Krony-PT: GPT-2 Compressed with Kronecker Products

Mohamed Ayoub Ben Ayad
Chair of Data Science
University of Passau
Passau, Germany
Mohamed.Benayad@uni-passau.de

Jelena Mitrović
Chair of Data Science
University of Passau
Passau, Germany
Jelena.Mitrovic@uni-passau.de

Michael Granitzer
Chair of Data Science
University of Passau
Passau, Germany
Michael.Granitzer@uni-passau.de

***Abstract*—We introduce Krony-PT, a compression technique for GPT-2 based on Kronecker products. We specifically target the feed-forward weights of each transformer block, and systematically compress the feed-forward layer matrices to various degrees. We introduce a modified Van Loan decomposition to initialize new Kronecker factors, and also propose a new pruning-based initialization technique. Our method compresses the original 124M-parameter GPT-2 to various smaller models, ranging from 80M to 96M. Our 81M model variant outperforms DistilGPT2 on next-token prediction across all standard language modeling datasets, and shows competitive or comparable performance with significantly larger Kronecker-based compressions of GPT-2.**

***Index Terms*—LLMs, Neural Networks Compression**

I. Introduction

Given their rapid development, Large Language Models (LLMs) have revolutionized numerous industries and changed the way we perceive and interact with machines. Their ability to understand, generate, and generalize across various domains, including automated text generation, language translation, and notably code synthesis, has led many to believe in the scaling hypothesis [1], the idea that scaling model size is "all you need" to achieve Artificial General Intelligence (AGI). Despite their advanced features, the core of LLMs still fundamentally relies on the original Transformer architecture [2], particularly the decoder-only variants popularized by the release of GPT-2 [3].

While large corporations can afford the vast computational resources and data required to train state-of-the-art models like GPT-4 [4] and its competitors [5]–[7], individuals and smaller organizations face significant barriers to reproduce these frontier models. Therefore, advancing research on LLM compression and factorization is essential to make these powerful tools more accessible, sustainable, and deployable on resource-constrained hardware, such as edge devices.

Matrix Factorization techniques leverage ideas from classical linear algebra (e.g., SVD, QR, and Kronecker products) to reduce the size and parameter count of LLMs. In particular, the Kronecker product [8] offers an effective method for decomposing large matrices into structured, smaller blocks. For instance, using the Van Loan decomposition (detailed in Section III-E), a singular value decomposition (SVD)-based method, we can optimally[1] express any given matrix as a sum of multiple Kronecker products of smaller matrices. Kronecker product decompositions (with appropriate factor dimensions, see Section III-C) can achieve a significant parameter reduction while preserving the rank of the original matrix, in contrast to methods that specifically aim for a low-rank approximation [9], [10]. Kronecker products also provide a natural framework for vectorizing operations and accelerating computation.

This paper builds on these methods by applying Kronecker-based factorizations to the MLP layers of GPT-2. We extend the existing literature on Kronecker-based compression of LLMs by introducing various rank-invariant compression schemes, with key architectural changes to the MLPs of each transformer layer, leading to models of different sizes. We also introduce two lightweight initialization techniques that significantly improve and stabilize convergence.

Our contributions[2] could be summarized as follows:

- We explore different compression schemes that produce models of various sizes, including 81M, 92M, and 96M.
- We propose a new pruning-based initialization for the 96M model, and an improved scale-invariant Van Loan (VL) decomposition for the other models.
- We enhance the decomposition by introducing learnable scalars for each Kronecker factor, providing an additional degree of freedom without a significant increase in parameters or inference overhead.
- We apply a consistent compression strategy across all transformer layers, ensuring a uniform architectural modification, unlike prior work that selectively compresses layers.
- We use weight tying for the embedding and final output matrices. This practice, standard for models like GPT-2, reduces the parameter count by 38M. Other Kronecker-based methods [11], [12] do not employ this, resulting in parameter counts that are not directly comparable to ours (see Appendix B for details).

[1] In the Frobenius norm sense.

