HEAT: Hardware-Efficient Automatic Tensor Decomposition for Transformer Compression

Jiaqi Gu[2]*, **Ben Keller**[1], **Jean Kossaifi**[1],
Anima Anandkumar[1,3], **Brucek Khailany**[1], **David Z. Pan**[2]
[1]NVIDIA, [2]The University of Texas at Austin, [3]Caltech
jqgu@utexas.edu

Abstract

Self-attention and feedforward layers in large-scale Transformer models are overparameterized, limiting inference speed and energy efficiency. Tensor decomposition is a promising technique to reduce parameter redundancy by expressing weight matrices in an efficiently factorized form. Prior efforts used manual or heuristic decomposition settings without hardware-aware customization, resulting in poor hardware efficiencies and large performance degradation.

In this work, we propose a hardware-aware tensor decomposition framework, dubbed HEAT, that enables efficient exploration of the exponential space of tensor decompositions and automates the choice of tensorization shape and decomposition rank with hardware-aware co-optimization. We jointly investigate tensor contraction path optimizations and a fused Einsum mapping strategy to bridge the gap between theoretical benefits and real hardware efficiency improvement. Our two-stage knowledge distillation flow resolves the trainability bottleneck and thus significantly boosts the final accuracy of factorized Transformers. We find that our hardware-aware factorized BERT variants reduce the energy-delay product by $5.7\times$ with less than 1.1% accuracy loss and achieve a better efficiency-accuracy Pareto frontier than hand-tuned and heuristic baselines.

1 Introduction

Transformers have demonstrated record-breaking performance on natural language processing (NLP) [15, 7, 2]. However, the overparametrized linear layers in multi-head self-attention and feedforward networks (FFNs) limit the efficient deployment of Transformers. Compressing large-scale Transformers is an essential problem in practical NLP tasks.

Tensor factorization can be applied to matrices by first *tensorizing* [1, 12] them (reshaping them into a higher-order tensor) and then factorizing that tensor. Prior work has successfully applied this method to reduce the number of parameters and computations in NNs by manually selecting the tensorization shapes and ranks [12, 19, 9, 4, 10, 6, 24, 21, 8].

However, three critical issues remain unresolved. First, **improvements in compression metrics do not necessarily translate to better hardware efficiency**, a distinction ignored by most prior work. The number of multiply-accumulate operations (MACs) and parameters are imprecise indicators of energy efficiency and execution speed on real hardware, so high compression ratios may not translate to real hardware efficiency benefits. Moreover, we observe **heterogeneous low-rank characteristics** in different weight matrices, but previous methods ignore this heterogeneity and manually assign a global setting to all matrices based on heuristics [6, 9, 4, 11], failing to explore the huge design space.

*Work done during an internship at NVIDIA

36th Conference on Neural Information Processing Systems (NeurIPS 2022), Workshop on Machine Learning for Systems

An additional challenge of factorized Transformers is the non-trivial performance drop after tensor decomposition. Direct re-training cannot recover the accuracy of factorized Transformers due to the **optimization difficulty** of cascaded tensor contractions, which hinders their practical deployment.

To solve these challenges, we propose HEAT, a hardware-efficient tensor decomposition framework that features automated tensor decomposition with hardware-aware optimization. HEAT can efficiently explore the huge design space of tensorization while significantly improving hardware efficiency based on the following approaches: (1) compared to prior hardware-unaware tensor decomposition work, HEAT incorporates hardware cost feedback in the tensorization optimization flow to find expressive and hardware-efficient tensorization settings; (2) instead of manually selecting a global rank setting via trial-and-error, HEAT leverages the heterogeneous low-rank property of different tensors and adopts a novel Rank SuperNet-based method to automatically search for efficient per-tensor rank settings in the exponentially large space with one-shot re-training cost; and (3) HEAT resolves the trainability challenge of factorized Transformers by introducing a two-stage knowledge distillation flow to significantly boost the task performance.

