TensorSLM: Energy-efficient Embedding Compression of Sub-billion Parameter Language Models on Low-end Devices

Mingxue Xu [1] **Yao Lei Xu** [1] **Danilo P. Mandic** [1]

Abstract

Small Language Models (SLMs, or on-device LMs) (Lu et al., 2024) have significantly fewer parameters than Large Language Models (LLMs). They are typically deployed on low-end devices, like mobile phones (Liu et al., 2024) and single-board computers. Unlike LLMs, which rely on increasing model size for better generalisation, SLMs designed for edge applications are expected to have **adaptivity** to the deployment environments and **energy efficiency** given the device battery life constraints, which are not addressed in datacenter-deployed LLMs. This paper addresses these two requirements by proposing a training-free token embedding compression approach using Tensor-Train Decomposition (TTD). Each pre-trained token embedding vector is converted into a lower-dimensional Matrix Product State (MPS). We comprehensively evaluate the extracted low-rank structures across compression ratio, language task performance, latency, and energy consumption on a typical low-end device, i.e. Raspberry Pi. Taking the sub-billion parameter versions of GPT-2/Cerebres-GPT and OPT models as examples, our approach achieves a comparable language task performance to the original model with around $2.0\times$ embedding layer compression, while the energy consumption of a single query drops by half.

1. Introduction

Modelling complex language patterns and solving complex language tasks are two of the primary reasons that Large Language Models (LLMs) have attracted considerable attention in recent years. While the LLMs track thrives on increasing model sizes and tackling more difficult tasks, another track is considering putting such capable models on lower-end devices. These models are called Small Language Models (SLMs) (Lu et al., 2024) or on-device language models (Liu et al., 2024; Mehta et al., 2024; hfs, 2024).

SLMs may have less than one billion parameters (Mehta et al., 2024; Liu et al., 2024; Laskaridis et al., 2024). Though such a size is already a few tenths or even hundreds of what common LLMs usually are, it can still be burdensome for some low-end devices. As listed in (Liu et al., 2024, Fig. 2), some prevalent mobile devices (e.g. iPhone 14 and iPhone 15) only have 6GB DRAM. For some SLMs like Gemma2-2B, running the uncompressed version causes a system crash on Raspberry Pi-5 with 8GB DRAM.

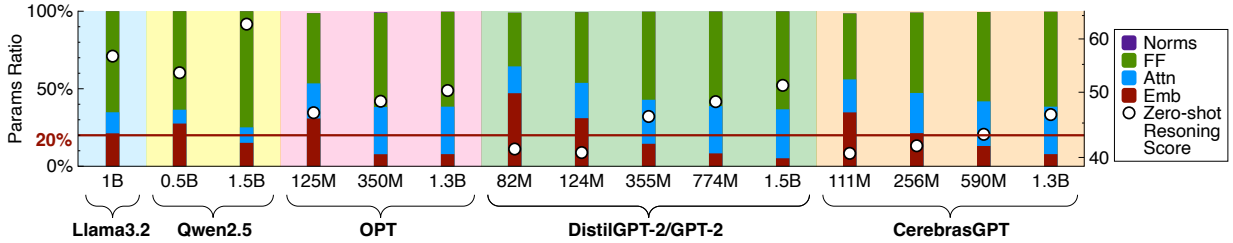
Compared with LLMs, SLMs on low-end devices have different layer compositions of the model and different on-board operations due to the absence of server-level GPUs. As shown in Figure. 1a, around half of the investigated open-source models have more than 20% of the parameters attributed to token embedding layers, which is consistent with the previous findings, i.e. (Liu et al., 2024, Section 2.2.3). Additionally, since no server-level GPU is on board to support massive parallel operations for matrix multiplication, block-wise approaches that rely on parallelism (Dao et al., 2022; Qiu et al., 2024) are not suitable for low-end deployment scenarios.

To this end, this paper proposes TensorSLM, a tensor-based approach to compress SLMs for low-end devices (i.e. Raspberry Pi without GPU). Together with matrix-based low-rank approaches (Chen et al., 2018a; Hrinchuk et al., 2020; Lioutas et al., 2020; Acharya et al., 2019; Chen et al., 2021; Hsu et al., 2022; Dao et al., 2022; Qiu et al., 2024), this kind of approach forms a broader field named low-rank factorization. The comparison of these works regarding methodologies (e.g. matrix/tensor, with/without training) and applications (e.g. high-end/low-end devices, large/small models) are clarified in Table. 4.

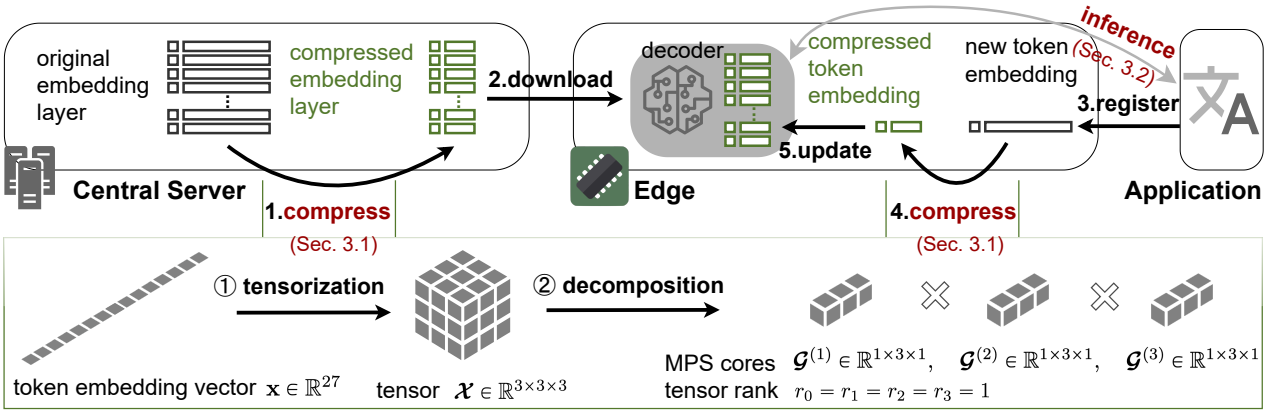
Compared with two-dimensional matrices or their finer-grained block-wise forms (Chen et al., 2018a; Dao et al., 2022), higher-order tensors provide more diverse representation alternatives through their inter-order information, which is more suitable for small-size models to model complex

[1]Imperial College London, London, United Kingdom. Correspondence to: Mingxue Xu <mx1221@ic.ac.uk>.

ICML 2025 Workshop on Tiny Titans: The next wave of On-Device Learning for Foundational Models (TTODLer-FM), Vancouver, Canada, 2025. Copyright 2025 by the author(s).



(a) The parameter ratio of Norms (including layer norms), feed-forward layers (FF), attention layers (Attn), and embedding layers (Emb), and the average zero-shot reasoning score (Zellers et al., 2019; Clark et al., 2018; 2019; Bisk et al., 2020) of several open-source model series. In a model series, smaller models have a higher token embedding layer ratio and lower feed-forward layer ratio, while the attention layer ratio is maintained.



(b) The workflow of SLM compression in edge computing scenario with our approach.

Figure 1. Typical SLM layer composition and the SLM application requirement of adaptability.

patterns. This superiority is more pronounced when no fine-tuning data is available to adjust model parameters for specific deployment environments.

The contributions of this paper are summarised as follows:

1. We systematically analyse LLMs on high-end GPU servers and SLMs on low-end edge devices to address the two unique requirements of SLM compression: *adaptability* to specific deployment environments and *energy efficiency* for better user experience.
2. To our knowledge, we are the first to compress SLMs for low-end device use cases using low-rank factorization. We adjust Tensor-Train Decomposition for non-parallel operations in the forward passes, where block-wise approaches (Dao et al., 2022; Qiu et al., 2024) are incompetent.
3. We gave the measured latency and estimated energy consumption of SLMs on the typical low-end device, Raspberry Pi 5, finding that our approach reduces half of the inference energy with negligible latency increase.
4. We evaluated both simple and complex language tasks. We found that our tensor-based approach is better at unprompted and unconstrained question answering than the matrix-based SVD approach, and herein sheds light on selecting appropriate algebraic structures for language model compression according to the specific tasks.

