# NeuKron: Constant-Size Lossy Compression of Sparse Reorderable Matrices and Tensors (Supplementary Document)

NOTE: If a preview does not appear properly, please download this file.

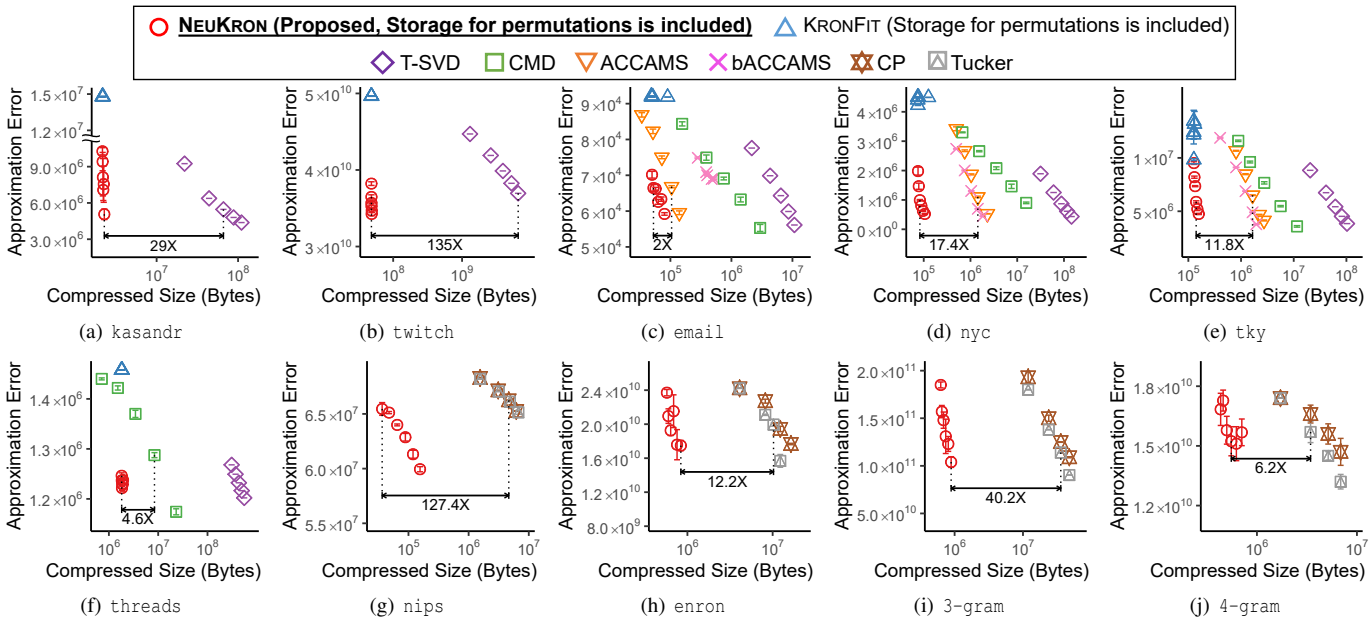



Fig. 1. NEUKRON significantly outperforms the competitors even in *non-reorderable* matrices and tensors. Even when the permutations of indices for all modes are included, the outputs of NEUKRON require up to two orders of magnitude smaller space than those of the competitors with similar approximation error.

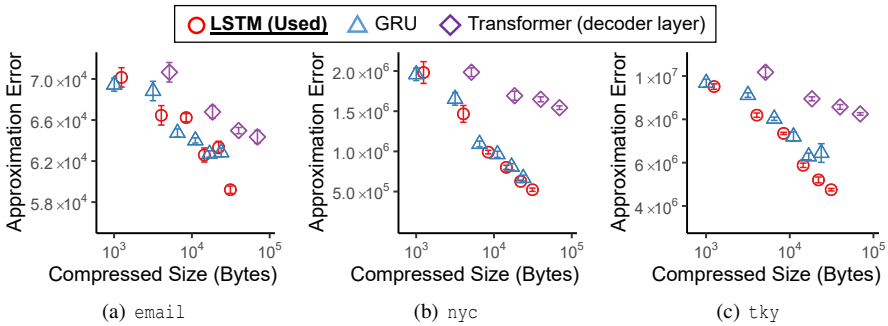



Fig. 2. When equipped with NEUKRON, LSTM leads to concise and accurate compression. NEUKRON with LSTM and that with GRU show perform similarly, but NEUKRON with the decoder layer of Transformer requires significantly more space for the same level of approximation error.