Our searched factorized BERT models outperform the original BERT with an estimated 5.7× lower energy-delay product (EDP) and surpass hand-tuned and heuristic baselines with 25%-30% lower EDP and 1-3% accuracy improvement on the SQuAD-v1.1 and SST-2 datasets.

2 HEAT Automatic Tensor Decomposition Framework

2.1 Understanding Hardware-Efficient Tensor Decomposition

In tensor decomposition, a matrix $\mathbf{W} \in \mathbb{R}^{M\times N}$ is tensorized into a high-order tensor $\mathcal{X}$, which is then approximated by the product or summation of a series of smaller core tensors. Representative decompositions include CP [16], Tucker [22], and tensor-train matrix (TTM) [13]. For example, the order-d TTM decomposition is formulated by

$$\mathcal{X}((i_1,j_1),\cdots,(i_d,j_d)) = \mathcal{G}^{(1)}((i_1,j_1),:)\cdots\mathcal{G}^{(d)}(:,(i_d,j_d)), \tag{1}$$

where each $\mathcal{G}^{(i)} \in \mathbb{R}^{r_{i-1}\times m_i\times n_i\times r_i}$ is called a core tensor, the size of tensorized $\mathcal{X}$ is called tensorization shape, i.e., $\boldsymbol{s} = (m_1,\cdots,n_1,\cdots)$, where $M = \prod_i m_i$ and $N = \prod_i n_i$. The variable dimensions of cores are called decomposition ranks, i.e, $\boldsymbol{r} = (r_0, r_1, \cdots, r_{d-1}, r_d)$. The compression ratio is $c = \sum_i r_{i-1}m_i n_i r_i / MN$.

To find a hardware-efficient decomposition, the goal is to determine the tensorization shape $\boldsymbol{s}$ and rank $\boldsymbol{r}$ for each matrix $\mathbf{W}$ that minimizes energy-delay product while maintaining high accuracy. However, the design space is too large and discrete to explore with brute-force methods. For example, for a 12-layer BERT model, there are 10^{632} different possible tensorization settings.

We formulate the search as a *three-level hierarchical optimization* as follows,

$$\begin{aligned}
&\texttt{Level 3}: \text{ Train factorized model: } \Theta^*(\boldsymbol{s}^*,\boldsymbol{r}^*) = \arg\min_{\Theta} \mathcal{L}(\Theta(\boldsymbol{s}^*,\boldsymbol{r}^*),\mathcal{D}_{trn}),\\
&\texttt{Level 2}: \text{ Search rank: } \boldsymbol{r}^* = \arg\min_{\boldsymbol{r}} \ \big(1-\texttt{Acc}(\Theta^*(\boldsymbol{s}^*,\boldsymbol{r}))\big)\texttt{Cost}^*(\boldsymbol{s}^*,\boldsymbol{r}|\alpha)^\gamma,\\
&\qquad\qquad \Theta^*(\boldsymbol{s}^*,\boldsymbol{r}) = \arg\min_{\Theta} \mathcal{L}(\Theta(\boldsymbol{s}^*,\boldsymbol{r}),\mathcal{D}_{trn}),\\
&\texttt{Level 1}: \text{ Search shape: } \boldsymbol{s}^* = \texttt{Pareto}_{\boldsymbol{s}}\big(\texttt{Cost}^*(\boldsymbol{s},\boldsymbol{r}|\alpha),\epsilon,c\big)\\
&\epsilon = \|\mathbf{W}-\mathbf{W}'(\boldsymbol{s},\boldsymbol{r})\|_{\mathcal{F}}/\|\mathbf{W}\|_{\mathcal{F}},\ \texttt{Cost}^*(\boldsymbol{s},\boldsymbol{r}|\alpha) = \min_m \min_p \texttt{Cost}(\boldsymbol{s},\boldsymbol{r},m,p|\alpha).
\end{aligned} \tag{2}$$

