Saten: Sparse Augmented Tensor Networks for Post-Training Compression of Large Language Models

Ryan Solgi[1], Kai Zhen[2], Rupak Vignesh Swaminathan[2], Nathan Susanj[2], Athanasios Mouchtaris[2], Siegfried Kunzmann[2], Zheng Zhang[1]
[1]University of California-Santa Barbara, USA
[2]Amazon, USA
solgi@ucsb.edu, zhengzhang@ece.ucsb.edu

Abstract

The efficient implementation of large language models (LLMs) is crucial for deployment on resource-constrained devices. Low-rank tensor compression techniques, such as tensor-train (TT) networks, have been widely studied for over-parameterized neural networks. However, their applications to compress pre-trained large language models (LLMs) for downstream tasks (post-training) remains challenging due to the high-rank nature of pre-trained LLMs and the lack of access to pretraining data. In this study, we investigate low-rank tensorized LLMs during fine-tuning and propose sparse augmented tensor networks (Saten) to enhance their performance. The proposed Saten framework enables full model compression. Experimental results demonstrate that Saten enhances both accuracy and compression efficiency in tensorized language models, achieving state-of-the-art performance. [1]

1 Introduction

Transformers have shown great success in various natural language processing tasks (Vaswani et al., 2017; Devlin et al., 2019; Raffel et al., 2020). However, their large number of parameters and computational demands hinder their implementation on resource-constrained devices. Consequently, various methods for LLM compression have been studied, including pruning (Sanh et al., 2020), distillation (Sanh et al., 2019), quantization (Shen et al., 2020), and matrix factorization (Lan et al., 2020; Hsu et al., 2022; Gao et al., 2024).

Low-rank tensor factorization is one of the prominent neural network compression techniques that has been widely studied and applied to various neural network architectures (Lebedev et al., 2014; Yang et al., 2017; Hawkins and Zhang, 2019). The tensor-train (TT) format is the most common tensor decomposition method used for neural network compression (Novikov et al., 2015). Furthermore, some studies successfully applied the idea of low-rank plus sparsity in tensor completion for signal processing (Mateos and Giannakis, 2012; Driggs et al., 2019).

Despite their success in improving the efficiency of LLM fine-tuning and pre-training (Yang et al., 2024a,b,c), low-rank tensor compression of pre-trained language models has not yet been successfully reported. This task mainly faces two challenges. First, almost all open-source LLMs are trained using uncompressed approaches, and the resulting model parameters of many layers may not exhibit a low-rank structure. Second, there is a lack of access to pre-training data to fully retrain low-rank parameters. These challenges result in a significant performance drop when post-training compression of language models is performed during fine-tuning using low-rank tensor factorization.

Paper Contributions. In this study, we address the challenges mentioned above by introducing sparse augmented tensor networks (Saten). Our specific contributions are summarized below:

- We propose Saten to compress LLMs with TT decomposition by incorporating a sparse error approximation. We study both unstructured and structured sparsity patterns in the Saten model.
- We analyze the model and computational complexities of the model compressed by Saten. When the tensor ranks are low and sparsity ratios are high, our model shows both memory and computational savings in inference.
- We compare Saten, TT and the recent SVD with adaptive rank selection (Gao et al., 2024) on BERT-Base and LlaMA. The experiments demonstrate that Saten achieves state-of-the-art performance in both compression efficiency and model accuracy.

[1]Representative code and implementation details are available at: https://github.com/rmsolgi/saten.git.

2 Background

Tensors are generalizations of matrices to higher orders and can be represented by multi-dimensional data arrays (Kolda and Bader, 2009). A real tensor of order d and dimension can be denoted as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, where n_j is the size of dimension j. The $(i_1, i_2, \cdots, i_d)$-th element of $\boldsymbol{\mathcal{A}}$ is $a_{i_1 i_2 \cdots i_d}$. Obviously, matrices and vectors are tensors of order $d = 2$ and $d = 1$, respectively.

Tensor Train (TT) Decomposition. Many practical tensors have low ranks and can be approximated using low-rank tensor decompositions. The TT decomposition approximates an order-d tensor $\boldsymbol{\mathcal{A}}$ with d factors $\{\boldsymbol{\mathcal{G}}_j \in \mathbb{R}^{r_{j-1} \times n_j \times r_j}\}_{j=1}^{d}$:

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}}_1 \times_3^1 \boldsymbol{\mathcal{G}}_2 \times_3^1 \cdots \times_3^1 \boldsymbol{\mathcal{G}}_d, \tag{1}$$

where $r_0, r_1, \cdots, r_d$ are TT ranks and $r_0 = r_d = 1$. Here, operator $\times_3^1$ refers to contracting the third and first dimensions of the left and right tensors, respectively (see Appendix A for more details).