2. Unique Requirements of SLM Applications

This section clarifies the main application differences between LLMs and SLMs, which will then guide the design of SLMs compression on low-end devices.

2.1. Adaptability

Unlike the current LLM applications, which are mostly running on high-end GPU servers (e.g. in the data centres with numerous NVIDIA A100), SLMs are mainly for edge (or mobile) applications that require adapting to the environment with limited resources on lower-end devices. A common approach to adapting to the dynamic environment is updating the vocabulary according to the changes in input text distribution (Chen et al., 2018a). The reasons for this distribution change vary from case to case. For example, new user registration, or the frequently used tokens update with the users' changing daily lives.

To cope with the ever-changing input tokens and vocabulary, a straightforward strategy is to build a could-edge system, as shown in Figure. 1b, which is similar to the workflows

in the field of edge computing, e.g. (Laskaridis et al., 2024, Fig.1). There are two kinds of devices in this workflow: 1) the central server, which is possibly a server in public or private cloud services, or a higher-end personal computer, and 2) the low-end edge device. In this paper, we only talk about a typical edge device - Raspberry Pi. Over a fairly long period (e.g. months or years), the central server only communicates with the edge device once to provide a brand-new pre-trained language model. Afterwards, the edge device should update the vocabulary on board according to the changes in the environment.

A detailed explanation of Figure. 1b is as follows:

Step 1. The central server compresses the whole token embedding matrices on the token embedding level, according to Algorithm 1.

Step 2. The compressed vocabulary and other parts of the language model (e.g. the decoder) are downloaded and then deployed on a low-end device.

Step 3. During the application runs, the vocabulary updates for two cases:

1. a new token is required according to the actual application requirements, it will be registered by the service on the edge device. Jump to **Step 4**.
2. an old token is required to be removed (e.g. it has not been used for a long time), the edge device simply deletes the corresponding token embedding vector. Meanwhile, the application deregisters this token.

Step 4. The low-end device compresses the added token embedding vector as described in Algorithm 1.

Step 5. The current vocabulary of the language model. The compression process of a single token embedding follows a pipeline of ① tensorization and ② decomposition.

2.2. Energy Efficiency

From the workload of the high-end GPU servers (e.g. those equipped with NVIDIA A100) and low-end edge devices (e.g. Raspberry Pi 5) described in Section 2.1, we know that the edge device only takes charge of light-weight essential tasks, since it has strict limitations in computation, memory and communication. Furthermore, since battery life directly impacts the user experience, energy consumption is also a significant concern.

The actual energy consumption of a device depends on various factors, like the semiconductor temperature, system workload, operating environment, etc. Thus, it is hard to precisely calculate the exact energy consumption of an algorithm on a certain hardware. However, we can still estimate the range of energy consumption in the system as Table. 1, where we can have the following remarks:

Remark 2.1. Memory operations are more "expensive" than computation in terms of energy.

Table 1. Approximate energy consumption of different operations (1nJ=1000pJ). For servers, communication with the wired network (e.g. ethernet or optical fibre) is preferred; for edge devices, it is preferred to use wireless networks (e.g. Wi-Fi or cellular network).

Energy Consumption		Raspberry Pi 5 (Cortex-A76 CPU)	GPU server (A100 GPU)
Computation (pJ/`float32`)	**Add**	1.0-2.5	5-12
	Mult	1.2-3	6-15
Memory (pJ/`float32`)		70-260	100-450
Communication (nJ/`float32`)	**Wired**	50-350	
	Wireless	400-6000	

Remark 2.2. Non-essential communication should be avoided for energy concerns.

The workflow in Figure. 1b has already satisfied Remark 2.2. For Remark 2.1, if real-time is *not* the most important concern in the edge application, we "exchange" memory with computation for longer battery life. Further discussion and evaluation around these are in Section 4.1 and Appx. F.

3. Preliminaries

This section gives the essential concepts related to tensor, tensor operations and Tensor-Train Decomposition.

Order-N Tensor. An order-N real-valued tensor, $\mathcal{A}$, is a high-dimensional matrix (or multi-way array), denoted by $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, where N is the order of the tensor (i.e., number of its modes), and I_k ($1 \leq k \leq N$) is the size (i.e., the dimension) of its k-th mode. In this sense, matrices (denoted as $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2}$) can be seen as order-2 tensors ($N = 2$), vectors (denoted as $\mathbf{a} \in \mathbb{R}^{I}$) can be seen as order-1 tensors ($N = 1$), and scalars (denoted as $a \in \mathbb{R}$) are order-0 tensors ($N = 0$).

Tensor-Train Decomposition (TTD). The most common Tensor-Train Decomposition (Oseledets, 2011) formats a tensor into a Matrix Product State (MPS) form, which applies the Tensor-Train Singular Value Decomposition (TT-SVD) algorithm to an order-N tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. This results in N smaller 2-nd or 3-rd order tensors, $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times I_k \times r_k}$ for $k = 1, \ldots, N$, such that

$$\mathcal{X} \approx \mathcal{G}^{(1)} \times_2^1 \mathcal{G}^{(2)} \times_3^1 \mathcal{G}^{(3)} \times_3^1 \cdots \times_3^1 \mathcal{G}^{(N)}. \quad (1)$$

Tensor $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}$ are referred to as the tensor cores, while the set $\{r_0, r_1, \ldots, r_N\}$ represents the TT-rank of the TT decomposition ($r_0 = r_N = 1$).

4. Methodology

This section clarifies the technical cornerstones of our approach. A practical pipeline of our approach is depicted in Figure. 1b. The whole vocabulary is processed on higher-end servers, while inference and vocabulary updates happen

Algorithm 1 `TT_SVD`(Oseledets, 2011) for a Single Token Embedding Compression

Input : 1. d-dimensional token embedding vector $\mathbf{x} \in \mathbb{R}^d$, approximation accuracy ϵ;
2. Tensor dimension $\{I_1, I_2, \ldots, I_N\}$ and TT ranks $\{r_0, r_1, \ldots, r_N\}$.

Output : TT cores $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}$.

Initialize : Tensor $\mathcal{X} \leftarrow \texttt{reshape}(\mathbf{x}, [I_1, I_2, \ldots, I_N])$, temporary matrix $\mathbf{Z} \leftarrow \texttt{reshape}(\mathcal{X}, [r_0 I_1, \prod_{j=2}^{N} I_j])$, truncation parameter $\delta = \frac{\epsilon}{\sqrt{N-1}} \|\mathcal{X}\|_F$.

1 **for** $k = 1$ *to* $N-1$ **do**
2 $\quad \mathbf{U}, \mathbf{S}, \mathbf{V}, \mathbf{E} \leftarrow \texttt{truncSVD}(\mathbf{Z}, \delta, r_k)$
$\quad$ // s.t. $\mathbf{U} \in \mathbb{R}^{r_{k-1} I_k \times r_k}$, $\|\mathbf{E}\|_F \leq \delta$
3 $\quad \mathcal{G}^{(k)} \leftarrow \texttt{reshape}\,(\mathbf{U}, [r_{k-1}, I_k, r_k])$
$\quad$ // get kth TT core
4 $\quad \mathbf{Z} \leftarrow \texttt{reshape}\left(\mathbf{S}\mathbf{V}^T, [r_k I_{k+1}, \prod_{j=k+2}^{N} I_j]\right)$
$\quad$ // $\mathbf{S}\mathbf{V}^T \in \mathbb{R}^{\prod_{i=k+2}^{N} I_i}$
5 $\mathcal{G}^{(N)} \leftarrow \mathbf{Z}$
6 **return** $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(N)}$

on lower-end edge devices.

4.1. Individual Embedding Vector Compression

For the compression of the embedding matrix, rather than decomposing the whole embedding weight matrix, we propose to decompose each embedding vector. The lower half of Figure. 1b is a simplified illustration of such a process, with a detailed description in Algorithm 1.

