), Qwen2.5-7B (Yang et al., 2024a), Qwen2.5-Math-

Model	Dataset	FP16		INT4		INT3		CARVQ-4		INT2		CARVQ-3		CARVQ-2	
		Prec.	Acc.	Prec.	Acc.	Prec.	Acc.	Prec.	Acc.	Prec.	Acc.	Prec.	Acc.	Prec.	Acc.
LLaMA-3.2-1B	Hella	16	47.72	4	47.66	3	46.83	3.201	**47.09**	2	30.43	2.451	**45.86**	1.701	40.58
	Wino		60.69		60.85		**60.46**		59.98		54.93		**60.30**		56.04
	Piqa		74.48		74.70		73.23		**73.76**		60.83		**72.31**		70.46
LLaMA-3.2-3B	Hella	16	55.31	4	54.22	3	50.46	3.155	**55.38**	2	35.15	2.405	**54.63**	1.655	50.33
	Wino		69.85		68.98		**69.38**		68.67		56.83		**67.80**		64.48
	Piqa		76.66		76.39		75.57		**76.88**		67.57		**76.33**		74.59
LLaMA-3.1-8B	Hella	16	60.01	4	60.07	3	**60.00**	3.133	59.83	2	54.74	2.383	**59.41**	1.633	55.11
	Wino		73.88		73.72		73.24		**73.72**		70.00		**73.24**		70.32
	Piqa		80.14		79.82		79.33		**79.82**		75.57		**79.33**		77.15
Qwen2.5-7B	Hella	16	60.02	4	59.97	3	**60.03**	3.131	59.90	2	59.11	2.381	59.98	1.631	**60.05**
	Wino		72.93		73.32		**72.69**		**72.69**		70.80		**72.14**		71.11
	Piqa		78.73		78.45		**78.73**		78.62		**78.89**		78.56		78.78
Qwen2.5-Math-7B	GSM8K	16	83.62	4	83.78	3	83.02	3.131	**84.08**	2	47.54	2.381	**83.70**	1.631	83.24
Phi-4 (14B)	GSM8K	16	88.78	4	88.25	3	88.02	3.138	**88.63**	2	**88.55**	2.388	88.10	1.638	87.79
	ARC_C		55.55		55.72		**56.74**		54.86		55.03		55.20		**55.38**
	ARC_E		81.48		81.61		**81.90**		81.65		82.24		81.82		80.43
Avg. Precision / Mean Δ Acc.		16	-	4	-0.15	3	-0.64	3.148	**-0.27**	2	-8.23	2.398	**-0.70**	1.648	-2.75

Table 4: Comparison of accuracy on discriminative and math/reasoning tasks. Phi-4's embedding matrix is of size (100352×5120). Exact-match accuracy with flexible extraction is reported for GSM8K. ARC_C and ARC_E represent ARC challenge and easy tasks, respectively. CARVQ-L represents L-iteration RVQ. For CARVQ, the centroid count K is fixed to 16 (4-bit), the sub-vector dimension h to 8, and the group size g to 1024. Precision in the table above corresponds to average bitwidth-per-parameter B.

7B (Yang et al., 2024b), and Phi-4 (14B) (Abdin et al., 2024). All pretrained models are open-source from HuggingFace, with our Corrective adaptor implemented in PyTorch. We do not consider weight-tying (Press and Wolf, 2017) in our work due to its limited usage in recent high-complexity tasks. We assess four types of LLM tasks for evaluation: generative, discriminative, math, and reasoning. For the generative task, we evaluate perplexity on Wikitext-2 (Merity et al., 2016) following previous quantization works (Lin et al., 2024; Chen et al., 2025). Discriminative tasks are evaluated with three benchmarks: HellaSwag (Zellers et al., 2019), WinoGrande (Keisuke et al., 2019), and Piqa (Bisk et al., 2020). Math tasks are evaluated with GSM8K (Cobbe et al., 2021). We evaluate on ARC Challenge and ARC Easy (Clark et al., 2018) for the reasoning task.