3 The Saten Method

In this section, we present the Saten framework to compress a pre-trained LLM. Since many layers of a pre-trained LLM may not exhibit a low-rank property, compressing a pre-trained model directly using TT decomposition can result in huge accuracy drop. To address this challenge, Saten approximates the matrix $\mathbf{W}$ as follows:

$$\mathbf{W} \approx \hat{\mathbf{W}}_{\text{TT}} + \mathbf{E} \tag{2}$$

where $\mathbf{E}$ is a sparse matrix, $\hat{\mathbf{W}}_{\text{TT}}$ is the matrixization of a tensor $\hat{\boldsymbol{\mathcal{W}}}_{\text{TT}}$ with a TT representation:

$$\hat{\boldsymbol{\mathcal{W}}}_{\text{TT}} = \boldsymbol{\mathcal{G}}_1 \times_3^1 \boldsymbol{\mathcal{G}}_2 \times_3^1 \cdots \times_3^1 \boldsymbol{\mathcal{G}}_{d+k}. \tag{3}$$

The decomposition in Saten is illustrated in Fig. 1.

3.1 Computing the Saten Model

Low-rank TT component. We fist get the low-rank TT repsentation via the following steps:

- **Step 1:** We factor the dimensions of the matrix $\mathbf{W}$ as $N = n_1 \times n_2 \cdots \times n_k$ and $M = m_1 \times m_2 \cdots \times m_d$, and fold $\mathbf{W}$ to an order-$(d+k)$ tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1 \times \cdots m_d \times n_1 \cdots n_k}$.
- **Step 2:** We perform TT decomposition over $\boldsymbol{\mathcal{W}}$:

$$\boldsymbol{\mathcal{W}} \approx \hat{\boldsymbol{\mathcal{W}}}_{\text{TT}} = \boldsymbol{\mathcal{G}}_1 \times_3^1 \boldsymbol{\mathcal{G}}_2 \times_3^1 \cdots \times_3^1 \boldsymbol{\mathcal{G}}_{d+k}. \tag{4}$$

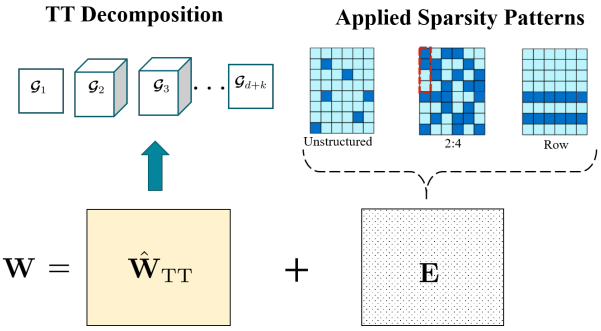

Figure 1: The sparse + low-rank tensor train representation for a weight matrix (or embedding table).

The accuracy of the TT decomposition highly depends on two factors: the TT ranks $(r_0, r_1 \cdots r_{d+k})$, and tensor shape $(n_1, \cdots n_d, m_1, \cdots m_k)$. In Saten, we apply an error-based TT-SVD method (see Appendix A) in each layer to automatically determine the TT ranks, which greatly reduces the number of hyper-parameters. In order to find a memory-efficient tensor shape in the compression process, Saten solves the tensor shape optimization problem using the method described in Appendix B.

Sparse component. After getting the TT factors in (3), we further determine the sparse matrix $\mathbf{E}$. We assume that $\mathbf{M} \in \{0,1\}^{N \times M}$ is a sparsity mask for $\mathbf{E}$, and the sparse matrix is decided as:

$$\mathbf{E} = \left(\mathbf{W} - \hat{\mathbf{W}}_{\text{TT}}\right) \odot \mathbf{M}, \tag{5}$$

where $\odot$ denotes element-wise product operations. To compress the linear layers, we have implemented two different methods: Saten(u) and Saten(2:4).

- **Saten(u).** In this method, we consider unstructured sparsity: the error is pruned (by setting the associated elements in $\mathbf{M}$ as zero) to preserve the top percent absolute values of matrix $\mathbf{W} - \hat{\mathbf{W}}_{\text{TT}}$.
- **Saten(2:4).** In this method we consider the 2:4 structured sparsity to enhance the efficiency on GPU. Specifically, of every 4 elements along a column of the error matrix $\mathbf{W} - \hat{\mathbf{W}}_{\text{TT}}$, the 2 largest-magnitude elements are preserved in $\mathbf{E}$.

When $\mathbf{W}$ is the parameter in an embedding layer, a different sparsity pattern (**row sparsity**) is used to compute $\mathbf{E}$. Specifically, the error $\mathbf{W} - \hat{\mathbf{W}}_{\text{TT}}$ is pruned to retain only the rows corresponding to the most frequent tokens in the training dataset.

