EFFICIENT PRETRAINING OF LLMS THROUGH TRUNCATED SVD REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM pretraining is extremely costly; therefore, parameter-efficient LLM architectures have recently emerged as a compelling research direction. One such promising approach is to represent the parameters as orthonormal low-rank weight matrices. However, maintaining orthonormality during training is computationally expensive, making it impractical. This paper presents the TSVD (Truncated Singular Value Decomposition) framework which efficiently maintains orthonormality through QR decomposition and caching. Furthermore, a spectral energy heuristic is introduced to select the rank of the resulting low-rank weight matrices. Empirical evaluations across model sizes show that TSVD matches or outperforms full-parameter baselines at a fraction of the compute cost. TSVD thus provides a scalable, computationally efficient foundation for LLM pretraining.

1 INTRODUCTION

The number of parameters in LLMs has grown exponentially in recent years, scaling from millions to trillions (Gu, 2025; Minaee et al., 2025; Zhao et al., 2026). To pretrain such models, immense computational resources are needed (Hoffmann et al., 2022; Kaplan et al., 2020). To mitigate these costs, various optimization strategies have been developed. In particular, representing LLM weight matrices via low-rank approximations has gained prominence as an effective approach (Kamalakara et al., 2022; Khodak et al., 2022; Savostianova et al., 2023; Sui et al., 2024; Wei et al., 2024).

Representing LLM weight matrices as low-rank products drastically reduces the number of parameters and the computational overhead. However, this approach typically increases training and evaluation loss compared to full-parameter models (Zhao et al., 2024; Han et al., 2024; Wei et al., 2024). A potential solution is to make these matrices orthonormal, which alleviates vanishing and exploding gradients, reduces overfitting, enhances generalization, and leads to higher accuracy and faster convergence rates (Arjovsky et al., 2016; Bansal et al., 2018; Huang et al., 2017; see App. A for details). However, strict orthonormality is computationally expensive. It can be approximated through orthonormality regularization (Yang et al., 2020), which introduces an extra hyperparameter to balance the penalty against the primary loss. Alternatively, low-rank initialization with periodic orthonormalization balances computational overhead and performance (Mo et al., 2025).

This paper proposes Truncated Singular Value Decomposition (TSVD) as an efficient low-rank representation for LLM weight matrices. Unlike existing low-rank methods, TSVD maintains strict orthonormality with minimal computational overhead. Preserving orthonormality yields lower training and evaluation loss than other low-rank approximations. Empirically, models with TSVD layers perform on par with or better than full-parameter models. Furthermore, their reduced memory and compute footprint make it possible to scale to larger LLM architectures with the same hardware.

Thus, this research closes the performance gap between compressed, low-rank models and their full-parameter counterparts. The main contributions are:

1. An efficient three-fold strategy for maintaining orthonormality:
   (a) QR-based retraction: Replaces the expensive SVD steps required by standard Riemannian optimization with fast QR decomposition.
   (b) Amortized computation: Deploys a caching mechanism that amortizes the remaining orthonormalization overhead across gradient accumulation steps.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(c) Rank selection: A spectral-energy heuristic for selecting different ranks for weight matrices within a transformer decoder layer prior to training, yielding better pretraining performance than uniform rank selection.

2. By substantially reducing the parameter count, the TSVD framework reduces LLM pretraining FLOPs, lowering wall-clock time and GPU costs. Its lower memory footprint also enables training larger models on existing hardware or cheaper GPUs.
3. A demonstration that TSVD achieves competitive or superior performance compared to dense, full-parameter models, despite significantly fewer parameters.

Thus, TSVD offers a practical framework for efficient pretraining, reducing computational costs while maximizing LLM performance within fixed compute budgets.

2 Related Work

This section reviews previous approaches to reducing LLM pretraining costs. They are categorized into two main groups: (1) parameter-preserving methods, which improve training efficiency without modifying model scale, and (2) parameter-reducing methods, which factorize or compress weight matrices directly. For each category, the key methodologies are discussed alongside their inherent limitations, followed by the subsequent techniques developed to address those shortcomings.

Parameter-preserving methods. GaLore (Zhao et al., 2024) reduced the pretraining memory requirements without reducing the number of parameters. They projected the gradient matrix into a low-rank form, substantially reducing the memory cost for the optimizer states. However, during pretraining, the optimizer states only constitute a small proportion of the total memory, limiting the effectiveness of the approach (Shamshoum et al., 2025).

