FFT-BASED DYNAMIC SUBSPACE SELECTION FOR LOW-RANK ADAPTIVE OPTIMIZATION OF LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank optimization has emerged as a promising direction in training large language models (LLMs) to improve running time and reduce the memory usage of adaptive optimizers by constraining learning to a lower-dimensional space. Prior work typically projects gradients of linear layers using approaches based on Singular Value Decomposition (SVD) or QR-decomposition. Applying these techniques individually to each layer in large models is computationally expensive and incurs additional memory costs due to storing the projection matrices. In this work, we propose a computationally efficient and conceptually simple, two-step procedure to approximate SVD/QR-based gradient projections into lower-dimensional spaces by using a predefined orthogonal matrix of the Discrete Cosine Transform (DCT). We dynamically select columns from the DCT matrix based on their alignment with the gradient of each layer. The effective projection matrices are obtained via a simple `matmul` with the DCT matrix in $O(n^3)$ time, followed by a lightweight sorting step to identify the most relevant basis vectors. For large layers, DCT can be computed via `Makhoul`'s N-point algorithm based on Fast Fourier Transform (FFT) in $O(n^2 \log(n))$ time. Due to the predefined nature of the orthogonal bases, they are computed once at the start of training. Our numerical experiments on both pre-training and fine-tuning tasks demonstrate the effectiveness of our dual strategy in approximating optimal low-rank projections, obtaining an approach with rank-independent running time that matches the performance of costly SVD/QR-based methods while achieving faster runtime and reduced memory usage by up to 25% across different model sizes.

1 INTRODUCTION

The Adam optimizer (Kingma & Ba, 2014) and its regularized version AdamW (Loshchilov & Hutter, 2019) have become the standard approach for optimizing deep neural networks in various settings. With the recent increase in scale of LLMs up to billions and trillions of parameters, training with AdamW becomes more and more challenging, as its internal state requires two momentum buffers that scale with the model size. This practical problem paved the way for a line of research that focuses on reducing the memory usage of optimizer states in the context of adaptive gradient optimization. These approaches range from quantizing the states to 8 bits (Dettmers et al., 2021) to the recent GaLore optimizer (Zhao et al., 2024) inspired from the LoRA techniques (Hu et al., 2021; Lialin et al., 2023), that compresses the gradient matrix using low-rank decomposition based on SVD. Several improvements to GaLore have been proposed to enhance its performance, such as LDAdam (Robert et al., 2025), FRUGAL (Zmushko et al., 2024), FIRA (Chen et al., 2024), BAdam (Luo et al., 2024), Q-GaLore (Zhang et al., 2024). The key aspect of GaLore and its later improvements is the low-rank decomposition based on matrix factorization, such as SVD or QR decomposition.

Recently, the Muon (Jordan et al., 2024) optimizer sparked the community's attention due to the faster convergence in pretraining settings, achieved by orthogonalizing the momentum matrix using Newton-Schulz, an iterative procedure difficult to parallelize for large-scale pretraining runs *as the full-sized matrices must be materialized* on GPU. The Dion optimizer (Ahn et al., 2025) aims to reduce this overhead by employing low-rank, orthogonal updates. However, it requires storing a projection matrix for each layer and uses QR-decomposition to orthogonalize the low-rank components, making its running time dependent on the rank.

All these SVD/QR-based techniques are known to be computationally intensive as they have to be invoked for each linear layer, either at each step (for, e.g., LDAdam, Muon) or once at a few steps (for, e.g., GaLore). To mitigate the high computational and memory costs of these procedures, we ask: *can we find an alternative low-rank projection approach to serve as an accurate replacement for the orthogonal matrices in SVD/QR for low-rank compression of optimizer states, that is much cheaper to compute, and portable across memory-efficient optimizers?*

Contributions. In this work, we address our research question by proposing a cheaper alternative to the orthogonal matrices computed via SVD/QR. Concretely, we propose using the orthogonal matrices from the general class of Discrete Fourier Transforms, such as the Discrete Cosine Transform (DCT), which has been successfully used in image compression for the JPEG algorithm. To the best of our knowledge, we are the first to use it in the context of low-rank adaptive gradient optimization. We summarize our contributions as follows:

- We propose a dynamic column selection approach to adaptively choose columns from a fixed orthogonal matrix to compute a low-rank projection of the input matrix. The effective projection matrix is obtained from the fixed orthogonal matrix by indexing the columns (Section 2.1);
- We motivate that DCT is a good candidate for our dynamic column selection approach due to the reduced time complexity to compute the alignments with the input matrix via the Makhoul's N-point algorithm (Makhoul, 1980) to compute a fast DCT with $O(n^2 \log(n))$ time complexity compared to $O(n^3)$ incurred by a basic matrix multiplication. This can yield speedups of up to $8 - 50\times$ in computing the alignments for large layers (Appendix C).
- We theoretically justify our dynamic column selection approach to compute the most significant columns of any fixed orthogonal matrix (including the DCT-based) to obtain a projection matrix tailored to each layer (Section 4);
- We show the DCT-based projection is a fast and accurate replacement for the orthogonal matrices computed via SVD/QR for the Muon- and Adam-like optimizers (such as Dion, FRUGAL, FIRA, LDAdamW) in the context of low-rank compression of optimizer states. We reduce the running time and improve the memory usage as we store only one DCT matrix per GPU for the entire network, computed once at the beginning of the training. In addition, we store only r (rank) integers, representing the indices of the most significant columns for each layer instead of storing one low-rank projection matrix.
- We propose two standalone optimizers that use the DCT-based dynamic column selection approach:
  1. **Trion** (Section 2.3), which improves Dion by replacing the Power-Iteration with our DCT-based dynamic column selection approach to compute a low-rank representation of the momentum buffer, followed by orthogonalization via Newton-Schulz iteration. We provide a DDP-compatible implementation that communicates orthogonal, low-rank momentum across devices and compute the final layer updates locally using the projection matrix obtained from DCT. To the best of our knowledge, we are the first ones to reduce the complexity of Newton-Schulz using a low-rank approximation of momentum.
  2. **DCT-AdamW** (Section 2.4), which replaces the SVD-based low-rank projections and optionally adds quantized error feedback for the projection error. Further, DCT-AdamW rotates the momentum buffers such that new low-rank gradients are correctly incorporated, and thus allows changing the low-rank subspace every step.