Algorithm 1 Language Model Compression using Saten

Require: A pre-trained model
  for layer **in** targeted layers **do**
    Extract layer's weight ($\mathbf{W}$);
    Fold $\mathbf{W}$ into a high-order tensor $\boldsymbol{\mathcal{W}}$;
    Compute TT factors for $\boldsymbol{\mathcal{W}}$ [Eq. (4)]
    Compute sparse error; $\mathbf{E}$ with a chosen sparsity pattern;
    Replace layer by the Saten layer [Eq. (7)].
  end for
  Fine-tune the Saten model.
  Return the Saten model

Fine-tuning. After deciding the values of low-rank TT factors and the sparse matrix, we still need to fine-tune the compressed model to further boost the performance. This can be done directly in the **compressed format**. Specifically, the matrix $\mathbf{W}$ does not need to be recovered, and we can use automatic differentiation to compute directly the stochastic gradient w.r.t. the TT factors $\{\boldsymbol{\mathcal{G}}_j\}_{j=1}^{d+k}$ and w.r.t. the sparse matrix $\mathbf{E}$. Then these compressed components are updated via stochastic gradient descent.

The overall flow of the Saten framework is summarized in Algorithm 1.

3.2 Complexity of Saten

In this subsection, we analyze the model complexity and computational complexity of the compressed model by Saten.

Number of Parameters: The number of model parameters of a Saten representation of a linear transformation is computed as follows:

$$P = \sum_{j=1}^{d+k} r_{j-1} s_j r_j + \rho NM. \tag{6}$$

where $\mathbf{s} = (n_1, \cdots, n_k, m_1, \cdots, m_d)$, and ρ is the density of non-zero elements in $\mathbf{E}$. When the TT ranks and sparsity are sufficiently small, P becomes smaller than the number of parameters in a linear layer (NM).

Computational Complexities: Let $\mathbf{x} \in \mathbb{R}^N$ be the input of a linear layer $\mathbf{y} = \mathbf{W}^T\mathbf{x}$. This linear layer can be written as $\mathbf{y} = \hat{\mathbf{W}}_{\text{TT}}^T\mathbf{x} + \mathbf{E}^T\mathbf{x}$ in the matrix format after applying the Saten compression flow. However, since $\hat{\mathbf{W}}_{\text{TT}}$ is parameterized in a TT format, we rewrite the linear layer by using the TT factors directly, leading to

$$\mathbf{y} = \mathbf{E}^T\mathbf{x} + f(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{G}}_1, \cdots, \boldsymbol{\mathcal{G}}_{d+k}). \tag{7}$$

Here $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_k}$ is obaianed by folding $\mathbf{x}$ into an order-k tensor, f denotes the process of contracting $\boldsymbol{\mathcal{X}}$ with the TT factors as described in Appendix A. This tensor network contraction leads to the following number of MACs (multiplication and accumulation)

$$C_{\text{TT}} = \sum_{i=1}^{k} \frac{N r_{i-1} r_i}{\prod_{j=1}^{i-1} n_j} + \sum_{i=k+1}^{d} \Big(\prod_{j=k+1}^{i} m_j \Big) r_{i-1} r_i, \tag{8}$$

which is usually much smaller than that of a standard matrix-vector multiplication. The total number MACs for the inference of a Saten layer is

$$C_{\text{Saten}} = C_{\text{TT}} + (\rho N + 1)M. \tag{9}$$

When the TT ranks and ρ are sufficiently small, C_{Saten} becomes smaller than the typical linear layer MAC count, which is equal to $M(N+1)$.

4 Experimental Results

In this section, we conduct experiments to compress BERT-Base (Devlin et al., 2019) and LlaMA-3.2-1B (AI, 2024). For details regarding hyperparameter for fine-tuning and Saten compression, see Appendix D. In Appendix C, we also show additional results regarding the compression of DistilBERT (Sanh et al., 2019), with a particular focus on (1) the effectiveness of Saten on the embedding layer, (2) the balance between sparsity and low-rank parameterization. In the following, we focus on discussing the overall performance on BERT-Base and LlaMA-3.2-1B.

4.1 BERT Compression for GLUE

Table 1 compares Saten with other compression methods, including TT, singular value decomposition (SVD), and the most recent SVD with adaptive rank selection (SVD-ARS) (Gao et al., 2024), on the GLUE benchmarks (Wang et al., 2018) for BERT-Base. It can be seen that Saten achieves the **highest compression ratio** (by nearly a factor of two) while preserving the performance of the network compared to the BERT-Base model. Meanwhile, Saten produces much **better accuracy** compared to SVD and TT. Although Saten is not equipped with adaptive rank selection, it still

Table 1: The experimental results of the end-to-end compression of BERT for GLUE datasets (SVD and SVD-ARS are cited from Gao et al. (2024)). (For SST-2, MNLI, and QNLI, the values represent accuracy. For MRPC and QQP, they correspond to the F1 score. Values for CoLA and STS-B refer to Matthew's correlation and Pearson correlation, respectively).