Low-Rank Adaptation (LoRA; Hu et al., 2021) is a popular *fine-tuning* method that efficiently adapts a pretrained LLM to a specific application or problem. LoRA freezes the weights of the pretrained model and adds two low-rank trainable matrices; during the forward pass, it multiplies the input by both the frozen weights and the product of the low-rank matrices. Inspired by LoRA, ReLoRA (Lialin et al., 2023) *pretrains* an LLM using multiple low-rank updates. Instead of updating the full weights, ReLoRA trains low-rank matrices and periodically merges the low-rank updates into the main weights. It then reinitializes the low-rank matrices and repeats this process. Over time, multiple low-rank updates accumulate into a high-rank update, enabling efficient training without updating all parameters at once. However, ReLoRA requires a ”warm start” full-parameter training for good performance and depends on an intricate learning-rate schedule. GaLore and ReLoRA successfully lower memory overhead for optimizer states and gradients but do not decrease the total number of trained parameters for even greater computational efficiency.

Parameter-reducing methods. Several studies investigated pretraining models via low-rank weight matrix representations, effectively reducing the number of trainable parameters (Kamalakara et al., 2022; Khodak et al., 2022; Savostianova et al., 2023; Sui et al., 2024; Wei et al., 2024). While this approach reduced pretraining memory and compute requirements, its training and evaluation loss was worse than that of full-parameter training (Zhao et al., 2024; Sui et al., 2024; Wei et al., 2024).

Incorporating orthonormality into low-rank representations serves as a strategy to mitigate the performance trade-offs typical of low-rank methods (Fernandez-Lopez et al., 2024; Han et al., 2024; Li et al., 2025; Yang et al., 2020). To enhance the performance of low-rank pretraining, LR-SMS (Fernandez-Lopez et al., 2024) proposed SVD initialization, wherein low-rank factors are initialized via SVD (Eckart & Young, 1936) to ensure initial orthonormality. SLTrain (Han et al., 2024) and LOST (Li et al., 2025) decomposed each weight matrix into additive low-rank and sparse components. Both methods also utilized SVD initialization to provide a stable starting point, where the sparse component served as a complementary mechanism to recover information lost during compression by capturing salient features within the residual space. While LR-SMS, SLTrain, and LOST utilize SVD to initialize low-rank factors as orthonormal, this property is not preserved during training. Because the parameters remain unconstrained during optimization, they inevitably deviate from orthonormality. To address this limitation, SVD Training (Yang et al., 2020) uses orthonormality regularizers to maintain orthonormality during training. It requires an additional hyperparameter to balance the regularization penalty against the primary loss objective.

The Low-Rank Riemannian Optimizer (LORO) (Mo et al., 2025) treated weight matrices as elements of a fixed-rank Riemannian manifold. During each update, low-rank factors were jointly optimized following the Riemannian gradient, facilitating effective descent of the loss function. This retraction-based process involved: (1) projecting the Euclidean gradient onto the tangent space to update the weights, and (2) retracting the updated weights back onto the manifold via SVD. In contrast to SVD Training, LORO avoided explicit orthonormality regularizers. To maintain computational efficiency, the method bypassed the costly SVD retraction for the majority of iterations, performing the operation only every K steps (e.g., $K = 500$). Consequently, the weight matrices remain off the intended manifold for the bulk of the training process. StelLA (Li et al., 2026) is conceptually similar to LORO; however, it is designed for parameter-efficient fine-tuning (PEFT) rather than pretraining. This shift introduces fundamental differences in key training dynamics, including gradient behavior, optimization stability, and parameter scaling.

Distinct from the weight-centric strategies mentioned above, COLA (Liu et al., 2025) leverages the low-rank nature of LLM activations by replacing Multilayer Perceptron (MLP) and attention projection layers with low-rank autoencoders. Its memory-efficient variant, COLA-M, further reduces overhead by utilizing low-rank activations, while both frameworks preserve the original query, key, and value projection structures. Finally, to control the spectral norm of weight matrices and improve generalization, POET (Qiu et al., 2025) and its more efficient variant, POET-X (Qiu et al., 2026), reparameterize W as RWP. Both methods freeze W and learn orthonormal matrices R and P using distinct approximations; however, the frozen weights must still be retained in memory.

In summary, low-rank weight matrix representations offer the most significant reductions in pretraining memory and computational overhead. The integration of orthonormality constraints helps mitigate the performance trade-offs typical of low-rank methods. However, due to high computational costs, previous techniques have not maintained orthonormality throughout the entire training duration. The TSVD method proposed in this paper ensures strict orthonormality at every iteration and, to the best of our knowledge, is the first approach to achieve this goal in a pretraining setting.

3 The TSVD Framework

In this section, SVD and TSVD are reviewed, and the TSVD layer, model, and its corresponding training methodology are introduced. Next, QR decomposition, caching, and rank selection methods are described. The TSVD algorithm is summarized in 1 and compared to the state of the art in B.