[2] The implementation is available at: https://github.com/eigenAyoub/krony-PT/

II. Related Work

There are three major lines of research in the compression of LLMs: **Distillation** [13]–[15], **Quantization** [16]–[18], and **Factorization** [19]. Our work focuses on the latter category, Matrix Factorization provides a straightforward framework to compress models. Generally speaking, Matrix Factorization techniques aim to reconstruct an original (typically larger) matrix by combining smaller, often lower-rank, matrices (e.g., via a product). With techniques ranging from SVD [20], [21], to other low-rank factorization methods [22], while the use of Kronecker products remains less common.

Kronecker products [23], [24] provide a powerful framework for matrix manipulation and can simplify complex computations through more compact and intuitive notation (e.g., computing the Kronecker-based Fisher matrix approximations [25]). Their use naturally arises in machine learning, where they help model and accelerate computations in large neural networks, and reduce the notational burden, as demonstrated in various applications [26], [27]. Additionally, Kronecker products can also be used to approximate a given matrix by two smaller matrices, e.g., using decomposition methods like Van Loan's [23], hence greatly reducing the total number of parameters of the model.

Using Kronecker products as a factorization technique first appeared in the literature to approximate the weight matrices in fully connected neural networks by a sum of Kronecker products [28], and was then used to compress LLMs. This approach was most notable for encoder-only models like BERT [29], and more recently applied to decoder-only methods like GPT-2 [11], [12].

III. Methodology

This section details our compression methodology for the GPT-2 Small (124M) model, which we refer to as GPT-2 for the remainder of this paper. We describe our factorization approach, define the resulting compression schemes, and highlight key design choices that distinguish our work from prior art.

A. Compression Strategy and Key Differentiators

Our compression strategy targets the most parameter-dense components of the GPT-2 architecture. A standard GPT-2 model consists of 12 transformer layers, each containing a self-attention module and a two-layer feed-forward network (FFN). We exclusively compress the weight matrices of these FFNs[3]. Collectively, these matrices account for 56.6M parameters, amounting to 45% of the model's total 124M parameters. Compressing them therefore offers a path to significant model size reduction. Our methodology differs from prior Kronecker-based compression work [11], [12] in two fundamental ways:

- **Uniform Layer Compression:** We apply our factorization scheme identically to all 12 transformer layers, creating a homogeneous architecture, in contrast to methods that only compress odd-numbered layers.
- **Weight Tying:** We employ weight tying for the input embedding and final output projection matrices, a standard practice that reduces the parameter count by 38M. This is a critical difference that makes direct parameter-count comparisons more nuanced (see Appendix B).

[3]Referred to as `c_fc.weight` and `c_proj.weight` in the code.

B. An Introductory Example: The 96M Model

The most direct factorization strategy is one that effectively halves one dimension of each of the two large FFN weight matrices. This approach leads to our 96M-parameter model.

Let the two FFN matrices for a given layer be $p_1 \in \mathbb{R}^{3072\times768}$ and $p_2 \in \mathbb{R}^{768\times3072}$, which we decompose using a Kronecker product in the following fashion:

$$p_1 = \underbrace{W_{11}}_{(3072,384)} \otimes \underbrace{W_{12}}_{(1,2)} \quad \& \quad p_2 = \underbrace{W_{21}}_{(384,3072)} \otimes \underbrace{W_{22}}_{(2,1)}$$

This factorization reduces the FFN parameters by 50% (a 28M reduction), resulting in a final model of approximately 96M parameters. While this is a substantial compression, our ultimate goal is to achieve an 81M model for a fair and direct comparison with DistilGPT-2 [15]. To reach this more aggressive compression target, we must move beyond this simple dimensional split and explore a broader design space of factorization schemes.

C. The Kronecker Factorization Design Space

The 96M model represents a single point in a design space of possible decompositions. This space is governed by a fundamental trade-off: minimizing the parameter count versus preserving the expressive capacity of the original matrix. A key measure of this capacity is matrix rank. For Kronecker products, rank is multiplicative: $\text{rank}(A \otimes B) = \text{rank}(A) \cdot \text{rank}(B)$.[4]

Formally, for an FFN matrix $W \in \mathbb{R}^{m\times n}$ (where $m = 3072, n = 768$), we seek an approximation $W \approx A \otimes B$ with factors $A \in \mathbb{R}^{m_1\times n_1}$ and $B \in \mathbb{R}^{m_2\times n_2}$. The dimensions must satisfy $m = m_1m_2$ and $n = n_1n_2$. The number of parameters in this factorized form is $m_1n_1 + m_2n_2$. Our goal is to find factor dimensions that significantly reduce this parameter count while preserving the maximal rank of the original matrix, which is $\min(m, n) = 768$.