## I. Effectiveness of NeuKron on a non-reorderable setting (Related to Section I)

NeuKron can also be applied to non-reorderable matrices and tensors if the mapping between the original and new orders of mode indices are stored additionally. Even with this additional space requirement, NeuKron still yielded the best trade-off between the approximation error and the compressed size, as seen in Figure 1. Especially, the compression size of NeuKron was two orders of magnitude smaller than those of the competitors with similar approximation error on the `twitch` and `nips` datasets. Remark that KronFit also needs space for storing orders when it is applied to non-reorderable matrices.

## II. Analysis for the type of the sequential model (Related to Section IV.A)

We compared the performances of auto-regressive sequence models, when they are equipped with NeuKron. We varied the hidden dimesion $h$ of NeuKron from 5 to 30 for LSTM and GRU, and the model dimension $d_{model}$ from 8 to 32 for the decoder layer of Transformer. As seen in Figure 2, when equipped with NeuKron, LSTM and GRU performed similarly, outperforming the decoder layer of Transformer.

## III. Analysis on inference time (Related to Section IV.A)

We measure the inference time for $10^6$ elements randomly chosen from square matrices of which numbers of rows and cols vary from $2^7$ to $2^{16}$. We ran 5 experiments for each size and report the average of them. As expected from Theorem 1 of the main paper, the approximation of each entry by NeuKron is almost in $\Omega(\log M)$ (see Figure 4).

## IV. Analysis of the Tensor Extension (Related to Section V)

Below, we analyze the time and the space complexities of NeuKron extended to sparse reorderable tensors. For all proofs, we assume a $D$-order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_D}$ where $N_1 \leq N_2 \leq \cdots \leq N_D$, without loss of generality. The complexities are the same with those in Section IV of the main paper if we assume a fixed-order tensor (i.e., if $D = O(1)$).

**Theorem 1** (Approximation Time for Each Entry)**.** *The approximation of each entry by* NeuKron *takes* $\Theta(D \log N_D)$ *time.*

*Proof.* For encoding, NeuKron subdivides the input tensor $O(\log N_D)$ times and each subdivision takes $O(D)$. For approximation, the length of the input of the LSTM equals to the number of the subdivisions, so the time complexity for retrieving each entry is $O(D \log N_D)$. □

**Theorem 2** (Training Time)**.** *Each training epoch in* NeuKron *takes* $O(nnz(\mathcal{X}) \cdot D \log N_D + D^2 N_D)$.

*Proof.* Since the time complexity for inference is $O(D \log N_D)$ for each input, model optimization takes $O(\text{nnz}(\mathcal{X}) \cdot D \log N_D)$. For reordering, the time complexity of matching the indices as pairs for all the dimensions is bounded above to $O(D \cdot (DN_D)) = O(D^2 N_D)$. For checking the criterion for all pairs, we need to retrieve all the non-zero entries, and it takes $O(\text{nnz}(\mathcal{X}) \cdot D \log N_D)$. Therefore, the overall training time per epoch is $O(\text{nnz}(\mathcal{X}) \cdot D \log N_D + D^2 N_D)$. □

TABLE I
Semantics of Real-world Datasets

| Name | Description |
|---|---|
| `email` | e-mail addresses × e-mails [binary] |
| `nyc` | venues × users [check-in counts] |
| `tky` | venues × users [check-in counts] |
| `kasandr` | offers × users [clicks] |
| `threads` | users × threads [participation] |
| `twitch` | streamers × users [watching time] |
| `nips` | papers × authors × words [counts] |
| `4-gram` | words × words × words × words [counts] |
| `3-gram` | words × words × words [counts] |
| `enron` | receivers × senders × words [counts] |

**Theorem 3** (Space Complexity during Training)**.** NeuKron *requires* $O(D \cdot nnz(\mathcal{X}) + D^2 N_D)$ *space during training.*

*Proof.* The bottleneck is storing the input tensor in a sparse format, the random hash functions and the shingle values, which require $O(D \cdot \text{nnz}(\mathcal{X}))$, $O(\sum_{i=1}^{D} N_i)$, and $O((D-1) \cdot \sum_{i=1}^{D} N_i)$, respectively. Thus, the overall complexity during training is $O(D \cdot \text{nnz}(\mathcal{X}) + (D-1) \cdot \sum_{i=1}^{D} N_i) = O(D \cdot \text{nnz}(\mathcal{X}) + D^2 N_D)$. □

**Theorem 4** (Space Complexity of Outputs)**.** *The number of model parameters of* NeuKron *is* $\Theta(2^D)$.

*Proof.* In NeuKron, the number of parameters for LSTM does not depend on the order of the input tensor; thus, it is still in $\Theta(1)$. The embedding layer and the linear layers connected to the LSTM require $\Theta(2+2^2+\cdots+2^D) = \Theta(2^D)$ parameters. □

## V. Semantics of Datasets (Related to Section VI.A)

We provide the semantics of the datasets in Table I.