In Level 1, given an accelerator architecture α, we find a Pareto optimal tensorization shape $\boldsymbol{s}^*$ that minimizes decomposition error ϵ, compression ratio c, and the minimum hardware cost Cost* obtained by optimizing tensor contraction path (p) and hardware mapping (m). We use a standard *energy-delay product (EDP)* as the hardware cost to reflect both energy consumption and runtime cost. Then in Level 2, we search for optimal per-tensor rank settings $\boldsymbol{r}^*$ while minimizing hardware cost and maximizing the task-specific performance. In Level 3, we train the factorized Transformer model with the optimal $(\boldsymbol{s}^*, \boldsymbol{r}^*)$ settings to find its optimal parameters $\Theta^*(\boldsymbol{s}^*,\boldsymbol{r}^*)$.

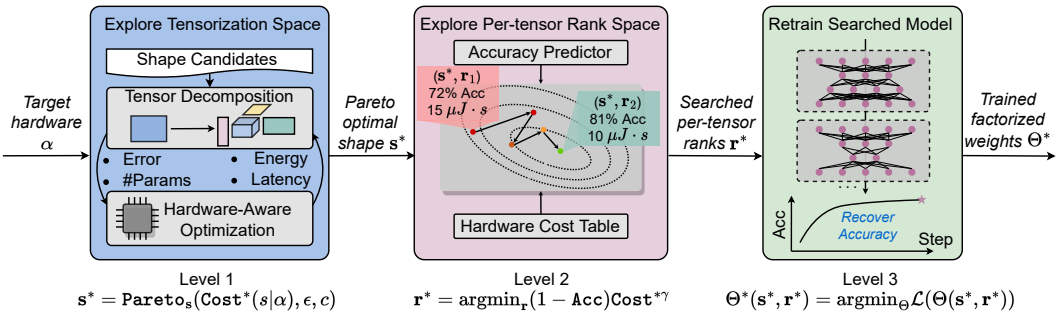

Figure 1: Overview of our hardware-efficient automatic tensor decomposition framework HEAT.

2.2 The Proposed HEAT Framework

To solve this three-level optimization problem efficiently, we propose a three-stage framework HEAT, which is summarized in Fig. 1. In the first stage, we simulate each shape candidate on a given inference accelerator architecture α and build a *hardware cost table* for all shapes $\mathcal{T}$, from which we select one Pareto optimal shape $\boldsymbol{s}^*$ with low decomposition error ϵ, low compression ratio c, and low hardware cost. All matrices with the same size share this tensorization shape. We import the optimal shape $\boldsymbol{s}^*$ to the second stage, a one-shot rank search flow that can efficiently explore per-tensor rank settings with minimum model re-training cost. With the searched optimal shape and rank $(\boldsymbol{s}^*, \boldsymbol{r}^*)$, we enter the last step, a knowledge distillation-based re-training flow to recover the accuracy.

2.2.1 Level 1: Pareto Optimal Tensorization Shape Search

To determine the real hardware cost of a given tensorization, we must find a near-optimal tensor contraction path p and map the operation to the target architecture α with an optimal mapping m.

Tensor Contraction Path Optimization. A factorized linear layer requires a series of tensor contractions, which can be described by a symbolic Einsum equation as shown in Fig. 2. The order in which these tensors are contracted, or the *contraction path*, is critical to hardware efficiency. Simply following the left-to-right association order leads to considerable computation and intermediate storage overhead due to the many outer product operations. In contrast, a MAC-optimal path reduces hardware cost by orders of magnitude [20]. Note that we can pre-compute certain einsum nodes that have static inputs to eliminate redundant memory and computation cost. Hence, we only need to implement the pre-computed MAC-optimal path on the hardware accelerator.