Table I summarizes several decomposition schemes that satisfy this maximal-rank criterion, detailing their resulting FFN parameter counts and total model sizes. Our experimental investigation focuses primarily on the models in the 96M, 81M.

Furthermore, the single-factor decomposition ($W \approx A \otimes B$) can be extended to a sum of multiple factors ($W \approx \sum_{i=1}^{k} A_i \otimes B_i$). This provides another degree of freedom to control the parameter-performance trade-off. Table II shows how the model size increases as we add more factors for specific base dimensions. Our work also explores a 4-factor model which results in a total size of 92M parameters.

[4]For a proof, see: https://math.stackexchange.com/questions/707091/elementary-proof-for-textrank-lefta-otimes-b-right-textranka-cdot

TABLE I
RANK-PRESERVING KRONECKER DECOMPOSITION SCHEMES FOR FFN LAYERS

Name	Dimension	# Params	Model size
67M	(64, 32)	3200	67,893,504
	(64, 48)	3840	67,908,864
	(96, 32)	3840	67,908,864
	(64, 64)	4672	67,928,832
	(128, 32)	4672	67,928,832
	(96, 48)	5120	67,939,584
	(96, 64)	6528	67,973,376
	(128, 48)	6528	67,973,376
68M	(128, 64)	8480	68,020,224
	(96, 96)	9472	68,044,032
	(192, 48)	9472	68,044,032
	(128, 96)	12480	68,116,224
	(192, 64)	12480	68,116,224
MF1	(128, 128)	16528	68,213,376
	...	...	...
MF2	(1024, 256)	262153	74,108,376
	(768, 384)	294920	74,894,784
	(1024, 384)	393222	77,254,032
81M	(768, 768)	589828	81,972,576
	(1536, 384)	589828	81,972,576
	(1024, 768)	786435	86,691,144
96M	(1536, 768)	1179650	96,128,304
GPT-2	(3072, 768)	2359297	124,439,832

TABLE II
ADDING MULTIPLE KRONECKER FACTORS

Name	Dimension	#params	1 factor	2 factors	3 factors
MF1	(256, 64)	16 528	68.2 M	68.6 M	69 M
MF2	(1024, 256)	262 153	74 M	80 M	86 M

D. Initialization Strategies

Having defined the target architecture of the compressed model, the crucial next step is to initialize the new factor weights. Since we start from a pre-trained GPT-2 checkpoint, our goal is to initialize the factors in a way that preserves as much knowledge from the original pre-trained model as possible. While parameters common to both models can be copied directly, initializing the new Kronecker factors is more challenging.

In this work, we propose and evaluate two distinct initialization strategies:

(i) A novel rescaling of the standard Van Loan (VL) decomposition that preserves the norm of the original weight matrix, detailed in Section III-F.

(ii) A new pruning-based initialization that forgoes SVD entirely, described in Section III-G.

Both approaches are compared against the standard VL decomposition, which we describe first as a baseline.

E. Baseline: The Van Loan Decomposition

The standard method for initializing Kronecker factors from an original matrix is the Van Loan (VL) decomposition [23].