Corrective Adaptor Implementation. Our method consists of two components: group RVQ and our Corrective Adaptor method. We acquire RVQ by applying K-Means clustering at each iteration for each group with a tolerance of $1e^{-4}$. Corrective Adaptor hyper-parameters are $[m_1, m_2, m_3] = [16, 384, 512]$ for all models. Note that we promote overfitting during training as we want the network output to be as close as possible to the original embedding vector.

Training. As the input to embedding layers are a scalar pointing to a row in the embedding matrix, the data used to train the Corrective Adaptor is limited to the input dimension of embedding matrix. This also means that we can overfit the MLP as the input is fixed to a specific set, further lowering the expressiveness requirement of the adaptor. Then, with roughly 150,000 data points (English), we train such that the L1 loss of the final output sum of RVQ and Corrective Adaptor with the original embedding vector for each vocabulary is minimized. We train for 500 iterations with Adam for learning rate of 1e-3 on a RTX 4090 for 2 minutes.

Quantization Scheme for Comparison. To our knowledge, there are no open-source post-training compression methods targeting embedding layers. Thus, we compare Corrective Adaptor with scalar quantization, as prior quantization works (Lin et al., 2024; Liu et al., 2024) utilize scalar quantization for the embedding layer. To ensure fair comparison, we keep the transformer layers in their original precision (FP16/BF16). Although scalar quantization can achieve a slightly lower average error per dimension, its uniform bins make it highly sensitive to large-magnitude embedding values, producing a skewed error distribution. In contrast, group RVQ's iterative residual steps produce errors that are approximately zero-mean and more concentrated. Such zero-centered, low-variance errors are better suited for being corrected by the corrective adaptor.

5.2 Evaluation Details

Generative Tasks. Table 1 presents perplexity changes when applying different post-training compression methods to the embedding matrix, namely scalar quantization and our CARVQ-L method for L-iteration RVQ. For 4, 3 and 2-layer CARVQ, we observe a 0.238, 0.532, and 3.544 perplexity increase on average compared to the FP16 baseline models. On the other hand, scalar quantization works well up to 3-bit quantization with average perplexity increase of 0.098 and 0.75. However, 2-bit quantization fails to capture enough information for most models with an average perplexity loss of 84. In comparison, the proposed CARVQ is very stable at 3 iterations with an average bit-per-parameter of 2.405, even preserving most of the structure at 2 iterations with 1.655 bit-per-parameter. We sometimes also notice marginal improvements ($< 1\%$) over baseline when CARVQ is applied. We visualize the comparison in Fig. 3 with LLaMA-3.2-3B and LLaMA-3.1-8B. Furthermore, compared to INT3 and INT2 quantization, which require hardware optimization due to limited architecture support, CARVQ-3 and CARVQ-2 only utilize 4-bit and 16-bit datatypes for storage, making them compatible with all recent architectures supporting INT4. By adapting the centroid bitwidth κ, we can adapt CARVQ to any appropriate hardware. This demonstrates that the combination of group RVQ and corrective adaptors in CARVQ significantly reduces quantization error, achieving low-bit embedding without sacrificing hardware compatibility.

Discriminative Tasks. Table 4 exhibits a comparison of evaluation accuracy on discriminative tasks with CARVQ and scalar quantization applied. With L=2, 3 and 4, we attain mean accuracy loss of 0.34, 0.88, and 3.45 for discriminative tasks, showing decent performances even at average bitwidth-per-parameter of 1.68. In comparison, scalar quantization mean losses are at 0.87 for INT3 and 7.97 for INT2, so 2-bit quantization would risk significant loss in answer quality. Moreover, this bigger gap is more pronounced in smaller models such as LLaMA-3.2-1B, where the accuracy gap between INT2 and CARVQ-2 is more than 10%. This shows that CARVQ can achieve much smaller bidwidth-per-parameter while maintaining good accuracy compared to scalar quantization for this task.