2 Method

This section presents our dynamic column selection approach that works with any orthogonal matrix, briefly introduces the Discrete Cosine Transform (DCT) and the motivation of using it and finally introduces the two algorithms we propose: Trion as an improvement to Dion to replace Power-Iteration and DCT-AdamW as an improvement to low-rank AdamW variants to replace SVD/QR.

2.1 Dynamic Column Selection

General View. Given an orthogonal matrix $Q \in \mathbb{R}^{n \times n}$, we want to compute a projection matrix $Q_r \in \mathbb{R}^{n \times r}$ by selecting r columns from Q to project the gradient $G \in \mathbb{R}^{n \times n}$ to an r-dimensional space $g = GQ_r \in \mathbb{R}^{n \times r}$. First, we compute the similarities matrix $S = GQ$ containing the scalar products between rows of G and columns of Q. We rank the columns of S based on their ℓ_1- or ℓ_2-norm and then pick the indices of the largest r columns according to the chosen norm, which we use to index columns in Q to obtain Q_r. The i^{th} column in matrix S contains the scalar products between each row of G and i^{th} column of Q, as detailed in Appendix B. We view these scalar products as similarities (or alignments) of rows in G with columns of Q and we want to choose the columns of Q with largest similarity with rows of G. Using this approach, we ensure a dynamic mechanism to choose the most appropriate columns of Q for the current gradient matrix G to minimize the projection error (see Section 4). Each layer will have its own set of r indices for the columns. The dynamic behavior comes from the n-choose-r possible sets of indices to select from.

The rule of thumb in the low-rank factorization methods for optimization we are targeting in this work is to compress the smallest dimension of a matrix to r dimensions (e.g. from $\mathbb{R}^{n \times m}$ to $\mathbb{R}^{n \times r}$ for $n \geq m$). Usually, the smallest dimension of the gradient matrix is d_{model}, the hidden (embedding) size of the model. As a result, the memory overhead of our dynamic column selection approach is the cost of storing only one orthogonal matrix $Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ (DCT in our case) per GPU for the entire model and r integers for the corresponding column indices from Q for each layer to create Q_r.

2.2 Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is widely used in the signal processing and data compression literature (e.g., JPEG algorithm for image compression) and consists of n orthogonal basis vectors whose components are cosines (Strang, 1999). There are minor variations of DCT and in this work we will use DCT-II/III. We denote the DCT-III matrix of order n by $Q \in \mathbb{R}^{n \times n}$, defined as $Q_{ij} = \sqrt{2/n} \cdot \cos \frac{i(2j+1)\pi}{2n}$, with $i, j \in [n]$, where the first row has to be divided by $\sqrt{2}$ in order for Q to be orthogonal, i.e. $Q^\top Q = I_n$. The DCT-II matrix is the transpose of DCT-III. In Appendix A, we provide details on how we can efficiently materialize the DCT matrix on GPU. Once we compute Q, we can use it in the dynamic column selection approach to efficiently compute low-rank projections of any two-dimensional matrix.

In Appendix C, we discuss why we choose DCT matrix. In short, the particular structure of DCT allows us to enable fast computation for the similarities matrix S in $O(n^2 \log(n))$ time using the Makhoul's N-point algorithm (see Appendix D) instead of $O(n^3)$ time for basic matmul.

2.3 Trion: DCT-based improvement to Dion

In this section, we present how we can apply the DCT-based dynamic column selection approach to compute low-rank projection of momentum in the context of Dion (Ahn et al., 2025) to obtain a faster and more accurate optimizer, which we call Trion, presented in Algorithm 1.

Dion uses Power-Iteration to compute a low-rank projection and orthogonalizes it using QR-decomposition, whose running time depends on the rank r. Instead, we propose replacing these techniques with a rank independent approach to compute the low-rank projection based on DCT matrix, which requires computing the similarity matrix S_t, followed by a top-r ranking to determine the indices of columns that best align with the momentum matrix, denoted by the set i_t. The matrix S_t represents the DCT of the momentum matrix and it can be computed using the Makhoul's algorithm or simply by only one matrix multiplication $S_t = B_t D_C$, where $B_t \in \mathbb{R}^{R \times C}$ is the momentum and $D_C \in \mathbb{R}^{C \times C}$ is the DCT matrix.

Once we determine the indices i_t of the most significant columns in D_C, we can extract the low-rank momentum b_t from the similarity matrix S_t, as well as the projection matrix Q_t by indexing D_C, which will be used in (a) computing the projection error Δ_t and (b) projecting the orthogonal low-rank momentum back to the higher dimensional space to update the model.

We would like to emphasize that we input the low-rank momentum $b_t \in \mathbb{R}^{R \times r}$ to Newton-Schulz and not the original momentum buffer $B_t \in \mathbb{R}^{R \times C}$, which significantly reduces the computational overhead. Moreover, we can use the efficient triton kernels provided in the Muon implementation from the official Dion repository[1] to speed up the computations even further for large ranks r, as Newton-Schulz will operate with $r \times r$ matrices.