Tensorization. Each token embedding $\mathbf{x} \in \mathbb{R}^d$ is reshaped (or folded and tensorized into an order-N tensor. Denote $\texttt{reshape}(\cdot)$ as the reshape function, $\mathcal{X} = \texttt{reshape}(\mathbf{x}, \{I_1, I_2, \ldots, I_N\})$ and $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ such that $d = \prod_{k=1}^{N} I_k$. In the example in Figure. 1b, the token embedding vector $\mathbf{x}$ is a 27-dimensional vector, $d = 27$. In this way, vector $\mathbf{x}$ is reshaped into an order-3 ($N = 3$) tensor $\mathcal{X}$, with tensor size for each mode $I_1 = I_2 = I_3 = 3$.

Tensor Decomposition. Tensor $\mathcal{X}$ is then decomposed and stored in a Matrix Product State (MPS) form as $\mathcal{X} \approx \mathcal{G}^{(1)} \times_3^1 \cdots \times_3^1 \mathcal{G}^{(N)}$, with hyperparameters as TT ranks $r_0, r_1, \ldots, r_N$. For the case in Figure. 1b, the MPS cores are $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}$, with TT ranks $r_0 = r_1 = r_2 = r_3 = 1$. In other words, instead of storing the entire token embedding vector $\mathbf{x} \in \mathbb{R}^d$, we store the corresponding MPS cores, $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times I_k \times r_k}$, for $k = 1, \ldots, N$. The parameter count of the MPS cores $\{\mathcal{G}^{(k)}\}$ is $\sum_{k=1}^{N} |\mathcal{G}^{(k)}| = \sum_{k=1}^{N} r_{k-1} I_k r_k$, where $|\cdot|$ represents the parameter count.

A more detailed explanation of individual token embedding compression is given in Algorithm 1, where $\|\cdot\|_F$ denotes the Frobenius norm. Although the embedding vector is reshaped into a tensor, the decomposition for each mode of this tensor is still based on the matrix-level SVD (line 2). Then the complexity of `TT_SVD` can be derived from SVD and its variants, such as truncated SVD (Oseledets, 2011). Given the vocabulary size V, the original parameters of the embedding layers are compressed from Vd to $V \sum_{k=1}^{N} r_{k-1} I_k r_k$, and the compression ratio can be obtained via $\eta_{\texttt{TTD}} = \frac{d}{\sum_{k=1}^{N} r_{k-1} I_k r_k} - 1$. The computation and memory complexities for all the above processes are summarized in Table. 2.

Energy Consumption Analysis. Recall in Section 2.2 we have Remark 2.1 to guide the choice between memory and computation for the same functionalities from the perspective of energy cost. Based on Remark 2.1 and Table. 2, we can initially give the estimated energy costs when the SLM processes an input token (only before the decoder), which is similar with (Yang et al., 2017). Assuming in the same operating environment and other conditions (e.g. temperature), the memory energy cost of each `float32` is ν, and the computation energy cost of each `float32` is τ, all the model weights are represented in `float32`.

When inputting a text of length l, denote original model energy cost regarding memory as $\mathcal{E}_\nu$, model energy cost regarding computation is $\mathcal{E}_\tau$,

$$\mathcal{E}_\nu = \nu(dV + ld), \quad \mathcal{E}_\tau = 0, \tag{2}$$

and after compression, the energy costs are

$$\mathcal{E}_\nu^{'} = \nu(VNIr^2 + lNIr^2 + ld), \quad \mathcal{E}_\tau^{'} = \tau NIr^2. \tag{3}$$

Denote the SVD rank k, the energy cost after compressing with matrix-based SVD is

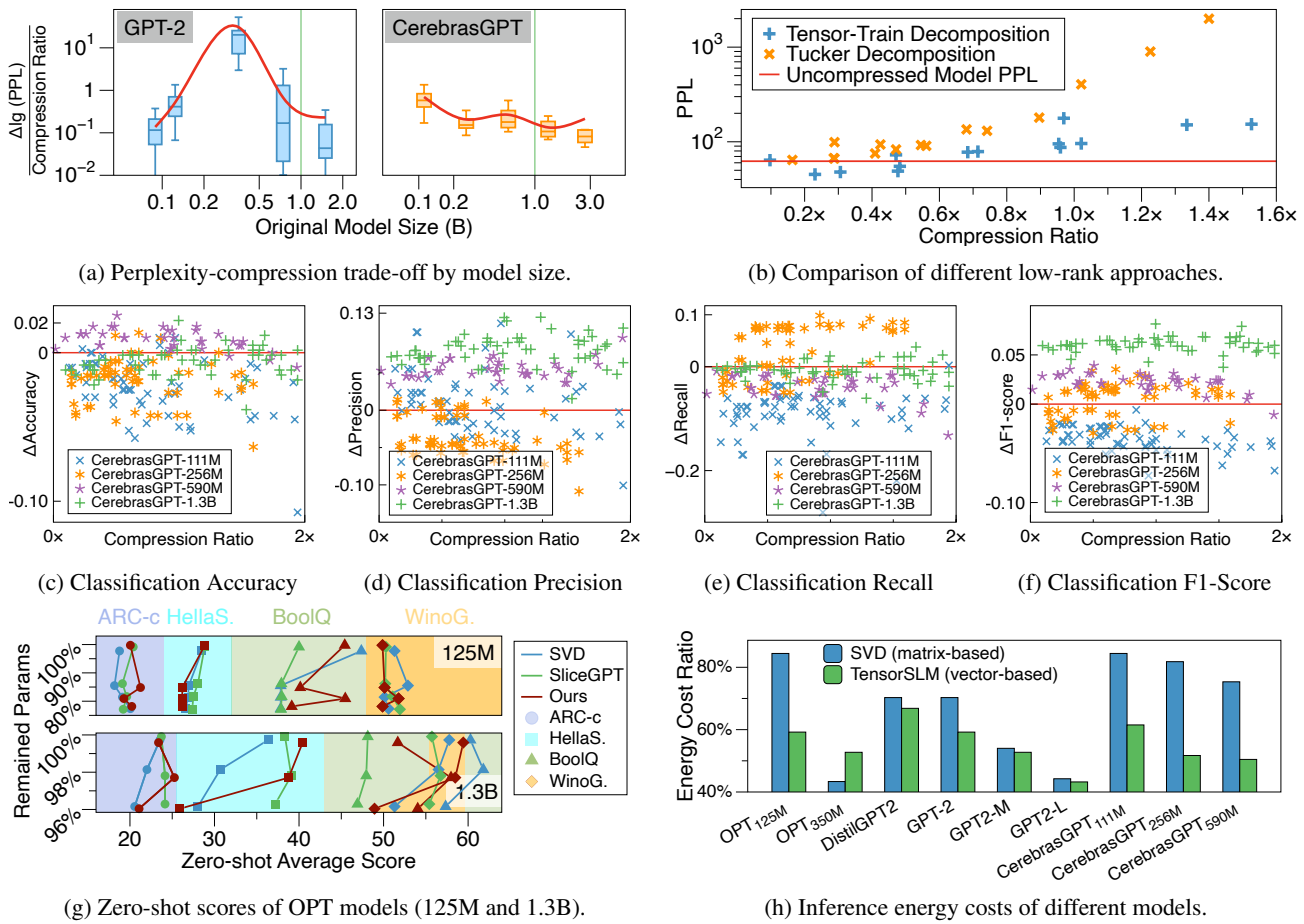
$$\mathcal{E}_\nu^{''} = \nu\left[k(V + 2d + l + 1) + ld\right], \tag{4}$$

$$\mathcal{E}_\tau^{''} = \tau(2ldk - ld + kd). \tag{5}$$

Therefore, we have the ratio of inference energy ω, between the compressed language models and the uncompressed models. Denote $\omega_{\texttt{TT}} = \frac{\mathcal{E}_\nu^{'} + \mathcal{E}_\tau^{'}}{\mathcal{E}_\nu + \mathcal{E}_\tau}$ as the ratio with TensorSLM, and $\omega_{\texttt{SVD}} = \frac{\mathcal{E}_\nu^{''} + \mathcal{E}_\tau^{''}}{\mathcal{E}_\nu + \mathcal{E}_\tau}$ as the ratio with SVD. We will give the estimated values of $\omega_{\texttt{TT}}$ and $\omega_{\texttt{SVD}}$ in Section 5 according to the hyperparameters of the investigated open-source SLMs.