## VI. Implementation Details (Related to Section VI.A)

We implemented NeuKron in PyTorch. We implemented the extended version of KronFit in C++. For ACCAMS, bACCAMS, and CMD, we used the open-source implementations provided by the authors. We used the `svds` function of SciPy for T-SVD. We used the implementations of CP and Tucker decompositions in Tensor Toolbox [1] in MATLAB. Below, we provide the detailed hyperparameter setups of each competitor.

- **KronFit**: The maximum size of the seed matrix was set as follows - `email`: 32 × 161, `nyc`: 33 × 196, `tky`: 14 × 40, `kasandr`: 75 × 80, `threads`: 57 × 85, `twitch`: 30 × 63. We fixed $\gamma$ to 1, and performed a grid search for the learning rate in $\{10^{-1}, 10^{-2}, \cdots, 10^{-8}\}$.
- **T-SVD**: The ranks were up to 50 for `email`, 460 for `nyc`, 200 for `tky`, 15 for `kasandr`, 90 for `threads`, and 50 for `twitch`.

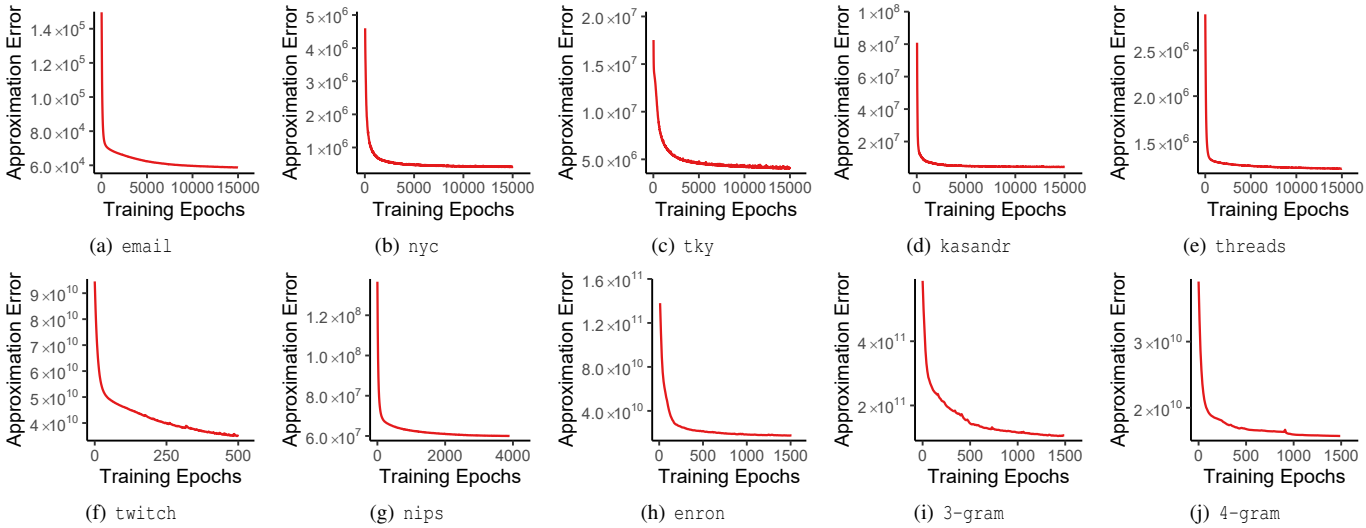



Fig. 3. Approximation error of NEUKRON after each epoch. In most cases, the approximation error drops rapidly in early iterations, especially within one third of the total epochs that are determined by the termination condition in Section 6.1.

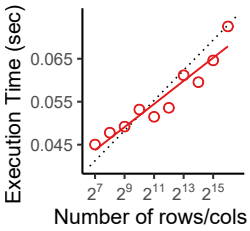



Fig. 4. Inference time of NEUKRON is nearly proportional to the logarithm of the number of rows and columns.