3.1 Singular Value Decomposition (SVD)

SVD (Eckart & Young, 1936) is a fundamental matrix factorization method widely used in linear algebra, statistics, and machine learning. At its core, it expresses any matrix as a product of three simpler matrices. For any $m \times n$ matrix W, the SVD is

$$W = U\Sigma V^T, \tag{1}$$

where $U \in \mathbb{R}^{m\times m}$ comprises the left singular vectors (orthonormal matrix), $\Sigma \in \mathbb{R}^{m\times n}$ are singular values (diagonal matrix with non-negative entries), and $V \in \mathbb{R}^{n\times n}$ comprises the right singular vectors (orthonormal matrix). Thus, SVD decomposes a linear transformation into three distinct geometric operations: V^T rotates the input into a new coordinate system, Σ scales the vector along orthogonal axes, and U rotates the result into the output's coordinate system.

For any $m \times n$ matrix W, Truncated SVD (TSVD) is a low-rank approximation of SVD, defined as

$$W \approx U_k\Sigma_k V_k^T, \tag{2}$$

where U_k is the first k columns of U, Σ_k is the diagonal matrix with the top k singular values, and V_k is the first k columns of V. Thus, the truncated version discards the smaller singular values and their corresponding directions.

3.2 The TSVD Terminology

The TSVD framework consists of three core abstractions. A **TSVD layer** approximates a full weight matrix $W \in \mathbb{R}^{m\times n}$ via truncated factorization $W \approx U_k\Sigma_k V_k^T$. Here, $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and

$V_k^T \in \mathbb{R}^{k \times n}$ are initialized as unconstrained trainable parameters. During the forward pass, the layer enforces structural constraints: U_k and V_k are projected onto orthonormal matrices via QR decomposition (Chan, 1987), while Σ_k is constrained to a positive-valued diagonal vector. Constraining U_k and V_k to be orthonormal is what provides the stability benefits established in prior work on orthogonal and semi-orthogonal parameterizations, summarized in App. A. Second, a **TSVD model** is a neural network that replaces standard dense weight matrices with TSVD layers. Third, **TSVD training** is the end-to-end training process of a TSVD model, wherein structural orthonormality constraints are maintained across every training iteration.

3.3 QR Decomposition

Low-rank approximation of W reduces the number of parameters from mn to $mk + k + kn = k(m+n+1) \approx k(m+n)$. For example, if W is a 1000 by 800 matrix, and $k = 128$, the number of parameters reduces from $800,000$ to $128*(1000+800+1) = 230,528$, i.e. by $\approx 71\%$. The benefits are twofold: (1) lower memory requirements for storing parameters, gradients, and optimizer states, and (2) reduced computational cost, as measured in FLOPs, because multiplying the input by a low-rank approximation of W is less costly than multiplying the input by W (see App. C for details). The TSVD framework relies on QR decomposition in the forward pass to enforce orthonormality on unconstrained trainable parameters. The cost, in the number of FLOPs, for a matrix $\in \mathbb{R}^{m \times k}$ is

$$2mk^2 - \frac{2}{3}k^3. \tag{3}$$

3.4 Caching

To mitigate the computational overhead of QR decomposition, the structure of gradient accumulation can be leveraged (Goodfellow et al., 2016). While larger batch sizes make training of LLMs more stable, hardware constraints often require splitting an effective batch into smaller mini-batches. For example, an effective batch of 1024 is processed as four mini-batches of 256, with gradients accumulated before a single parameter update. The TSVD training includes a caching mechanism that computes the QR decomposition during the initial mini-batch and reuses the result for all subsequent mini-batches within the same update. This approach ensures that the expensive orthonormalization occurs only once per parameter update, effectively amortizing the computational cost while preserving the benefits of orthonormality. For ablation studies of these mechanism, see Sec. 5.3.2.

3.5 Rank Selection

Existing low-rank methods (Han et al., 2024; Mo et al., 2025; Liu et al., 2025) typically apply a uniform rank across all layers and weight matrices, regardless of their functional roles. However, this choice may be suboptimal, because weight matrices in different components of a transformer decoder layer can require different ranks to perform well.

The TSVD framework employs the spectral energy of weight matrices of pretrained models as a heuristic for rank selection. First, SVD is applied to the weight matrices of the pretrained model, obtaining the diagonal matrix Σ with singular values $\sigma_1, \sigma_2, \ldots, \sigma_n$. The total energy of a full-rank matrix W is then defined as the sum of these values. Second, to determine the rank for the low-rank approximation, the energy for a truncated set of ranks is calculated. The *compression rank*, $r_\alpha(W)$, to preserve at least α (e.g., 95%) of the total energy is then defined as

$$r_\alpha(W) \;=\; \min\left\{ k \;\middle|\; \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2} \geq \alpha \right\}. \tag{4}$$

The value of α is selected based on the desired reduction in model's parameter count and is used to determine the compression rank of the weight matrices across different layers. Since compression rank generally varies across layers – for example, MLP weight matrices typically exhibit higher compression ranks than attention weight matrices – this approach supports variable rather than uniform rank selection. In the next section, we analyze the compression ranks across models of various sizes within an LLM family to determine their configurations.