4.2. Language Model Inference Process with the Compressed Embeddings

The original inference process with embedding vectors is as follows: when the encoded texts (separated as tokens) are forwarded to the embedding layer, the embedding layer outputs the embedding vectors according to the input tokens; the embedding layer here acts like a look-up table. The embedding vectors are then forwarded to the hidden layers of the transformer, whose size is the same as the dimension of the embedding vectors. Thus, if there is no internal change



(a) Perplexity-compression trade-off by model size.

(b) Comparison of different low-rank approaches.

(c) Classification Accuracy

(d) Classification Precision

(e) Classification Recall

(f) Classification F1-Score

(g) Zero-shot scores of OPT models (125M and 1.3B).

(h) Inference energy costs of different models.

Figure 2. Experimental results. (a): Perplexity-compression trade-off across different model sizes. This trade-off is measured by the ratio between perplexity and compression ratio of embedding layers; lower ratio values indicate better trade-offs. (b): Perplexity of the compressed models with different tensor decomposition approaches. (c)-(f): Task performance on sentiment classification with increasing compression ratio. Higher values indicate better classification performance. (g): Zero-shot reasoning scores of OPT series models on four different tasks. Our approach demonstrates competitive performance. (h): Energy cost ratio of compressed to uncompressed models, where 100% represents original energy consumption. Our approach overall outperforms the SVD-based approach.

in the hidden layers, the dimension of the embedding vectors should compile with the dimension of the hidden layers. The compressed embeddings should be reconstructed to the original dimension to enable the forwarding process. This inference happens at the application phase shown in the upper right of Figure. 1b.

Thus just before forwarding embedding vectors to the hidden layers, the memory usage increases from $l\sum_{k=1}^{N} r_{k-1}I_k r_k$ to ld. However, given that the vocabulary size V is normally much larger than the input token number l, that means $V \gg l$. Thus our approach can still significantly reduce the memory usage if the embedding layer takes a significant part of the whole model parameters. The reconstruction process follows the tensor contraction in Eq. (7), turning the TT cores $\{\mathcal{G}^{(k)}\}$ into a N-order tensor $\mathcal{X}$ according to Eq. (1), and then vectorizing $\mathcal{X}$ into a full-size embedding vector according to Appx. B.1.

5. Experimental Evaluation

Our comprehensive experimental evaluation covers compression ratio, language task performance changes, runtime (flops and latency), and energy consumption.

5.1. Changes of Language Task Performance

Perplexity-compression Trade-off. In most cases, the shrinkage of model size leads to a drop in the language task performance (though there are exceptions like the accuracy improvement of CerebrasGPT-590M in Figure. 2c). There should be approaches to measure such a trade-off, with the benefits of a more affordable model size, and how much language task performance has been sacrificed. Here we gave

Table 2. Computation and memory complexity during the compression (Section 4.1) and inference(Section 4.2) of TensorSLM. $\mathcal{M}_{\texttt{trans}}$ is the transformer module, V denotes the vocabulary size, d is the original token embedding dimension, and l is the token number of the input text. For simplicity, the dimensions for each mode of the tensor and TT rank are represented as I and r.

Memory	
Original Embedding Layers	$\mathcal{O}(Vd)$
Compressed Embedding Layers	$\mathcal{O}(VNIr^2)$
Compressed Encoded Texts	$\mathcal{O}(lNIr^2)$
Intermediate input to $\mathcal{M}_{\texttt{trans}}$	$\mathcal{O}(ld)$
Computation	
`TT-SVD` for single token embedding	$\mathcal{O}(NIr^3)$
Reconstruction of single token embedding	$\mathcal{O}(NIr^2)$

a simple approach for the task evaluated with perplexity, $\frac{\Delta \lg \texttt{PPL}(S,\mathcal{M})}{\eta_{\text{emb}}}$, with the measurements on GPT-2 and CerebrasGPT shown in Figure. 2a. We found that larger model sizes achieve better trade-offs, with CerebrasGPT showing a smoother trend compared to GPT-2.

Language Modelling. Due to the combination of tensor size and TT ranks exponentially exploding, we could not test all the possible combinations. However, we can still observe that independent of the tensor orders and the models used for the compression, significant language modelling performance loss tends to appear when the compression ratio exceeds $2.0\times$. We further compared our proposed approach with the Tucker decomposition in Figure. 2b with the same tensorization strategy in Section 4.1, and found our adopted Tensor-Train Decomposition outperforms the Tucker Decomposition in perplexity.

Sentiment Classification. The results of the sentiment classification task are shown in Figure. 2c to 2f, also indicate that the robustness of larger-scale models (Cerebras-590M and Cerebras-1.3B) is better than that of the smaller models (Cerebras-111M and Cerebras-256M), similar to the trend in language modelling tasks mentioned above. The compressed larger-scale models tend to outperform the original model in precision and F1-score, indicating that our compression improves the ability of the larger models to recognise the positive texts. In contrast, the smaller models tend to have worse performance when the compression ratio increases.

Zero-shot Reasoning. Since SLMs are incapable of the tasks that are too complex, we only evaluate the relatively simple reasoning tasks (e.g. those that do not involve multi-hop questioning, mathematics or multilingual), and the results are shown in in Figure. 2g. The bold numbers are the cases that outperform the uncompressed models, or the best in all the compressed cases.

Our approach has a higher chance of achieving better average reasoning task scores than the SVD-based approach, which implies that our tensors are better at extracting implicit representations in small size models than matrices. Moreover, in our evaluation, our approach generally has higher scores than the SVD-based approach in ARC-challenge and BoolQ. Both of these datasets are more unprompted and unconstrained compared to the other evaluated datasets. This fact implies that our approach may be better at these difficult, unconstrained reasoning tasks.

5.2. Latency

While TensorSLM significantly reduces the model parameters and even improves the language tasks performance, in practice it also introduced extra latencies - compression latency (Section 4.1) and inference latency(Section 4.2).

In our experimental evaluation, a typically induced latency for an input text was no more than 0.3 seconds, which is acceptable for edge applications. Due to space constraints, the comprehensive results and detailed analysis of the on-device latency evaluation are provided in Appx. G.

5.3. Energy Consumption

The estimated inference energy costs are shown in Figure. 2h. The Y-axis indicates the ratio between the inference energy costs of the compressed model and that of the uncompressed model; the lower, the better energy saving. For each language model, we select the compression case that has a similar language task performance according to Section 5.1.

We can observe that our approach is mostly better than the SVD-based approach. Furthermore, TensorSLMsupports adaptivity in edge applications, while the SVD-based approach does not.

6. Conclusion and Future Work

This paper addresses two unique requirements of Small Language Models (SLMs) deployed on low-end devices: *adaptivity* and *energy efficiency*. We propose a training-free approach to compress token embeddings using Tensor-Train Decomposition, enabling dynamic vocabulary adjustment and memory-computation trade-offs for extended battery life. We evaluated our approach on GPT-2, CerebrasGPT, and OPT models across language modeling, classification, and zero-shot reasoning tasks. Systematic measurements on Raspberry Pi 5 show that our method reduces inference energy costs by half, with negligible performance degradation and minimal latency increase.

Future work includes extending tensorization to hidden layers for native compilation and developing accelerated tensor operations to optimize CPU arithmetic requirements.