Communication in Distributed Training. We develop the Trion optimizer on top of the published code for the Muon [2] optimizer that leverages the ZeRO (Rajbhandari et al., 2020) approach in Distributed Data Parallel (DDP) settings. Specifically, the model update O_t for a layer is computed only on one GPU, and the result is communicated to other GPUs using `all-gather`, drastically reducing the computation costs for large models (under the assumption that communication is cheaper than computation itself). Since we replicate the DCT matrix on each GPU, we would like to

[1] `github.com/microsoft/dion`
[2] `Keller Jordan's Muon implementation`

emphasize that we communicate only low-rank terms o_t from the source GPU (the one computing the update) to other devices and perform the step $O_t = o_t Q_t^\top$ locally on each device. This way, we do not communicate full size, orthogonal matrices O_t, but only low-rank versions o_t, thus reducing the overall iteration time.

This is particularly useful in the FSDP-compatible implementation, where we can materialize the full low-rank matrices o_t on each device using `all-to-all` (see this Muon implementation [3]) to perform Newton-Schulz on a low-rank input, followed by a resharding. After resharding, each GPU can compute the individual slice of O_t and update its own shard. However, the FSDP implementation requires deciding how each layer should be sharded based on whether that particular layer requires a left- or right-projection in order to avoid unwanted calls to communication primitives to materialize full tensors to compute the alignments S.

Algorithm 1 Trion Optimizer

```
1:  Define the DCT-II/III matrix $D_C \in \mathbb{R}^{C\times C}$
2:  for $t = \{1, 2, \dots, T\}$ do
3:      $G_t = \nabla_\theta L(\theta_t) \in \mathbb{R}^{R\times C}$
4:      $B_t = M_{t-1} + G_t \in \mathbb{R}^{R\times C}$
5:      $S_t = \textsc{Makhoul}(B_t) \in \mathbb{R}^{R\times C}$      ▷ or $S_t = B_t \cdot D_C$ to compute similarities;
6:      $i_t = \textsc{DynamicColumnSelection}(S, r) \in \mathbb{N}^r$ ▷ select largest $r$ columns by $\ell_1/\ell_2$-norm
7:      $Q_t = D_C[:, i_t] \in \mathbb{R}^{C\times r}$      ▷ projection matrix containing most significant $r$ columns
8:      $b_t = S_t[:, i_t] \in \mathbb{R}^{R\times r}$      ▷ extract low-rank momentum from the similarity matrix $S$
9:      $\Delta_t = B_t - b_t Q_t^\top$      ▷ the error is $b_t Q_t^\top$ and replaces $P_t R_t^\top$ error from Dion
10:     $M_t = \mu B_t + (1-\mu)\Delta_t = B_t - (1-\mu) b_t Q_t^\top$      ▷ update momentum with error feedback
11:     $o_t = \textsc{NewtonSchulz}(b_t) \in \mathbb{R}^{R\times r}$      ▷ Newton-Schulz applied on low-rank $b_t$
12:     $O_t = o_t Q_t^\top \in \mathbb{R}^{R\times C}$      ▷ project orthogonalized low-rank momentum to original size
13:     $\theta_{t+1} = (1 - \lambda\eta_t)\theta_t - \eta_t \ \max(1, \sqrt{R/C})\ O_t \in \mathbb{R}^{R\times C}$
14: end for
```

2.4 DCT-AdamW

We propose DCT-AdamW, a standalone low-rank version of AdamW with DCT-based projection that has the option to use quantized error feedback (EF) (Seide et al., 2014; Karimireddy et al., 2019; Alistarh et al., 2018) and ensures momentum buffers integrate gradients from the same lower dimensional subspaces in a similar way as in LDAdamW (Robert et al., 2025). In contrast to LDAdamW, which has to store two consecutive projection matrices per layer, we only have to store two sets of r indices per layer. Prior work MicroAdam (Modoranu et al., 2024) quantized the error feedback down to 4-bits in the context of compressing the optimizer state using sparsity. In our setup, the lowest resolution we can quantize EF to is 8-bits without degrading the optimizer performance. For space constraints reasons, we present the pseudocode of our DCT-AdamW optimizer in Appendix E.

3 Experiments

In this section we present our numerical results. Our main goal is to show that our DCT-based dynamic column selection approach at least recovers the performance of the original algorithms which we integrate it in. Specifically, we evaluate Trion against Dion, where we directly compare the DCT-based projection followed by Newton-Schulz iteration with QR-based Power-Iteration. In addition, we replace the SVD-based projection in FRUGAL and FIRA optimizers with our DCT approach. In the end, we compare LDAdamW with our standalone DCT-AdamW optimizer. In our pretraining experiments, we compare training/validation perplexity and for fine-tuning we compare the evaluation accuracy, as well as memory usage and running time for both scenarios. In the remaining part of the paper, we focus on presenting our pretraining (PT) results and the fine-tuning (FT) results are presented in Appendix H.

We train from scratch models from the Llama family with 350M, 800M and 1.3B parameters using Chinchilla-optimal token counts (20 tokens per parameter) from the C4 (Raffel et al., 2020) dataset and sequence length 512. All PT experiments are run in Distributed Data Parallel (DDP) settings on 8x H100 Nvidia GPUs using global batch size 512 with local batch size 64 per GPU (unless otherwise specified explicitly).

