PCA-Based Token Embedding Compression for Efficient Small Language Models

Gilhyeon Lee
Department of Electrical Information Engineering, Research Center for Electrical and Information Technology
Seoul National University of Science and Technology
Seoul 01811, Korea
gilhyeonlee@seoultech.ac.kr

***Abstract*—Large language models (LLMs) often devote a large fraction of their parameters to the token embedding layer, which limits deployment in memory-constrained environments. Motivated by prior observations that embedding spaces are anisotropic and low-dimensional, this work applies principal component analysis (PCA) to the token embedding of a GPT-style small language model and replaces it with a low-rank factorized embedding, optionally followed by LoRA-based fine-tuning. Using a GPT-small model trained on WikiText-103, we compare PCA-compressed embeddings, PCA+LoRA models, and low-rank factorized models trained from scratch, while sweeping the PCA rank k and measuring validation loss, parameter count, and embedding-layer FLOPs. The results show that PCA-based initialization is consistently superior to training low-rank embeddings from scratch, and that combining PCA with LoRA recovers most of the accuracy loss at moderate ranks (*e.g.*, reducing embedding parameters by about 32% at $k = 512$ with only minor degradation in validation loss and a modest increase in FLOPs). These findings indicate that PCA-based embedding compression is a practical post-hoc technique for small- to medium-scale LLMs under tight memory budgets, and highlight the value of classical statistical methods for understanding and compressing modern LLM representation spaces.**

***Index Terms*—Machine Learning, Principal Component Analysis, LLM, Embedding Layer**

I. Introduction

Large language models (LLMs) such as the GPT family have become core components of modern natural language processing systems thanks to their strong performance in translation, question answering, code generation, reasoning, and other general-purpose text generation tasks [1]. However, these models typically contain hundreds of millions to billions of parameters and require substantial computational resources, making them difficult to deploy on resource-constrained environments such as consumer GPUs, mobile devices, or edge platforms [2]. A variety of model-compression methods—quantization, pruning, knowledge distillation, and low-rank adaptation (LoRA)—have been proposed to address these issues, but most of them focus on the attention and MLP layers, leaving the overall model structure largely unchanged [3], [4].

Transformer-based LLMs include a token embedding layer that maps discrete tokens into fixed-dimensional continuous representations. This layer is implemented as a matrix $E \in \mathbb{R}^{V\times d}$, where V is the vocabulary size and d is the embedding dimension. Although the embedding lookup is a simple linear operation accessed at every decoding step, it still accounts for a substantial portion of the model parameters [2]. For example, in the 4-bit quantized sLLM Qwen2.5-0.5B [4], the embedding layer alone has been reported to occupy about 59% of all parameters. Despite this, most existing LLM-compression studies primarily focus on optimizing the attention and MLP layers, and relatively few approaches attempt to structurally reduce or redesign the embedding layer itself.

Meanwhile, several studies that analyze the geometric structure of word embeddings and Transformer representation spaces report that these spaces exhibit strong anisotropy, with variance concentrated in a small number of principal components, as well as low intrinsic dimensionality [5]–[8]. Mu *et al.* showed that simply removing the mean vector and a few dominant principal components as a post-processing step can improve performance on various similarity and classification tasks [5]. Raunak *et al.* further combined this post-processing with principal component analysis (PCA), reporting that the embedding dimension can be reduced by up to 50% while achieving comparable or even better performance [6]. Bi *et al.* observed that the embeddings of Transformer language models become increasingly concentrated along a few common directions as training progresses, and demonstrated that a simple linear transformation can substantially restore isotropy [7]. More recently, Bengtsson *et al.* applied PCA directly to the embeddings and internal representations and internal representations of GPT-2 models, showing that the number of parameters can be reduced to 17% or less while preserving about 97% of the original performance [9]. These results suggest that the token embeddings of LLMs in practice compress much of their information into a low-dimensional linear subspace, leaving significant room to reduce memory usage and computation through appropriate dimensionality reduction.