References

HuggingFace. SmolLM. https://huggingface.co/huggingface/Smol, 2024. [Accessed 20-11-2024]. 1

Abronin, V., Naumov, A., Mazur, D., Bystrov, D., Tsarova, K., Melnikov, A., Oseledets, I., and Brasher, R. Tq-compressor: improving tensor decomposition methods in neural networks via permutations. *arXiv preprint arXiv:2401.16367*, 2024. 11

Acharya, A., Goel, R., Metallinou, A., and Dhillon, I. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 6196–6203, 2019. 1, 11, 12

Bałazy, K., Banaei, M., Lebret, R., Tabor, J., and Aberer, K. Direction is what you need: improving word embedding compression in large language models. *arXiv preprint arXiv:2106.08181*, 2021. 12

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 7432–7439, 2020. 2, 14

Brown, T. B. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 13

Chekalina, V., Novikov, G., Gusak, J., Panchenko, A., and Oseledets, I. Efficient GPT model pre-training using tensor train matrix representation. In Huang, C.-R., Harada, Y., Kim, J.-B., Chen, S., Hsu, Y.-Y., Chersoni, E., A, P., Zeng, W. H., Peng, B., Li, Y., and Li, J. (eds.), *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, pp. 600–608, Hong Kong, China, December 2023a. Association for Computational Linguistics. URL https://aclanthology.org/2023.paclic-1.60. 11, 12

Chekalina, V., Novikov, G. S., Gusak, J., Oseledets, I., and Panchenko, A. Efficient gpt model pre-training using tensor train matrix representation. *ArXiv*, abs/2306.02697, 2023b. 12

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013. 13

Chen, P., Si, S., Li, Y., Chelba, C., and Hsieh, C.-J. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a. 1, 2, 11, 12

Chen, P., Yu, H.-F., Dhillon, I., and Hsieh, C.-J. Drone: Data-aware low-rank compression for large nlp models. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29321–29334. Curran Associates, Inc., 2021. 1, 11, 12

Chen, T., Min, M. R., and Sun, Y. Learning k-way d-dimensional discrete codes for compact embedding representations. In *International Conference on Machine Learning*, pp. 854–863. PMLR, 2018b. 12

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019. 2, 13, 14

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018. 2, 13, 14

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Re, C. Monarch: Expressive structured matrices for efficient and accurate training. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4690–4721. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/dao22a.html. 1, 2, 11, 12

Dey, N., Gosal, G., Khachane, H., Marshall, W., Pathria, R., Tom, M., Hestness, J., et al. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023. 13

Edalati, A., Tahaei, M., Rashid, A., Nia, V., Clark, J., and Rezagholizadeh, M. Kronecker decomposition for GPT compression. In Muresan, S., Nakov, P., and Villavicencio, A. (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 219–226, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.24. URL https://aclanthology.org/2022.acl-short.24. 11

Hrinchuk, O., Khrulkov, V., Mirvakhabova, L., Orlova, E., and Oseledets, I. Tensorized embedding layers. In Cohn, T., He, Y., and Liu, Y. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4847–4860, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.

436. URL https://aclanthology.org/2020.findings-emnlp.436. 1, 11, 12

Hsu, Y.-C., Hua, T., Chang, S., Lou, Q., Shen, Y., and Jin, H. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uPv9Y3gmAI5. 1, 11, 12

Laskaridis, S., Kateveas, K., Minto, L., and Haddadi, H. Melting point: Mobile evaluation of language transformers. *arXiv preprint arXiv:2403.12844*, 2024. 1, 3

Lin, C.-H., Gao, S., Smith, J. S., Patel, A., Tuli, S., Shen, Y., Jin, H., and Hsu, Y.-C. Modegpt: Modular decomposition for large language model compression. *arXiv preprint arXiv:2408.09632*, 2024. 12

Lin, J., Chen, W.-M., Lin, Y., cohn, j., Gan, C., and Han, S. Mcunet: Tiny deep learning on iot devices. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 11711–11722. Curran Associates, Inc., 2020. 11

Lioutas, V., Rashid, A., Kumar, K., Haidar, M. A., and Rezagholizadeh, M. Improving word embedding factorization for compression using distilled nonlinear neural decomposition. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 2774–2784, 2020. 1, 11

Liu, S., Lu, H., and Shao, J. Improved residual vector quantization for high-dimensional approximate nearest neighbor search. *arXiv preprint arXiv:1509.05195*, 2015. 12

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., Lai, L., and Chandra, V. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32431–32454. PMLR, 21–27 Jul 2024. 1, 12

Lu, Z., Li, X., Cai, D., Yi, R., Liu, F., Zhang, X., Lane, N. D., and Xu, M. Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790*, 2024. 1

Luo, H. and Sun, W. Addition is all you need for energy-efficient language models. *arXiv preprint arXiv:2410.00907*, 2024. 14

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015. 13

Mao, Y., Wang, Y., Wu, C., Zhang, C., Wang, Y., Yang, Y., Zhang, Q., Tong, Y., and Bai, J. Ladabert: Lightweight adaptation of bert through hybrid model compression. *arXiv preprint arXiv:2004.04124*, 2020. 11

Mehta, S., Sekhavat, M. H., Cao, Q., Horton, M., Jin, Y., Sun, C., Mirzadeh, I., Najibi, M., Belenko, D., Zatloukal, P., et al. Openelm: An efficient language model family with open-source training and inference framework. *arXiv preprint arXiv:2404.14619*, 2024. 1

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe. 13

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286. URL https://doi.org/10.1137/090752286. 3, 4

Qiu, S., Potapczynski, A., Finzi, M., Goldblum, M., and Wilson, A. G. Compute better spent: Replacing dense layers with structured matrices. *arXiv preprint arXiv:2406.06248*, 2024. 1, 2, 11, 12

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 13

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. 13, 14

Sanh, V. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. 13

Sap, M., Rashkin, H., Chen, D., Le Bras, R., and Choi, Y. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019. 14

Tahaei, M., Charlaix, E., Nia, V., Ghodsi, A., and Rezagholizadeh, M. KroneckerBERT: Significant compression of pre-trained language models through kronecker decomposition and knowledge distillation. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2116–2127, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.154. URL https://aclanthology.org/2022.naacl-main.154. 11

Wang, H., Li, R., Jiang, H., Wang, Z., Tang, X., Bi, B., Cheng, M., Yin, B., Wang, Y., Zhao, T., and Gao, J. Lighttoken: A task and model-agnostic lightweight token embedding framework for pre-trained language models. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 2302–2313, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599416. URL https://doi.org/10.1145/3580305.3599416. 12

Yang, T.-J., Chen, Y.-H., Emer, J., and Sze, V. A method to estimate the energy consumption of deep neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 1916–1920, 2017. doi: 10.1109/ACSSC.2017.8335698. 4

Yuan, Z., Shang, Y., Song, Y., Wu, Q., Yan, Y., and Sun, G. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023. 12

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. 2, 13, 14

A. Notation

Table 3. Notation in this paper.

Symbol	Meaning
a	Scalar.
$\mathbf{x}$	Vector.
$\mathbf{A}$	Matrix.
$\mathcal{X}$, $\mathcal{A}$, $\mathcal{B}$	Tensor.
N	Tensor order.
$\mathcal{X}[i_1, \ldots, i_N]$	The $(i_1, i_2, \ldots, i_N)$th entry of the tensor.
I, I_k	Tensor dimension, tensor dimension for the kth mode.
$\mathcal{M}$	Model module set.
$\lvert\mathcal{M}\rvert$, $\lvert\mathcal{G}\rvert$, $\lvert S\rvert$	Parameter count of the model module set $\mathcal{M}$, tensor $\mathcal{G}$ or cardinality of set S.
V	Vocabulary of the language model.
d	Token embedding dimension.
l	Input text length.
r, r_k	TT rank, TT rank of the kth mode of the tensor.
$\mathcal{G}^{(k)}$	TT(MPS) core of the kth mode of the tensor.
$\times_k^p$	Tensor contraction for the pth (formal tensor) and kth (latter tensor) mode.
η	Compression ratio of the entire model.
$\eta_{\texttt{emb}}$	Compression ratio of the embedding layer.
φ	Parameter reduction ratio of the whole model.
$\varphi_{\texttt{emb}}$	Parameter reduction ratio of the embedding layer.
ν	Memory energy consumption per `float32` data.
τ	Computation energy consumption per `float32` data.
$\mathcal{E}_\nu$	Estimated energy cost regarding memory.
$\mathcal{E}_\tau$	Estimated energy cost regarding computation.
$\omega_{\texttt{TT}}$	Estimated energy cost ratio between the compressed model with TensorSLMand uncompressed model.
$\omega_{\texttt{SVD}}$	Estimated energy cost ratio between the compressed model with SVD and the uncompressed model.