Model	MRPC	STSB	CoLA	SST-2	MNLI	QNLI	QQP	# Params (M)
BERT-Base	91.50	89.42	58.92	92.78	84.31	91.38	87.95	109.5
Saten(2:4)	**90.19**	**87.23**	45.28	91.86	81.74	**89.82**	86.87	**59.9**
TT	84.57	84.32	10.25	89.10	79.02	87.71	86.33	64.8
SVD (Gao et al. (2024))	83.60	85.67	29.02	91.28	83.02	89.35	87.05	66.5
SVD-ARS (Gao et al. (2024))	85.57	86.30	**47.08**	**91.97**	**83.55**	89.44	**87.39**	65.1
Saten(u)	**83.38**	**83.43**	23.60	**90.25**	80.87	**87.61**	**86.91**	**50.3**
SVD (Gao et al. (2024))	81.06	79.35	9.83	89.11	81.61	86.99	86.35	52.4
SVD-ARS (Gao et al. (2024))	81.42	82.85	**27.62**	89.22	**83.07**	87.50	86.68	52.6

Table 2: The experimental results of the compression of LlaMA-3.2-1B using Saten(2:4), Saten(u), and TT.

Model	$L_{\text{lin}}(\%)$	$\rho_{\text{lin}}(\%)$	BoolQ(%)	CB(%)	WSC(%)	COPA(%)	#Params (B)	#MACs (M)
Llama-3.2-1B	-	-	66.48	87.50	63.46	65.00	1.24	973
Saten(2:4)	13	50	66.29	**91.07**	**64.42**	**65.00**	0.78	630
TT	68	0	64.83	58.93	50.00	55.00	0.93	706
Tucker	71	0	63.33	66.07	63.46	47.00	0.95	744
SVD	68	-	65.66	69.64	63.46	51.00	0.93	615
Saten(u)	16	5	**67.03**	71.43	57.69	52.00	**0.38**	**226**
TT	27	0	62.08	51.79	42.31	51.00	0.53	293
Tucker	29	0	63.94	66.07	55.76	49.00	0.55	320
SVD	27	-	65.50	67.86	61.54	50.00	0.53	270

achieves **competitive accuracy with fewer model parameters** compared to SVD-ARS, which optimizes SVD ranks.

Both Saten(2:4) and Saten(u) have 8.8 million parameters in the word embedding layer. Table 3 presents details of the encoder (linear layers) compressed by Saten, including their low-rank and sparsity budgets and MAC count relative to the base model. ρ_{lin} and L_{lin} denote the proportion of nonzero elements and low-rank parameters relative to the original total number of parameters in the linear layers, respectively. In Saten(u), linear layers have 95% sparsity and 40% low-rank parameters, leading to $1.9\times$ reduction in the network's MAC count. Meanwhile, Saten(2:4), which achieves performance comparable to the base model, reduces the MAC count by $1.7\times$.

4.2 LLaMA Compression

We further apply Saten to compress LLaMA-3.2-1B across multiple datasets from SuperGLUE benchmarks (Wang et al., 2019)

Table 2 presents the specifications and results of Saten(u) and Saten(2:4), compared with TT, Tucker, and SVD. Both Saten(2:4) and Saten(u) have 170 million low-rank parameters with 92% sparsity in their embedding layer. Saten(2:4) outperforms the base model on CB and WSC while reducing model size by about 40% and maintaining accuracy on the BoolQ dataset. Its improved accuracy stems from better generalization and reduced overfitting. Meanwhile, Saten(u) achieves over 3× and 4× reductions in model size and MAC counts, respectively.

Table 3: Specification of the Saten layers relative to the BERT-Base linear layers for GLUE benchmarks.

Model	$L_{\text{lin}}(\%)$	$\rho_{\text{lin}}(\%)$	MAC Reduction
Saten(u)	40	5	$1.9\times$
Saten(2:4)	8	50	$1.7\times$

5 Conclusion

Pre-trained model parameters may exist high-rank properties in many layers, leading to performance drops when using tensor factorization for LLM post-training compression. We have presented Saten, which integrates a sparse component into tensor networks to improve model performance in compression. Saten has been tested on BERT-Base and LlaMA-3.2-1B, showing best compression ratios and sate-of-the-art accuracy compared to SVD, TT and the most recent SVD-ARS.

6 Limitations

We demonstrated that Saten reduces the theoretical computational time complexities (i.e. MACs) compared to the uncompressed model. Current mainstream platforms for implementing large language models are not optimized for TT plus sparse operations. Nevertheless, there is ongoing interest and progress in improving sparse operations for AI and machine learning tasks (Fu et al., 2024; Hsu et al., 2023). Meanwhile, in this study, we utilized structured sparsity, which facilitates the implementation of Saten on GPUs (Zhou et al., 2021) and kept the unstructured sparsity at high levels of 95% for which sparse algebra packages typically achieve significant speedup. Our findings motivate future research on custom hardware accelerators as the actual inference speedup of Saten requires both software- and hardware-level optimization, which is the subject of our future studies. Another promising direction for future work is the application of quantization to further enhance compression efficiency and reduce inference cost (Saha et al., 2024). Finally, we evaluated the LLaMA 3.2 models, which are already distilled and thus more difficult to compress. This highlights Saten's practical utility in resource-constrained, low-redundancy settings. Although we do not include further larger-scale models, Saten's architecture-agnostic design and layer-wise compression make it scalable, and we expect the observed improvements to generalize across model sizes and architectures.