[3] github.com/microsoft/dion/blob/main/dion/muon.py#L22

PT with Trion. Our purpose is to show that our DCT-based dynamic column selection approach can successfully replace the QR-based Power-Iteration procedure in Dion and at least recovers the performance for the best hyper-parameters reported by the original work. In Table 1, we present our results, which are obtained using the optimal learning rate $\eta = 0.01$ (as reported by the original Dion paper) and weight decay $\lambda = 0.01$. We report the average values across 3 seeds for train/validation loss and perplexity, maximum allocated memory (read directly from the GPU) and the lowest running time across all runs. We experiment with ranks $128, 256$ and 512, the rank-to-dimension ratio is $r/d \in \{1/16, 1/8, 1/4, 1/2\}$, equivalent to $6.25\%, 12.5\%, 25\%$ and 50% of the full rank d, where d is model's embedding dimension.

Rank r	Metric	350M ($d = 1024$)		800M ($d = 2048$)		1.3B* ($d = 2048$)	
		Trion	Dion	Trion	Dion	Trion	Dion
128	r/d	1/8		1/16		1/16	
	Train Loss	**2.726**	2.764	**2.532**	2.555	**2.475**	2.503
	Train PPL	**15.29**	15.89	**12.59**	12.89	**11.90**	12.24
	Val Loss	**2.736**	2.773	**2.505**	2.527	**2.422**	2.448
	Val PPL	**15.43**	16.00	**12.25**	12.52	**11.27**	11.56
	Memory (GB)	**42.42**	45.56	**67.45**	71.64	**63.62**	68.57
	Runtime	**1h 53m**	1h 58m	**8h 5m**	8h 24m	**19h 13m**	19h 42m
256	r/d	1/4		1/8		1/8	
	Train Loss	**2.715**	2.741	**2.533**	2.546	**2.476**	2.492
	Train PPL	**15.11**	15.51	**12.61**	12.78	**11.92**	12.11
	Val Loss	**2.727**	2.750	**2.503**	2.519	**2.423**	2.440
	Val PPL	**15.30**	15.64	**12.22**	12.42	**11.28**	11.47
	Memory (GB)	**42.42**	45.59	**67.45**	71.75	**63.62**	68.58
	Runtime	**1h 53m**	2h 3m	**8h 3m**	8h 36m	**19h 13m**	20h 6m
512	r/d	1/2		1/4		1/4	
	Train Loss	**2.708**	2.718	**2.528**	2.535	**2.470**	2.481
	Train PPL	**15.01**	15.16	**12.54**	12.63	**11.84**	11.97
	Val Loss	**2.718**	2.727	**2.499**	2.509	**2.420**	2.430
	Val PPL	**15.15**	15.29	**12.18**	12.30	**11.25**	11.36
	Memory (GB)	**42.42**	45.99	**67.45**	71.81	**63.62**	68.73
	Runtime	**1h 54m**	2h 14m	**8h 3m**	8h 48m	**19h 13m**	20h 44m

Table 1: Perplexity, Memory, and Running Time Comparison for Trion and Dion. d stands for the embedding dimensionality of the model. The 1.3B model was trained with local batch size 32.

Performance In Table 1, we show Trion consistently achieves lower training and validation loss, which also translates to lower perplexities. At the top row of Figure 3 we show the training loss curve of Trion is lower than Dion across iterations. This is an indication that our DCT-based dynamic column selection algorithm, followed by orthogonalization via Newton-Schulz can successfully replace the Power-Iteration procedure in Dion.

Memory Usage. In Table 1, we show Trion has a lower memory requirement across all experiments since it allocates only one DCT matrix per GPU of size $d \times d$, from which we select r columns using the indices of the most significant columns. In contrast, Dion stores a projection matrix for each layer. This design difference translates to around 10% lower memory footprint for Trion compared to Dion.

Runtime. We would like to emphasize that the running time of Trion does not depend on rank, as can be seen from the reported runtime across different ranks in Table 1: Trion achieves nearly constant runtime for each model size across all ranks, while the runtime of Dion clearly depends on the rank because of the QR-decomposition. This represents an advantage of Trion for much larger scales compared to Dion. At the bottom row of Figure 3 we present the wall clock time for the two optimizers for rank 256 across all three models we tested on. Concretely, for a fixed running time budget, Trion consistently achieves lower training loss than Dion. Trion is faster than Dion by $2.5 - 4.5\%$ for rank 128, $4.5 - 9\%$ for rank 256 and $8 - 18\%$ for rank 512 (the overhead of Dion increases with rank). It is important to mention that the embedding size d of the models we used in our work is too small to see a significant difference in the runtime between Makhoul's algorithm and matmul when computing the column similarities. However, our benchmark presented in Appendix D shows the advantage of using Makhoul's algorithm at scale compared to matmul.

Factorization Accuracy. In order to understand why our low-rank projection in Trion is more accurate than Power-Iteration in Dion, we compute the ℓ_2-norm of projection errors between the accumulators B_t and the corresponding orthogonal updates that each optimizer uses to update the model: $\Delta_t^{\text{dion}} = ||B_t - P_t Q_t^\top||_2$ for Dion and $\Delta_t^{\text{trion}} = ||B_t - O_t||_2$ for Trion. In Figure 1 we show the projection errors for the linear layers of the first transformer block in a Llama-30M model and provide more information in Appendix F.

PT with LDAdamW. We train Llama-800M on 80B tokens (100 tokens per parameter) from C4 using LDAdam and DCT-AdamW described in Algorithm 2. In particular for our approach, we use EF quantized to 8-bits and some further optimizations from the ZeRO-redundancy optimizer (Rajbhandari et al., 2020), where one layer is updated on a single GPU and then it is broadcasted to the other GPUs. This way, we obtain lower memory usage by replacing redundant operations in the optimizer with communication, since the GPUs receiving the updated layer parameters do not allocate any state. In Figure 2 we present the training loss curves for full-rank AdamW (for reference), LDAdamW and DCT-AdamW and we are interested in directly comparing the last two. We observe that DCT-AdamW has lower training loss than LDAdamW, which also translates to lower perplexities in Table 2. Due to relatively high rank, the memory usage of LDAdamW is close to the memory usage of AdamW because it stores two projection matrices to be able to rotate the momentum buffers. In contrast, DCT-AdamW stores only two sets of r indices instead of storing the actual projection matrices, which drastically reduces the memory usage, coupled with the ZeRO-redundancy trick. In terms of running time, DCT-AdamW is faster than LDAdamW by 10h 7m ($\approx 25.75\%$) and slower than AdamW by 1h 55m ($\approx 5\%$).