Algorithm 1 TSVD Training with Caching

Require: TSVD model; Spectral energy compression ranks $\{r_i\}_{i=1}^{N}$ of weight matrices
for each TSVD weight matrix W_i **do**
  Initialize $A_{U,i} \in \mathbb{R}^{m_i \times r_i}$, $A_{V,i} \in \mathbb{R}^{n_i \times r_i}$, $A_{\Sigma,i} \in \mathbb{R}^{r_i}$
end for
for each optimizer step **do**
  for each mini-batch **do**
    for each TSVD weight matrix W_i **do**
      if cache is populated **then**
        Read cached (U_i, V_i, Σ_i)
      else
        $U_i \leftarrow \mathrm{qf}(A_{U_i})$
        $V_i \leftarrow \mathrm{qf}(A_{V_i})$
        $\Sigma_i \leftarrow \mathrm{diag}(\mathrm{softplus}(A_{\Sigma i}))$
        cache (U_i, V_i, Σ_i)
      end if
      run forward pass using $W_i := \frac{1}{\sqrt{r_i}} U_i \Sigma_i V_i^\top$
    end for
    run backward pass, and gradient accumulation
  end for
  update $\{A_{U,i}, A_{V,i}, A_{\Sigma,i}\}$ for $i = 1, \ldots, N$
  clear all TSVD caches
end for
Note: $\mathrm{qf}(\cdot)$ denotes the Q factor of a reduced QR decomposition.

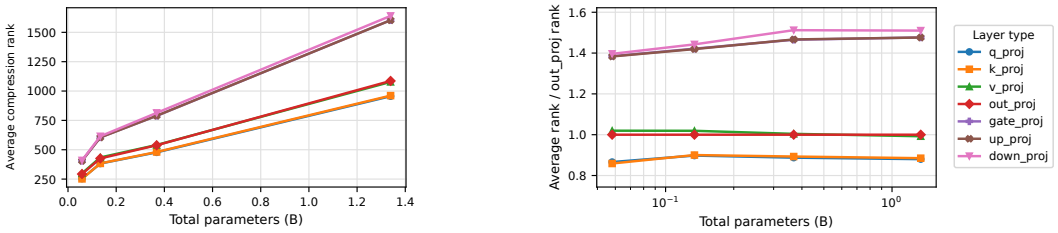
4 Configuring Compression Ranks

Compression ranks are analyzed across the experimental suite of full-parameter Llama-style LLMs (Touvron et al., 2023a), ranging from 60M to 1.3B parameters. Compression rank (Eq. 4) is computed for each linear weight matrix within the transformer decoder layers. Those are the query, key, value and out projection within the attention mechanism and the gate, up and down projections in the MLP. The compression ranks for the models in the experimental suite are provided in Fig. 4 in the Appendix. Across layers, the compression ranks of different projection types (query, key, etc.) are consistent, with the exception of the earliest layers. Motivated by this stability, compression ranks are averaged across layers for comparison across model scales.

Fig. 1a illustrates the compression ranks across the different model sizes in the experimental suite. The main observation is that they scale approximately linearly with model size. Notably, the major architectural parameters (Tab. 1), that is, the number of transformer layers n_{layers}, the number of heads per layer n_{heads}, the hidden dimension d_1, and the MLP intermediate dimension d_2, scale uniformly across the models. When normalizing compression ranks (we chose the compression rank of the out projection) of the model (Fig. 1b), the relative proportions between ranks of different projections are nearly constant across model sizes. Overall, these results indicate that *compression ranks measured on smaller models can be used to estimate those of larger models algorithmically*.

To assess the generality of the findings and highlight potential deviations, the analysis was extended to large, publicly available model families: Llama-2 (Touvron et al., 2023b), Llama-3.1 (Dubey et al., 2024), and Olmo 3 (Olmo et al., 2026). Detailed results are provided in Appendix D. The results show that the compression ranks determined for small models can be used to predict the compression ranks at larger model sizes given the correct scaling of its hyperparameters.

For the pretraining experiments in Sec. 5.2.1, the consistent relationships observed between the compression ranks of different functional components in Fig. 1a guided the rank selection; E.g., the compression ranks for the value and output projections layers were 18% larger than those for the query and key projections. The compression rank ratios were then maintained when scaling to larger target architectures in the rest of the experiments. That is, while absolute ranks are scaled according to a target parameter budget, the relative rank selections adhere to these relationships.