References

Meta AI. 2024. Llama 3.2-1b model card. Retrieved from https://huggingface.co/meta-llama/Llama-3.2-1B.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the NAACL-HLT*.

Derek Driggs, Stephen Becker, and Jordan Boyd-Graber. 2019. Tensor robust principal component analysis: Better recovery with atomic norm regularization. *arXiv preprint arXiv:1901.10991*.

Qiang Fu, Thomas B. Rolinger, and H. Howie Huang. 2024. Jitspmm: Just-in-time instruction generation for accelerated sparse matrix-matrix multiplication. *Proceedings of the 2024 IEEE/ACM International Symposium on Code Generation and Optimization*.

Shangqian Gao, Ting Hua, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. 2024. Adaptive rank selections for low-rank approximation of language models. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*.

Cole Hawkins and Zheng Zhang. 2019. Bayesian tensorized neural networks with automatic rank selection. *arXiv:1905.10478*.

Olivia Hsu, Maxwell Strange, Ritvik Sharma, Jaeyeon Won, Kunle Olukotun, Joel S Emer, Mark A Horowitz, and Fredrik Kjølstad. 2023. The sparse abstract machine. *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems,*.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *International Conference on Learning Representations (ICLR)*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Tamara G. Kolda and Brett W. Bader. 2009. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. *International Conference on Learning Representations (ICLR)*.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*.

Gonzalo Mateos and Georgios B. Giannakis. 2012. Robust pca as bilinear decomposition with outlier-sparsity regularization. *IEEE Transactions on Signal Processing*, 60(10):5176–5190.

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

C. Mu, B. Huang, J. Wright, and D. Goldfarb. 2013. Square deal: Lower bounds and improved relaxations for tensor recovery. *arXiv:1307.5870v2*.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing neural networks. *Advances in neural information processing systems*.

Ivan Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*.

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea Goldsmith, and Mert Pilanci. 2024. Compressing large language models using low rank and low precision decomposition. In *Advances in Neural Information Processing Systems*, volume 37, pages 88981–89018.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: Smaller, faster, cheaper, and lighter. *arXiv:1910.01108*.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems (NeurIPS)*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. *AAAI Conference on Artificial Intelligence*.

Ryan Solgi. 2024. Low-rank tensorized neural networks with tensor geometry optimization. Master's thesis, Department of Electrical and Computer Engineering, University of California Santa Barbra.

Ryan Solgi, Zichang He, William Jiahua Liang, Zheng Zhang, and Hugo A Loaiciga. 2023. Tensor shape search for efficient compression of tensorized data and neural networks. *Applied Soft Computing*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Aidan N. Gomez Llion Jones, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *Association for Computational Linguistics*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yifan Yang, Kai Zhen, Ershad Banijamal, Athanasios Mouchtaris, and Zheng Zhang. 2024a. Adazeta: Adaptive zeroth-order tensor-train adaption for memory-efficient large language models fine-tuning. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yifan Yang, Jiajun Zhou, Ngai Wong, and Zheng Zhang. 2024b. Loretta: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models. *Conf. Northern American Association of Computational Linguistics (NAACL)*.

Yinchong Yang, Denis Krompass, and Volker Tresp. 2017. Tensor train recurrent neural networks for video classification. *Proceedings of the British Machine Vision Conference (BMVC)*.

Zi Yang, Ziyue Liu, Samridhi Choudhary, Xinfeng Xie, Cao Gao, Siegfried Kunzmann, and Zheng Zhang. 2024c. Comera: Computing- and memory-efficient training via rank-adaptive tensor optimization. *arXiv:2405.14377*.

Z. Zhong, F. Wei, Z. Lin, and C. Zhang. 2019. Adatucker: Compressing deep neural networks via adaptive dimension adjustment tucker decomposition. *arXiv:1906.07671*.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. 2021. Learning n:m fine-grained structured sparse neural networks from scratch. *arXiv:2102.04010*.

A Tensor Train Network

Tensor: A tensor is a generalization of scalars, vectors, and matrices to higher dimensions. A tensor of order d over the real numbers is an array of components indexed by d indices as follows:

$$\begin{aligned} &\boldsymbol{\mathcal{A}} = \left(a_{i_1,i_2,\ldots,i_d}\right) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}, \\ &\forall i_t = 1, 2, \ldots, I_t, \quad t = 1, 2, \ldots, d. \end{aligned} \tag{10}$$

Tensor Contraction: A tensor contraction generalizes matrix multiplication to tensors. Without loss of generality, the contraction of two tensors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_v \times T_1 \times \cdots \times T_h}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{T_1 \times \cdots \times T_h \times J_1 \times \cdots \times J_u}$ over the shared dimensions $T_1, \cdots, T_h$ results in a new tensor $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_1 \times \cdots \times I_v \times J_1 \times \cdots \times J_u}$ with elements given by:

Algorithm 2 TT Contraction Network

Require: $\mathcal{X}, \mathcal{G}_1, \cdots, \mathcal{G}_{k+d}$
  $\mathcal{Y} = \mathcal{X} \times_1^2 \mathcal{G}_1$
  for $t = 2$ **to** $t = k$ **do**
    $\mathcal{Y} = \mathcal{Y} \times_{1,4}^{2,1} \mathcal{G}_t$
  end for
  for $t = 1$ **to** $t = d$ **do**
    $\mathcal{Y} = \mathcal{Y} \times_{t+1}^{1} \mathcal{G}_{k+t}$
  end for
  Reshape $\mathcal{Y} \in \mathbb{R}^{1 \times m_1 \times \cdots \times m_d \times 1}$ to $\mathbf{y} \in \mathbb{R}^M$
  Return $\mathbf{y}$

$$
\begin{aligned}
&c_{i_1,\cdots,i_v,j_1,\cdots,j_u} \\
&= \sum_{t_1,\cdots,t_h} a_{i_1,\cdots,i_v,t_1,\cdots,t_h} b_{t_1,\cdots,t_h,j_1,\cdots,j_u}, \\
&\forall \quad i_1, \cdots, i_v, \quad j_1, \cdots, j_u.
\end{aligned} \tag{11}
$$

For arbitrary tensors $\mathcal{A}$ and $\mathcal{B}$ we define a contraction pattern as a set of index pairs $\{(l_1, l_1'), \cdots, (l_h, l_h')\}$ where the indices l_i in $\mathcal{A}$ correspond to indices l_i' in $\mathcal{B}$ for all i. The contraction operation can then be written as follows:

$$
\mathcal{C} = \mathcal{A} \times_{l_1,\cdots,l_h}^{l_1',\cdots,l_h'} \mathcal{B}. \tag{12}
$$

Frobenius norm: The Frobenius norm of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_t}$, denoted by $\|\mathcal{A}\|_F$, is defined as follows:

$$
\|\mathcal{A}\|_F = \sqrt{\sum_{i_1,i_2,\cdots,i_t} (a_{i_1,i_2,\cdots,i_t})^2}. \tag{13}
$$

Tensor-Train (TT) Decomposition: For a given tensor $\mathcal{W}$ and an error bound $\epsilon = \frac{\|\mathcal{W} - \hat{\mathcal{W}}\|_F}{\|\mathcal{W}\|_F}$, the TT factors $\{\mathcal{G}_j\}_{j=1}^t$, are computed by conducting a hierarchical σ-truncated singular value decomposition (SVD) for unfolded tensor $\mathcal{W}$ along its different dimensions, where $\sigma = \frac{\epsilon}{\sqrt{t-1}} \|\mathcal{W}\|_F$ (Oseledets, 2011).

B Tensor Geometry Optimization

To construct a low-rank tensorized layer, a weight $\mathbf{W}$ is folded to a $(k+d)$ dimensional tensor $\mathcal{W}$. The memory requirement of a tensorized layer depends on the tensor geometry (shape) selected in the folding process (Mu et al., 2013; Zhong et al., 2019; Solgi et al., 2023). It is necessary to find an optimal tensor geometry that minimizes the memory requirement of a low-rank tensorized layer in the TT format as follows:

$$
\begin{aligned}
&\min_{\mathbf{s}} \sum_{j=1}^{k+d} r_{j-1}^{(\mathbf{s})} \times s_j \times r_j^{(\mathbf{s})}, \\
&\text{subject to } \prod_{j=1}^{k} s_j = N, \prod_{j=k+1}^{k+d} s_j = M, \\
&\qquad s_j \geq 2, s_j \in \mathbb{Z}, j = 1, \cdots, k+d,
\end{aligned} \tag{14}
$$

where $\mathbf{s} = (n_1, \cdots, n_k, m_1, \cdots, m_d)$ and $r_j^{(\mathbf{s})}$ is the TT rank associated with the shape $\mathbf{s}$. When the ranks are unknown, the above problem is hard to solve. It has been shown that $\min_{\mathbf{s}} \sum_{j=1}^{k+d} s_j$ serves as an upper bound surrogate for the cost function of Eq. (14) for all ranks $r_j^{(\mathbf{s})} \in \mathbb{R}$. This upper bound surrogate leads to two independent integer programming problems of minimization of sum under product constraint whose solutions are the feasible and most balanced shapes for the input $(n_1, \cdots, n_k)$ and output dimensions $(m_1, \cdots, m_d)$, separately (Solgi, 2024).

Table 4: Experimental results demonstrate the compression of the word embedding layer of DistilBERT using Saten(e) and TT(e) (γ_{emb} denotes the compression factor of the embedding layer, defined as the ratio of the compressed size to the uncompressed size).