However, existing work has three main limitations. First, many PCA-based studies focus on analyzing the geometric properties of static word embeddings such as word2vec [10] or GloVe [11], or only examine the dimensionality reduction effect on embedding vectors in relatively small models. These studies mainly evaluate embedding quality using similarity measures or simple downstream metrics, and do not systematically analyze how PCA-based embedding compression affects end-to-end language modeling performance and parameter count, memory footprint, and computation when applied to practical decoder-only LLMs such as GPT-3 [1]. Second, there is little empirical work directly comparing two scenarios: (i) compressing the embedding of a sufficiently trained full-rank LLM with PCA and then fine-tuning it, and (ii) training from scratch a smaller model with the same low-rank structure using random initialization. In other words, we still lack a careful comparison of what benefits PCA-based compression with different configurations can provide over simply training a small model. Third, few studies systematically sweep the PCA rank on decoder-only models and explicitly map out quantitative trade-offs between model size (*i.e.*, the number of parameters) and resource usage (*e.g.*, memory and FLOPs). In particular, on standard language modeling benchmarks such as WikiText-103 [12], there is still limited empirical guidance on how far the embedding dimension can be reduced before performance degrades gradually and where a sharp collapse begins.

The goal of this work is to fill this gap by applying PCA-based dimensionality reduction to the token embedding layer of a small LLM and analytically evaluating its effects. To this end, we first train a GPT-style model on the WikiText-103 dataset, then apply PCA to the learned embedding matrix to obtain a compressed representation, and finally apply the representative fine-tuning method LoRA [3] to recover any loss in performance. During this process, we keep the Transformer block architecture identical except for the token embedding layer, so that the impact of PCA on the embedding can be isolated in terms of language modeling performance and resource usage.

To evaluate this setup, we design experiments along two axes. The first axis is a model-variant axis that compares (i) a model trained from scratch with a low-rank factorized embedding of a given dimension against (ii) a model obtained by applying PCA to compress the embedding of a full-rank baseline and then fine-tuning it. The second axis is a rank axis that sweeps the PCA rank k and analyzes the quantitative trade-offs among embedding dimension, explained variance, resource usage, and performance.

The contributions of this report can be summarized as follows:

- We propose a model that applies PCA to the token embedding and then performs fine-tuning, and compare its validation loss against a full-rank baseline and several variants to demonstrate the practical usefulness of PCA-based compression.
- We measure the relationship between PCA rank and parameter count/FLOPs, and thus provide a quantitative characterization of how much the token embedding can be practically compressed.
- We discuss the usefulness and limitations of the proposed compression method and how it could be combined with other techniques in future work.

Through these results, we systematically shed light on how effectively the classical statistical method PCA can compress the token embedding of modern LLMs, and what performance–resource trade-offs arise in the process.

II. Approach Details

A. Large Language Model and Token Embedding

The base model used in this study is GPT, a decoder-only Transformer language model that has demonstrated strong performance across various natural language processing tasks and is widely adopted in practice [1].

The overall model architecture is illustrated in Fig. 1(a). Each token x_t is first mapped into a d-dimensional continuous vector $e_t \in \mathbb{R}^d$ through the embedding matrix $E \in \mathbb{R}^{V\times d}$. The decoder block then performs a series of computations, including query (q), key (k), and value (v) generation, attention, projection, and a feed-forward network (FFN). Stacking multiple decoder blocks allows the model to capture token-level dependencies and predict the next word. Finally, the language model head produces the output logits using the same parameters as the token embedding layer.

The token embedding layer is implemented as a matrix $E \in \mathbb{R}^{V\times d}$, where V denotes the vocabulary size and d denotes the embedding dimension. In the GPT-small configuration, V is 50,257, which is 65.44× larger than d. Prior work reports that, in small LLMs, more than half of all model parameters can be attributed solely to the embedding matrix [4]. Therefore, structurally reducing E serves as an important solution for decreasing memory usage and computational cost [2].

B. PCA-based low-rank embedding

To efficiently reduce the size of the token embedding while preserving accuracy, we apply PCA to the token embedding parameters and replace the original embedding layer with a low-rank structure. In the following, we describe how PCA is applied to the trained embedding matrix and how it induces a low-rank parameterization.