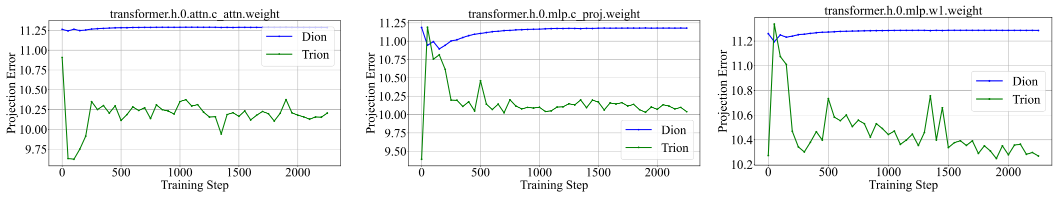


Figure 1: Projection errors for Dion and Trion for a few layers from the first transformer block on Llama-30M ($d = 640, r = 128$).

	AdamW	LDAdamW	DCT-AdamW
Train PPL	12.88	15.10	14.95
Val. PPL	11.73	13.91	13.69
Mem. (GiB)	73.72	72.10	57.82
Time	1d 13h 22m	2d 1h 24m	1d 15h 17m

Table 2: Pre-training results on Llama-800M for AdamW, LDAdamW and DCT-AdamW with 100 tokens/parameter. AdamW is the full-rank optimizer and is added for reference.

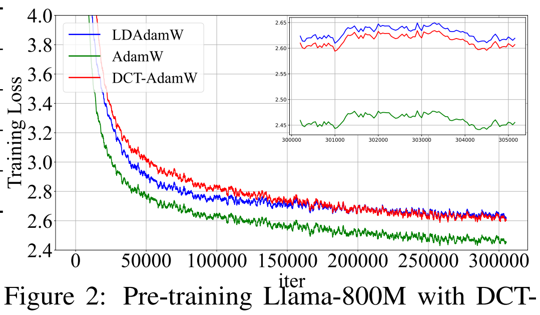


Figure 2: Pre-training Llama-800M with DCT-AdamW and LDAdam on 80B tokens from C4.

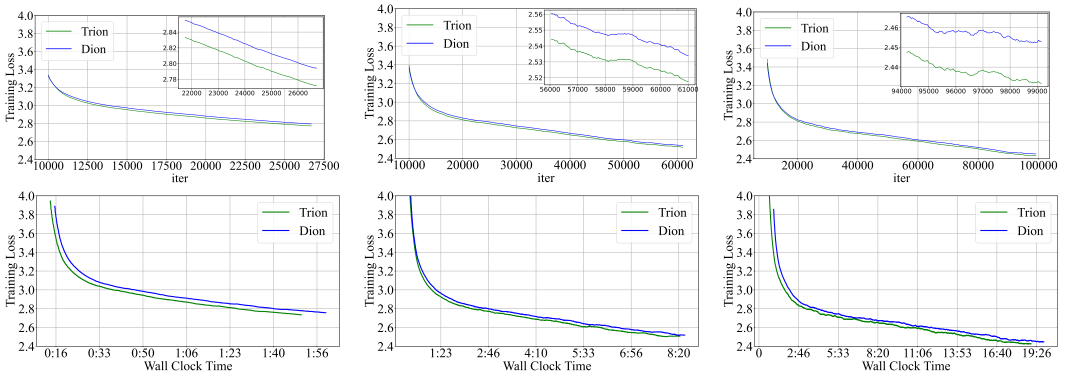
PT with FRUGAL/FIRA. We replace the SVD projection with our DCT approach and test it on Llama-800M model using 16B tokens from the C4. For space constraints reasons, we present the results in Appendix G.

4 Theoretical Guarantees

In this section, we provide a theoretical justification for our two-step procedure to approximate SVD-based gradient projections. First, we rigorously show that adaptively selecting columns of an orthogonal matrix based on their alignment with the gradient matrix is an optimal strategy for minimizing the reconstruction error. This approach leads to a contractive compression scheme, which is commonly exploited in the analysis of compressed adaptive optimization algorithms. Next, to justify the specific use of the DCT matrix, we demonstrate that it naturally serves as a linear approximation of the left or right eigenbases of the gradient matrices.

4.1 Optimality of norm-based ranking procedure

Let $G \in \mathbb{R}^{n\times m}$ be the gradient matrix and $Q \in \mathbb{R}^{n\times n}$ is orthogonal, namely $QQ^\top = I_n$ (without loss of generality, we consider left multiplication). For a given rank $r \leq n$, let Q_r be a $n \times r$



(a) Llama-350M (b) Llama-800M (c) Llama-1.3B

Figure 3: Training Loss across iterations (top row) and wall clock time (bottom row) for three Llama models trained with Dion and Trion optimizers on C4 using 20 tokens per parameter and rank $r = 256$. For a clearer visualization, the training loss curve was smoothed using a moving average with a window size of 200 and we zoom-in on the last 5000 training steps.