Model	γ_{emb}	SST2 (Acc%)	MRPC (F1%)
Base	1.00	91.28	90.22
Saten(e)	0.46	91.17	90.03
TT(e)	0.43	88.18	81.97
TT(e)	0.58	90.37	84.97

C DistilBERT Compression Results

C.1 Word Embedding Compression

When the embedding layer is compressed using TT decomposition, the approximation error associated with high-frequency tokens in a sequence accumulates, significantly perturbing the input to the encoder network—even if the individual errors are small. Consequently, incorporating a sparse component for highly frequent tokens helps mitigate error accumulation and the resulting perturbations.

Table 4 lists the scores and compression factor of the embedding layer denoted by γ_{emb} for different datasets and different networks. In Table 4, base network refers to the uncompressed DistilBERT network, Saten(e) refers to a network in

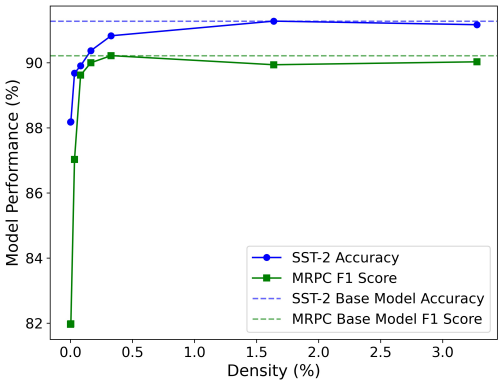

Figure 2: Accuracy versus density of saten(e) for SST2 and MRPC datasets.

which only the word embedding layer is replaced by the Saten layer, while the rest of the network (encoder) remains uncompressed. For Saten(e) we set row sparsity to keep only 1,000 tokens resulting in about 96.7% sparsity ($\rho = 0.033$). TT(e) refers to the network whose word embedding layer is compressed with TT without sparsity while the rest of the network remains uncompressed. For TT(e) the low-rank budget has been changed to study the effect of low-rank parameters. Both Saten(e) and TT(e) have been fine-tuned with the same training arguments.

As shown in Table 4, Saten(e) compresses the embedding layer by more than two times without a significant reduction in the evaluation metrics. However, TT(e) shows a significant drop in evaluation metrics while having almost the same compression efficiency. TT(e) with less memory reduction compared to that of Saten(e), has worse evaluation metrics, emphasizing that sparsity played a crucial role. Therefore, simply increasing the rank of the TT does not provide a matching evaluation metric to that of Saten(e).

Fig. 2 illustrates the accuracy versus the density of the sparse component of the saten(e) for SST2. It is observed that even a high sparsity significantly enhances the network's accuracy. For instance, from TT only (without sparsity component) to Saten(e) with 99.84% sparsity ($\rho = 0.0016$, corresponding to only the 50 most common tokens), the accuracy of the network increases from 88.18 to 90.37.

C.2 Low-Rank Versus Sparsity

Table 5 lists the results of end-to-end compression of DistilBERT. In Table 5, Saten(u) and Saten(2:4) refer to models in which the encoder network is compressed using unstructured and 2:4 sparsity patterns, respectively, while the word embedding layer is compressed using row sparsity, as in Saten(e). TT refers to a network where the word embedding layer is compressed using Saten, while the encoder is compressed using TT. In Table 5, the scores for SST-2 and MRPC refer to Accuracy and F1-Score, respectively.

For Saten(u) the low-rank budget is kept the same, but the sparse budget has been changed to show the effect of sparsity on Saten versus the model accuracy. On the other hand, for Saten(2:4) the sparsity budget is constant due to the pattern of sparsity, but the low-rank budget has been changed to investigate the effect of low-rank component on the network's performance score. Note that for Saten(2:4) with $L_{lin} = 0$, although the low-rank budget is set to zero, the sparse pattern has been derived based on the low-rank decomposition error. For all Saten networks, the total embedding parameters are 11.3 million with near 97% sparsity for the word embedding layer.

For SST2 dataset we can observe that both Saten(u) and Saten(2:4) were able to compress the network by 30% and 40%, resepctively with no drop in accuracy. For both MRPC and SST2 we were able to compress the DistilBERT by more than two times with at most 2% drop in the network's score. Considering the fact that DistilBERT network is already a compressed model, this level of compression with no significant drop of accuracy is notable and shows the capacity of the Saten framework for compressing language models.

Comparing the TT and Saten models for both SST2 and MRPC, TT leads to a more significant drop in network's score than Saten even with almost the same or worse compression ratios. Comparing TT and Saten(u) that have the same low-rank budget, we can observe that by adding a small sparse budget the network achieves significantly higher scores for both MRPC and SST2 dataset. On the other hand, by increasing the low-rank budget for TT, we do not observe the same amount of improvement.

Studying the effect of sparsity and the low-rank component, it is evident that both play a key role in preserving the performance of the models. Comparing Saten(u) and Saten(2:4) with TT, we can observe that the Saten models not only improve the compression efficiency but also achieve higher scores.

Table 5: The experimental results of full model compression of DistilBERT on the SST-2 and MRPC datasets using Saten(u), Saten(2:4), and TT.