Algorithm 1 Kronecker Product Decomposition

Require: Matrix A, integers m, m_2, n, n_2, and rank k
Ensure: Kronecker factors $\{U_i\}$ and $\{V_i\}$

1: Rearrange A into a matrix suitable for Kronecker decomposition
2: Compute thin SVD:

$$A \approx U\,S\,V^T$$

retaining the top k singular values
3: Reshape and scale the columns of U and V to the desired factor dimensions
4: **return** $\{U_i\}$ and $\{V_i\}$ scaled by $\sqrt{S}$

TABLE III
COMPARISON OF DIMENSIONS, FACTORS, PARAMETERS, AND MODEL NAMES

Model name	A_i dim	B_i dim	# factors	# parameters
KronyPT-80M-2	(1024, 256)	(3, 3)	2	80 M
KronyPT-92M-4	(1024, 256)	(3, 3)	4	92.2 M

This SVD-based algorithm finds a set of factor pairs $\{U_i, V_i\}$ whose Kronecker product sum, $\sum_{i=1}^{k} U_i \otimes V_i$, is the optimal rank-$k$ approximation of a target matrix W in the Frobenius norm sense ($\arg\min \left\| W - \sum_{i=1}^{k} U_i \otimes V_i \right\|_F$). The algorithm (detailed in Algorithm 1) essentially rearranges the target matrix W to isolate the structure of the factors, performs an SVD to find the most significant components, and then reshapes these components back into the desired factor dimensions. A Python implementation is provided in Appendix A.

a) Multiple factors with scalars.: Algorithm 1 returns k factors. In our work, we take advantage of each Kronecker factor $A_i \otimes B_i$ and add it with an additional degree of freedom in the form of a scalar s_i, resulting in our MLP weight being represented as follows $W = \sum_{n=1}^{k} s_i (A_i \otimes B_i)$, Table III details two multi-factor configurations used in our experiments (see Sections IV-C and IV-E).

While the VL framework is mathematically optimal for reconstruction, we identified a key limitation of this approach when applied to pre-trained neural networks.

F. Method 1: Adaptive Normalization for VL

A critical issue with the standard VL decomposition is its failure to preserve the norm of the original weight matrices. This can disrupt the carefully balanced signal magnitudes of the original pre-trained model. Table IV quantifies this problem, showing that the Frobenius norm of the factorized weights can be as low as 67% of the original's, a significant drop in magnitude.

To address this, we introduce a simple but effective adaptive normalization step. Let W be the original weight matrix from GPT-2 and $\hat{W} = \sum_{i=1}^{k} A_i \otimes B_i$ be its standard VL approximation up to k factors. We compute a rescaling factor

TABLE IV
COMPARISON OF NORMS OF SOME PARAMETERS ACROSS DIFFERENT LAYERS FOR THREE MODELS: GPT-2, KRONY-PT-96M, AND KRONY-PT-92M.

Layer	Param.	Norm	GPT-2	96M	92M	%
Layer 1	p_1^1	Frobenius	216	153	146	67.6%
		ℓ_1	248 k	154 k	161 k	
	p_2^1	Frobenius	135	95	91	67.4%
		ℓ_1	151 k	84 k	94 k	
Layer 2	p_1^2	Frobenius	200	142	134	67%
		ℓ_1	235 k	132 k	157 k	
	p_2^2	Frobenius	133	94	92	69.1%
		ℓ_1	151 k	103 k	91 k	
Layer 11	p_1^{11}	Frobenius	199	141	134	67.3%
		ℓ_1	241 k	170 k	161 k	
	p_2^{11}	Frobenius	304	217	204	67.1%
		ℓ_1	335 k	243 k	223 k	

α_W as the ratio of their norms:

$$\alpha_W = \frac{\|W\|_F}{\|\hat{W}\|_F}$$

We then define our initialized matrix, $\hat{W}_{\text{norm}}$, as the rescaled approximation:

$$\hat{W}_{\text{norm}} = \alpha_W \hat{W}$$

This ensures our initialized weights have the exact same Frobenius norm as the original weights, i.e., $\|\hat{W}_{\text{norm}}\|_F = \|W\|_F$. When using multiple factors with learnable scalars ($W = \sum s_i (A_i \otimes B_i)$), we simply initialize each scalar s_i to α_W. Except for the 96M model using pruning (detailed in the next section), all of our models are initialized with this adaptive normalization technique.

G. Method 2: Pruning-Based Initialization

Prior work has shown that optimal, task-agnostic weight approximations do not necessarily translate to optimal performance in compressed neural networks [10]. This is mostly because these methods aim to minimize reconstruction error independently of the downstream task.