sub-matrix of Q composed of r columns. We are interested in the reconstruction error between G and $Q_r Q_r^\top G$, where $Q_r^\top$ performs projection and Q_r performs back-projection. While we may choose any matrix norm to measure the reconstruction error, the standard Frobenius norm makes the derivation cleaner. Specifically, using the definition of the Frobenius norm $\|G\|_{\mathrm{F}}^2 = \mathrm{tr}(G^\top G)$, linearity of trace and $Q_r^\top Q_r = I_r$ due to orthogonality of Q, we decompose the reconstruction error:

$$\begin{aligned}
\|G - Q_r Q_r^\top G\|_{\mathrm{F}}^2 &= \mathrm{tr}(G^\top (I - Q_r Q_r^\top)^\top (I - Q_r Q_r^\top) G) = \mathrm{tr}(G^\top (I - Q_r Q_r^\top) G) \\
&= \mathrm{tr}(G^\top G - G^\top Q_r Q_r^\top G) = \mathrm{tr}(G^\top G) - \mathrm{tr}(G^\top Q_r Q_r^\top G) \\
&= \|G\|_{\mathrm{F}}^2 - \|Q_r^\top G\|_{\mathrm{F}}^2 = \|G\|_{\mathrm{F}}^2 - \textstyle\sum_{i=1}^r \|q_i^\top G\|_2^2,
\end{aligned}$$

where q_i's are the columns of Q_r. From this identity, we conclude that to minimize the reconstruction error, the optimal strategy is to maximize the alignments $\|q_i^\top G\|_2^2$ for all selected columns q_i. Furthermore, since $\|q_1^\top G\|_2^2 + \|q_2^\top G\|_2^2 + \cdots + \|q_n^\top G\|_2^2 = \|Q^\top G\|_{\mathrm{F}}^2 = \mathrm{tr}(G^\top Q Q^\top G) = \mathrm{tr}(G^\top G) = \|G\|_{\mathrm{F}}^2$, following the optimal strategy of selecting r columns from Q, we have $\|G - Q_r Q_r^\top G\|_{\mathrm{F}}^2 = \|G\|_{\mathrm{F}}^2 - \sum_{i=1}^r \|q_i^\top G\|_2^2 \le \|G\|_{\mathrm{F}}^2 - \frac{r}{n}\sum_{i=1}^n \|q_i^\top G\|_2^2 = \left(1 - \frac{r}{n}\right)\|G\|_{\mathrm{F}}^2$. Thus, the proposed norm-based selection strategy for any orthogonal matrix Q induces a low-rank compression scheme that is contractive with a factor $1 - r/n$. Contractivness of the compression scheme is the key property in the convergence analysis of compressed optimization (Stich et al., 2018; Richtárik et al., 2021; Li et al., 2022; Modoranu et al., 2024; Robert et al., 2025).

We can extend the above argument for any p-norm over vectorized matrices, for which the Frobenius norm is the special case of $p = 2$. Then, we have:

$$\|G - Q_r Q_r^\top G\|_p = \left\|\sum_{i=1}^n q_i q_i^\top G - \sum_{i=1}^r q_i q_i^\top G\right\|_p = \left\|\sum_{i=r+1}^n q_i q_i^\top G\right\|_p \overset{(a)}{\le} \sum_{i=r+1}^n \left\|q_i q_i^\top G\right\|_p$$

$$\overset{(b)}{=} \sum_{i=r+1}^n \|q_i\|_p \left\|q_i^\top G\right\|_p \overset{(c)}{\le} \max(1, n^{\frac{1}{p}-\frac{1}{2}}) \sum_{i=r+1}^n \left\|q_i^\top G\right\|_p,$$

where (a) follows from the triangle inequality of the p-norm, (b) follows from the definition of p-norms for matrices, namely $\|uv^\top\|_p = (\sum_{i,j} u_i^p v_j^p)^{1/p} = (\sum_i u_i^p \sum_j v_j^p)^{1/p} = \|u\|_p \|v\|_p$ for two column-vectors u and v of the same size, (c) follows from the relationship between ℓ_p norms and that $\|q_i\|_2 = 1$, i.e., $\|v\|_p \le n^{\frac{1}{p}-\frac{1}{2}}\|v\|_2$ if $p \le 2$ and $\|v\|_p \le \|v\|_2$ if $p \ge 2$.

4.2 DCT AS LINEAR APPROXIMATION OF THE GRADIENT EIGENBASIS

Given a real-valued gradient matrix $G \in \mathbb{R}^{n\times m}$ and its SVD decomposition $G = U\Sigma V^\top$, our goal is to find fast approximation of (without loss of generality) its left eigenvectors stacked in $U \in \mathbb{R}^{n\times n}$. As $GG^\top = U\Sigma^2 U^\top$, it is equivalent to approximate the eigenvectors of symmetric matrix $GG^\top$.

Our argument starts from a linear algebra decomposition result, originally motivated by the optical information processing literature to factorize linear transformations that can be implemented optically

(Müller-Quade et al., 1998; Schmid et al., 2000). The result states that any square matrix M of shape $n \times n$ over the complex numbers $\mathbb{C}$ can be decomposed into a product of diagonal and circulant matrices, i.e., $M = D_1 C_2 D_3 \dots D_{2k-3} C_{2k-2} D_{2k-1}$, where D's are diagonal matrices and C's are circulant matrices. Circulant matrices are a special class of Toeplitz matrices in which each row is a cyclic right shift of the previous one as shown below:

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & \cdots & c_{n-2} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ c_1 & c_2 & \cdots & c_{n-1} & c_0 \end{bmatrix}, \quad F = \frac{1}{\sqrt{n}} \left[w^{ij} \right]_{i,j=0}^{n-1}, \text{ where } w = e^{\frac{2\pi i}{n}}. \tag{1}$$