Model	L_{lin} (%)	ρ_{lin} (%)	Score (%)	#Params (M)	#MACs (M)
DistilBERT-SST2	-	-	91.28	67.5	43.1
Saten(u)	31	50	91.51	47.6	38.6
Saten(u)	31	20	90.94	34.4	25.4
Saten(u)	31	10	89.11	28.7	19.7
Saten(2:4)	20	50	91.63	41.6	32.2
Saten(2:4)	8	50	90.83	36.1	25.9
Saten(2:4)	0	50	89.44	32.8	21.5
TT	76	0	90.02	43.8	35.5
TT	31	0	82.22	24.1	15.9
DistilBERT-MRPC	-	-	90.22	67.5	43.1
Saten(u)	31	25	88.48	35.4	26.4
Saten(u)	31	20	88.04	33.2	24.2
Saten(u)	31	10	85.11	29.9	20.9
Saten(2:4)	20	50	88.40	41.6	32.2
Saten(2:4)	8	50	87.67	36.1	25.9
Saten(2:4)	0.0	50	82.22	32.8	21.5
TT	76	0	83.75	43.0	35.5
TT	31	0	82.21	43.0	35.5

D Implementation and Fine-Tuning Details

For our experiments we downloaded the pre-trained models and data from Huggingface (Wolf et al., 2020). The BERT models are under Apache 2.0 license. In this we also applied LLaMA 3.2-1B model, developed by Meta Platforms, Inc., available under the LLaMA 3.2 Community License. For all applied datasets, we used all training data for fine-tuning and we evaluated all models using all the evaluation datasets. All reported evaluation metrics are one-time run. For fine-tuning of the base models we applied the default setting of the downloaded models. For both BERT-Base and LlaMA models we used learning rate 2e-5 and 5 epochs. We used batch size 8 and 2 for BERT-Base and LlaMA models, respectively. We fine tuned the models on single NVIDIA RTX 6000 ADA generation GPU using the default AdamW optimizer with weight decay of 0.01. After compressing the models we fine-tuned for another 5 epochs with the same settings but we used learning rate 5e-6 for compressed BERT-Base models and learning rate 2e-5 (2e-6 for BoolQ) for compressed LlaMA-3.2-1B models.

For compressing the models using Saten we need to set TT error bound ϵ and sparsity ratio for each layer in the network. For BERT-Base experiments for Saten(u) we set $\epsilon = 0.75$ for all linear layers and for Saten(2:4) we used $\epsilon = 1$ for all linear layers. For LlaMA experiments, Saten(u) uses an error bound $\epsilon = 1$ for all linear layers but for query and key layers we used $\epsilon = 0.4$. For Saten(2:4) we used $\epsilon = 1$ for all layers but for query and key layers we used $\epsilon = 0.65$. For embedding layers for both BERT-Base and LlaMA we applied $\epsilon = 0.5$ and used 1,000 ($\rho = 0.03$) and 10,000 ($\rho = 0.08$) tokens for sparsity, respectively. For the sparsity of Saten(u) in all experiments of BERT-Base and LlaMA-3.2-1B, we set the sparsity to be 95% for all linear layers, but 90% for query and key layers, which on average is about 95% sparsity given the smaller size of query and key layers.

For TT compression of BERT-Base we used $\epsilon = 0.65$ for all compressed linear layers. We did not compress the query and key linear layers as it degraded its performance. In LlaMA-3.2-1B experiments we run TT twice with different budgets to compare with both Saten(u) and Saten(2:4). For comparison with Saten(u) we used $\epsilon = 0.4$ for query and key layers and $\epsilon = 0.9$ for the rest of linear layers. This setting was set to be similar to Saten(u) but gives a higher number of parameters to TT. For comparison with Saten(m:n) we set $\epsilon = 0.6$ and $\epsilon = 0.55$ for the query/key linear layers and the rest of the linear layers, respectively. For Tucker decomposition, we applied similar setting as the TT but to match the compression ratio with Saten(u) we set $\epsilon = 1.0$ for all layers.

E Llama-3.2-3B

Additionally, we compressed Llama-3.2-3B for Wikitext-2 dataset (Merity et al., 2016). To ensure the generalization of our approach to

Table 6: Results of compressing LlaMA-3.2-3B using Saten versus SVD and TT for WikiText-2 dataset (CR denotes compression ratio).

Model	Zero-Shot (PPL)	LoRA (PPL)	CR
Base	7.60	7.18	1.0
Saten(2:4)	780.66	13.37	0.6
TT	312755.56	1813.55	0.6
SVD	120484.91	776.16	0.6

larger models, we directly compressed the pre-trained model without initial fine-tuning, followed by parameter-efficient fine-tuning using low-rank adapters (LoRA), which is standard practice for large-scale models (Hu et al., 2021). We also reported perplexity as a proxy for the model evaluation across different tasks. The results presented in Table 6 show that Saten significantly improves perplexity (PPL).