We start from the token embedding of the fully trained baseline model. Let

$$E_{\text{base}} \in \mathbb{R}^{V\times d} \tag{1}$$

denote the weight matrix of the token embedding layer, where V is the vocabulary size and d is the embedding dimension. Each row of E_{base} corresponds to the embedding vector of a single token type, and this matrix is the object to which we apply PCA.

To apply PCA, we first compute the mean embedding across the vocabulary:

$$\mu = \frac{1}{V}\mathbf{1}^\top E_{\text{base}} \in \mathbb{R}^{1\times d}, \tag{2}$$

where $\mathbf{1} \in \mathbb{R}^V$ is the all-ones vector. This mean vector μ represents the average token embedding and is used to center the embedding matrix.

Using the mean, we construct the centered embedding matrix:

$$\tilde{E} = E_{\text{base}} - \mathbf{1}\mu. \tag{3}$$

Here, each row of $\tilde{E}$ is a token embedding with the global mean removed. Centering ensures that PCA captures directions of variation around the average embedding rather than around the origin.

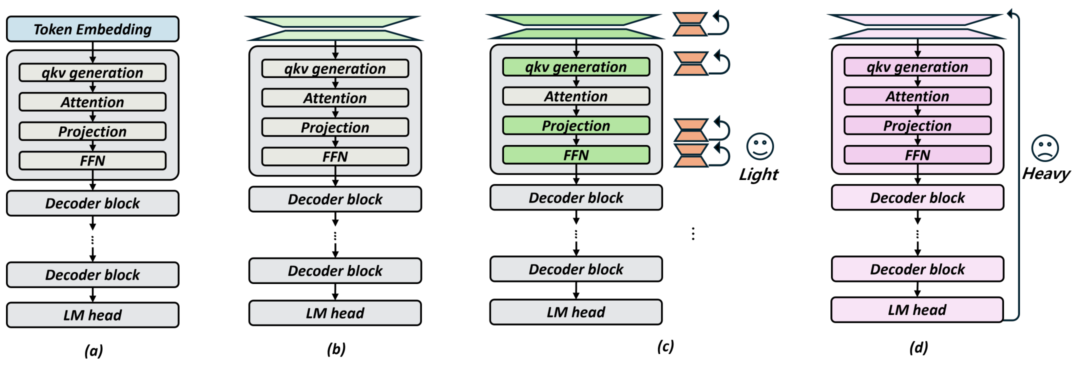

Fig. 1: Overview of PCA-based token embedding compression. (a) Original GPT-style model (baseline). (b) Baseline model with PCA-based token embedding. (c) Fine-tuned model with PCA-based token embedding. (d) Model trained from scratch with a low-rank factorized token embedding.

Based on the centered embeddings, we form the covariance matrix over the embedding dimensions:

$$C = \frac{1}{V}\tilde{E}^{\top}\tilde{E}, \tag{4}$$

where $C \in \mathbb{R}^{d\times d}$ summarizes how the d embedding dimensions co-vary across the vocabulary. This covariance matrix is the standard starting point for PCA.

We then perform eigen decomposition of the covariance matrix:

$$C = U\Lambda U^{\top}, \tag{5}$$

where $U \in \mathbb{R}^{d\times d}$ is an orthogonal matrix whose columns are eigenvectors, and $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_d)$ is a diagonal matrix of eigenvalues satisfying $\lambda_1 \geq \cdots \geq \lambda_d \geq 0$. The eigenvectors in U provide orthogonal directions in embedding space, ordered by the amount of variance (eigenvalues) they explain.

To obtain a low-rank subspace, we keep only the top-k principal components:

$$U_k = [u_1, \dots, u_k] \in \mathbb{R}^{d\times k}, \tag{6}$$

where $u_1, \dots, u_k$ are the eigenvectors associated with the largest k eigenvalues. The columns of U_k span a k-dimensional subspace that captures most of the variance in E_{base}.

We then project the centered embeddings onto this k-dimensional PCA subspace:

$$Z = \tilde{E}U_k \in \mathbb{R}^{V\times k}. \tag{7}$$

Each row of Z contains the coordinates of the corresponding token embedding in the reduced k-dimensional space. Thus, Z represents a compressed embedding table in the PCA subspace.

