Algorithms for accelerating vision transformers (ViT) using NLA

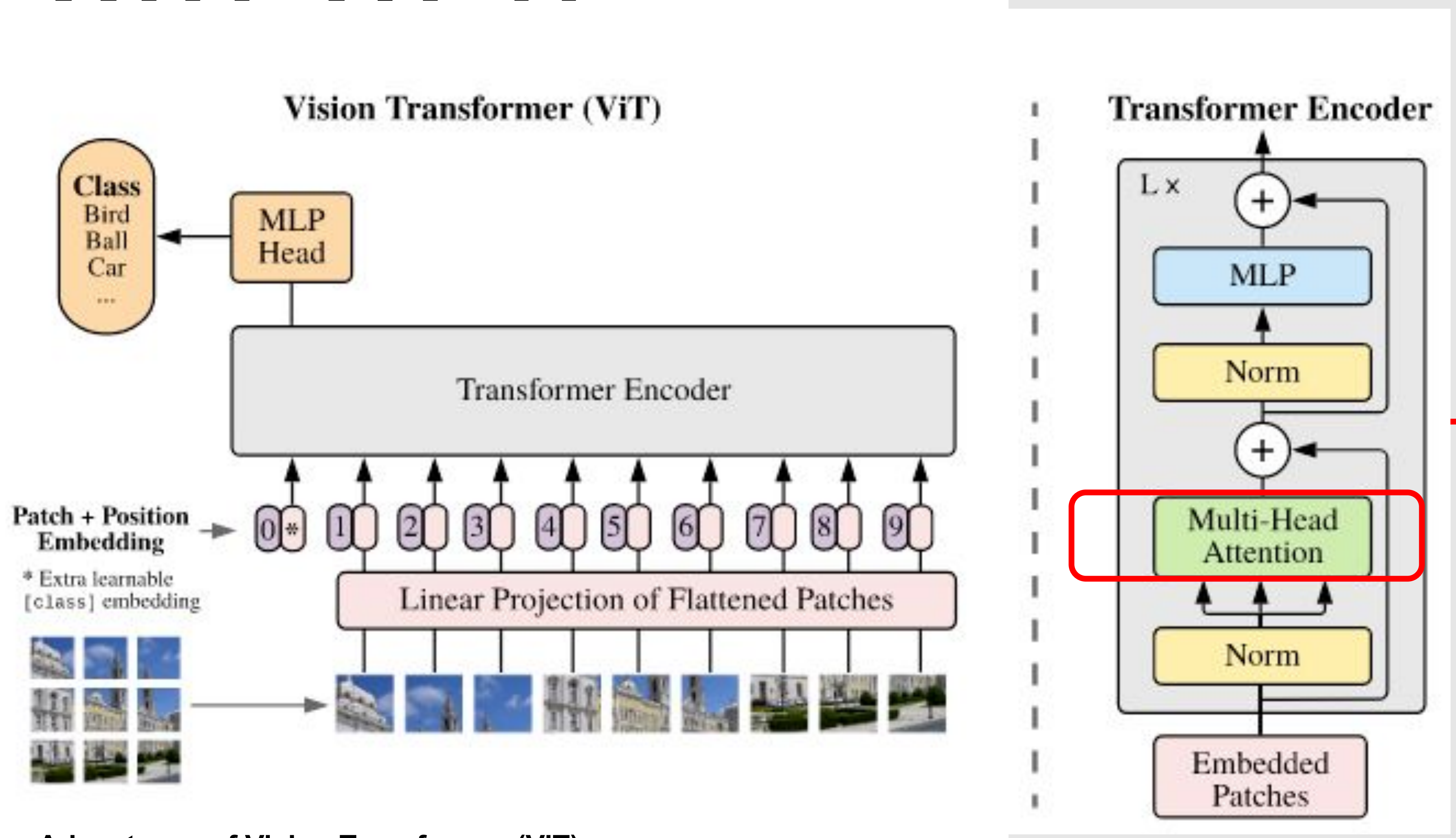
Team 7

Kornienko Dmitrii
Lukina Svetlana
Rahmatulin Oleg
Zavadskaya Lyudmila
Filin Andrey

Moscow, 12.17.2024

Skoltech

WHY VIT ?!



Patch Embedding:
The input image is split into fixed-size patches, each of which is flattened and linearly projected into an embedding vector.

Positional Encoding:
Positional embeddings are added to the patch embeddings to retain positional information, as transformers do not inherently account for spatial structure.

Transformer Encoder:
A series of transformer layers (Multi-Head Self-Attention, Feed-Forward Networks, Layer Normalization) processes the patch embeddings, allowing the model to learn global dependencies.

Classification Token:
A special `[CLS]` token is prepended to the sequence of embeddings. The final representation of this token is used for image classification.

MLP Head:
A Multi-Layer Perceptron processes the `[CLS]` token representation to predict the image class.

Advantages of Vision Transformer (ViT):

- **Flexibility of Transformers Adapted to Vision Tasks:**
  ViT leverages the flexibility of transformer models from NLP for image processing.
- **Efficient Global Dependency Modeling:**
  Through the self-attention mechanism, ViT can effectively capture relationships between any regions in the image, regardless of their spatial location.
- **Superior Performance on Large Datasets (e.g., ImageNet):**
  ViT achieves state-of-the-art accuracy when trained on large-scale datasets, outperforming traditional convolutional neural networks (CNNs).
- **Key Idea — Image Patches Instead of Pixels:**
  By dividing an image into small fixed-size patches (e.g., 16x16 pixels), ViT adapts the transformer model to vision tasks while reducing computational overhead.

Skoltech

WHY MSA ?!

Core Problem: MSA requires computing attention matrices with a complexity of **$O(N^2)$**, where N is the number of patches (i.e., the sequence length).
Solution: The use of decomposition methods, such as **SVD**, sparsity, or random projections, reduces computational complexity and optimizes matrix multiplications.
Application in our case:

Cholesky Decomposition

- **Why Cholesky?**
  Cholesky decomposition is efficient for positive-definite matrices, which often arise in attention computations. It enables faster and more stable matrix factorization compared to general decompositions.
- **Application:** It reduces the cost of matrix operations by decomposing the attention matrix into a lower-triangular form, simplifying further computations.

Singular Value Decomposition (SVD)

- **Why SVD?**
  SVD allows for low-rank approximations of the attention matrix. By retaining only the most significant singular values, we can significantly reduce the matrix rank while preserving essential information.
- **Application:**
  - Compresses attention matrices by removing redundant information.
  - Reduces computational complexity.
  - Improves generalization and reduces overfitting through rank regularization.

Principal Component Analysis (PCA)

- **Why PCA?**
  PCA is a dimensionality reduction technique that identifies principal components capturing the most variance in the data. It simplifies the representation of the attention mechanism.
- **Application:**
  - Reduces dimensionality of key, query, and value matrices.
  - Lowers the cost of attention calculations without significant information loss.

Tensor Decomposition

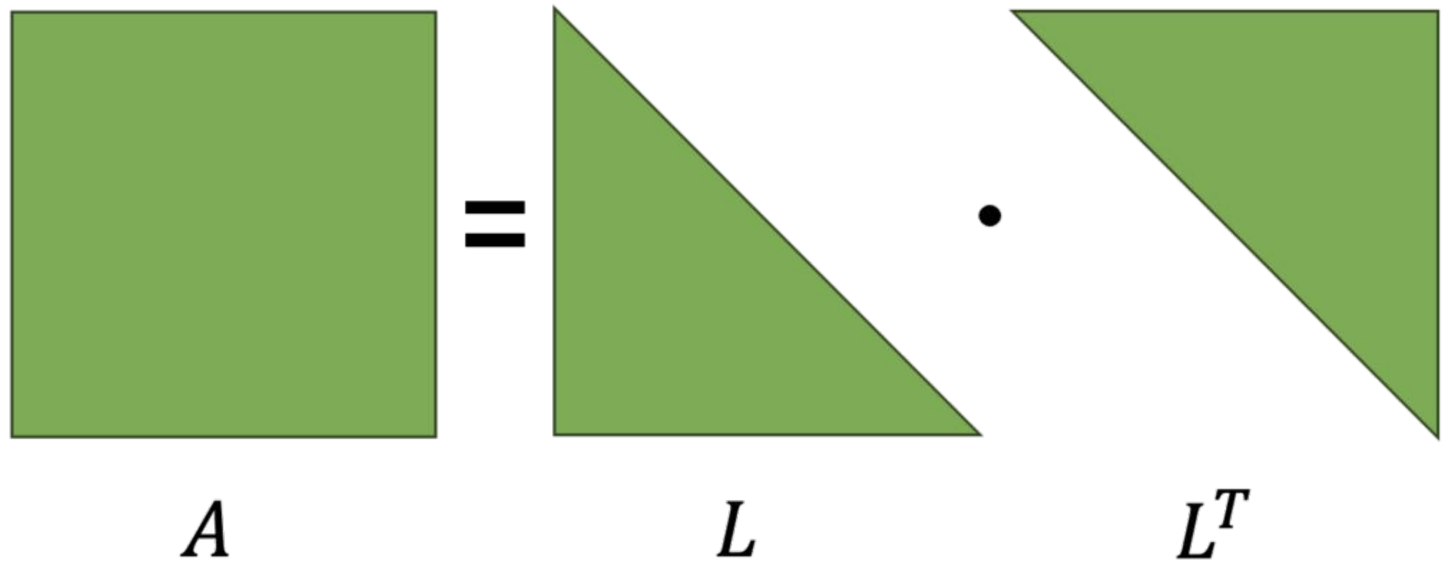
- **Why Tensor Decomposition?**
  Tensor decomposition generalizes matrix factorization methods to higher dimensions, making it suitable for multi-dimensional attention computations. It decomposes tensors into smaller, manageable components.
- **Application:**
  - Reduces memory and computational requirements for high-dimensional attention representations.
  - Enables efficient approximation of attention tensors.