Motivated by these potential limitations, we propose an alternative initialization strategy that completely avoids the SVD-based VL decomposition. Our method induces sparsity in the first factor of the Kronecker product, which is equivalent (for the 96M model variant) to pruning the original matrix by half. This is equivalent to setting one of the Kronecker factors to a fixed vector like $[1, 0]^T$, as illustrated in Figure 1.

To ensure stable training and avoid the zero initialization issue, we instead use $[1, 0.1]^T$ for the second factor. This straightforward, SVD-free method provides a serious alternative to the standard VL initialization used in prior work, and we compare their performance in Table VII.

IV. EXPERIMENTS AND RESULTS

A. Training setup

For pre-training, we follow general industry standards, namely those recommended by the Chinchilla paper [30].

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \xrightarrow[\text{pruning}]{} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Fig. 1. An illustration of pruning. Zeroing out alternating rows in a matrix is equivalent to a Kronecker product with the vector $\begin{bmatrix} 1 & 0 \end{bmatrix}^T$.

TABLE V
PERPLEXITY RESULTS OF KRONY-PT AND DISTILGPT2

Model		Datasets		
# Params	Model Name	wikitext-103	wikitext-2	Lambada
124 M	GPT-2	29.16	24.67	45.28
82 M	DistilGPT2	44.53	36.48	76.00
82 M	**KronyPT-81M**	42.74	35.75	**61.02**
80 M	**KronyPT-80M-2**	**41.06**	**34.56**	66.88

We observed more stable loss curves with larger batch sizes. Consequently, we typically use a batch size of 120 sequences, attained with a gradient accumulation of 5, with each forward pass seeing a batch of 24 sequences, which amounts to approximately 122k tokens per step. We use a cosine scheduler for the **learning rate**, peaking at 6×10^{-5} and diminishing to 6×10^{-6}, with a warm-up ranging from 500 to 1500 steps, depending on the model. We pre-train most our compressed models for 3 epochs, and use 4 epochs for the smaller models, on the OpenWebText corpus [31].

After pre-training, we use the best-performing checkpoint and continue training for one epoch with approximately 327K tokens per step (320 sequences per step), with a very small constant learning rate, ideally 6×10^{-6}. All experiments were conducted on a single A100 80GB GPU.

B. Model Naming, and Evaluation

We adopt the nomenclature **Krony-PT-XM{-Y-factors}**, where X denotes the parameter count and optionally Y the number of Kronecker factors. Omitting the number of factors implicitly implies that only one factor was used. We focus on two model classes: **Krony-PT-81M** and **Krony-PT-96M**. To evaluate their performance, we measure next-token prediction perplexity on three standard benchmarks: wikitext103, wikitext2 [32] and Lambada [33].

C. The 81M class

Table V shows the results of our 81M model compared to other models of the same weight count category, namely DistilGPT2 [15], a distilled [13] version of GPT-2. Our model outperforms DistilGPT2 on all datasets, but especially on Lambada, which has a larger test set compared to both wikitext datasets. Perplexity scores of GPT-2 are also added for reference, all models significantly drop in performance, notably on Lambada.

Table VI compares **Krony-PT-81M** with larger Kronecker-based GPT-2 compressions. Section B details our parameter-counting method and how it differs from previous work. Although the 81-million-parameter variant lags behind the larger models on WikiText-103, it still outperforms all of them on LAMBADA. Note that the LAMBADA validation set we

TABLE VI
PERPLEXITY OF KRONY-PT AGAINST OTHER KRONECKER-BASED MODELS

Model		Datasets		
# Params	Model Name	wiki-103	wiki-2	Lambada
81 M	**KronyPT-81M**	42.74	35.75	**61.02**
119 M	TQCompressedGPT2	**40.28**	**32.25**	64.72
119 M	KnGPT-2	40.97	32.81	67.62

TABLE VII
RESULTS OF PRUNING-BASED INITIALIZATION

Model		Datasets		
# Params	Model Name	Lambada	wiki-103	wiki-2
96 M	**KronyPT-96M-prune**	**54.52**	**38.22**	**32.19**
96 M	**KronyPT-96M-VL**	64.92	41.98	34.99
119 M	TQCompressedGPT2	64.72	40.28	32.25
119 M	KnGPT-2	67.62	40.97	32.81

evaluate on ($\approx$ 400 k tokens) is considerably larger than that of WikiText-103 ($\approx$ 230 k tokens).