(a) Unnormalized (b) Normalized by out projection

Figure 1: Compression ranks measured across layers and model sizes. (a) Compression ranks of the different weight matrices (listed in the legend) in the transformer layers of the models in the experimental suite. The size of the model is shown in the x axis; the ranks are averaged for each layer type within the same model. The ranks grow approximately linearly over the model size. (b) Compression ranks in Fig. 1a normalized by the size of the out projection for each model size. The proportions remain nearly constant across model scales. These observations suggest that compression ranks measured on smaller models can be scaled to estimate those in larger models.

Table 1: Major architectural parameters of models in the experimental suite. Those are embedding dimension d_1, the hidden MLP dimension d_2, the number of layer n_{layers} and the number of heads n_{heads}. Gray values denote scaling factors relative to the smallest model in the model family. Most architectural parameters scale proportionally with model size. As shown in Fig. 1, compression ranks scale consistently, allowing smaller models to estimate ranks for larger ones.

Model size / factor	60m	130m	× 60m	350m	× 60m	1.3B	× 60m
d_1	512	768	1.5	1024	2	2048	4
d_2	1376	2048	1.5	2736	2	5461	4
n_{layers}	8	12	1.5	24	3	24	3
n_{heads}	8	12	1.5	16	2	32	4

5 Experiments

This section details preliminary experiments establishing hyperparameters, followed by a performance comparison across model scales for TSVD, full-parameter, and low-rank architectures during pretraining and evaluation. Finally, ablation studies examine the impact of the QR decomposition, caching mechanism, and the role of variable rank selection.[1]

5.1 Hyperparameter Setup

To tune hyperparameters across the TSVD experimental suite, the Maximal Update Parametrization (μP; Yang et al., 2022) framework was employed, which mitigates the prohibitive computational cost of optimization in large neural networks. By properly scaling initialization and learning rates, μP guarantees stable forward-pass activations and backward-pass gradient updates as model width increases. Crucially, this property enables zero-shot hyperparameter transfer, allowing optimal learning rates identified on small, computationally cheap models to transfer directly to significantly larger models without additional tuning. Accordingly, TSVD hyperparameter searches used the lightweight Llama-60M variant, requiring approximately 30 minutes of compute.

5.2 Performance Evaluation

This section provides a performance comparison across model scales for TSVD, full-parameter, and low-rank architectures during pretraining and evaluation.

[1]The source code will be made publicly available on GitHub upon publication of the paper and is included as a ZIP archive in the supplementary material.

Table 2: Pretraining results on C4 dataset, including validation perplexity (PPL), number of parameters in millions (Param), and memory usage in GiB (Mem) for parameters, gradients, and activations at 16-bit precision. ↓ indicates lower is better. The symbol r is the rank for low-rank and CoLA models. K, Q, V, Out, and MLP are ranks for attention, out projection, and MLP layers. Results show that TSVD outperforms full-parameter and low-rank methods with lower computational cost.

Model (# Tokens)	Llama-60M (1.1B)			Llama-130M (2.2B)			Llama-350M (6.4B)			Llama-1.3B (13.1B)		
r	128			256			256			512		
K, Q / V, Out / MLP	96 / 112 / 160			168 / 192 / 280			168 / 192 / 280			392 / 456 / 640		
Method	PPL ↓	Param	Mem ↓	PPL ↓	Param	Mem ↓	PPL ↓	Param	Mem ↓	PPL ↓	Param	Mem ↓
Full-parameter	16.02	58	0.43	12.08	134	1.00	9.20	368	2.74	7.68	1339	9.98
Low-rank	14.99	**43**	**0.32**	11.73	94	0.70	9.23	185	1.37	7.72	**609**	**4.54**
CoLA	14.84	46	0.34	11.58	101	0.75	9.23	223	1.66	7.74	760	5.66
TSVD	**14.40**	**43**	**0.32**	**11.36**	**91**	**0.68**	**9.14**	**177**	**1.32**	**7.56**	644	4.80

Table 3: Model perplexity (↓) during training steps 50k to 150k for 7B full-parameter and TSVD models. TSVD is more parameter and memory efficient and closes the perplexity gap to the full parameter model as training progresses.

Steps (# Tokens)	50k (6.5B)	75k (9.8B)	100k (13.1B)	120k (15.7B)	150k (19.6B)		
Method	PPL ↓					Param	Mem ↓
Full-parameter	7.35	6.84	6.60	6.51	6.38	6,738	50.20
TSVD	7.83	7.27	6.90	6.70	6.57	4,574	34.08

5.2.1 Pretraining Performance

A series of Llama-style LLMs (Touvron et al., 2023a) with scales ranging from 60M to 7B parameters were pretrained in the language modeling task. The experimental framework strictly followed the benchmark protocols established in the literature (Zhao et al., 2024; Han et al., 2024). However, the benchmarks were ported to the HuggingFace ecosystem (Wolf et al., 2020), using `accelerate` for launching distributed runs across GPUs. Four architectural configurations were evaluated: dense full-parameter models, models with low-rank approximations of linear and attention layers, models with CoLA-based linear layers, and the proposed TSVD approach. This enables comparison of TSVD with traditional and state-of-the-art reparameterization methods.