The matrix that maps from the reduced space back to the original embedding dimension is

$$P = U_k^{\top} \in \mathbb{R}^{k\times d}. \tag{8}$$

Given a k-dimensional coordinate vector, multiplication by P approximately reconstructs the original d-dimensional embedding.

Combining these pieces, we approximate the original embedding matrix as

$$E_{\text{base}} \approx \hat{E}_k = \mathbf{1}\mu + ZP. \tag{9}$$

In this reconstruction, $\mathbf{1}\mu$ restores the global mean embedding, while ZP reconstructs the principal-variance component from the low-dimensional PCA coordinates.

We use this PCA reconstruction to define a low-rank token embedding module parameterized by the tuple (μ, Z, P). The effective embedding weight used during inference is always given by

$$E_{\text{lowrank}}(\mu, Z, P) = \mathbf{1}\mu + ZP. \tag{10}$$

Under this parameterization, the number of learnable parameters in the embedding module is reduced from Vd (full rank) to

$$Vk + dk + d, \tag{11}$$

since $\mu \in \mathbb{R}^{1\times d}$, $Z \in \mathbb{R}^{V\times k}$, and $P \in \mathbb{R}^{k\times d}$. This corresponds to a parameter ratio of

$$\frac{Vk + dk + d}{Vd}$$

relative to the original full-rank embedding layer, yielding a more compact embedding representation with minimal changes to the rest of the model.

C. Fine-tuning with low-rank adaptation

Low-rank adaptation (LoRA) [3] is a widely used method for improving performance in various neural networks with small additional training cost by adding a low-rank correction term BA to the weight $W \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ of a linear layer:

$$W' = W + BA. \tag{12}$$

Here, $A \in \mathbb{R}^{r\times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}}\times r}$ are adaptation parameters with rank $r \ll \min(d_{\text{in}}, d_{\text{out}})$, and the base weight W is kept frozen while only A and B are trained. In this work, we attach LoRA adapters to the query, value, and output projections in self-attention, as well as to selected linear layers in the FFN, and update only these adapters together with the PCA-compressed embedding parameters.

With this strategy, our goal is threefold. First, we aim to preserve as much of the representation learned by the baseline model as possible. Second, we seek to partially recover the performance loss induced by PCA-based embedding compression. Third, we aim to minimize the additional memory and computation overhead incurred during the fine-tuning stage.

Building on this methodology, the next section presents experimental results from (i) comparing four models at a fixed rank and (ii) sweeping the PCA rank, and analyzes how PCA-based token embedding compression affects the performance and resource efficiency of small LLMs.

III. Experiments

A. Experimental setup

In this section, we describe the experimental setup used to evaluate the proposed PCA-based token embedding compression method. All experiments are conducted using PyTorch 2.5.1, CUDA 12.1, and cuDNN 9.1.0 on a single GPU system equipped with an NVIDIA GeForce RTX 3090. The same hardware and software environment is used for all compared models.

The base language model in our experiments is a GPT-style decoder-only Transformer. The detailed configuration is summarized in Table I. Across all comparisons, we fix the token embedding dimension d and vocabulary size V, and only change the internal structure and training procedure of the token embedding layer so that the effect of the proposed method can be isolated.

Our implementation is based on the open-source ReaLLM codebase [13], which we extend by adding PCA analysis for the token embedding, a low-rank factorization module, and a LoRA-based fine-tuning module. The baseline model is trained from scratch on the training split of the WikiText-103 dataset [12], and all performance metrics are reported in terms of validation loss on the corresponding validation split. For LoRA-based fine-tuning, we use a randomly sampled subset of the original training split to evaluate how much of the accuracy loss due to PCA can be recovered with relatively small additional training cost.