Used methods. Cholesky

We use Cholesky decomposition to the **attention matrix** derived from the input tensor x. Here's how it's implemented:

1. **Compute the Attention Matrix:**
   A=x · x^T
   - **Explanation:** This computes a symmetric matrix by multiplying the input tensor xx with its transpose.
2. **Apply Cholesky Decomposition:**
   A=L · L^T
   - **Explanation:** Decomposes the attention matrix A into a lower triangular matrix L and its transpose L^T.
3. **Store the Cholesky Factor:**
   - **Explanation:** The lower triangular matrix LL is stored for potential future use, such as efficient computation or analysis within the transformer block.

By decomposing the attention matrix A using Cholesky decomposition in model can operations involving A can be performed more efficiently using its Cholesky factors, especially in solving linear systems or computing determinants



Skoltech

Used methods. Cholesky corrected, ok?

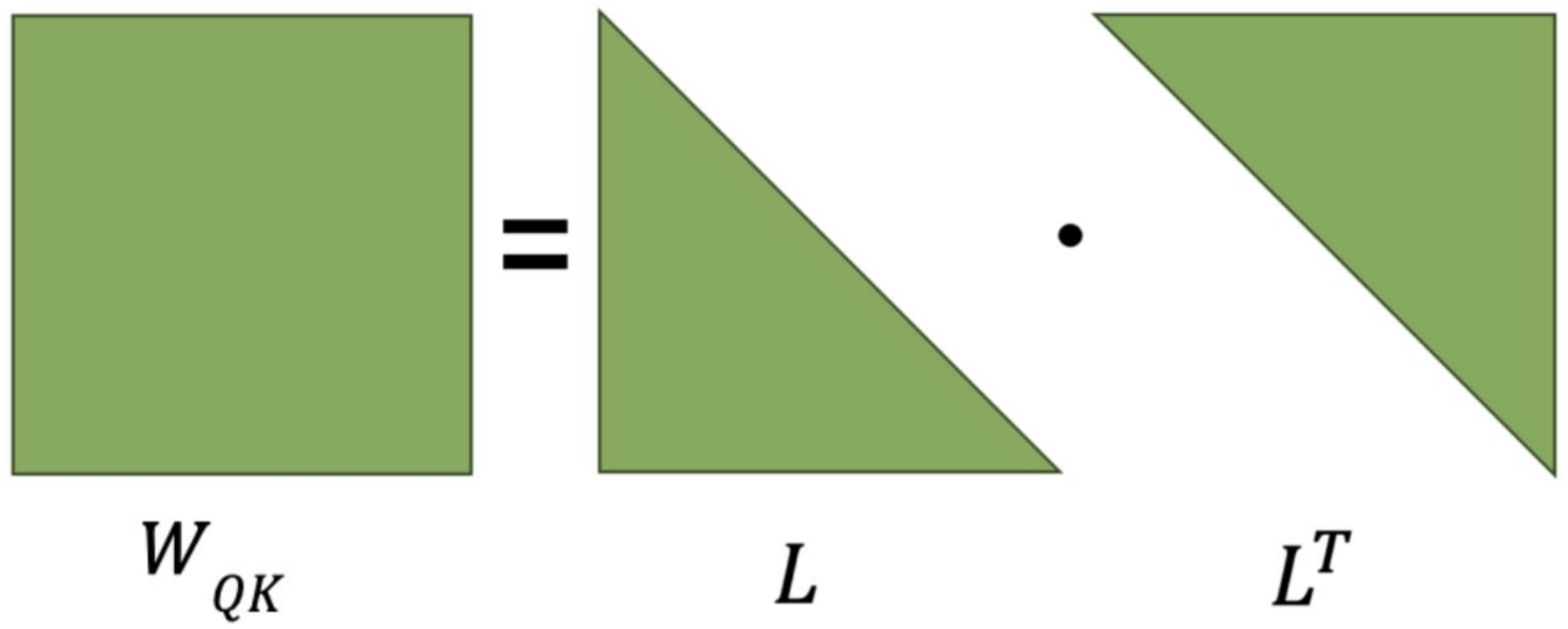
We use Cholesky decomposition to the **attention score matrix**. Here's how it's implemented:

1. **Compute the attention scores matrix:**

   S = X · W_QK · X^T, where W_QK - symmetric weight matrix for queries and keys.
   - **Explanation:** This computes a symmetric matrix S by multiplying the input tensor X with with the weight matrix W_QK and its transpose .
2. **Apply Cholesky Decomposition:**
   W_QK =L · L^T
   Explanation: Decomposes the symmetric weight matrix for queries and keys W_QK into a lower triangular matrix L and its transpose L^T.
3. **Store the Cholesky Factor:**
   - **Explanation:** The lower triangular matrix L is stored for potential future use, such as efficient computation or analysis within the transformer block.

By decomposing the attention score matrix using Cholesky decomposition, operations involving this matrix can be performed more efficiently using its Cholesky factors, especially when solving linear systems or computing determinants.

Cholesky-Based Symmetric Attention: d_k(d_k+1)/2 parameters instead of d_k^2



Skoltech

Used methods. PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving as much variance as possible. It achieves this by identifying the principal components—directions in the data that account for the most variance.

Mathematically, PCA seeks to find a matrix WW that projects the original data X onto a lower-dimensional space:

$X\_reduced = X \cdot W$, where:X is the original data matrix, W contains the principal components as its column, X_reduced is the reduced-dimensionality representation of X

In our code, PCA is used to perform a **low-rank approximation** of the attention matrix.

1. **Compute the Attention Matrix:**
   $A = x \cdot x\^T$
   - **Explanation:** This operation computes the attention matrix AA by multiplying the input tensor x with its transpose, resulting in a symmetric matrix.
2. **Apply PCA for Low-Rank Approximation:**
   $A\ low\_rank = U \cdot \Sigma \cdot V\^T$
   - **Explanation:**.
     - **Transformation:** The attention matrix A is transformed into its principal component representation $Apca = X \cdot W$.
     - **Inverse Transformation:** The low-rank approximation $A\ low\_rank = Apca \cdot W\^T$ reconstructs AA using only the top principal components.
3. **Convert Back to Tensor -** the low-rank approximated attention matrix is converted back to a PyTorch tensor and moved to the appropriate device

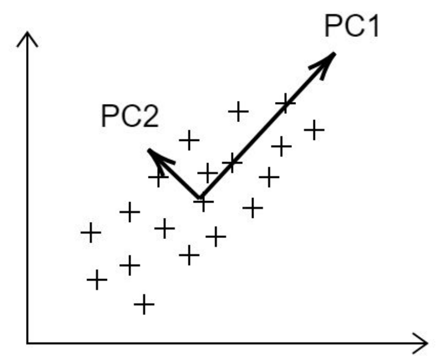
Mathematical Purpose

By applying PCA to the attention matrix AA, the model achieves the following:

- **Dimensionality Reduction:** Reduces the complexity of the attention mechanism by approximating AA with a lower-rank matrix, effectively decreasing the number of parameters and computational overhead.

- **Noise Reduction:** By retaining only the principal components that capture the most variance, PCA helps in filtering out noise and redundant information, potentially enhancing the model's performance.

- **Efficiency:** Low-rank approximations lead to faster c



PC1 = 1st principal component
PC2 = 2nd principal component

Skoltech

Used methods. SVD

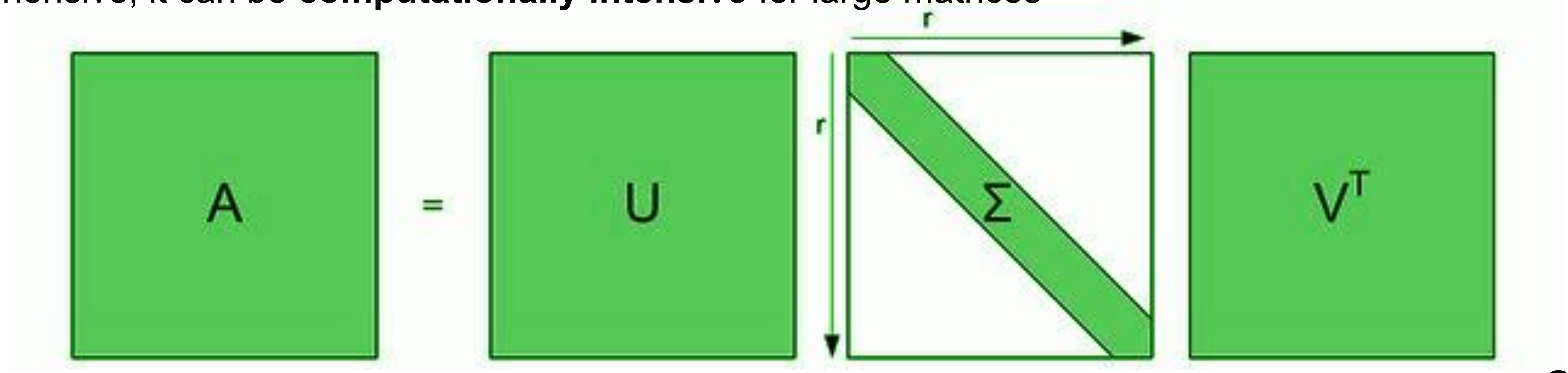
Let consider the matrix multiplication Y=X*W^T + b, where: X∈R(m×p) is the input matrix; W∈R(n×p) is the weight matrix; b∈R(n) is the bias vector
Resulting in Y∈R (m×n)

Full-Rank SVD

Decompose W using Singular Value Decomposition (SVD):

W=UΣV^T, where: U∈R n×n and V∈R p×p are orthogonal matrices and Σ is a diagonal matrix of singular values.