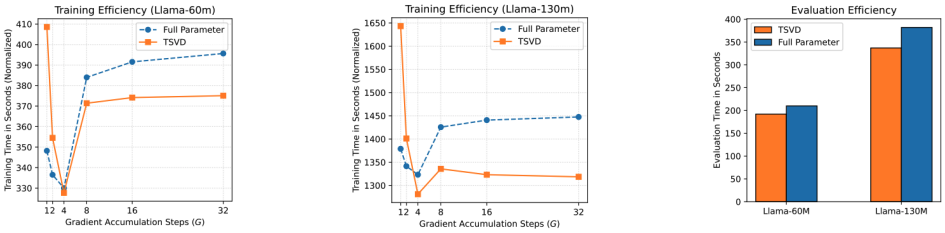
Models were trained on the Colossal Clean Crawled Corpus (C4; Raffel et al., 2019; Dodge et al., 2021), a massive web-scale dataset derived from Common Crawl. C4 uses strict heuristic filtering to remove low-quality, repetitive, and offensive data. Comprising of 365M documents and 150B tokens, C4 serves as a standard benchmark for pretraining prominent models like T5 (Dodge et al., 2021).

The results for models up to 1.3B are included in Tab. 2. Across all model scales, TSVD models consistently achieved lower perplexity than their full-parameter counterparts. TSVD models contained between 24% and 51% fewer parameters, yielding a corresponding reduction in the memory footprint for parameters, gradients, and optimizer states. Note that while the full-parameter, low-rank, and CoLA experiments were designed to replicate results reported in the literature, the modern Hugging Face implementation yielded significantly improved accuracies for them. These enhancements may be attributed to several factors, including modernized weight initialization schemes, refined tokenization processes, and the utilization of better optimization hyperparameters.

The scale-up properties of TSVD were demonstrated further on a Llama-7B model, which takes an average of 10 full days to pretrain on an NVIDIA H200 GPU node. The results are presented in Tab. 3. Although the TSVD model's perplexity is $\approx 3\%$ higher than that of the full-parameter model, it contains over 2B fewer parameters – significantly reducing memory overhead for parameters, gradients, and optimizer states. Furthermore, as shown in Tab. 3, the perplexity gap between the full-parameter and TSVD models narrows as training progresses, decreasing from 0.48 at 50k steps to 0.30 at 100k, and 0.19 at 150k. Given the observed trajectory, further training would likely narrow this performance gap even more. These results demonstrate that TSVD scales effectively to larger models, offering a robust and memory-efficient approach to large-scale pretraining.

Table 4: Accuracy across model scales for full-parameter and TSVD architectures on commonsense reasoning and text/context understanding benchmarks, HellaSwag, LAMBADA, and PIQA.

	Llama-60M			Llama-130M			Llama-350M			Llama-1.3B		
Method	HS	LAMB	PIQA	HS	LAMB	PIQA	HS	LAMB	PIQA	HS	LAMB	PIQA
Full-parameter	25.4%	12.6%	58.5%	26.0%	15.9%	60.5%	29.1%	**27.8%**	**65.2%**	32.7%	**33.5%**	68.7%
TSVD	**25.9%**	**15.5%**	**60.4%**	**27.0%**	**17.9%**	**63.2%**	**29.8%**	27.4%	64.6%	**33.5%**	**33.5%**	**68.9%**



(a) Llama-60M (b) Llama-130M (c) Normalized

Figure 2: The impact of caching on 2a Llama-60M training, 2b Llama-130M training, and 2c Llama60m and Llama130m evaluation. As gradient accumulation steps increase, the TSVD model achieves greater gains in training efficiency compared to the full-parameter model, while maintaining consistent evaluation efficiency regardless of step size.

Table 5: Comparison of uniform vs. compression rank selections. In the 'Compression-rank' column, the triplet represents the ranks for the query/key projection, value/output projection, and MLP layer, respectively. Values in the 'Uniform-rank' column are applied uniformly across all layers. The compression-rank TSVD models outperform their uniform-rank counterparts, with the exception of the 60M model. Moreover, its performance advantage grows with model scale.