B. Preliminaries

B.1. Tensors and Tensor Operations

This section gives brief mathematical preliminaries of tensor algebra, and basic knowledge in LLMs to facilitate the understanding of our proposed methodology in Section 4.

Order-N Tensor. An order-N real-valued tensor is a multi-dimensional array, denoted by a calligraphic font, e.g., $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, where N is the order of the tensor (i.e., number of modes), and I_n ($1 \leq n \leq N$) is the size (i.e., the dimension) of its n-th mode. Matrices (denoted by bold capital letters, e.g., $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2}$) can be seen as order-2 tensors ($N = 2$), vectors (denoted by bold lower-case letters, e.g., $\mathbf{a} \in \mathbb{R}^I$) can be seen as order-1 tensors ($N = 1$), and scalars (denoted by lower-case letters, e.g., $a \in \mathbb{R}$) are order-0 tensors ($N = 0$).

Tensor Entries. The $(i_1, \ldots, i_N)$-th entry of an order-N tensor is denoted by $a_{i_1,\cdots,i_N} \in \mathbb{R}$, where $i_n = 1, \ldots, I_n$ for $n = 1, \ldots, N$. A tensor fiber is a vector of tensor entries obtained by fixing all but one index of the original tensor (e.g., $\mathbf{a}_{:,i_2,i_3,\ldots,i_N} \in \mathbb{R}^{I_1}$). Similarly, a tensor slice is a matrix of tensor entries obtained by fixing all but two indices of the original tensor (e.g., $\mathbf{A}_{:,:,i_3,i_4,\ldots,i_N} \in \mathbb{R}^{I_1 \times I_2}$).

Tensorization. A vector $\mathbf{a} = (a_1, a_2, \ldots, a_{I_1 I_2 \cdots I_N}) \in \mathbb{R}^{I_1 I_2 \cdots I_N}$, can be tensorized (or "folded", "reshaped") into an order-N tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, so that

$$\mathcal{A}[i_1, i_2, \ldots, i_N] = a_{1+\sum_{k=1}^{N}(i_k-1)\prod_{p=1}^{k-1} I_p}, \qquad 1 \leq i_k \leq I_k, \tag{6}$$

where $\mathcal{A}[i_1, i_2, \ldots, i_N]$ denotes the $(i_1, i_2, \ldots, i_N)$-th entry of tensor $\mathcal{A}$.

Vectorization. Given an order-N tensor, $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, its vectorization reshapes the high-dimensional matrix into a vector, $\texttt{vec}(\mathcal{A}) = \mathbf{a} \in \mathbb{R}^{I_1 \cdots I_N}$.

Tensor Contraction. The contraction of $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, over the kth and pth modes respectively, where $I_k = J_p$ is denoted as $\mathcal{A} \times_k^p \mathcal{B}$ and results in a tensor $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times I_{k+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{p-1} \times J_{p+1} \times \cdots \times J_M}$, with entries

$$\begin{aligned} &\mathcal{C}[i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_N, j_1, \ldots, j_{p-1}, j_{p+1}, \ldots, j_M] \\ &= \sum_{q=1}^{I_k} \Big(\mathcal{A}[i_1, \ldots, i_{k-1}, q, i_{k+1}, \ldots, i_N] \\ &\qquad \cdot \mathcal{B}[j_1, \ldots, j_{p-1}, q, j_{p+1}, \ldots, j_M] \Big) \end{aligned} \tag{7}$$

Matricization (Mode-n unfolding). Mode-n matricization of a tensor, $\texttt{mat}(\mathcal{A}, n) = \mathbf{A}_{\{n\}} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$, is a procedure of mapping the elements from a multidimensional array to a two-dimensional array (matrix). Conventionally, such procedure is associated with stacking mode-n fibers (modal vectors) as column vectors of the resulting matrix. For instance, the mode-1 unfolding of $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is represented as $\texttt{mat}(\mathcal{A}, 1) = \mathbf{A}_{\{1\}} \in \mathbb{R}^{I_1 \times (I_2 \cdots I_N)}$, where the subscript, $\{1\}$, denotes the mode of matricization, and is given by

$$\mathbf{A}_{(1)} \left[i_1, \overline{i_2 i_3 \ldots i_N} \right] = \mathcal{A}[i_1, i_2, \ldots, i_N] \tag{8}$$

Note that the overlined subscripts refer to linear indexing (or Little-Endian), given by:

$$\begin{aligned} \overline{i_1 i_2 \ldots i_N} &= 1 + \sum_{n=1}^{N} \left[(i_n - 1) \prod_{n'=1}^{n-1} I_{n'} \right] \\ &= 1 + i_1 + (i_2 - 1) I_1 + \cdots + (i_n - 1) I_1 \ldots I_{N-1} \end{aligned} \tag{9}$$

B.2. Related Work in Detail

Low-rank factorization can break the high-dimensional weight matrices into smaller matrices or tensors, so that the overall size of the model can be shrunk. According to the dimensions of the structure that the original weight matrices are broken into, these approaches can be divided into matrix-based and tensor-based.

Matrix-based Approaches. A straightforward way to shrink the model size is to decompose weight matrices via singular value decomposition (SVD) (Acharya et al., 2019), which can be further improved by the weighted approach considering the model performance afterwards (Hsu et al., 2022), knowledge distillation (Lioutas et al., 2020; Mao et al., 2020) and pruning (Mao et al., 2020). There are also some block-wise decomposition approaches used in language model compression, like Kronecker Products (Tahaei et al., 2022; Edalati et al., 2022) and data-driven block-wise partitioning (Chen et al., 2018a; 2021).

(Dao et al., 2022; Qiu et al., 2024) used the block-diagonal matrices to reduce the FLOPs in the linear layers computation, with the bonus of shrinking the model size. However, our paper focuses on reducing the parameters of embedding layers, and there is no monotonous relationship between the FLOPs (computation cost) and parameters (memory usage) (Lin et al., 2020). Also, their investigated matrix multiplication only occurs in feed-forward layers, thus their approaches do not fit the embedding layer compression. Moreover, block-diagonal matrices are optimised for GPUs for better parallelization. Our aim of minimizing the number of parameters, makes it optimized for lower-end edge devices rather than GPUs. Indeed, on Raspberry Pi 5, the additional forwarding latency due to compressed embeddings (0.330 - 0.364ms /token in Table. 5) is even faster than that on GPU (measured as 0.463ms /token in our setting), since there is no parallelization during this forwarding process.

Tensor-based Approaches. Despite some efforts to use tensor decomposition to compress the language model size, all come with an extra training process. The works in (Abronin et al., 2024) use Kronecker decomposition with row-column permutation during the GPT model fine-tuning process, while (Hrinchuk et al., 2020) and (Chekalina et al., 2023a) propose a tensor-train structured embedding layer and GPT model respectively, yet both train the new-structured model from scratch.

C. Why not existing solutions?

Table 4. Comparison with our approach and the relevant research.