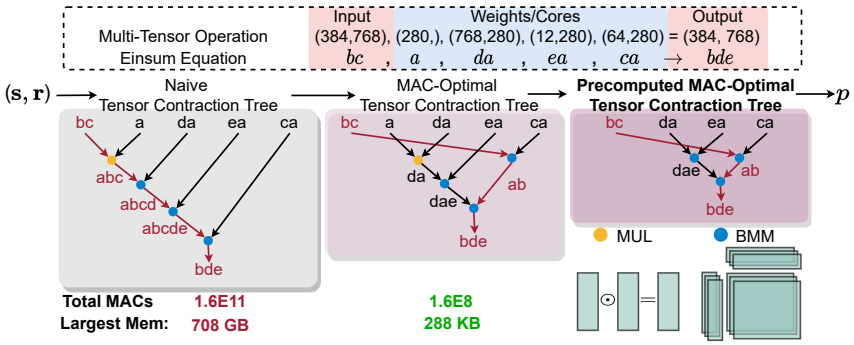

Figure 2: Our pre-computed MAC-optimal contraction path can significantly reduce hardware cost.

Mapping from Fused Einsum to Hardware. To efficiently implement factorized tensor operations, we customize our accelerator Simba-L [18] to perform a 1024-element matrix-vector multiplication each cycle to achieve high utilization on factorized Einsum workloads [18]. Based on this architecture, we implement a memory-efficient fused Einsum, minimizing redundant DRAM accesses by storing intermediate results in on-chip SRAM whenever possible to obtain the minimum hardware cost $\texttt{Cost}^*(\boldsymbol{s}, \boldsymbol{r}|\alpha)$. Then we use Timeloop [14] to search for an efficient mapping m while minimizing energy-delay product, map the fused Einsum to the accelerator, and evaluate the energy and runtime.

Search for the Pareto Optimal Shape: s^*. We perform integer factorization on the matrix height/width and select the top candidates with maximum entropy, since we prefer uniform shapes across axes. Then we decompose the matrix to get the approximation error and evaluate their costs to form a cost table $\mathcal{T}$. We automatically detect the points on the Pareto-optimal surface and heuristically select the best shape s^* with the lowest decomposition error. We repeat this process for all different sizes of matrices in the model, completing the `Level 1` optimization.

2.2.2 Level 2: Weight-Sharing Per-Tensor Rank Optimization

One-Shot Rank Search via Weight-Sharing SuperNet. The challenges in the `Level 2` optimization are twofold: (1) the exponentially large per-tensor rank search space and (2) the prohibitive cost of accuracy evaluation on a shape-rank pair. These barriers make it impossible to select the best rank by enumeration. Inspired by the high efficiency in weight-sharing neural architecture search (NAS) [23, 3], we build a rank SuperNet where each SubNet corresponds to a per-tensor rank setting. As shown in Fig. 3, at each iteration, we randomly sample valid ranks for each tensor, and different SubNets share the same set of parameters. Hence, we can efficiently explore a large space of different rank settings.

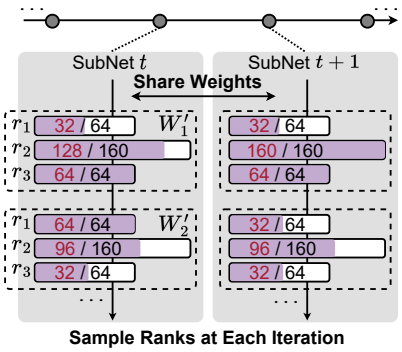

Figure 3: Rank SuperNet training flow.

Per-Tensor Rank Selection via Evolutionary Search. We randomly sample 2,560 SubNets from the Rank SuperNet and train a random forest $\mathcal{P} : (\boldsymbol{s}, \boldsymbol{r}) \rightarrow \texttt{Acc}$ to predict the validation accuracy based on the factorization settings, which is a fast proxy to reduce validation cost. In addition to accuracy maximization, network hardware cost must also be considered. The total energy E_{tot} and runtime T_{tot} of the model is simply the sum of all layer costs. We use evolutionary search to find the optimized factorization settings: $\min_{\boldsymbol{r}} \ (1 - \texttt{Acc})(E_{tot} \cdot T_{tot})^{\gamma}$, where γ is empirically set to 0.25.