D. Pruning initialization compared to Van Loan:

Table VII presents results for our proposed pruning-based initialization applied to **Krony-PT-96M** (denoted **Krony-PT-96M-prune**) alongside other Kronecker-compressed GPT-2 variants [12], [29]. Our method consistently outperforms every competitor, including **Krony-PT-96M-VL**, which shares the same architecture but uses the conventional Van Loan initialization. Our pruning based method is also faster to compute because it avoids the SVD step required by Van Loan. As in earlier experiments, the improvement is most pronounced on LAMBADA.

E. Multiple Kronecker Factors with Normalized Scalars

In this section, we study the effect of using multiple Kronecker factors, each multiplied by a scalar that is normalized as described in Section III-F. To keep the total parameter count comparable, we evaluate two settings:

1) **Baseline (96M):** Single-factor model described in Section III-B, initialized with the standard Van Loan (VL) decomposition.
2) **Multiple factors setting (92.2M):** each MLP weight W is initialized with a sum of 4 Kronecker factors, each scaled by a learned scalar (i.e., $W = \sum_{i=1}^{4} s_i (A_i \otimes B_i)$). The scalars s_i follow our scaled VL initialization; each A_i has size $(256, 1024)$.

Table VIII reports the perplexity of both models.[5] The multiple-factor model with adaptive normalization consistently outperforms the baseline, achieving lower perplexity on every dataset after one epoch and retaining the lead after full training, with the largest gain on LAMBADA.

Additionally, adding a scalar per Kronecker factor increases the parameter count only marginally. For instance, the 92.2M model introduced above adds a negligible 96 scalars [6] to the total parameter count. The inference cost is also unchanged: after training, each scalar can be absorbed into A_i (i.e., $A'_i = s_i A_i$), and serve the model for inference with each MLP weight represented as $W = \sum_{i=1}^{4} A'_i \otimes B_i$.

[5]Training hyper-parameters might not be fully optimal for the baseline; see Section VI.

TABLE VIII
EFFECT OF ADDING MULTIPLE FACTORS WITH NORMALIZED SCALARS AFTER ONE EPOCH AND FULL TRAINING.

Model spec.	Training	Lambada	wiki-103	wiki-2
Classic VL	1 epoch	76.95	47.42	56.69
Adap. normalized VL	1 epoch	**68.38**	**45.13**	**39.50**
Classic VL	Full training	64.92	41.98	34.99
Adap. normalized VL	Full training	**60.74**	**40.78**	**34.75**

TABLE IX
COMPARING PERPLEXITY DURING EPOCHS.

Model	Epoch %	Datasets		
		Lambada	wikitext-103	wikitext-2
96 M	100% epoch	76.95	57.46	47.42
92 M	30% epoch	**74.57**	**45.19**	**36.65**

Convergence speed. Adding multiple Kronecker factors with adaptive, normalized scalars accelerates convergence, as shown in Table IX. The 92 M model with adaptive normalization achieves the baseline one-epoch perplexity after only processing 30 % of the first epoch and performs especially well on both WikiText datasets. Training curves show the same trend. The negative log-likelihood on the *OpenWebText* validation set drops faster for the 92 M model than for the 96 M baseline with standard VL initialization. Fig. 2 shows the first 30 % of the epoch, highlighting this earlier convergence trend. This faster training translates into better generalization, as the same improvements appear in downstream benchmarks that are observed in Table IX.

V. CONCLUSION:

In this work, we introduced Krony-PT, a compression technique for decoder-only models like GPT-2 using Kronecker Products. We systematically compress the feed-forward matrices of each transformer block, and explore different compression schemes resulting in models of various sizes. Our 81M model outperformed DistilGPT2 on next-token prediction and performed competitively with other compressed models of larger size. We also presented a new pruning-based initialization technique, and an improved scale-invariant Van Loan decomposition. We also used weight tying to further reduce model size. Our results highlight the potential of Kronecker-based compression to make large language models more efficient and accessible.