- **CMD**: We sampled (# rows, # columns) up to (150, 2500) for `email`, (1000, 34000) for `nyc`, and (700, 21000) for `twitch`.
- **ACCAMS**: We used 5, 50, and 50 stencils for `email`, `nyc`, and `tky`, respectively. We used up to 48, 64, and 40 clusters of rows and columns for the aforementioned datasets, respectively.
- **bACCAMS**: We set the maximum number of clusters of rows and columns to 48, 48, and 24 for `email`, `nyc`, and `tky`, respectively. We used 50 stencils for the datasets.
- **CP**: The ranks were set up to 40 for `nips`, 8 for `enron`, 20 for `3-gram`, and 4 for `4-gram`.
- **Tucker**: We used hypercubes as core tensors. The maximum dimension of a hypercube for each dataset is as follows - `nips`: 40, `enron`: 6, `3-gram`: 20, and `4-gram`: 4.

We followed the default setting in the official code from the authors for the other hyperparameters of ACCAMS and bACCAMS. The implementations of KronFit, T-SVD, CP, and Tucker used 8 bytes for real numbers. The implementations of ACCAMS and bACCAMS used 4 bytes for real numbers and assumed the Huffman coding for clustering results.

## VII. HYPERPARAMETER ANALYSIS (RELATED TO SECTION VI.A)

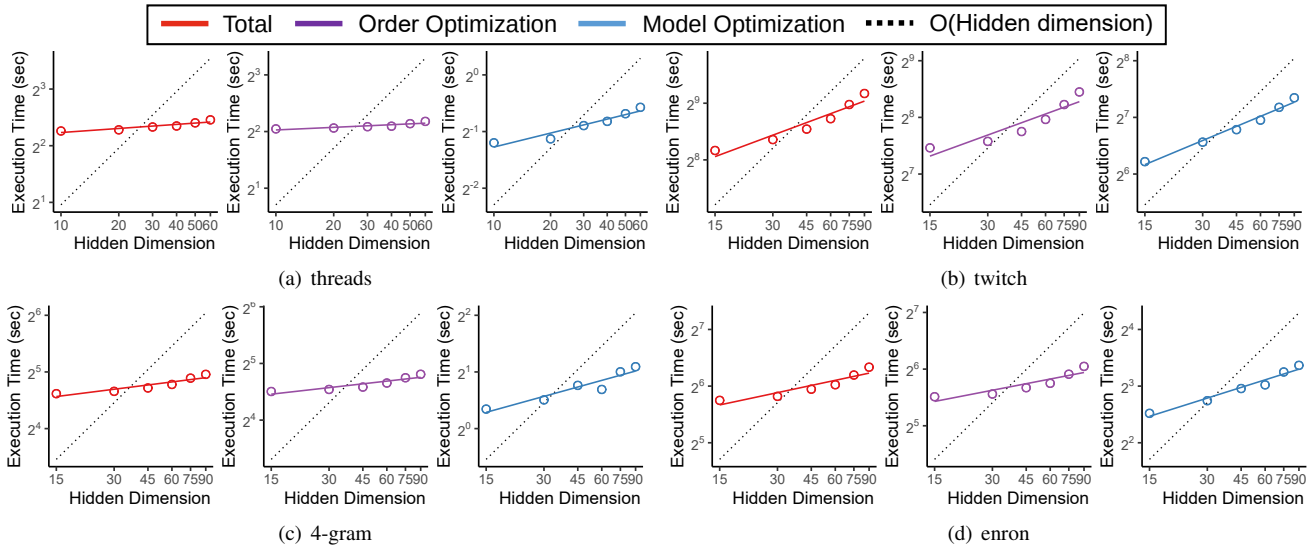
We investigate how the approximation error of NEUKRON varies depending on $\gamma$ values. We considered three $\gamma$ values and four datasets (`email`, `nyc`, `tky`, and `kasandr`) and reported the approximation error in Table II. Note that setting $\gamma$ to $\infty$ results in a hill climbing algorithm that switches rows/column in pairs only if the approximation error decreases. The results show that, empirically, the approximation error was smallest when $\gamma$ was set to 10 on all datasets except for the `nyc` dataset.