Full-Rank SVD retains **all** singular values and vectors, allowing an exact reconstruction of WW. While comprehensive, it can be **computationally intensive** for large matrices



SVD of A full-rank

Skoltech

Used methods. SVD

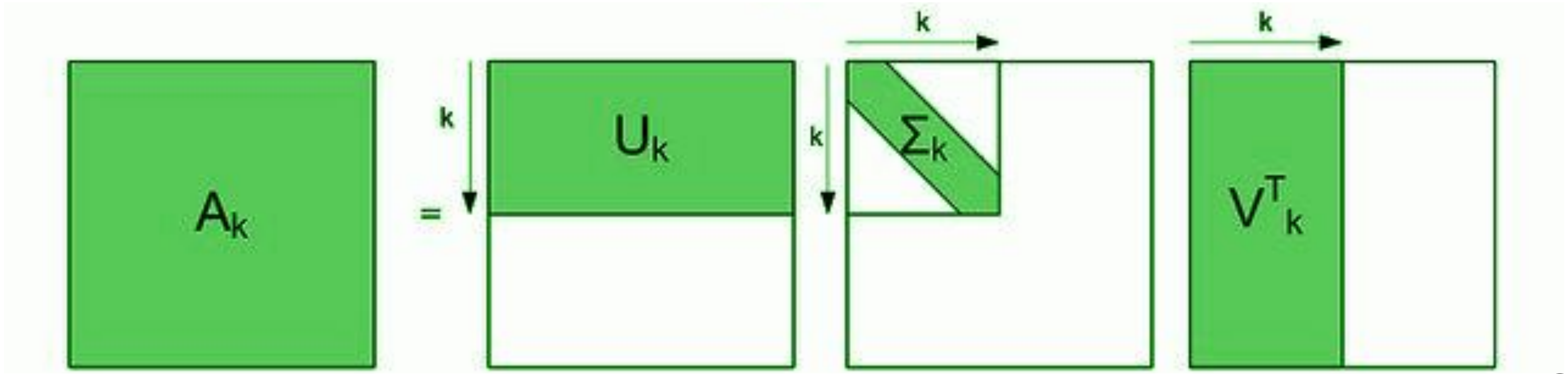
Low-Rank SVD Approximation

To enhance efficiency, retain only the top r < min(p,n) singular values and their corresponding vectors:

W≈UrΣrVr^T, where:Ur∈R n×r; Σr∈R r×r; Vr∈R p×r.

This **low-rank approximation** reduces computational and memory costs while preserving the most significant features:

Y=XVrΣrUr^T+b



SVD of A rank k approximation

Used methods. SVD

In our PyTorch implementation, we enhance Transformer blocks with SVD-based low-rank approximations to improve efficiency and performance:

- **TransformerBlockSVD**:
  - Multi-Head Self-Attention: Captures dependencies in the data.
  - **SVD Decomposition**: Applies SVD to the attention output, retaining only the top singular values and vectors for a low-rank approximation
    - .Operation: Applies Singular Value Decomposition to the attention output to obtain a low-rank approximation.
    - Benefits**:** we reduces the number of parameters and operations from O(m^2) to O(m*r)
  - Feed-Forward Network: Includes a two-layer network with ReLU activation.
  - Layer Normalization: Stabilizes training by normalizing inputs before attention and feed-forward layers.
- **CustomVisionTransformerSVD**:
  - Embedding Layer: Transforms image patches into embedding vectors.
  - **Stack of TransformerBlockSVD**: Incorporates multiple SVD-enhanced Transformer blocks.
    - Operation: Processes embeddings through multiple Transformer blocks enhanced with SVD-based low-rank approximations.
    - Benefits: When calculate X′=TransformerBlockSVD(X) L times for L layers with SVD we canmaintains efficiency even as the number of layers increases.
  - Normalization & Classification: Applies final layer normalization and a fully connected layer to classify images into categories.

Skoltech

Goyal, N., & Kumar, S. Lossy color image compression based on singular value decomposition and GNU GZIP // *International Journal of Computer Applications.* 2011. № 12(9). С. 1-5.

Used methods. Tensor decomposition

In our code we use tensor decomposition to perform a **low-rank approximation** of the attention tensor.

1. Compute the Attention Tensor: A=X⊗X^T

- **Explanation:** This operation computes the attention tensor A by performing a matrix multiplication between the input tensor X and its transpose, resulting in a symmetric tensor.

2. Reshape the Tensor for Decomposition: A reshaped=reshape (A,(−1,A.size(−1)))

- **Explanation:** The attention tensor A we reshape into a 2D matrix A reshaped to facilitate the application of matrix decomposition techniques like SVD.

3. Apply Singular Value Decomposition (SVD): A reshaped=U ⋅ Σ ⋅ V^T

4. Truncate to Desired Rank: U truncated=U[:,:r], Σ truncated=Σ[:r,:r], V truncated=V[:,:r]

- **Explanation:** Only the top rr singular values and their corresponding vectors are retained to form truncated matrices, where rr is the specified rank for the low-rank approximation.

5. Reconstruct the Low-Rank Approximation: A low_rank=U truncated ⋅ Σ truncated ⋅ V truncated^T

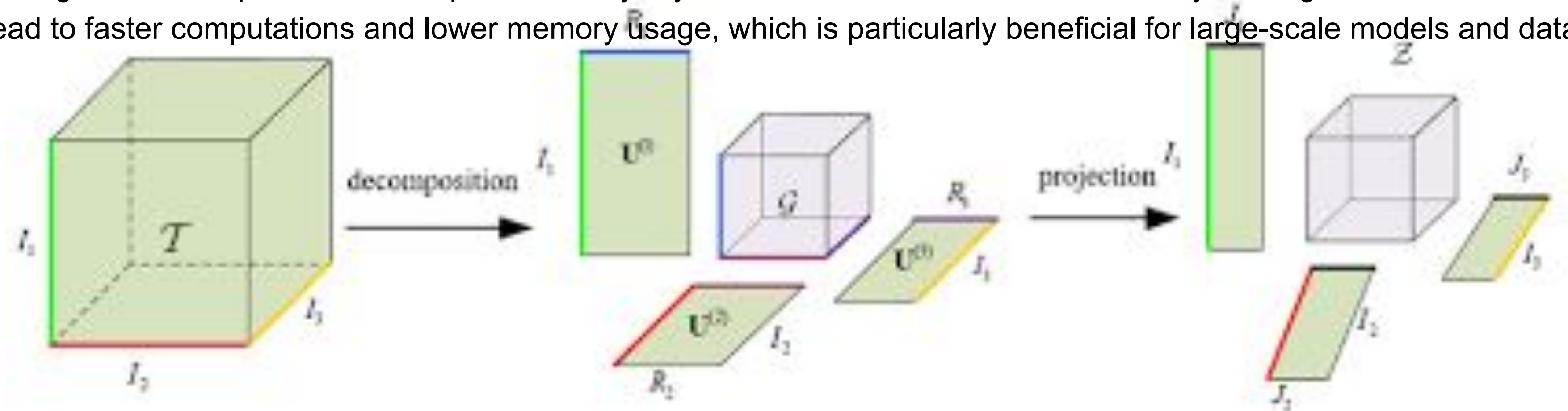
- **Explanation:** The low-rank approximation Alow_rank\mathcal{A}_{\text{low_rank}} is reconstructed by multiplying the truncated matrices, effectively reducing the dimensionality while preserving the most significant components of the original attention tensor.

6. Convert Back to Tensor and Reshape: A low_rank=reshape(A low_rank,A.shape)

- **Explanation:** The low-rank approximated attention matrix is reshaped back to its original tensor form and moved to the appropriate device (CPU or GPU).

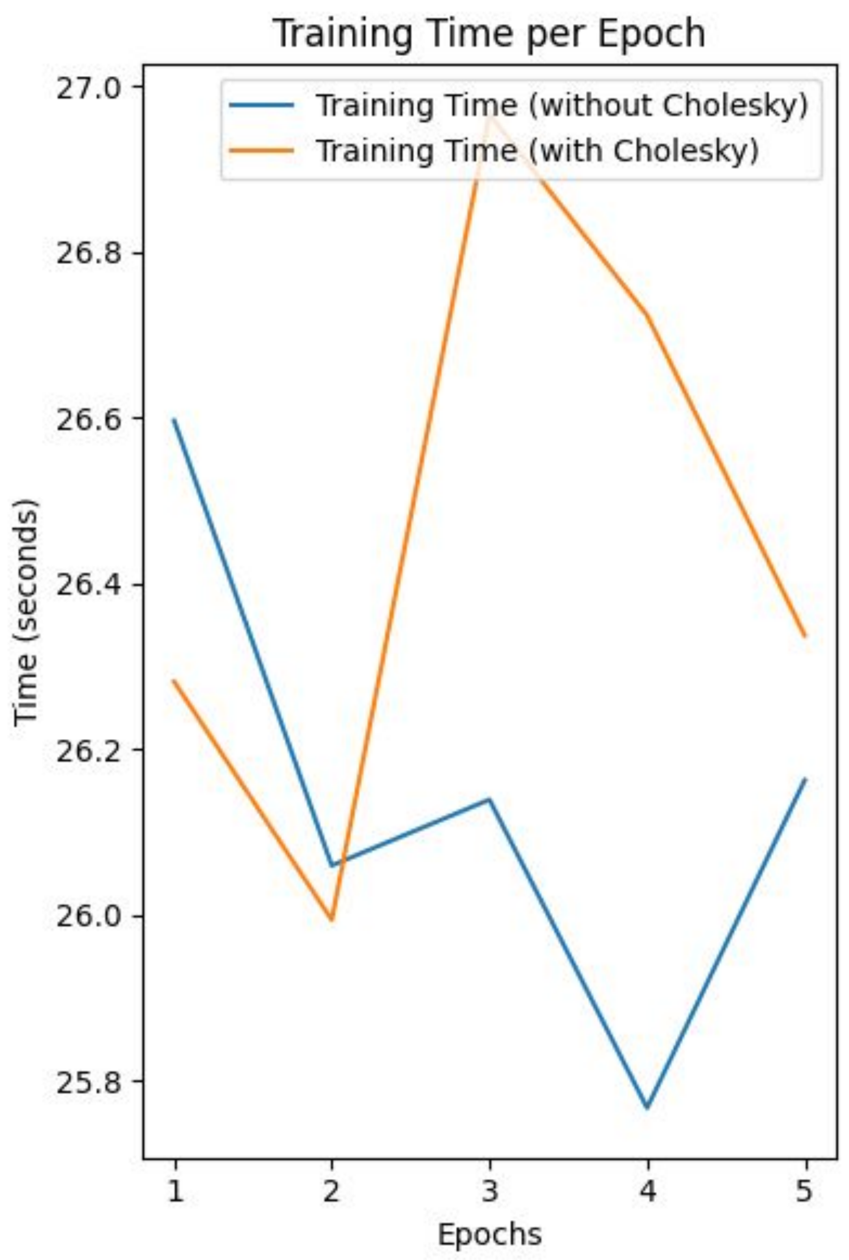
By applying Tensor Decomposition (specifically SVD) to the attention tensor A, the model achieves the following:

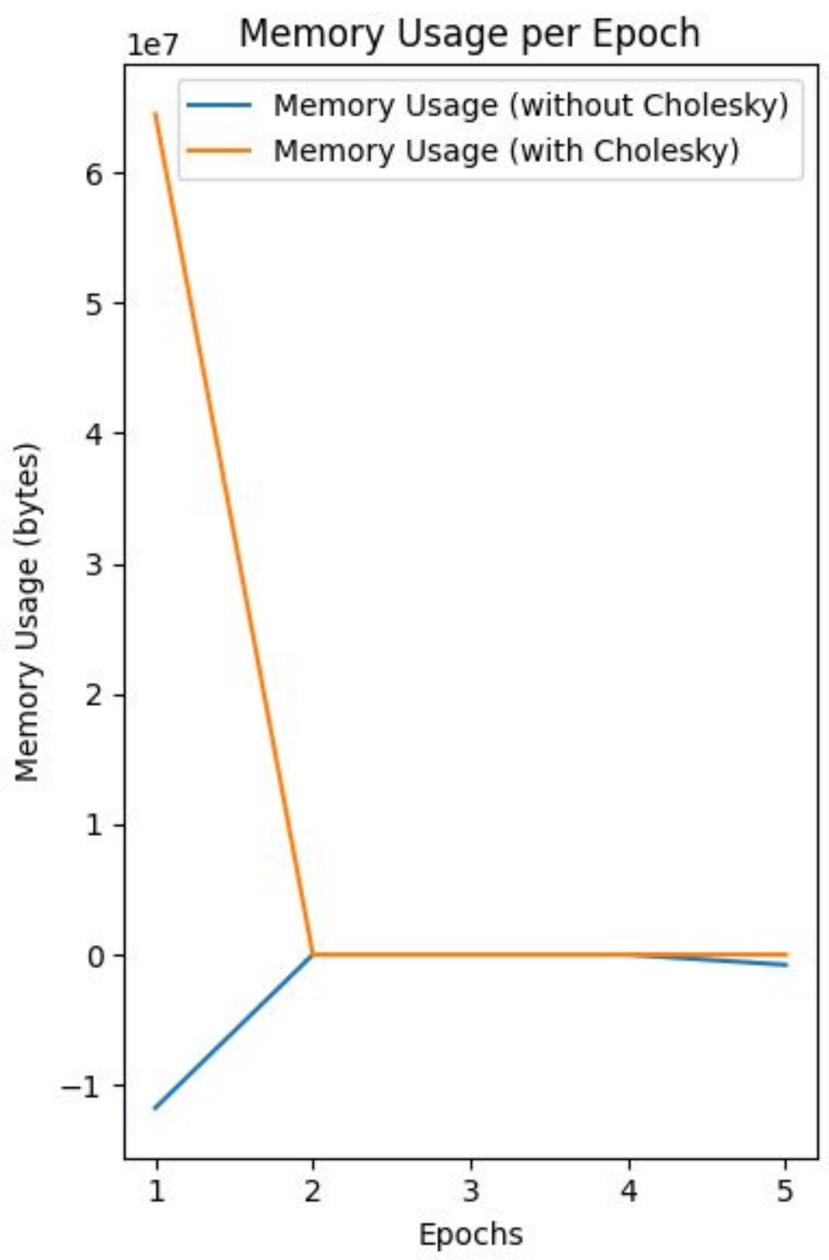
- **Dimensionality Reduction:** Reduces the complexity of the attention mechanism by approximating A\mathcal{A} with a lower-rank tensor, thereby decreasing the number of parameters and computational overhead.
- **Noise Reduction:** Retains only the most significant components that capture the majority of the variance in the data, effectively filtering out noise and redundant information.
- **Efficiency:** Low-rank approximations lead to faster computations and lower memory usage, which is particularly beneficial for large-scale models and datasets.