Models	Uniform-rank	Compression-rank	Uniform-rank PPL	Compression-rank PPL
Llama-60M	136	96, 112, 160	**14.38**	14.40
Llama-130M	240	168, 192, 280	11.40	**11.36**
Llama-350M	240	168, 192, 280	9.19	**9.14**
Llama-1.3B	552	392, 456, 640	7.62	**7.56**

5.2.2 Benchmark Performance

Having established that TSVD models perform well in pretraining task, It is important to assess pretrained model transfer to application benchmarks. To this end, the pretrained full-parameter and TSVD models were evaluated in key commonsense reasoning and text/context understanding tasks: HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2019), and LAMBADA (Paperno et al., 2016). Detailed accuracy results are reported in Tab. 4. The experimental results demonstrate that the TSVD models consistently match or outperform full-parameter models across the entire suite of evaluation benchmarks, with the sole exception of the Llama-350M variant on the LAMBADA dataset. Consequently, aligning closely with its strong pretraining trajectory, TSVD maintains competitive generalization and robust performance when transferred to external evaluation tasks.

5.3 Ablation Studies

This section examines the impact of the QR decomposition, caching mechanism, and rank selection.

5.3.1 Impact of QR decomposition

The TSVD framework uses QR decomposition to maintain orthonormality, avoiding the computationally expensive SVD operation in Riemannian optimization. The respective computational costs are given by Equations 3 and 5. Fig. 3 in Appendix plots the computational cost of SVD vs. QR decomposition for various ranks, showing that QR decomposition requires far fewer FLOPs.

5.3.2 Impact of Caching Mechanism

Gradient accumulation is a training technique that partitions an effective batch into smaller mini-batches, processing them sequentially while accumulating their gradients to execute a single parameter update once the full batch size is reached. The TSVD framework computes the QR decomposition during the initial mini-batch, caches the results, and reuses them across subsequent mini-batches, effectively amortizing orthonormalization costs. Consequently, an increased number of gradient accumulation steps leads to greater amortization of the QR decomposition overhead.

Full-parameter and TSVD Llama models (60M and 130M) were pretrained across $G \in \{1, 2, 4, 8, 16, 32\}$ gradient accumulation steps. Fig. 2 shows that larger gradient accumulation steps and model sizes amplify TSVD's training efficiency gains over full-parameter models. Note that at $G = 1$ (a single step), TSVD performs QR decomposition at every update without cache reuse, making it slower than the full-parameter model. However, as the number of gradient accumulation steps increases, the computational cost of QR decomposition is amortized across steps. The TSVD version thus outperforms the full-parameter version in wall-clock time, making it a more practical choice as the number of gradient accumulation steps increases.

5.3.3 Impact of Compression Rank Selection

While a uniform rank selection enforces a fixed rank across all layers and weight matrices irrespective of their functional roles, the TSVD framework computes the compression rank of each individual weight matrix based on its spectral energy, naturally accommodating variable rank selection across the architecture. To evaluate the impact of compression rank selections, four model scales were pretrained: Llama-60M, 130M, 350M, and 1.3B. At each scale, a TSVD model utilizing compression ranks (via spectral energy analysis) was compared against a uniform-rank baseline with fixed rank selections of 136, 240, 240, and 552, respectively. These uniform ranks were selected to match the total parameter count of the corresponding compression-rank models at each scale.

The experimental results are presented in Tab. 5. The compression-rank TSVD models consistently outperform their uniform-rank counterparts, with the exception of the 60M scale. Furthermore, the performance advantages of compression rank selection widen as model scale increases. The results show that compression rank selection outperforms uniform rank selection.

6 Discussion and Future Work

The TSVD experiments in this paper focused on dense models for two reasons: (1) to provide a fair comparison to prior low-rank pretraining methods, which were evaluated exclusively on dense architectures; and (2) to make sure that the approach scales to larger models. Having achieved these goals, a compelling next step is to extend it to Mixture-of-Experts (MoE) architectures (Shazeer et al., 2017), as the vast majority of state-of-the-art models rely on MoE architectures. Several other promising directions remain for future work. First, fine-tuning pretrained TSVD models on smaller, task-specific datasets is a natural next step. Second, investigating regularization strategies for Σ, to control the spectral norm of weight matrices and improve generalization. Third, exploring whether orthonormal matrices learned in smaller models can initialize corresponding matrices in larger models, while increasing the training efficiency.

7 Conclusion

This paper introduces TSVD, a novel framework designed to reduce the parameter count and computational overhead of LLM pretraining. To our knowledge, TSVD is the first approach to maintain strict weight orthonormality throughout training without incurring prohibitive overhead, enabled by QR decomposition, an optimized caching mechanism, and a rank selection heuristic. Empirical evaluations show that TSVD models match or outperform their full-parameter counterparts across pretraining metrics and evaluation benchmarks. TSVD reduces the FLOPs required for LLM pretraining, lowering wall-clock time and GPU cost. It also reduces memory requirements, enabling larger models on existing hardware or the use of lower-tier GPUs, further reducing pretraining costs. The TSVD framework provides a foundation for more efficient pretraining of LLMs.