Method	LLaMA-3.2-3B		Qwen2.5-7B	
	Prec.	PPL	Prec.	PPL
FP16	16	7.81	16	6.85
CA+INT3	3.155	**7.90**	3.131	**6.86**
CARVQ-4	3.155	7.97	3.131	**6.86**
CA+INT2	2.155	12.51	2.131	7.39
CARVQ-3	2.405	**8.43**	2.381	**6.87**
CA+INT1	1.155	14528	1.131	46480
CARVQ-2	1.655	**16.34**	1.631	**6.91**
CARVQ-2 (k=3)	1.155	3230	-	-

Table 5: Ablation of quantization with scalar quantization. We evaluate the perplexities of LLaMA-3.2-3B and Qwen2.5-7B on Wikitext-2.

Method	Llama-3.2-3B-Instruct	Qwen2.5-3B-Instruct
	Wiki PPL	Wiki PPL
AWQ	11.75	9.10
CARVQ-4+AWQ	12.12	9.19
CARVQ-3+AWQ	12.76	9.40
CARVQ-2+AWQ	15.91	10.61

Table 6: CARVQ applied on AWQ-quantized models. We evaluate perplexities of Llama-3.2-3B-Instruct and Qwen2.5-3B-Instruct on Wikitext-2.

Math Tasks. Table 4 also compares the accuracy on math tasks with Qwen2.5-Math-7B and Phi-4. For both models, CARVQ methods result in an average minimal accuracy loss of less than 0.5%. However, we see a sudden loss of accuracy in INT3 and INT2, each of 0.6% and 36%. As mathematical prompts require high retention of reasoning and memory, scalar quantization to INT2 likely simplifies the complex math problems too much. However, we observe CARVQ holding accuracy even at 1.63 bits, likely due to improved precision withheld from storing the VQ vectors in the original precision of FP16.

Reasoning Task. We include reasoning tasks evaluation with Phi-4 to demonstrate further generalization to different tasks in Table 4. As Phi-4 is larger at 14B parameters, we see retention of performance even with low-bit quantization, even showing improvements in both easy and challenge evaluations. Overall, both scalar quantization and CARVQ quantization result in a minimal ($< 1.1\%$) drop in accuracy under reasoning tasks when a sufficiently large model is used to counter the quantization loss in the embedding layer. Through extensive evaluation across various tasks, we conclude CARVQ-3 offers the best trade-off between accuracy retention and compression ratio.

5.3 Additional Analysis

Ablation of Quantization. To better understand the effect of RVQ in our method, we swap the RVQ operation with regular scalar quantization and train the Corrective Adaptor with the same configurations. Table 5 evaluates the effect of replacing RVQ with a scalar quantization. In both 3B and 8B pretrained models, RVQ clearly shows an advantage, especially below 2-bit, where INT1 (binary) quantization completely fails (the perplexity goes up to 14528 while with CARVQ-2, it remains 16.84).

For the comparison to be fair, the group RVQ's average bitwidth must be equal to the INT-N ($B_{RVQ} = N$). For that, we must say $B_{RVQ} = (pLh2^k + gLk)/gh = N$, where p is fixed to 16 and L fixed to the CARVQ-L. Due to the $gh|nV$ constraint as well as to avoid odd overflows into the next rows when grouping, it is unadvisable to manipulate $g = 1024$ and $h = 8$ unless we are scaling them by whole numbers. Thus, this leaves $k = 4$, the bit-width determining the centroid counts, to be manipulated. We calculate this for LLaMA-3.2-3B. For CARVQ-3 to be equal to INT-2, we would require $k = 3.704$, to which it rounds to the original $k = 4$. For CARVQ-2 to be equal to INT-1, we require $k = 3$, which we report in Table 5. However, keep in mind that we decrease the number of centroids, which significantly increases the quantization error that has to be regained by the proposed Corrective Adaptor.