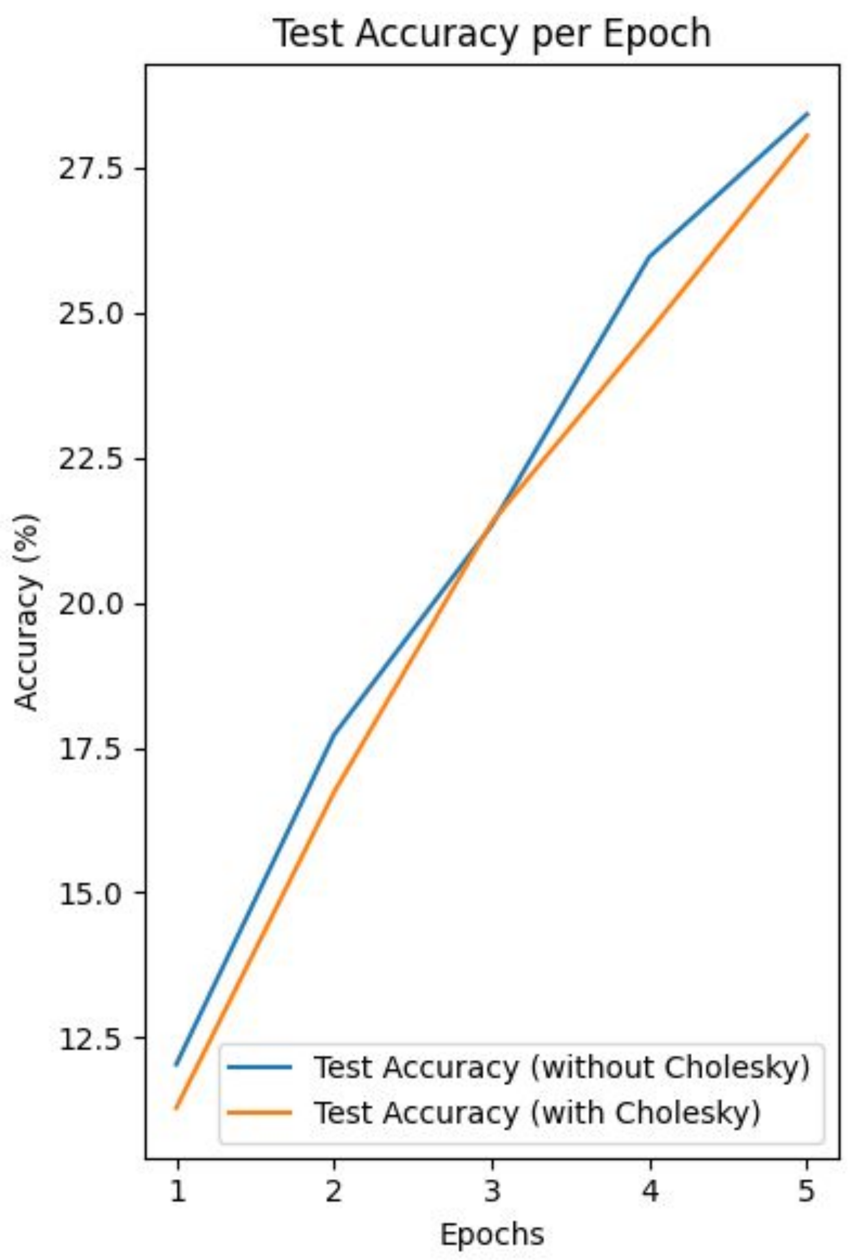


Skoltech

Methods validation. CIFIR100 - Cholesky



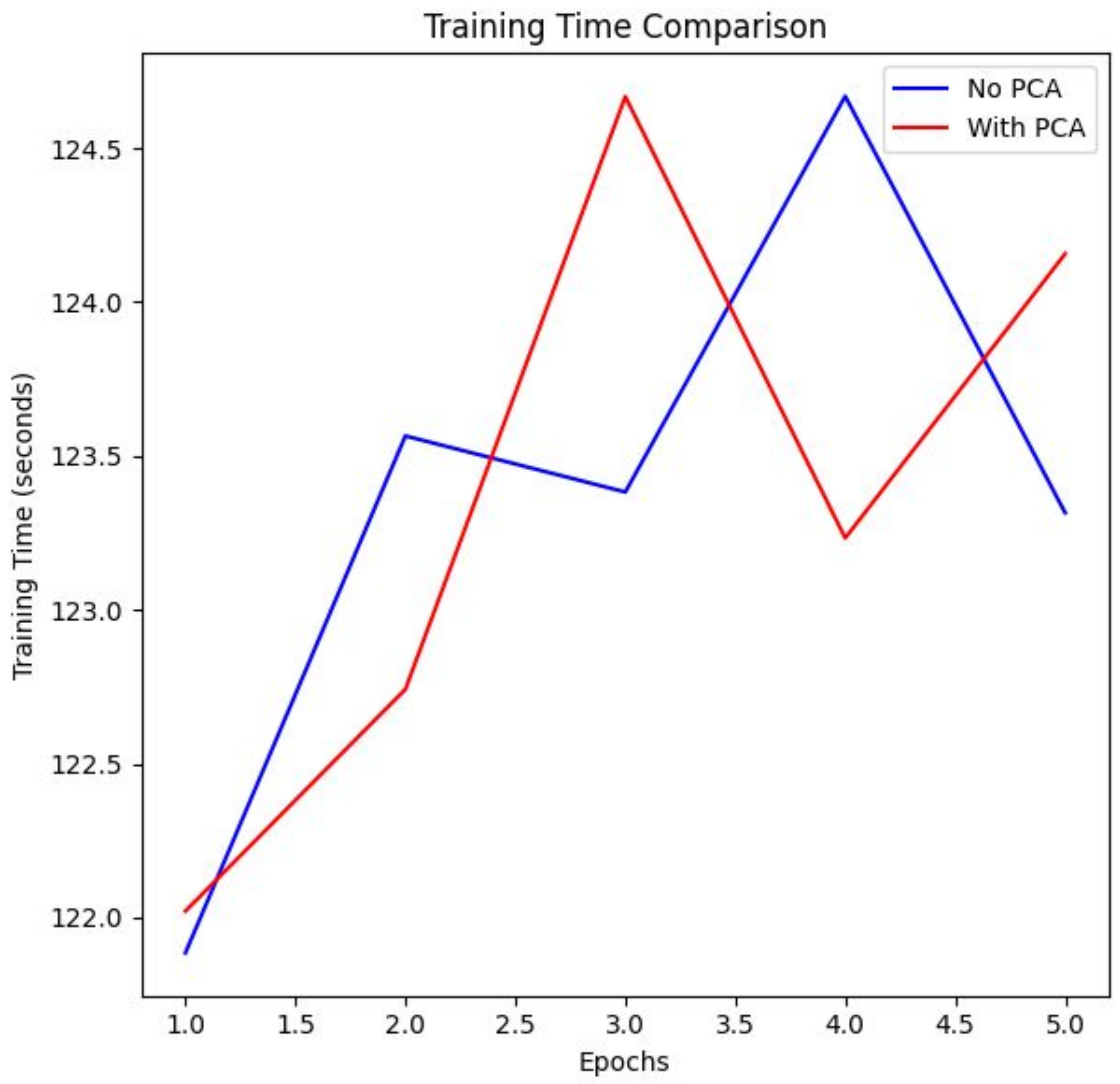


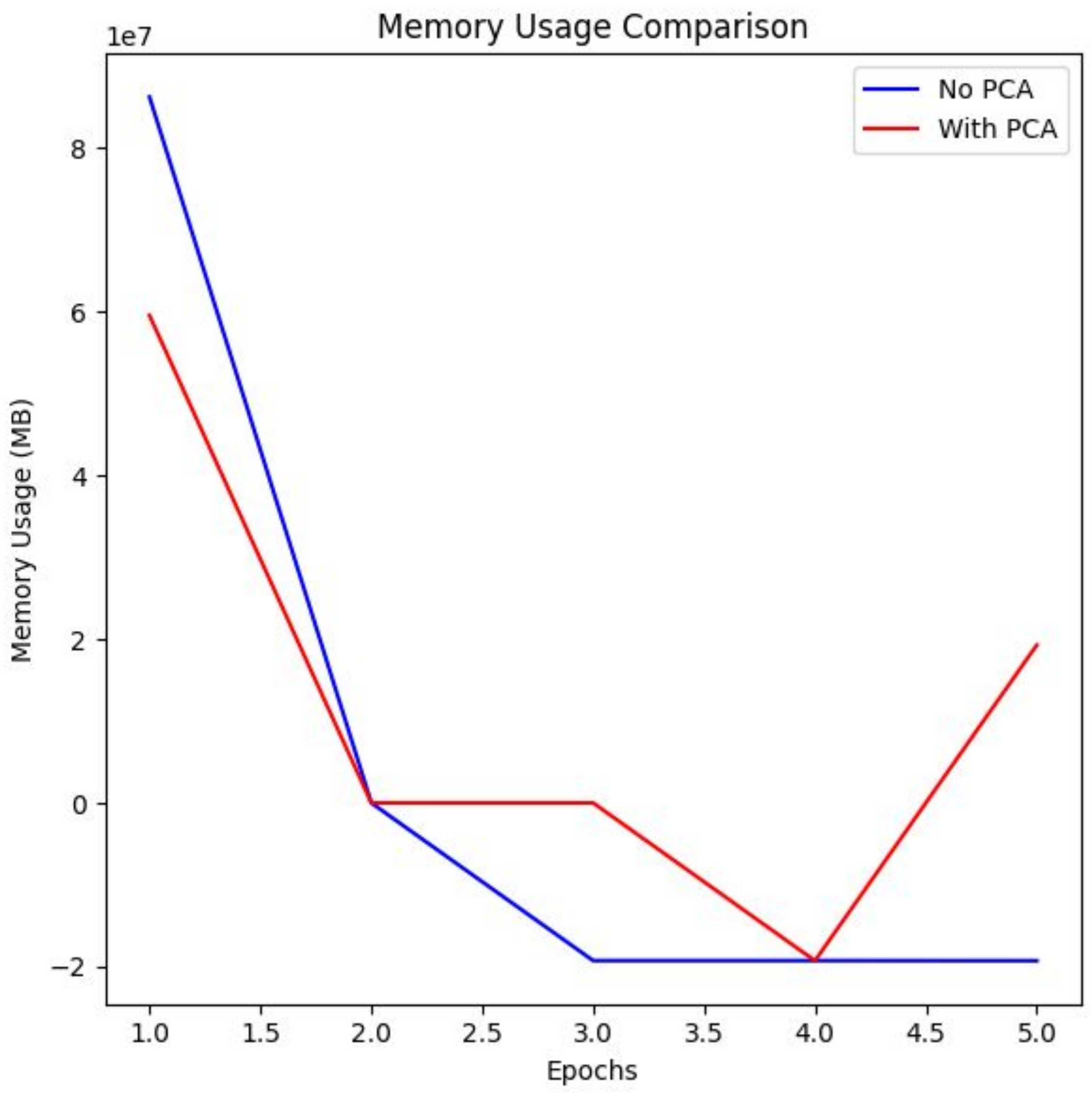


Total Time (without Cholesky): 130.72 seconds
Total Memory (without Cholesky): -12548096 bytes
Average Accuracy (without Cholesky): 21.09%
Total Time (with Cholesky): 132.30 seconds
Total Memory (with Cholesky): 64459776 bytes
Average Accuracy (with Cholesky): 20.42%

Implementing Cholesky decomposition in the transformer model resulted in marginally **increased** training time and significantly **higher** memory usage. Additionally, there was a slight **decrease** in average test accuracy. These outcomes indicate that the Cholesky-based approach **did not enhance** the model's performance and instead introduced additional computational and memory overhead. Overall, the application of Cholesky decomposition in this scenario did not provide any benefits and may have negatively impacted the model's efficiency and effectiveness.