[6]Depicted as follows: $96 = (\underbrace{4}_{\text{number of factors}} \times \underbrace{2}_{\text{MLP matrices per layer}}) \times \underbrace{12}_{\text{number of layers}}$

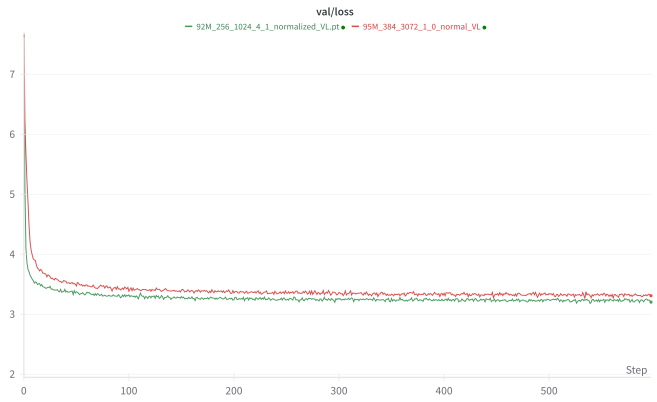

Fig. 2. Negative log-likelihood scores during the first 30% of an epoch on the OpenWebText validation set.

VI. Future work and limitations:

Future work includes investigating the impact of freezing scalar values at 4 when using four factors, as we observed that these scalars consistently converge to around 4 regardless of initialization. Additionally, we plan to explore inference speed improvements using advanced Kronecker Product tensor computations and further study the interpretability of each Kronecker factor. We also acknowledge the incomplete ablation studies due to limited computational resources, as we have not tested all training configurations for competing models.

REFERENCES

[1] G. Branwen, "The scaling hypothesis," 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[5] Anthropic, "The claude 3 model family: Opus, sonnet, haiku," https://www.anthropic.com/news/claude-3-family, 2024.

[6] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[7] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. Lillicrap, J.-b. Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[8] K. Schacke, "On the kronecker product," *Master's thesis, University of Waterloo*, 2004.

[9] M. Ben Noach and Y. Goldberg, "Compressing pre-trained language models by matrix decomposition," in *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, K.-F. Wong, K. Knight, and H. Wu, Eds. Suzhou, China: Association for Computational Linguistics, Dec. 2020, pp. 884–889. [Online]. Available: https://aclanthology.org/2020.aacl-main.88

[10] Y.-C. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen, and H. Jin, "Language model compression with weighted low-rank factorization," *arXiv preprint arXiv:2207.00112*, 2022.

[11] A. Edalati, M. Tahaei, A. Rashid, V. P. Nia, J. J. Clark, and M. Rezagholizadeh, "Kronecker decomposition for gpt compression," *arXiv preprint arXiv:2110.08152*, 2021.

[12] V. Abronin, A. Naumov, D. Mazur, D. Bystrov, K. Tsarova, A. Melnikov, I. Oseledets, S. Dolgov, R. Brasher, and M. Perelshtein, "Tqcompressor: improving tensor decomposition methods in neural networks via permutations," *arXiv preprint arXiv:2401.16367*, 2024.

[13] G. Hinton, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[14] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[15] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," in *NeurIPS EMC2̂ Workshop*, 2019.

[16] J. Chen, Y. Gai, Z. Yao, M. W. Mahoney, and J. E. Gonzalez, "A statistical framework for low-bitwidth training of deep neural networks," *Advances in neural information processing systems*, vol. 33, pp. 883–894, 2020.

[17] Y. Lin, Y. Li, T. Liu, T. Xiao, T. Liu, and J. Zhu, "Towards fully 8-bit integer inference for the transformer model," *arXiv preprint arXiv:2009.08034*, 2020.

[18] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 318–30 332, 2022.

[19] M. B. Noach and Y. Goldberg, "Compressing pre-trained language models by matrix decomposition," in *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, 2020, pp. 884–889.