2.2.3 Level 3: Trainability Boost with Two-stage Distillation

Optimization of factorized tensors is difficult, making trainability a bottleneck for factorized Transformers. To solve this issue, we propose a two-stage distillation flow. First, we perform optimal layer-wise projection to find the tensor decomposition that minimizes matrix approximation error, $\min_{\mathbf{W}_i'} \sum_i \|\mathbf{W}_i - \mathbf{W}_i'\|_{\mathcal{F}}$. Then we distill the layer-wise knowledge from the teacher T to the factorized student S both on the attention maps A and hidden states h on each Transformer block.

$$\begin{aligned} \mathcal{L}_{attn} + \mathcal{L}_{hidden} &= \sum_i \mathcal{L}^i_{attn} + \mathcal{L}^i_{hidden} \\ &= \sum_i \texttt{CosEmbed}(A^S_i, A^T_i) + \texttt{CosEmbed}(h^S_i, h^T_i) \end{aligned} \tag{3}$$

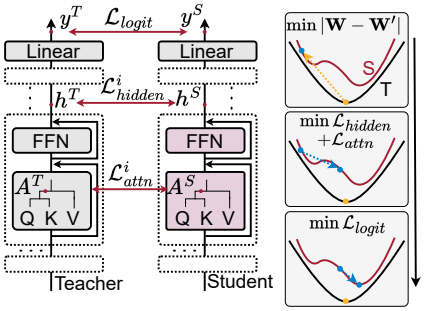

Figure 4: Layer-wise and logit distillation for factorized Transformers.

After the layer-wise alignment, we only apply last-layer logit distillation to provide more optimization freedom for the student: $\mathcal{L}_{logit} = \frac{1}{2}\mathcal{L}_{KL}(y^S/\tau, y^T/\tau) + \frac{1}{2}\mathcal{L}_{CE}(y^S, y^T)$.

3 Results

Results on BERT-base SQuAD-v1.1. We compare our searched factorization settings with (1) the original fine-tuned BERT, (2) `SR-Manual`: manually-selected shape and rank settings, and (3) `S(Ours)-R(TensorLy)`: searched optimal tensorization shapes and heuristic ranks by TensorLy [11] based on the target compression ratio. In Table 1, `HEAT` achieves the best performance-efficiency Pareto front, surpassing manual and heuristic tensor decomposition baselines.

With TTM decomposition, `HEAT` improves the F1 score by +0.65% with 8% fewer parameters and 8.8% lower hardware cost on average. The compact `HEAT-a1` benefits the most from our heterogeneous per-tensor rank settings and outperforms manual and heuristic decomposition with **+1.6%** higher F1

Table 1: Compare our HEAT-series with baseline decomposition methods on BERT-base SQuAD-v1.1 in terms of #Params, F1 score, and energy-delay product (EDP) across three decomposition methods.