Methods validation. CIFIR100 - PCA



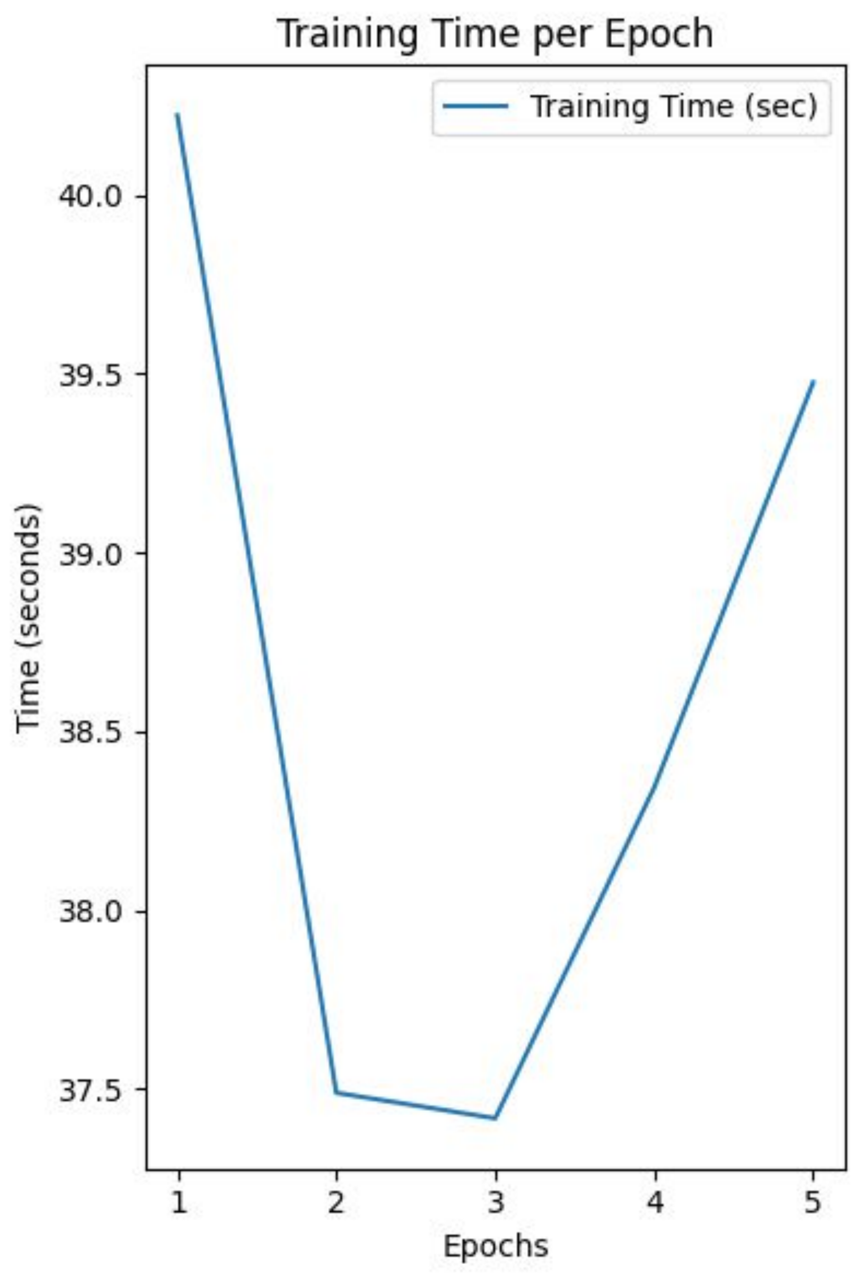


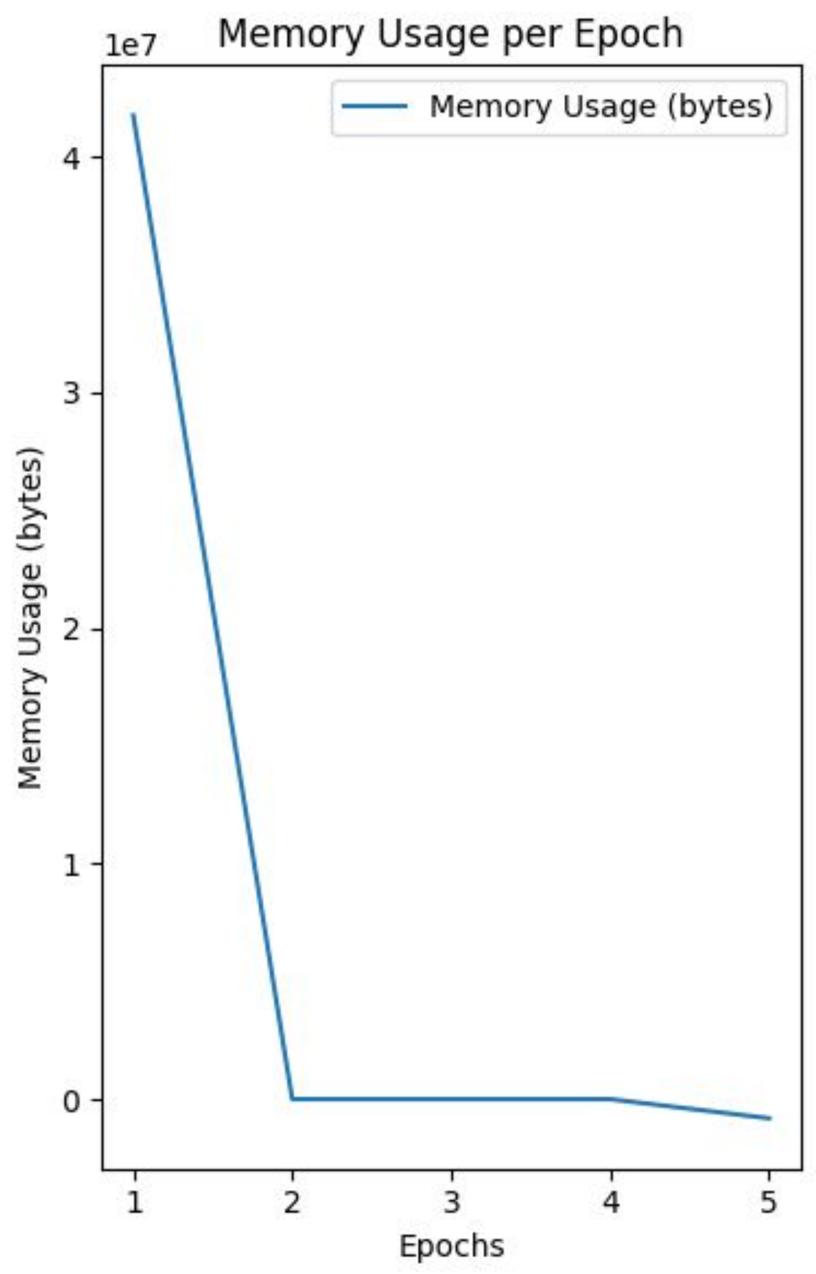
Average Training Time (without PCA): 123.36 seconds/epoch
Average Memory Usage (without PCA): 5682892.80 MB/epoch
Average Test Accuracy (without PCA): 14.81%
Average Training Time (with PCA): 123.36 seconds/epoch
Average Memory Usage (with PCA): 11909017.60 MB/epoch
Average Test Accuracy (with PCA): 14.63%

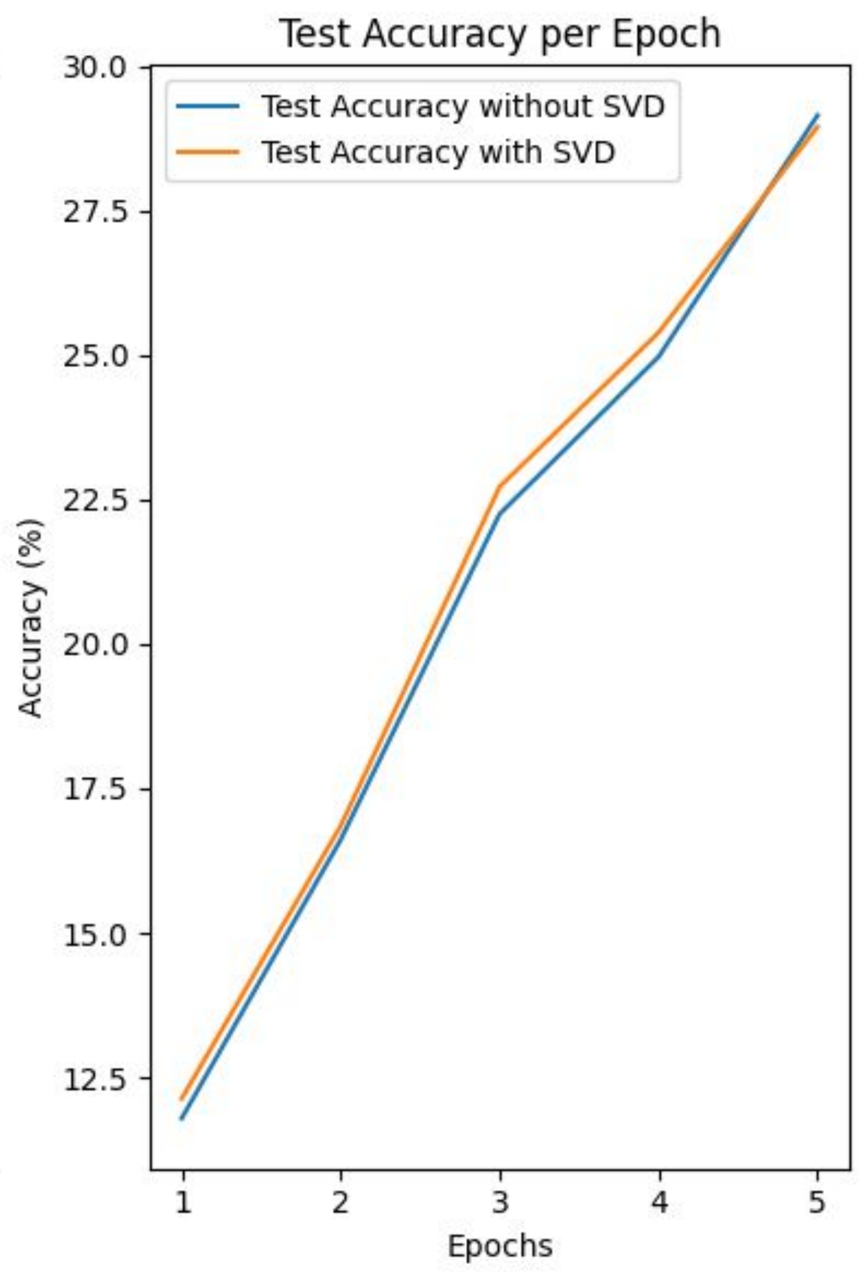
Using PCA in the transformer model did not yield the desired benefits. The training time remained **unchanged**, indicating that PCA did not enhance computational efficiency. Additionally, memory usage more than **doubled**, suggesting that the PCA implementation introduced significant overhead rather than reducing resource consumption. Furthermore, there was a slight **decline** in test accuracy, implying that the PCA-based low-rank approximation may have adversely affected the model's performance. Overall, the application of PCA in this scenario **did not improve** efficiency or accuracy and instead led to increased memory demands.