The compared model variants are defined as follows:

- `Baseline`. A GPT model with a 768-dimensional full-rank token embedding. The token embedding is implemented as a lookup matrix $E_{\text{base}} \in \mathbb{R}^{V \times d}$, and the rest of the Transformer blocks are trained from scratch with the same configuration.
- `PCA-`k. A model whose token embedding is initialized by applying PCA to the trained baseline embedding E_{base} and extracting a rank-k principal component basis to construct a low-rank token embedding. The remaining Transformer blocks share the same architecture as the baseline, but their parameters are randomly initialized. This variant is used to measure how expressive a PCA-based low-dimensional embedding is when trained from scratch. Here, k denotes the PCA rank.
- `PCA (with LoRA)`. A model obtained by first training the full baseline, then replacing the token embedding layer with the PCA-based low-rank structure and performing short LoRA-based fine-tuning. The PCA rank k is set to match that of `PCA-`k, and this variant is used to evaluate how much of the performance degradation from PCA can be recovered by LoRA.
- `Low-rank factorized (LRF)`. A model that uses the same low-rank embedding dimension as `PCA-`k and `PCA (with LoRA)`, but initializes the low-rank embedding completely at random and trains it from scratch without PCA-based initialization. This allows us to compare how far a simple low-rank factorization can go without PCA, and what additional benefit PCA-based initialization provides.

TABLE I: Configuration of GPT-small model.

Model	n_layer	n_head	block_size	inter_size	d	V
GPT	12	12	2048	3072	768	50257

In our LoRA-based experiments, we apply LoRA adapters to the qkv projections, the output projection, selected FFN linear layers, and the token embedding. The low-rank dimension of LoRA is fixed to $r = 32$ in all experiments.

B. Performance

Fig. 2 shows the validation loss on the WikiText-103 [12] validation set for each model variant and PCA rank. As the performance metric, we use cross-entropy validation loss, which directly reflects language modeling accuracy.

First, the `PCA-`k models, which only apply PCA without any additional fine-tuning, exhibit a significant degradation in performance compared to the baseline. For example, `PCA-64` with $k = 64$ reaches a validation loss of 7.14, which is about $2.08\times$ higher than the baseline. This can be attributed to two factors: (i) such a low rank is insufficient to preserve the information contained in the baseline embedding, and (ii) jointly retraining the entire Transformer from scratch with a low-dimensional embedding makes optimization more difficult.

In contrast, the `PCA (with LoRA)` models, which combine the same PCA-based low-dimensional embedding with LoRA fine-tuning, show a clear improvement. In particular, `PCA-512 (w/ LoRA)` with $k = 512$ achieves a validation loss of 3.44, which is nearly on par with the baseline. Even for smaller k, applying LoRA consistently improves performance over the corresponding `PCA-`k models. This suggests that the combination of PCA-based embeddings and LoRA effectively compensates for the information loss by leveraging the representations learned by the baseline.

The `Low-rank factorized (LRF)` models, which rely solely on a low-rank structure without PCA-based initialization, show higher validation loss than the baseline for all values of k. For instance, `LRF-512` with $k = 512$ records a validation loss about $1.52\times$ higher than the baseline, indicating that simply training a randomly initialized low-rank embedding is less effective at recovering accuracy than PCA initialization combined with LoRA. This highlights that extracting principal components from a pre-trained full-rank embedding is more efficient than learning a low-rank structure from scratch.

Overall, these results indicate that the proposed PCA + LoRA combination provides a practical compromise: it allows

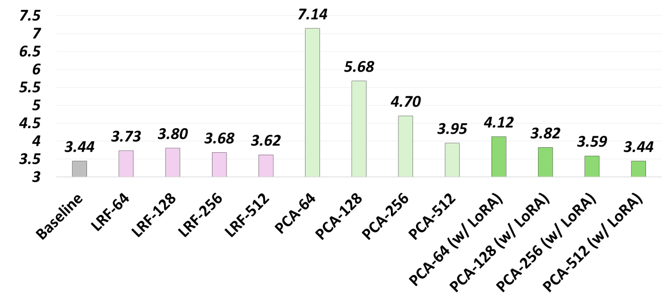

Fig. 2: Validation loss on WikiText-103 for different embedding variants and PCA ranks.

substantial reduction of the token embedding dimension while maintaining language modeling performance close to that of the full-rank baseline.