TABLE II
THE EFFECT OF $\gamma$ ON APPROXIMATION ERROR. WE REPORT THE MEANS AND STANDARD ERRORS OF APPROXIMATION ERRORS ON THE `email`, `nyc`, `tky`, AND `kasandr` DATASETS.

| Dataset | $\gamma$ | Approximation error |
|---|---|---|
| `email` | 1 | 90561.25 ± 467.996 |
| | **10** | **58691.88** ± 335.143 |
| | $\infty$ | 59113.75 ± 891.544 |
| `nyc` | 1 | 421451.2 ± 4842.068 |
| | 10 | 402673.6 ± 17291.959 |
| | **$\infty$** | **397947.5** ± 2393.016 |
| `tky` | 1 | 4166292.3 ± 143013.605 |
| | **10** | **3981669.6** ± 91907.201 |
| | $\infty$ | 4034389.1 ± 48117.964 |
| `kasandr` | 1 | 6315784.36 ± 140974.6535 |
| | **10** | **4300280.71** ± 488804.599 |
| | $\infty$ | 4385800.32 ± 496004.629 |

TABLE III
THE TRAINING TIME PER EPOCH ON ALL DATASETS. WE REPORT THE MEANS AND STANDARD ERRORS OF TRAINING TIME ON EACH DATASET.

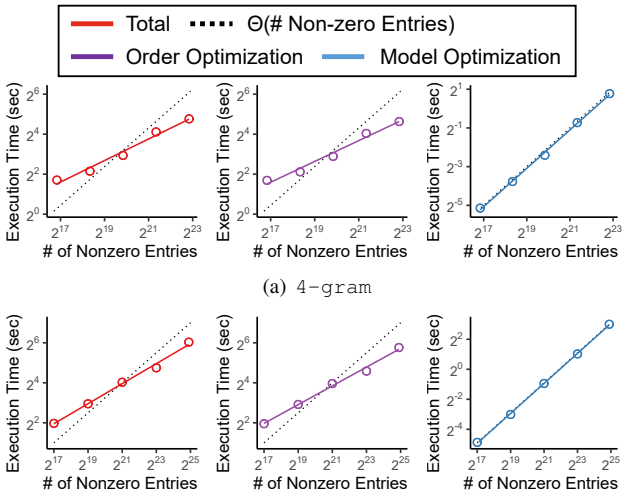
| Dataset (Hidden Dimension) | Training time |
|---|---|
| `email` (30) | 0.19 ± 0.010 |
| `nyc` (30) | 0.21 ± 0.004 |
| `tky` (30) | 0.32 ± 0.005 |
| `kasandr` (60) | 1.93 ± 0.005 |
| `threads` (60) | 5.49 ± 0.012 |
| `twitch` (90) | 566.82 ± 3.308 |
| `nips` (50) | 6.31 ± 0.081 |
| `enron` (90) | 80.69 ± 0.266 |
| `3-gram` (90) | 27.19 ± 0.089 |
| `4-gram` (90) | 41.09 ± 0.785 |




(a) threads (b) twitch

(c) 4-gram (d) enron

Fig. 5. The training time of NEUKRON is empirically sub-linear in the hidden dimension $h$ of NEUKRON. As in Figure 5 of the main paper, we measure the total elapsed time, the elapsed time for order optimization, and the elapsed time for model optimization.




(a) 4-gram

(b) enron

Fig. 6. The training process of NEUKRON on tensor is also scalable. Both model and order optimizations scale near-linearly with the number of non-zeros in the input.

## VIII. SPEED AND SCALABILITY ON HIDDEN DIMENSION (RELATED TO SECTION VI.D)

We report the average the training time per epoch of NEUKRON in Table III. The training time per epoch varied from less than 1 second to more than 9 minutes depending on the dataset. As seen in Figure 3, the training plots of all datasets dropped dramatically within one third of total epochs that were determined by the termination condition in Section 6.1. Thus, a model that worked well enough could be obtained before convergence.

We also analyzed the effect of the hidden dimension $h$ on the training time per epoch of NEUKRON. As seen in Figure 5, both the elapsed time for order optimization and the elapsed for model optimization were empirically sublinear in the hidden dimension.

## IX. SCALABILITY ON TENSOR DATASETS (RELATED TO SECTION VI.D)

For the `4-gram` and `enron` datasets, we generated multiple smaller tensors by sampling non-zero entries uniformly at random. The hidden dimension was fixed to 60. Consistently with the results on matrices, the overall training process of NEUKRON is also linearly scalable on sparse tensors, as seen in Figure 6.

## REFERENCES

[1] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2008.