Combining with Transformer Quantization. AWQ allows models to be quantized to 4-bit with minimal loss to quality and coherence. As the Corrective Adaptor only works on the embedding layer, it is inherently orthogonal to existing post-training quantization operations on transformer layers, assuming such methods do not amplify error cascades. We evaluate the perplexity of W4A16 AWQ (Lin et al., 2024) quantized models with CARVQ. Table 6 details the result of our comparison. The perplexity loss of CARVQ remains low (< 5) even on AWQ-quantized models, with CARVQ-3 keeping the loss less than 1.1 in all tests. Notably, the perplexity loss with CARVQ-4 and CARVQ-3 on Qwen2.5 architecture is kept below 0.3, showing significant bitwidth reduction with near-lossless performance. Further, as W4A16 AWQ natively uses 4-bit and 16-bit datatypes, we can set $\kappa = 4$ and $p = 16$ for the proposed CARVQ to be compatible with the machines running such models.

6 Conclusion

In this work, we introduced CARVQ, a novel post-training method for LLM embedding layer compression to $\sim$1.6 bits without requiring specialized hardware to support lower-bit storage. By carefully reorganizing the matrix into vector-quantized tables, CARVQ reduces the embedding layer precision without utilizing equally low-bit datatypes. We focused on evaluating varying-complexity models on a diverse set of tasks, showing that CARVQ can achieve lower average bitwidth-per-parameter while maintaining reasonable perplexity and accuracy compared to scalar quantization. Moreover, the CARVQ system can be seamlessly integrated into any hardware supporting 4-bit memory to reduce model memory footprint in significantly memory-constrained devices where every MB counts. CARVQ's compatibility with state-of-the-art transformer quantization methods, which can be highly prone to error propagation, unveils promising potential for future scalability.

7 Limitations

Despite its conceptual simplicity, CARVQ cannot be directly applied to transformer layers due to a substantial increase in computational complexity without a specialized lookup-table implementation. Such an implementation is feasible for embedding layers, which operate on discrete token indices, but not for continuous transformer activations. This constraint limits CARVQ's applicability in larger language models, where the proportion of the embedding layer to the overall model size becomes relatively small, even with compression applied to transformer layers. Furthermore, the Corrective Adaptor in CARVQ does not preserve the structural properties of the original embedding matrix, effectively acting as a coarse additional RVQ iteration. While we did not observe notable degradation in our experiments, this simplification may result in the loss of fine-grained semantic information, potentially affecting performance on tasks that depend on subtle representational nuances. Lastly, we observe that larger models tend to be more tolerant to quantization errors in the embedding layer; however, if the accompanying transformer layer compression is excessively lossy, it may fail to compensate for even small embedding-level errors, diminishing the overall robustness of the model.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, and 8 others. 2024. Phi-4 technical report. *Preprint*, arXiv:2412.08905.

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

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

Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273.

Yuzong Chen, Ahmed F. AbouElhamayed, Xilai Dai, Yang Wang, Marta Andronic, George A. Constantinides, and Mohamed S. Abdelfattah. 2025. Bit-MoD: Bit-serial mixture-of-datatype llm acceleration. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

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. *Preprint*, arXiv:2407.21783.

Robert Gray. 1984. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29.

David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. 2006. A closer look at skip-gram modelling. In *LREC*, volume 6, pages 1222–1225.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Iris AM Huijben, Matthijs Douze, Matthew Muckley, Ruud JG Van Sloun, and Jakob Verbeek. 2024. Residual quantization with implicit neural codebooks. *arXiv preprint arXiv:2401.14732*.

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

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 llm compression and acceleration. In *MLSys*.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024. Spinquant–llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Tomas Mikolov. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane

Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Mart van Baalen, Andrey Kuzmin, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*.

Jort Vincenti, Karim Abdel Sadek, Joan Velja, Matteo Nulli, and Metod Jazbec. 2024. Dynamic vocabulary pruning in early-exit llms. In *NeurIPS Workshop*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*.

Mingxue Xu, Yao Lei Xu, and Danilo P Mandic. 2023. Tensorgpt: Efficient compression of the embedding layer in llms based on the tensor-train decomposition. *arXiv preprint arXiv:2307.00526*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024b. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Dongchao Yang, Songxiang Liu, Rongjie Huang, Jinchuan Tian, Chao Weng, and Yuexian Zou. 2023. Hifi-codec: Group-residual vector quantization for high fidelity audio codec. *arXiv preprint arXiv:2305.02765*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.