The number of factors $2k-1$ in this decomposition has been shown to be up to $2n-1$ for almost all matrices (in the sense of Lebesgue measure), and it is further conjectured that a decomposition with up to n factors is sufficient (Huhtanen & Perämäki, 2015). Since circulant matrices can be diagonalized using the discrete Fourier transform (DFT) matrix F (see equation 1), i.e., $C = F^* D F$[4] (Golub & Van Loan, 2013), we can decompose the matrix $F^* M F$ with $M = GG^\top$ and arrive at the following decomposition for $GG^\top$:

$$GG^\top = (F D_1 F^*) D_2 (F D_3 F^*) D_4 \cdots D_{2k-2} (F D_{2k-1} F^*). \tag{2}$$

Notice the analogy between this matrix decomposition and the Taylor expansion for functions. Since the space of matrices is finite-dimensional, the signal-matrix can be recovered using finitely many "variable" D's, in contrast to the infinite series required for function expansions. Keeping this analogy in mind, just as loss functions are linearly approximated at each iteration in first-order optimization algorithms, we consider a "linear" approximation of the decomposition equation 2 by including only a single variable D, i.e., $GG^\top \approx F D_1 F^*$.

This directly implies that we approximate the eigenvectors U by the DFT matrix F. However, since the matrix $GG^\top$ is real and symmetric by design, its eigenvalues are real, and the real part Re(F) also forms an approximate eigenbasis that better aligns with U. Finally, we observe that the real part Re(F) corresponds to the discrete cosine transform (DCT), up to minor variations.

5 Related Work

Our approach aims to improve existing optimizers from prior work in the literature, which we group into two categories: optimizers that speed up convergence at the expense of increased FLOPs and optimizers focused on reducing the running time and memory usage by compressing the gradient via low-rank matrix factorization.

Fast Convergence Optimizers. Muon optimizer (Jordan et al., 2024) has recently stood up in the literature for its fast convergence rate due to the orthogonalized momentum update. The purpose of the orthogonalization is to push the singular values of the momentum matrix towards 1. It speeds up convergence by increasing the importance (singular values) of directions in the momentum matrix, which otherwise would have a low impact over the optimization.

The most straightforward approach to computing an orthogonal update is to use the $UV^\top$ from the SVD decomposition of the momentum matrix. Since SVD is expensive, it is replaced by an iterative procedure called Newton-Schulz, which involves computing the odd powers of the momentum matrix up to 5^{th} order and multiplying by some carefully chosen constants. While the Newton-Schulz procedure delivers an accurate approximation of $UV^\top$, the odd powers in the polynomial involve full-size matrix multiplications. This is in particular difficult for large scale settings, where the full matrices have to be materialized on a GPU before running Newton-Schulz, which increases communication and memory usage. Dion optimizer (Ahn et al., 2025) aims to reduce the communication overhead by using low-rank, orthogonal updates computed via Power-Iteration, that requires QR decomposition with running time depending on the rank.

Instead, we propose to reduce the overhead of Newton-Schulz in Muon and QR-decomposition in Dion by factorizing the momentum to a low-rank matrix using our DCT-based dynamic column selection approach and orthogonalizing the low-rank momentum using Newton-Schulz, then project back to the original space to update the model parameters.

[4] F^* is the conjugate transpose of the complex matrix F.

Low-Rank Compression of Optimizer States. Most approaches use individual projection matrices tailored to the gradient at each layer by invoking techniques based on quantization or matrix factorization, such as SVD, QR or PCA to reduce the memory usage of the **optimizer states stored in GPU memory**. The most common approach is to factorize the gradient to low-rank matrices. One pioneering work in the context of low-rank adaptive optimization is the recent GaLore optimizer (Zhao et al., 2024), which performs SVD once at a few steps to project the gradient to a lower-dimensional space. GaLore was followed by several other optimizers that improve certain aspects of it. LDAdam (Robert et al., 2025) aims to improve the computational runtime of GaLore by compressing the first order momentum in AdamW, replacing SVD with a Block Power-Iteration (Bentbib & Kanber, 2015) and performing a smooth subspace transition by rotating the first and second momentum accordingly to incorporate gradients from the same subspace at each step.

By default, GaLore discards the projection error, which LDAdam stores and incorporates into the gradient at the next step to improve convergence. In contrast, FRUGAL (Zmushko et al., 2024) makes the distinction between the low-rank gradient (called state-full) and the projection error (called state-free). The state-full gradient is used in AdamW, while the state-free gradient is fed to an optimizer without a state, such as SignSGD. The main idea is to leverage the remaining gradient information in the projection error since it is available at each step instead of discarding or storing it. A concurrent work to FRUGAL is FIRA (Chen et al., 2024), which preserves the low-rank constraint for memory efficiency while achieving full-rank performance by properly scaling the projection error.

Another approach worth mentioning is Online Subspace Descent Liang et al. (2024), which replaces SVD projection with Online PCA and involves computing the projection matrix as a solution of an optimization objective focused on: (a) minimizing the projection error and (b) forcing the projection matrix to be orthogonal. The authors indicate that performing one step with Adam to solve this additional optimization problem is enough to obtain a qualitatively good projection matrix P. In contrast, our method does not introduce such overheads during training since the DCT matrix is already computed at the beginning of training.

Despite all aforementioned approaches being similar in using SVD/QR-based projections, they differ in a few aspects. The most important one in our view is how they handle the projection error and how often they update the low-dimensional subspace, which we clarify in Table 3.

Our approach uses DCT projection coupled with a dynamic column selection to determine a projection matrix tailored to the gradient/momentum for a particular layer and can be integrated into any optimizer, regardless of the way it handles the projection error.