Skoltech

Methods validation. CIFIR100 - SVD





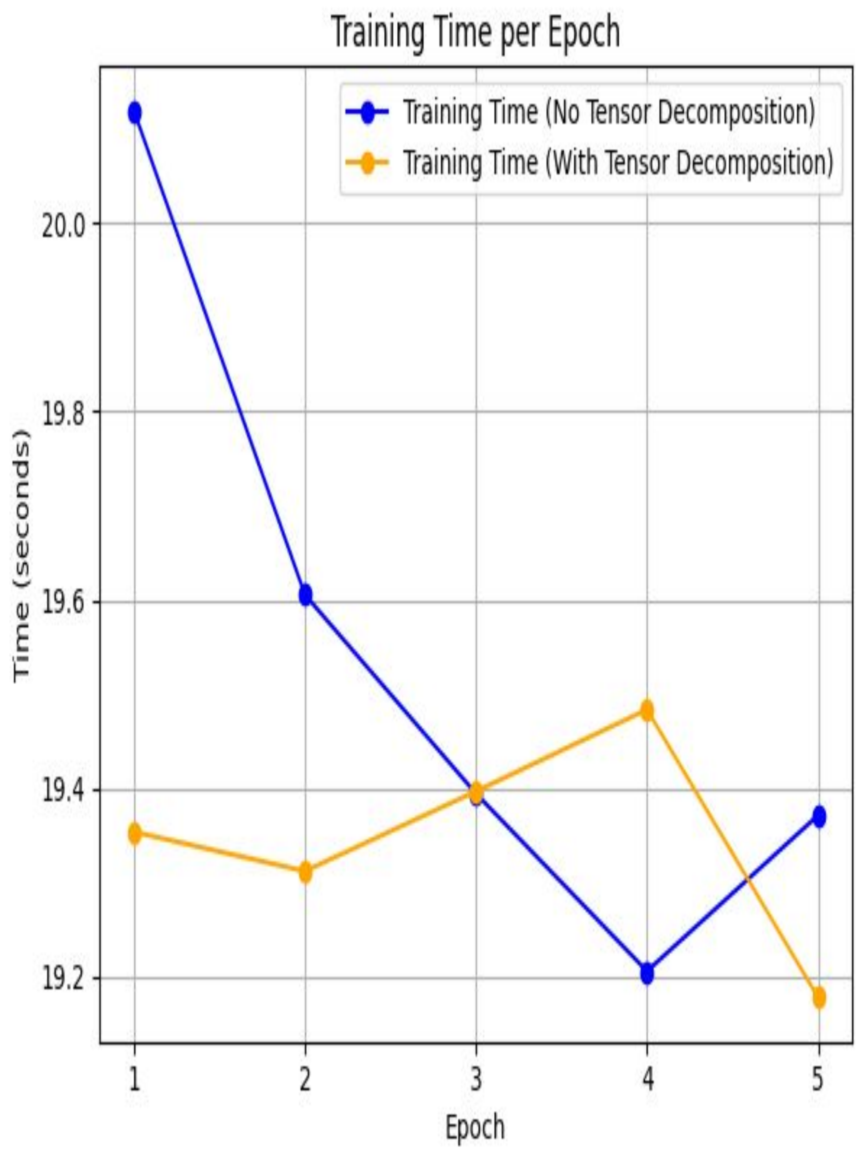


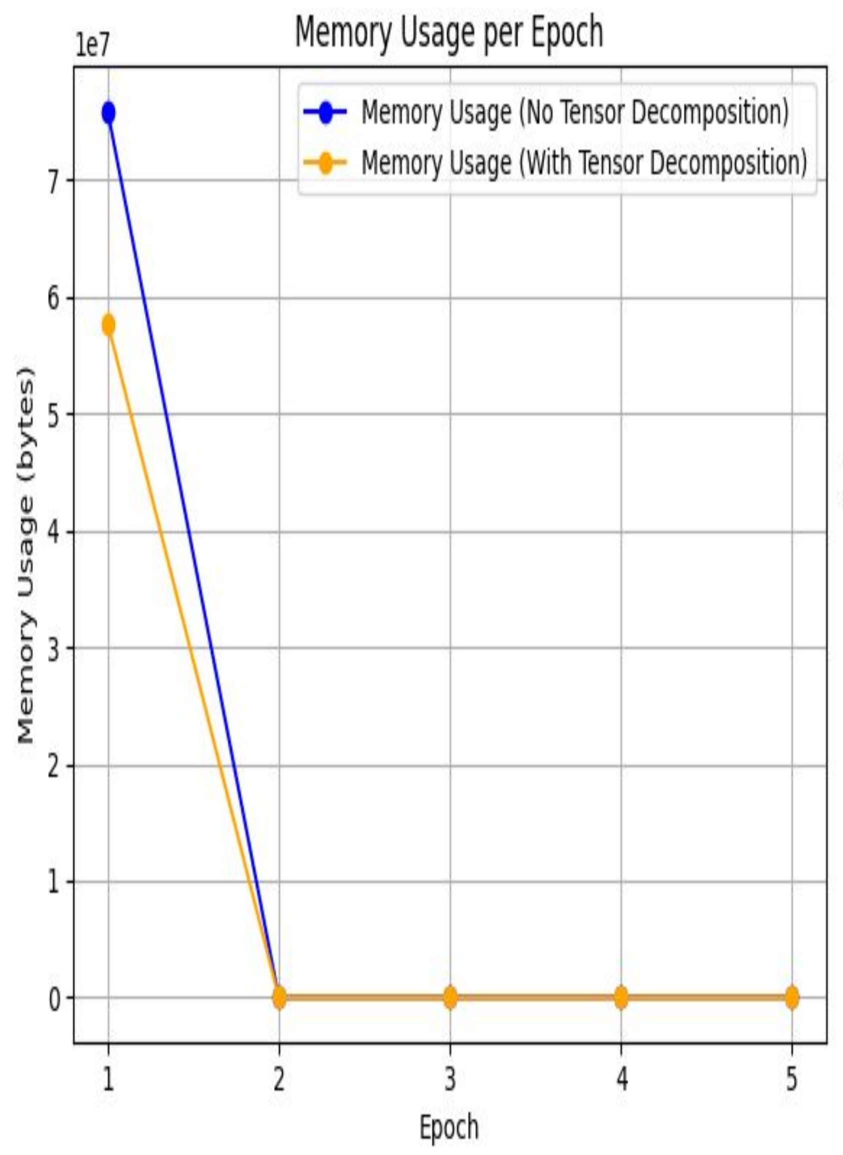
Total Time (without SVD): 192.95 seconds
Total Memory (without SVD): 40953856 bytes
Average Accuracy (without SVD): 20.95%
Total Time (with SVD): 192.95 seconds
Total Memory (with SVD): 40953856 bytes
Average Accuracy (with SVD): 21.21%

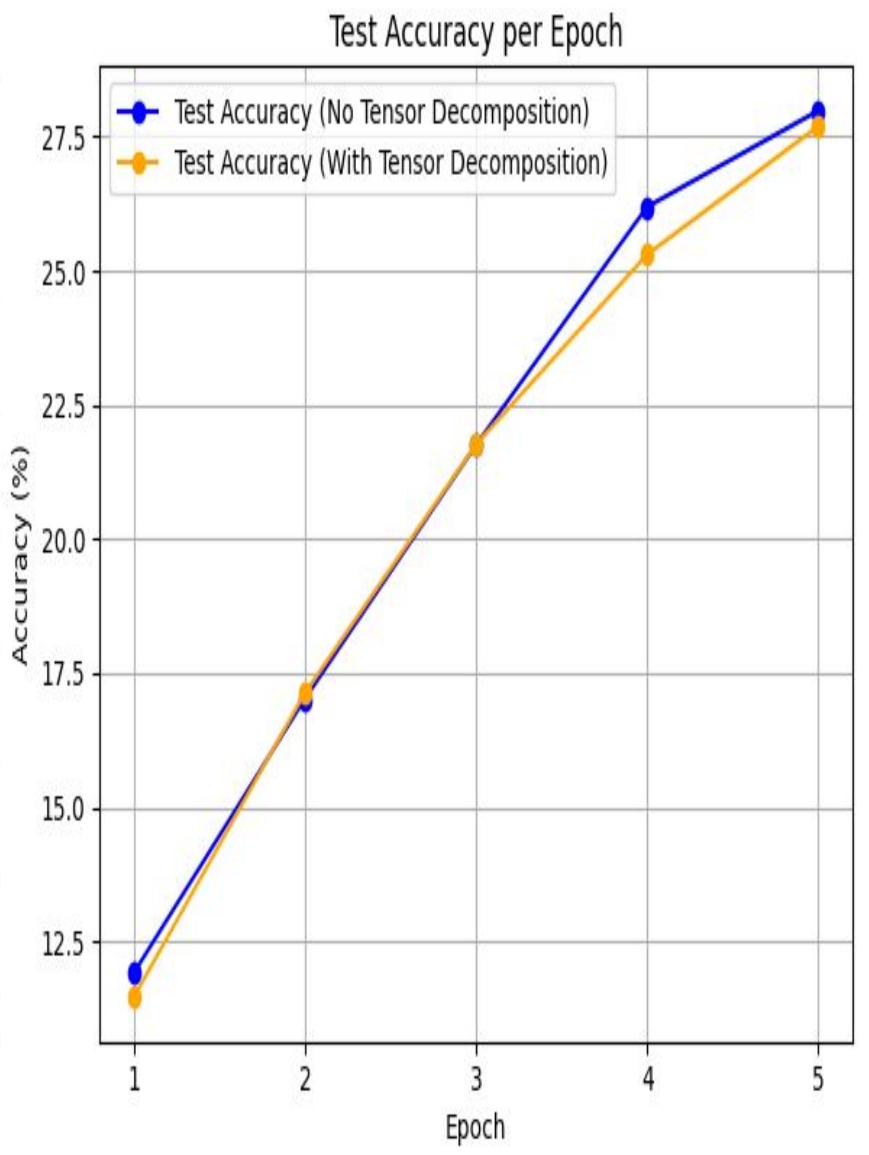
Implementing SVD in the transformer model maintained the training time and memory usage, indicating that it **did not introduce** additional computational or memory overhead. Importantly, there was a **slight improvement** in average test accuracy, suggesting that the SVD-based low-rank approximation may have enhanced the model's performance. Overall, SVD proved to be an **effective** method for achieving modest accuracy gains without compromising efficiency.

Skoltech

Methods validation. CIFIR100 - Tensor decomposition







Average Training Time (without PCA): 123.36 seconds/epoch
Average Memory Usage (without PCA): 5682892.80 MB/epoch
Average Test Accuracy (without PCA): 14.81%
Average Training Time (with PCA): 123.36 seconds/epoch
Average Memory Usage (with PCA): 11909017.60 MB/epoch
Average Test Accuracy (with PCA): 14.63%

Implementing Tensor Decomposition in the transformer model resulted in a slight **reduction** in training time and a notable **decrease** in memory usage, demonstrating improved resource efficiency. However, this optimization came with a minor **decline** in average test accuracy. Overall, Tensor Decomposition offers benefits in terms of memory conservation without adversely affecting training speed, but it may slightly compromise the model's predictive performance. This trade-off suggests that Tensor Decomposition can be a viable option when memory resources are limited, provided that the slight decrease in accuracy is acceptable for the specific application.

Skoltech

RESULTS

1. On the CIFAR-10 and CIFAR-100, the use of SVD shows a gradual improvement in accuracy compared to without SVD, however, the improvement is quite limited, especially on the CIFAR-100, where the difference between models with and without SVD is small. On SVD, SVD significantly improves accuracy throughout all epochs, especially in the early stages of learning. While on FOOD 101 and Imagenet, the improvement using SVD turned out to be less noticeable, perhaps due to the complexity of the data and the scale of the models.