C. Memory saving and FLOPs

Table II reports how the number of parameters and per-token FLOPs of the embedding layer change under PCA-based token-embedding compression. The parameter count includes the token-embedding lookup matrix, the PCA projection matrix, and, when applicable, the mean vector. FLOPs only account for the linear computation (matrix–vector multiply) performed inside the embedding layer. Because the baseline embedding is implemented as an index lookup, it adds no extra multiplications and additions; we therefore report its FLOPs as 0.

The baseline token embedding uses a total of $38.60M$ parameters, which constitutes a substantial portion of the overall model. Consequently, compressing only the embedding layer can meaningfully reduce the total model size and memory footprint.

When the PCA rank is set to $k = 64$, the embedding-related parameters drop to $3.27M$ (*i.e.*, about 8.47%) of the baseline. In this case, the per-token FLOPs increase to $6.53M$; however, this remains a relatively small fraction compared to the computation in the full transformer stack.

Even in the configuration where LoRA-based fine-tuning recovers accuracy close to the baseline (*i.e.*, $k = 512$), the embedding parameters are $26.13M$, which is only 67.69% of the baseline. The per-token FLOPs rise to $52.25M$, which can be regarded as a reasonable compute overhead in exchange for a fixed reduction in embedding parameters.

In summary, sweeping the PCA rank k produces a smooth trade-off between memory reduction and accuracy. In our setting, $k = 512$ nearly preserves baseline performance while reducing embedding parameters by roughly 32%, indicating that PCA-based embedding compression is a practical strategy for small LLMs where the embedding layer dominates memory.

IV. Discussion and Conclusion

This work studied PCA-based low-rank factorization of the token embedding in a GPT-style small language model and showed that, when combined with LoRA fine-tuning, it can substantially reduce embedding parameters (*e.g.*, about one third at $k = 512$) with only minor degradation in validation loss and a moderate increase in embedding-layer FLOPs. PCA-based initialization was consistently superior to training a low-rank embedding from scratch, indicating that the pre-trained full-rank embedding occupies a highly structured low-dimensional subspace that PCA can effectively capture. These results suggest that PCA-based embedding compression is a practical post-hoc technique for memory-constrained deployments of small- to medium-scale LLMs, where parameter count and VRAM are primary bottlenecks and a small compute overhead is acceptable. At the same time, this study is limited to a single model and dataset and focuses mainly on cross-entropy loss, so future work should examine larger architectures, diverse downstream tasks, and combinations with other compression methods such as quantization or pruning.

TABLE II: Parameter count and embedding FLOPs per token for different PCA ranks.

	Baseline	64	128	256	512
Params (M)	38.60	3.27	6.53	13.06	26.13
FLOPs / token (M)	0	6.53	13.06	26.13	52.25

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] J. Lee, J. Jang, and H. Kim, "Xnc: Xor and not-based lossless compression for optimizing unquantized embedding layers in large language models," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2025, pp. 1–5.

[3] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[4] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[5] J. Mu, S. Bhat, and P. Viswanath, "All-but-the-top: Simple and effective postprocessing for word representations," *arXiv preprint arXiv:1702.01417*, 2017.

[6] V. Raunak, V. Gupta, and F. Metze, "Effective dimensionality reduction for word embeddings," in *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, 2019, pp. 235–243.

[7] D. Biś, M. Podkorytov, and X. Liu, "Too much in common: Shifting of embeddings in transformer language models and its implications," in *Proceedings of the 2021 conference of the North American chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 5117–5130.

[8] J. Engels, E. J. Michaud, I. Liao, W. Gurnee, and M. Tegmark, "Not all language model features are one-dimensionally linear," *arXiv preprint arXiv:2405.14860*, 2024.

[9] M. Bengtsson, "Compressing large language models with pca without performance loss," *arXiv preprint arXiv:2508.04307*, 2025.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[11] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[12] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[13] ReaLLMASIC and Contributors, "ReaLLM-Forge: A Framework for Hardware-Aware LLM Exploration," 2025. [Online]. Available: https://github.com/ReaLLMASIC/ReaLLM-Forge

Code Submission

All source code and experimental scripts used in this project are available at https://github.com/purejomo/ML_2025_SNUT.