Relevant Study	Device		Training ?	Algebra Structure		Layer		Focused Size	
	high-end	low-end		matrix	tensor	Emb	Linear	large	small
(Chen et al., 2018a)	√		√	√		√			√
(Hrinchuk et al., 2020)	√				√	√			√
(Wang et al., 2023)	√		√	√		√			√
(Bałazy et al., 2021)		√	√	√		√			√
(Liu et al., 2015)	-		√	√		-			
(Chen et al., 2018b)	√		√	√		√			√
(Yuan et al., 2023)	√			√			√	√	
(Hsu et al., 2022)	√		√	√			√		√
(Chekalina et al., 2023b)	√		√		√		√	√	
(Lin et al., 2024)	√			√			√	√	
(Dao et al., 2022)	√		√	√			√		
(Qiu et al., 2024)	√		√	√	√		√		
(Liu et al., 2024)	√		√	-					√
TensorSLM(Ours)		√			√	√		√	

The field of language model compression with low-rank factorization has been booming in recent years. The recent relevant works are summarized in Table. 4. We can observe that for the current existing works, some are specialized for embedding layers (Chen et al., 2018a; Hrinchuk et al., 2020; Wang et al., 2023; Bałazy et al., 2021; Acharya et al., 2019; Liu et al., 2015) while others are not (Chekalina et al., 2023a; Chen et al., 2021; Hsu et al., 2022; Dao et al., 2022; Qiu et al., 2024). However, all of these require an extra training process, such as fine-tuning, meta-learning (Chen et al., 2018a; 2021; Hsu et al., 2022; Bałazy et al., 2021; Liu et al., 2015; Dao et al., 2022; Wang et al., 2023; Qiu et al., 2024) and training from scratch (Hrinchuk et al., 2020; Chekalina et al., 2023a).

There are two limitations to this extra training: 1) extra training involves additional computation and training data, which may be unavailable for low-end devices; 2) training the language model from scratch discards the valuable knowledge stored in the weights of the original models. However, we only focus on training-free low-end device applications. For a more detailed discussion of these relevant works, please refer to Appx. B.2.

D. Perplexity and Logarithmic Perplexity.

Perplexity is used as a performance evaluation metric of the language modelling task, which has the following form

$$\texttt{PPL}(S, \mathcal{M}) = \left(\prod_{i=1}^{|S|} p_{\mathcal{M}}(x_i | x_1, x_2, \ldots, x_{i-1}) \right)^{-1} \tag{10}$$

where S is an ordered set (token sequence), consisting of a set of tokens $\{x_t\}$, $t = 1, 2, \ldots, |S|$, and $\mathcal{M}$ is the model block that contains all the modules of the language model we evaluate.

Notice that the compression ratio Eq. (21) has a linear form, while perplexity Eq. (10) has an exponential form, so it is hard to combine them as a description of a model compression result, since when compression ratio η linearly increases, the perplexity $\texttt{PPL}$ explodes exponentially. To this end, we use the following logarithmic form to describe the language modelling performance

$$\ln \texttt{PPL}(S, \mathcal{M}) = -\sum_{i=1}^{|S|} \ln p_{\mathcal{M}}(x_i | x_1, x_2, \ldots, x_{i-1}) \tag{11}$$

Now, the language modelling performance change before and after compression is given by

$$\Delta \ln \texttt{PPL}(S, \mathcal{M}) = \ln \texttt{PPL}(S, \mathcal{M}_{\texttt{cmpr}}) - \ln \texttt{PPL}(S, \mathcal{M}_0) \tag{12}$$

$$= \sum_{i=1}^{|S|} \ln \frac{p_{\mathcal{M}_0}(x_i | x_1, x_2, \ldots, x_{i-1})}{p_{\mathcal{M}_{\texttt{cmpr}}}(x_i | x_1, x_2, \ldots, x_{i-1})}, \tag{13}$$

observe that Eq. (12) exhibits linearity, like Eq. (21).

E. Proof of the Highest Compression Ratio in Table. 2

Proposition E.1. *For an order-N tensor whose dimension for each order are I, its TT-format yields the highest compression ratio when $I = 2$ and TT rank $r = 1$.*

Proof. Assume the tensor size $[I_1, \dots, I_N]$ for the tensor $\mathcal{X}$ to achieve the highest compression rate, we next give the proof of this hyperparameter selection.

The compression ratio in Section 4.1 can be represented as

$$\eta = \frac{V \times d}{\sum_{j=1}^{V} \sum_{n=1}^{N} (r_{n-1} \times I_n \times r_n)_j} \tag{14}$$

$$= \frac{V \times d}{I_1 r_1 + r_1 I_2 r_2 + \cdots + r_{N-2} I_{N-1} r_{N-1} + r_{N-1} I_N} \tag{15}$$

$$= \frac{V \times d}{\sum_{k=1}^{\lfloor \frac{N+1}{2} \rfloor} r_{2k-1} \left(r_{2k-2} I_{2k-1} + I_{2k} r_{2k+1} \right)} \tag{16}$$

For the simplest case, assume $I_1 = \cdots = I_N = I$ and $r_1 = \cdots = r_N = r$. Given $d = \prod_{n=1}^{N} I_n = I^N$, we have $N = \log_I D$, and

$$\eta = \frac{V \times d}{rI\left[2 + (N-2)r\right]} = \frac{V \times d}{rI\left[2 + (\log_I d - 2)\right]}. \tag{17}$$

In Equation Eq. (17), the numerator is a constant, and in the denominator, R is a hyperparameter for the Tensor-Train Decomposition. Thus the objective function for the highest compression rate η is

$$\min_{I,N} rI\left[2 + (N-2)\right] \qquad \textbf{s.t.} \quad N = \log_I d \tag{18}$$

$$I, N, r \in \mathbb{Z}^+ \tag{19}$$

$$2 \le I \le N \le \lfloor \log_2 d \rfloor \tag{20}$$

Regarding Eq. (18), if eliminate N then we have a function $h = rI\left[2 + (\log_I d - 2)\right]$. Regarding d in Eq. (20), the largest token embedding size of recent GPT-3 (Brown, 2020) is 12,888. Thus, for the GPT series models no later than GPT-3, Eq. (18) should be $2 \le I \le N \le 13$. In this range, h is a monotonically increasing function, where the minimum h occurs at $I = 2$.

Therefore, for the simplest case, we have the best hyperparameter selection of $I_1 = I_2 = \cdots = I_N = 2$, and $N = \lfloor \log_2 d \rfloor$.

□

F. Experimental Setup

F.1. Models, Tasks and Dataset.

The sub-billion models we used are DistilGPT2 (Sanh, 2019), GPT2, GPT2-M/L (Radford et al., 2019), CerebrasGPT-111M/256M/590M (Dey et al., 2023), OPT-125M. We also tested the models of slightly over a billion parameters for language task performance with GPT2-XL (1.5 billion parameters), CerebrasGPT-1.3B and OPT-1.3B for the boundary tests.

Regarding the language tasks, we have two different level language tasks:

- **Simple Tasks**: language modelling and sentiment classification. For language modelling, the considered datasets are WikiText2, WikiText103 (Merity et al., 2017) and 1BW (Chelba et al., 2013). For sentiment classification, the considered dataset is IMDB (Maas et al., 2011).
- **Complex Tasks**: zero-shot common sense reasoning tasks. The common sense reasoning datasets include ARC-challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019) and WinoGrade (Sakaguchi et al., 2021).

F.2. Hardware.

Our main experiments were completed on a GPU workstation with an RTX A6000 48GB GPU and AMD Ryzen Threadripper PRO 5955WX CPU. The GPU resource was mainly used to fine-tune language modelling models for sequence classification, which is the requirement of the sentiment classification task. The inference latency of the low-end devices was measured on a Raspberry Pi 5, with a 64-bit Arm Cortex-A76 CPU and 8GB DRAM. The power meter we used is YOJOCK J7-c USB C Tester USB Power Meter, with a single refresh time of more than 500ms.

F.3. Evaluation Metrics

Compression Ratio. Denote $\mathcal{M}$ as a model block set containing a list of model modules like embedding layers and attention layers. With $\mathcal{M}_0$ as the original model block set, $\mathcal{M}_{\texttt{cmpr}}$ as the compressed version of $\mathcal{M}_0$, and $|\mathcal{M}|$ as the parameter count of $\mathcal{M}$. The compression ratio η is defined as

$$\eta = \frac{|\mathcal{M}_0| - |\mathcal{M}_{\texttt{cmpr}}|}{|\mathcal{M}_{\texttt{cmpr}}|}. \tag{21}$$

Specifically, the embedding compression rate is $\eta_{\text{emb}} = \frac{|\mathcal{T}_0| - |\mathcal{T}_{\texttt{cmpr}}|}{|\mathcal{T}_0|}$, where $\mathcal{T}$ only contains token embedding layer and position embedding layer.