Low-rank Projection	**Type**	**Frequency***	**Error**
GaLore (Zhao et al., 2024)	SVD	200	discard
FRUGAL (Zmushko et al., 2024)	SVD, Random, RandPerm	200	feed to SignSGD
FIRA (Chen et al., 2024)	SVD	200	norm-based scaling
LDAdam (Robert et al., 2025)	Block Power-Iteration	1	error feedback
Dion (Ahn et al., 2025)	Power-Iteration	1	save to momentum
Trion (this work)	DCT	1	same as Dion
DCT-AdamW (this work)	DCT	any	error feedback

Table 3: Properties of prior low-rank adaptive optimizers. The update frequency* 200 is the default in GaLore that made the approach computationally feasible.

6 Conclusion and Limitations

We introduced the Trion and DCT-AdamW optimizers that use our DCT-based dynamic column selection to replace two techniques used to perform low-rank decomposition, that is, the inaccurate Power-Iteration in Dion and the expensive SVD and QR-decomposition in FRUGAL/FIRA/LDAdam. We showed that our work improves running time and memory usage. Moreover, it recovers the accuracy of the original methods and thus serves as a cheaper alternative to these expensive and inaccurate methods used to perform low-rank decomposition in adaptive gradient methods for both pretraining and finetuning.

Our experiments are limited to models with at most 1.3B parameters for pretraining and additional work is required to test our technique for larger models and beyond the Chinchilla-optimal token counts, which would require significantly more computational resources.

References

Kwangjun Ahn, Byron Xu, Natalie Abreu, and John Langford. Dion: Distributed orthonormalized updates, 2025. URL https://arxiv.org/abs/2504.05295.

Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. The convergence of sparsified gradient methods, 2018. URL https://arxiv.org/abs/1809.10505.

Abdeslem Bentbib and A. Kanber. Block Power Method for SVD Decomposition. Analele Stiintifice ale Universitatii Ovidius Constanta, Seria Matematica, 23:45–58, 06 2015. doi: 10.1515/auom-2015-0024.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. Fira: Can we achieve full-rank training of llms under low-rank constraint?, 2024. URL https://arxiv.org/abs/2410.01623.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. arXiv preprint arXiv:2110.02861, 2021.

Gene H. Golub and Charles F. Van Loan. Matrix Computations - 4th Edition. Johns Hopkins University Press, Philadelphia, PA, 2013. doi: 10.1137/1.9781421407944. URL https://epubs.siam.org/doi/abs/10.1137/1.9781421407944.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Marko Huhtanen and Allan Perämäki. Factoring matrices into the product of circulant and diagonal matrices. Journal of Fourier Analysis and Applications, 2015.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes, 2019. URL https://arxiv.org/abs/1901.09847.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Xiaoyun Li, Belhal Karimi, and Ping Li. On distributed adaptive optimization with gradient compression. arXiv preprint arXiv:2205.05632, 2022.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates, 2023. URL https://arxiv.org/abs/2307.05695.

Kaizhao Liang, Bo Liu, Lizhang Chen, and qiang liu. Memory-efficient LLM training with online subspace descent. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL https://openreview.net/forum?id=P8rTCT6g45.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In Proceedings of the Seventh International Conference on Learning Representations, 2019.

Qijun Luo, Hengxu Yu, and Xiao Li. BAdam: A memory efficient full parameter training method for large language models. arXiv preprint arXiv:2404.02827, 2024.

J. Makhoul. A fast cosine transform in one and two dimensions. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(1):27–34, 1980. doi: 10.1109/TASSP.1980.1163351.

Ionut-Vlad Modoranu, Mher Safaryan, Grigory Malinovsky, Eldar Kurtic, Thomas Robert, Peter Richtarik, and Dan Alistarh. Microadam: Accurate adaptive optimization with low space overhead and provable convergence, 2024. URL https://arxiv.org/abs/2405.15593.

J. Müller-Quade, H. Aagedal, Th. Beth, and M. Schmid. Algorithmic design of diffractive optical systems for information processing. Physica D: Nonlinear Phenomena, 120(1):196–205, 1998. ISSN 0167-2789. doi: https://doi.org/10.1016/S0167-2789(98)00055-4. URL https://www.sciencedirect.com/science/article/pii/S0167278998000554. Proceedings of the Fourth Workshop on Physics and Consumption.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research, 21(140):1–67, 2020.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–16. IEEE, 2020.

Peter Richtárik, Igor Sokolov, and Ilyas Fatkhullin. EF21: A New, Simpler, Theoretically Better, and Practically Faster Error Feedback. arXiv preprint arXiv:2106.05203, 2021.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. Ldadam: Adaptive optimization from low-dimensional gradient statistics, 2025. URL https://arxiv.org/abs/2410.16103.

Michael Schmid, Rainer Steinwandt, Jörn Müller-Quade, Martin Rötteler, and Thomas Beth. Decomposing a matrix into circulant and diagonal factors. Linear Algebra and its Applications, 306(1):131–143, 2000. ISSN 0024-3795. doi: https://doi.org/10.1016/S0024-3795(99)00250-5. URL https://www.sciencedirect.com/science/article/pii/S0024379599002505.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In Fifteenth Annual Conference of the International Speech Communication Association, 2014.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. arXiv preprint arXiv:1809.07599, 2018.

Gilbert Strang. The discrete cosine transform. SIAM review, 41(1):135–147, 1999.

Zhenyu Zhang, Ajay Jaiswal, Lu Yin, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. Q-GaLore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. arXiv preprint arXiv:2407.08296, 2024.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. arXiv preprint arXiv:2403.03507, 2024.

Philip Zmushko, Aleksandr Beznosikov, Martin Takáč, and Samuel Horváth. Frugal: Memory-efficient optimization by reducing state overhead for scalable training, 2024. URL https://arxiv.org/abs/2411.07837.