Model	TTM			Tucker			CP		
	#Params (M)↓	F1 (%)↑	EDP ($\mu J \cdot s$)↓	#Params (M)↓	F1 (%)↑	EDP ($\mu J \cdot s$)↓	#Params (M)↓	F1 (%)↑	EDP ($\mu J \cdot s$)↓
BERT-base	109.50	88.16	34.31	109.50	88.16	34.31	109.50	88.16	34.31
SR-Manual-1	35.04	83.60	24.27	38.10	81.90	5.63	43.13	68.95	4.48
SR-Manual-2	38.71	84.74	25.86	39.71	82.31	6.85	54.27	79.22	7.57
S(Ours)-R(TensorLy)-1	36.15	83.89	24.42	35.81	80.43	4.06	46.38	85.81	5.45
S(Ours)-R(TensorLy)-2	39.78	85.49	27.18	39.48	82.00	4.93	55.52	87.21	9.37
HEAT-**a1**	**41.60**	**86.01**	**25.06**	**42.14**	**83.41**	**4.91**	**48.88**	**87.11**	**5.99**
SR-Manual-3	42.37	86.05	28.87	42.64	83.45	7.87	60.88	81.96	10.70
S(Ours)-R(TensorLy)-3	43.33	85.84	29.05	42.63	82.96	5.66	60.31	87.74	9.62
HEAT-**a2**	**46.42**	**86.71**	**28.93**	**50.96**	**85.51**	**7.45**	**59.08**	**87.79**	**9.01**
SR-Manual-4	50.90	86.99	32.57	52.94	85.05	12.21	69.12	84.31	12.40
SR-Manual-5	60.68	87.29	39.25	59.17	86.50	17.00	83.98	86.59	18.50
S(Ours)-R(TensorLy)-4	51.76	86.64	33.55	51.87	85.19	9.27	68.79	87.62	14.40
S(Ours)-R(TensorLy)-5	61.45	87.74	49.27	62.26	86.76	11.90	82.43	88.14	18.00
HEAT-**a3**	**54.68**	**87.36**	**32.72**	**63.62**	**86.35**	**10.40**	**74.15**	**87.91**	**14.50**
Avg. Improv.	**-8.15%**	**+0.65**	**-8.80%**	**+0.09%**	**+1.03**	**-30.28%**	**-9.07%**	**+3.45**	**-25.30%**

scores. With Tucker decomposition, HEAT-series boosts the F1 score by **+1.03** with **+30%** higher efficiency than handcrafted settings that typically reshape the matrix to a high-order tensor (e.g., order 6 or 8) [24, 21]. Though the accuracy per parameter of Tucker is slightly lower than TTM, its accuracy-to-EDP ratio is 3-5× higher than TTM. CP decomposition is usually disfavored due to unstable optimization [5], but our search framework finds an efficient CP tensorization shape and is able to recover accuracy through re-training. HEAT-series overall can boost the F1 score by **+3.45%** with **25.3%** less hardware cost. Compared to the original BERT, our searched HEAT-a1 can maintain accuracy (1.1% drop) with **5.7**× higher hardware efficiency.

Results on BERT-base SST-2. We re-train HEAT-variants on SST-2 to evaluate the generalization of the decomposition settings searched on SQuAD. In Table 2, we observe that when adapted to a new downstream task with a smaller sequence length (128), our searched Tucker and CP factorization can still largely maintain the F1 score with **4-10**× higher hardware efficiency.

Table 2: Evaluation of HEAT on SST-2 with decomposition settings searched on SQuAD-v1.1.

Model	TTM		Tucker		CP	
	F1 (%)↑	EDP↓	F1(%)	EDP↓	F1 (%)↑	EDP↓
BERT-base	91.74	5.21	91.74	5.21	91.74	5.21
HEAT-a1	90.02	4.24	87.27	0.45	91.40	0.50
HEAT-a2	90.90	5.65	88.42	0.69	91.51	0.79
HEAT-a3	91.20	7.34	89.91	1.02	91.17	1.34

Results on DistilBERT SQuADv-1.1 and SST-2. Our tensor decomposition method can also be applied to compact Transformers as an orthogonal compression technique. Based on DistilBERT [17], a 6-layer compact version of BERT-base, we searched three HEAT-variants in Table 3. Our HEAT-series can achieve comparable F1 scores with **5.7**× higher efficiency on SQuAD-v1.1. On SST-2, HEAT-series can maintain the accuracy while saving **3-10**× hardware cost.

Table 3: Compare three HEAT-variants with DistilBERT-base on SQuAD-v1.1 and SST-2 datasets.