Perplexity and Logarithmic Perplexity. We use perplexity (PPL) as our metrics of language modelling. Furthermore, we use the logarithmic form of perplexity ($\ln \texttt{PPL}$) and its change ($\Delta \ln \texttt{PPL}$) to align with the linearity of the compression ratio Eq. (21), as defined in Eq. (10).

Accuracy, Precision, Recall and F1-Score. We use these four common evaluation metrics for classification to analyze the classification performance of the compressed model comprehensively. To investigate the performance change before and after compression, we use the difference between the metric values after and before the compression.

Zero-shot Reasoning Scores. For the metrics of reasoning tasks, we use the scores from (Clark et al., 2018; 2019; Zellers et al., 2019; Bisk et al., 2020; Sap et al., 2019; Sakaguchi et al., 2021).

Energy Consumption. Since the actual energy consumption depends on multiple uncontrollable factors, as we discussed in Section 2.2, it is difficult to isolate compression energy cost from the actual measurements. Thus, we use similar approaches in (Luo & Sun, 2024) to estimate the energy consumption.

We use the notations in Table. 2 and Eq. (2) to (4), and approximate the ratio between computation energy cost and memory energy cost per `fload32` data as $\frac{\nu}{\tau} = 5$. Then, we got the configurations of the current open-source SLMs for the values of d, V in Eq. (2) to (4). Though we cannot get the actual energy costs, we can compare the inference energy costs of compressed and uncompressed models with this approach.

G. On-device Latency Explained with Experimental Results

For the compression latency, we investigated the compression latency on the token level, as shown in Table. 5. Here, "original" means the uncompressed model, while $\texttt{PPL}_\alpha$ means the compressed model with a negligible task performance drop. In our case, "negligible task performance drop" means in the language modelling task, the perplexity is no more than 100.0. The notation φ_{max} refers to the compressed model with maximum compression ratio. We observed that for individual token embeddings, there was no significant latency difference between high-end servers and Raspberry Pi, typically no more than 2 milliseconds for each token. Thus, it is acceptable for the Raspberry Pi to compress the individual token embeddings.

Table 5. The latency (ms/token) of tensorization & decomposition token embedding vectors and reconstruction on the high-end and lower-end devices. $\texttt{PPL}_\alpha$ means the compressed model with a negligible task performance drop, and the symbol φ_{max} represents the case with a maximum compression ratio. d_{emb} is the embedding dimension of the token embedding vector, and the tested models are GPT-2 and GPT-2-M. On the CPU level, for single token embedding vector decomposition and reconstruction, both server and edge devices have no significant computation overhead.

Device (CPU) (ms/token)	d_{emb}	**tensorization & decomposition**		**reconstruction**	
		$\texttt{PPL}_\alpha$	φ_{max}	$\texttt{PPL}_\alpha$	φ_{max}
Server	768	0.627	1.429	0.117	0.238
	1024	0.452	1.512	0.114	0.261
Raspberry Pi 5	768	0.760	1.948	0.330	0.468
	1024	0.612	2.148	0.364	0.614

For the inference latency of a single text, we chose a typical text length of 50 tokens, as shown in Table. 6. we used "original", $\texttt{PPL}_\alpha$, φ_{max} same as those in Table. 5, to represent the uncompressed model, the compressed model with a negligible task performance drop and the model with a maximum compression ratio. A typical induced latency for an input text was no more than 0.3 seconds, which is acceptable for edge applications.

It should be noted that the embedding reconstruction latency depends on both tensor shapes and flops, and the on-device memory management varies when models of different sizes are loaded. Consequently, in Table. 6, flops alone does not provide a complete predictor of on-device inference latency.

The cases of φ_{max} are typically slower than the cases of $\texttt{PPL}_\alpha$. We have demonstrated in Appx. E that the φ_{max} (the maximum compression ratio) corresponds to cases where embedding vectors are compressed into TT-formatted tensors of the highest orders. During the forward passes, the TT-format of these tensors is decompressed order by order. For example, for an N-order TT-formatted tensor, the decompression process involves $(N-1)$ *serial* matrix multiplications.

Table 6. Parameters, number of floating-point operations (flops) of the compressed and uncompressed sub-billion models, and latency on Raspberry Pi CPU. For flops, the token number of the input texts is 100, while for latency on Raspberry Pi, the token number is 50.

GPT Models		GPT2				CerebrasGPT		
		DistilGPT2	**GPT-2**	**GPT-2-M**	**GPT-2-L**	**111M**	**256M**	**590M**
# Params (M)	original	81.9	124.44	354.82	774.03	111.05	255.98	590.31
	$\texttt{PPL}_\alpha$	67.06	106.36	326.45	734.28	101.78	226.69	543.45
	φ_{max}	43.45	85.99	303.88	710.83	71.87	200.59	511.07
flops (10^6/text)	original	20250	40490	142250	330980	14470	40400	103060
	$\texttt{PPL}_\alpha$	+1.65	+1.88	+3.11	+2.30	+0.38	+1.63	+2.30
	φ_{max}	+0.13 -92%	+0.13 -93%	+0.20 -94%	+0.25 -89%	+0.13 -66%	+0.12 -93%	+0.26 -84%
Latency on Raspberry Pi (s/text)	original	$0.19_{\pm 0.02}$	$0.50_{\pm 0.19}$	$1.23_{\pm 0.12}$	$3.01_{\pm 0.47}$	$0.47_{\pm 0.21}$	$0.71_{\pm 0.02}$	$1.81_{\pm 0.25}$
	$\texttt{PPL}_\alpha$	$0.36_{\pm 0.19}$	$0.50_{\pm 0.16}$	$1.26_{\pm 0.22}$	$3.01_{\pm 0.29}$	$0.48_{\pm 0.23}$	$1.01_{\pm 0.29}$	$1.89_{\pm 0.28}$
	φ_{max}	$0.19_{\pm 0.03}$ -47%	$0.71_{\pm 0.38}$ $+42\%$	$1.55_{\pm 0.36}$ $+23\%$	$3.52_{\pm 0.44}$ $+16\%$	$0.72_{\pm 0.42}$ $+50\%$	$0.95_{\pm 0.27}$ -6%	$1.91_{\pm 0.24}$ $+1\%$

This implies that the higher the tensor order, the more matrix multiplication rounds are executed, potentially resulting in slower decompression. A (not representative) exception is the compression for CerebrasGPT-256M, which has the 5-order tensor shape $4 \times 2 \times 17 \times 4 \times 2$ for $\texttt{PPL}_\alpha$, and 7-order tensor shape $2 \times 2 \times 2 \times 2 \times 17 \times 2 \times 2$ for φ_{max}. The decompression process for each embedding vector involves 4 and 6 matrix multiplications respectively, which differ by only two matrix multiplications — a relatively small gap compared to other cases (e.g. 3-order for $\texttt{PPL}_\alpha$ and 10-order for φ_{max}). Thus, for CerebrasGPT-256M, the on-device inference latencies of $\texttt{PPL}_\alpha$ and φ_{max} are similar, as shown in Table. 6.

Though this compression approach does not provide latency reduction benefits, it does offer advantages in the reduction of memory usage and energy consumption.

DistilGPT2 exhibits different flops-latency trends from the others. In Table. 6 the compression for DistilGPT2 has significantly less latency for φ_{max} than $\texttt{PPL}_\alpha$, which contradicts the analysis in the preceding paragraph. A possible reason is the different memory scheduling processes of embedding layers and non-embedding layers. DistilGPT has the same embedding layer weight matrix size (50257×768) as GPT-2, yet has significantly fewer non-embedding layer parameters (and hence $\sim 50\%$ fewer non-embedding memory pages during inference). This difference may lead to distinct memory management dynamics.