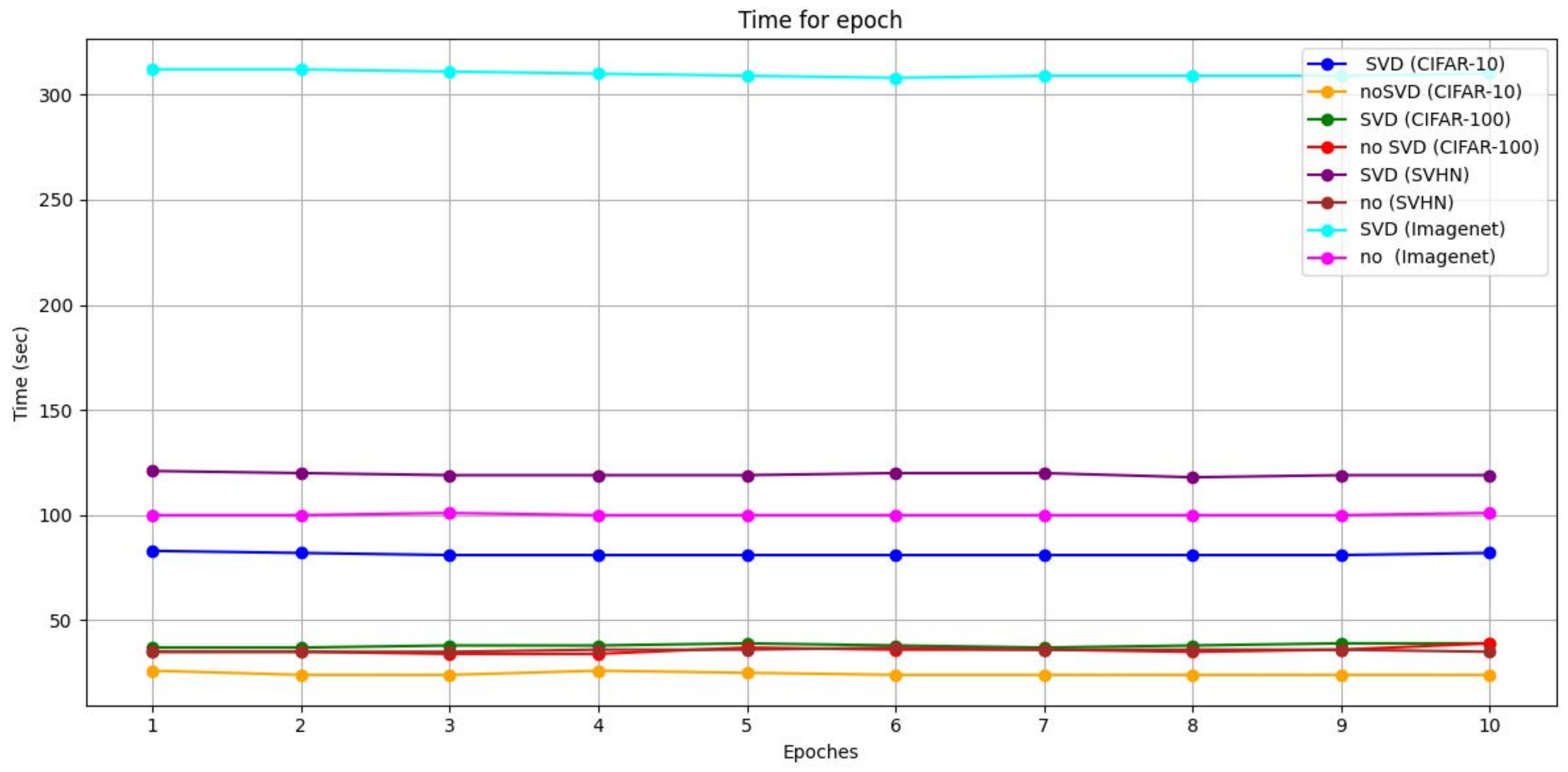
2. Based on the training data across various datasets (FOOD 101, CIFAR 10, SVHN, CIFAR 100, and ImageNet), we observe that the training times for both "with PCA" and "no PCA" approaches are quite comparable, with slight variations depending on the dataset. PCA (Principal Component Analysis) seems to have a marginal impact on the training speed, generally leading to slightly faster epochs in most cases. However, the training speed differences are not drastic, suggesting that PCA does not offer a significant boost in processing speed for these particular datasets, at least in terms of the raw training time for each epoch.

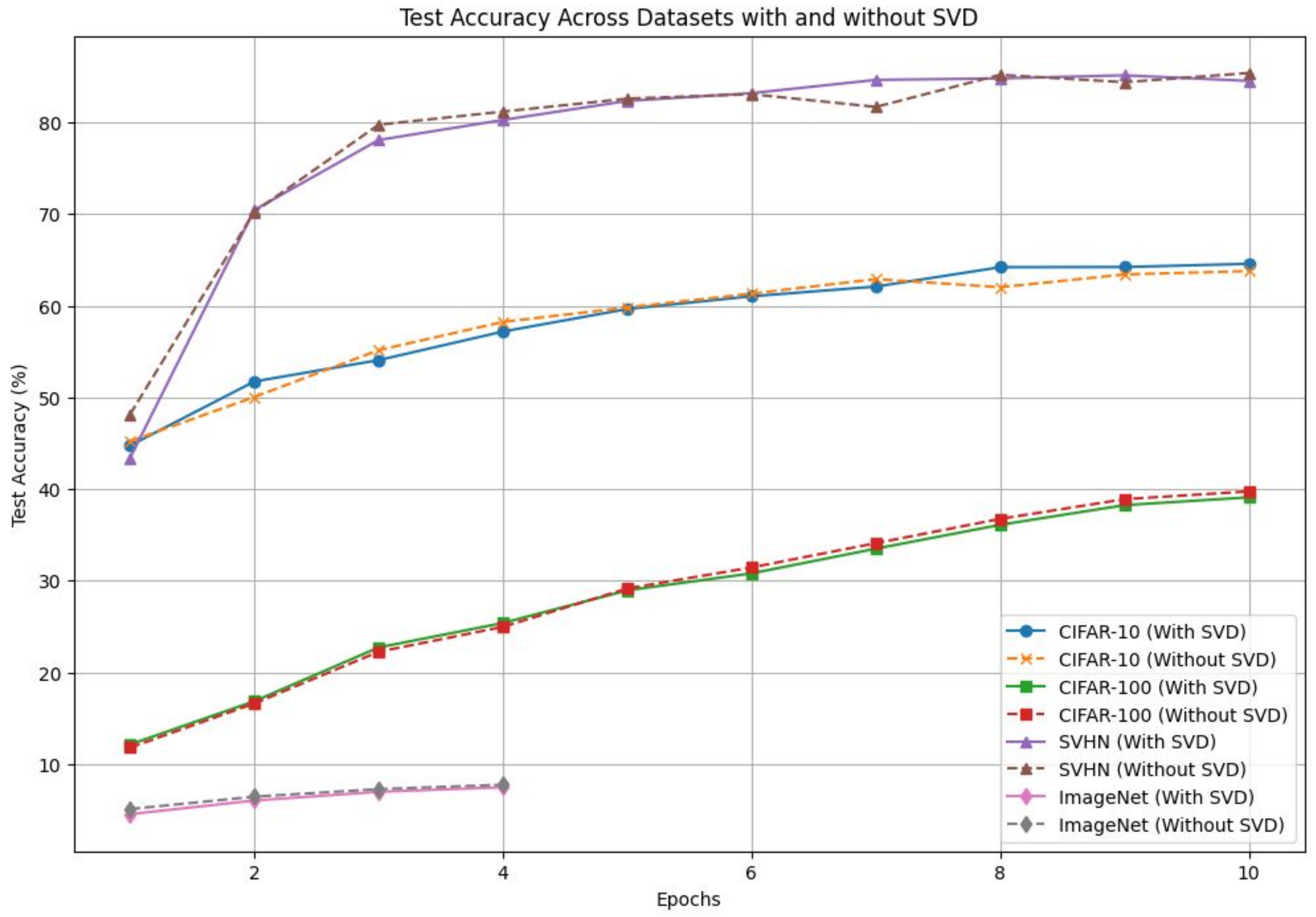
3. The use of tensor decomposition on various datasets (CIFAR-10, SVEN, ImageNet, Food-101) did not lead to significant improvements in either learning speed or model accuracy. Although decomposition sometimes accelerated learning, these improvements were minimal and had no noticeable effect on results.

4. For the CIFAR-100 set, the result of testing models without and using Cholesky showed only minor improvements. For example, in the first epoch, the accuracy of testing without Cholesky was 12.04%, and with it — 11.29%, with a gradual increase in the following epochs, where in the fifth epoch the accuracy without Cholesky reached 28.42%, and with Cholesky — 28.05%. The results were better on CIFAR-10, with test accuracy reaching 65.43% without Cholesky and 64.84% with Khaletsky on the tenth epoch. More obvious improvements could be seen on the SVHN set: accuracy without Cholesky in the first epoch was 44.74%, and with Cholesky — 49.42%, with subsequent improvements throughout the training. On ImageNet, the accuracy was significantly lower, with improvements, but still remained at 11%.

REFERENCES

1. BioMed Central. (2024). *Advantages of transformer and its application for medical image segmentation*. Biomedical Engineering Online.
2. https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/s12938-024-01212-4
3. PMC. (2023). *Comparison of Vision Transformers and Convolutional Neural Networks*. PubMed Central. https://pmc.ncbi.nlm.nih.gov/articles/PMC11393140/
4. Zhang, Y., & Wu, J. (2024). *A Review of Transformer-Based Models for Computer Vision Tasks*. arXiv. https://arxiv.org/html/2408.15178v1
5. PMC. (2023). *Transforming medical imaging with Transformers? A comparative review*. PubMed Central. https://pmc.ncbi.nlm.nih.gov/articles/PMC10010286/
6. Zhang, Y., & Li, X. (2023). *Medical Image Classification with a Hybrid SSM Model Based on Transformer*. MDPI. https://www.mdpi.com/2079-9292/13/15/3094
7. BioMed Central. (2024). *Application of visual transformer in renal image analysis*. Biomedical Engineering Online. https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/s12938-024-01209-z
8. Zhang, Y., & Li, Y. (2024). *Transformer-Based Visual Segmentation: A Survey*. arXiv. https://arxiv.org/abs/2304.09854
9. He, L., & Wang, Z. (2023). *ConvTransSeg: A Multi-resolution Convolution-Transformer Network for Medical Image Segmentation*. arXiv. https://arxiv.org/abs/2210.07072
10. Wu, X., & Zhang, S. (2023). *TransDeepLab: Convolution-Free Transformer-based DeepLab v3+ for Medical Image Segmentation*. arXiv. https://arxiv.org/abs/2208.00713
11. Li, H., & Li, Z. (2020). *Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers*. arXiv. https://arxiv.org/abs/2012.15840
12. Liu, J., & Wang, X. (2024). *Transformer-based Models for Image Classification and Segmentation: A Comprehensive Review*. IEEE Access, 12, 36587-36607. https://doi.org/10.1109/ACCESS.2024.3158992
13. Yu, C., Chen, T., Gan, Z., & Fan, J. (2024). Boost Vision Transformer with GPU-Friendly Sparsity and Quantization. *Academy for Engineering and Technology, Fudan University*. https://doi.org/10.1109/ACCESS.2024.3156890


Time for epoch
SVD (CIFAR-10)
noSVD (CIFAR-10)
SVD (CIFAR-100)
no SVD (CIFAR-100)
SVD (SVHN)
no (SVHN)
SVD (Imagenet)
no (Imagenet)
Time (sec)
Epoches


Test Accuracy Across Datasets with and without SVD
CIFAR-10 (With SVD)
CIFAR-10 (Without SVD)
CIFAR-100 (With SVD)
CIFAR-100 (Without SVD)
SVHN (With SVD)
SVHN (Without SVD)
ImageNet (With SVD)
ImageNet (Without SVD)
Test Accuracy (%)
Epochs