	SQuADv1.1						SST-2					
	TTM		Tucker		CP		TTM		Tucker		CP	
	F1↑	EDP↓	F1↑	EDP↓	F1↑	EDP↓	Acc↑	EDP↓	Acc↑	EDP↓	Acc↑	EDP↓
DistilBERT	86.90	8.58	86.90	8.58	86.90	8.58	91.97	1.30	91.97	1.30	91.97	1.30
HEAT-a1	85.26	6.71	83.66	1.72	87.09	1.50	91.40	1.23	90.30	0.17	91.28	0.13
HEAT-a2	86.42	7.76	85.21	2.56	87.89	2.26	90.60	1.64	91.30	0.23	91.06	0.20
HEAT-a3	86.92	8.74	86.15	3.21	88.18	3.63	90.60	2.10	90.60	0.29	91.63	0.34

4 Conclusion

In this work, we explore the large design space of hardware-efficient tensor decomposition using HEAT, an automatic decomposition framework for Transformer model compression. We consider hardware cost in the optimization loop and efficiently find expressive and hardware-efficient tensorization shapes. Our SuperNet-based one-shot rank search flow can efficiently generate optimized per-tensor decomposition rank settings. We employ a two-stage distillation flow to solve the trainability bottleneck of factorized Transformers and significantly boost their task performance. Experiments show that HEAT reduces up to 5.7× energy-delay product on our customized accelerator with less than 1.1% accuracy drop. Compared to manual and heuristic tensor decomposition methods, our searched HEAT-variants show 1-3% higher accuracy with ∼30% less hardware cost on average.

References

[1] Animashree Anandkumar et al. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 2014.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. In *Proc. NeurIPS*, volume 33, pages 1877–1901, 2020.

[3] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *Proc. ICLR*, 2020.

[4] Patrick H. Chen, Si Si, Yang Li, et al. GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking. In *Proc. NeurIPS*, 2018.

[5] Vin de Silva and Lek-Heng Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications*, pages 1084–1127, 2008.

[6] Chunhua Deng, Fangxuan Sun, Xuehai Qian, et al. TIE: Energy-efficient Tensor Train-based Inference Engine for Deep Neural Network. In *Proc. ISCA*, 2019.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019.

[8] Jiaqi Gu, Hanqing Zhu, Chenghao Feng, Mingjie Liu, Zixuan Jiang, Ray T. Chen, and David Z. Pan. Towards memory-efficient neural networks via multi-level in situ generation. In *Proc. ICCV*, pages 5229–5238, October 2021.

[9] Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, et al. Tensorized Embedding Layers. In *Proc. EMNLP*, 2020.

[10] Jean Kossaifi, Zachary C. Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *J. Mach. Learn. Res.*, 21(1), 2020.

[11] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019.

[12] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, , and Dmitry Vetrov. Tensorizing neural networks. In *Proc. NIPS*, 2015.

[13] Ivan V Oseledets. Tensor-train decomposition. *SIAM*, 2011.

[14] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *Proc. ISPASS*, pages 304–315, 2019.

[15] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL*, 2018.

[16] A.H. NFEA Phan. Nfea: Tensor toolbox for feature extraction and applications. *Technical Report*, 2011.

[17] Victor Sanh, Lysandre Debut, Julien Chaumond, et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Proc. NeurIPS*, 2019.

[18] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proc. MICRO*, page 14–27, 2019.

[19] Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Trans. on Signal Processing*, 65(13):3551–3582, 2017.

[20] Daniel G. A. Smith and Johnnie Gray. opt_einsum - a python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software*, 3(26):753, 2018.

[21] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Tensor decomposition for compressing recurrent neural network. In *Proc. IJCNN*, 2018.

[22] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 1966.

[23] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, et al. FBNet: Hardware-aware Efficient Convnet Design via Differentiable Neural Architecture Search. In *Proc. CVPR*, 2019.

[24] Miao Yin, Siyu Liao, Xiao-Yang Liu, Xiaodong Wang, and Bo Yuan. Towards extremely compact rnns for video recognition with fully decomposed hierarchical tucker structure. In *Proc. CVPR*, 2021.