[20] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," in *Handbook for Automatic Computation: Volume II: Linear Algebra*. Springer, 1971, pp. 134–151.

[21] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6655–6659.

[22] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *Interspeech*, 2013, pp. 2365–2369.

[23] C. F. Van Loan and N. Pitsianis, *Approximation with Kronecker products*. Springer, 1993.

[24] A. Graham, *Kronecker products and matrix calculus with applications*. Courier Dover Publications, 2018.

[25] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *International conference on machine learning*. PMLR, 2015, pp. 2408–2417.

[26] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: an approach to modeling networks." *Journal of Machine Learning Research*, vol. 11, no. 2, 2010.

[27] H. Gao, Z. Wang, and S. Ji, "Kronecker attention networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 229–237.

[28] J.-N. Wu, "Compression of fully-connected layer in neural network by kronecker product," in *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2016, pp. 173–179.

[29] M. Tahaei, E. Charlaix, V. Nia, A. Ghodsi, and M. Rezagholizadeh, "Kroneckerbert: Significant compression of pre-trained language models through kronecker decomposition and knowledge distillation," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 2116–2127.

[30] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark *et al.*, "Training compute-optimal large language models. arxiv," *arXiv preprint arXiv:2203.15556*, 2022.

[31] A. Gokaslan and V. Cohen, "Openwebtext corpus," http://Skylion007. github.io/OpenWebTextCorpus, 2019.

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

[33] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández, "The lambada dataset: Word prediction requiring a broad discourse context," *arXiv preprint arXiv:1606.06031*, 2016.

APPENDIX A
KRONECKER DECOMPOSITION ALGORITHM

As discussed in Section III-E, the optimal Van Loan decomposition can be generated with the following Python script:

```
def kronecker_decompose(A, m: int, n: int, *,
    k: int = 1, niter: int = 10):

    m2 = A.shape[-2] // m
    n2 = A.shape[-1] // n

    A = rearrange(A, "... (m m2) (n n2) -> ...
         (m n) (m2 n2)", m=m, m2=m2, n=n, n2=
        n2)

    u, s, v = torch.svd_lowrank(A, q=k, niter=
        niter)

    u = rearrange(u, "... (m n) k -> ... k m n
        ", m=m, n=n, k=k)
    v = rearrange(v, "... (m2 n2) k -> ... k
        m2 n2", m2=m2, n2=n2, k=k)

    scale = s[..., None, None].sqrt()
    return u * scale, v * scale
```

APPENDIX B
PARAMETER COUNT

Other papers [11], [12] report parameter counts differently from the convention we (and *DistilGPT-2*) follow, making direct comparisons misleading. Their key deviation is to exclude the output embedding matrix, which is substantial (about 40 million parameters) when inherited from pre-trained GPT-2. As KnGPT2 [11] notes: "the number of parameters of the models is reported excluding the output embedding layer in language modelling, which is not compressed, and is equal to row Parameters". TQCompressor [12] adopts the same practice, as clarified in a GitHub discussion[7]. This difference means that their 81 M-parameter model actually contains about 120 M parameters. We do not count the output matrix because we use weight tying, as in the original GPT-2 paper.

APPENDIX C
ONLY 3% OF DATA

Both [29] and [12] state using only 3% of the training data, this doesn't reflect a stronger compression scheme, for the following reasons:

1) They only factorize the odd-indexed transformer blocks, hence, over half the parameters of the compressed model fully match the fully pretrained checkpoint of GPT-2.
2) Within the odd layers, the Kronecker factors are not randomly initialized, as the Van Loan SVD-based decomposition is used which retains the top singular components (and thus the core pretrained features).
3) Most importantly, they reuse GPT-2's original 38 M-parameter output projection unchanged—and they do so without any weight tying—meaning the entire pretrained decoder vocabulary mapping is carried over intact.

Consequently, the precise amount of pretrained knowledge retained is unknown. A fair baseline would randomly initialize all compressed-model parameters, without reusing any weights from the original checkpoint, which is not the case here.

[7]Link to GitHub issue: https://github.com/terra-quantum-public/TQCompressedGPT2/issues/1