AI USE STATEMENT

Generative AI tools were used solely for sentence editing and polishing, assisting with literature discovery, and writing minor code scripts. The authors reviewed and verified all AI-generated content, including text, literature citations, and code, to ensure accuracy and correctness. The authors accept full responsibility for all content presented in this work.

REPRODUCIBILITY STATEMENT

The complete source code, along with scripts to reproduce all main experimental results, preprocessing steps, and hyperparameter configurations, is included in the supplementary material (and will be publicly released upon acceptance). Datasets used in our experiments are publicly available, and detailed instructions for downloading and preprocessing them are included in the code repository.

REFERENCES

P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks, 2016. URL https://arxiv.org/abs/1511.06464.

Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns?, 2018. URL https://arxiv.org/abs/1810.09102.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023.

Tony F. Chan. Rank revealing QR factorizations. *Linear Algebra and its Applications*, 88-89: 67–82, 1987. doi: 10.1016/0024-3795(87)90103-0. URL https://doi.org/10.1016/0024-3795(87)90103-0.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus, 2021. URL https://arxiv.org/abs/2104.08758.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and Aurelien Rodriguez et al. The llama 3 herd of models. *arXiv*, 2407.21783, 2024.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, Sep 1936. ISSN 1860-0980. doi: 10.1007/BF02288367. URL https://doi.org/10.1007/BF02288367.

Adriana Fernandez-Lopez, Shiwei Liu, Lu Yin, Stavros Petridis, and Maja Pantic. Full-rank no more: Low-rank weight training for modern speech recognition models, 2024. URL https://arxiv.org/abs/2410.07771.

Sylvestre Gallot, Dominique Hulin, Jacques Lafontaine, et al. *Riemannian geometry*, volume 2. Springer, 1990.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep feedforward networks. *Deep learning*, 1:161–217, 2016.

Jiarui Gu. From gpt to llama: Tracing the growth of large language models. *Theoretical and Natural Science*, 142:45–56, 11 2025. doi: 10.54254/2753-8818/2025.DL29003.

Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Jawanpuria, and Bamdev Mishra. Sltrain: a sparse plus low-rank approach for parameter and memory efficient pretraining, 2024. URL https://arxiv.org/abs/2406.02214.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

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.

Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks, 2017. URL https://arxiv.org/abs/1709.06079.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N. Gomez. Exploring low rank training of deep neural networks, 2022. URL https://arxiv.org/abs/2209.13569.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolò Fusi. Initialization and regularization of factorized neural layers, 2022. URL https://arxiv.org/abs/2105.01029.

Jiaxi Li, Lu Yin, Li Shen, Jinjin Xu, Liwu Xu, Tianjin Huang, Wenwu Wang, Shiwei Liu, and Xilu Wang. Lost: Low-rank and sparse pre-training for large language models, 2025. URL https://arxiv.org/abs/2508.02668.

Zhizhong Li, Sina Sajadmanesh, Jingtao Li, and Lingjuan Lyu. Stella: Subspace learning in low-rank adaptation using stiefel manifold, 2026. URL https://arxiv.org/abs/2510.01938.

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.

Ziyue Liu, Ruijie Zhang, Zhengyang Wang, Mingsong Yan, Zi Yang, Paul Hovland, Bogdan Nicolae, Franck Cappello, Sui Tang, and Zheng Zhang. Cola: Compute-efficient pre-training of llms via low-rank activation, 2025. URL https://arxiv.org/abs/2502.10940.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey, 2025. URL https://arxiv.org/abs/2402.06196.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank riemannian optimization. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 7907–7936, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/16fafdc5e8ab22780873d607ce4387a8-Paper-Conference.pdf.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2026. URL https://arxiv.org/abs/2512.13961.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016. URL https://arxiv.org/abs/1606.06031.

Zeju Qiu, Simon Buchholz, Tim Z. Xiao, Maximilian Dax, Bernhard Schölkopf, and Weiyang Liu. Reparameterized llm training via orthogonal equivalence transformation, 2025. URL https://arxiv.org/abs/2506.08001.

Zeju Qiu, Lixin Liu, Adrian Weller, Han Shi, and Weiyang Liu. Poet-x: Memory-efficient llm training by scaling orthogonal transformation, 2026. URL https://arxiv.org/abs/2603.05500.

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. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

Dayana Savostianova, Emanuele Zangrando, Gianluca Ceruti, and Francesco Tudisco. Robust low-rank training via approximate orthonormal constraints, 2023. URL https://arxiv.org/abs/2306.01485.

Yara Shamshoum, Nitzan Hodos, Yuval Sieradzki, and Assaf Schuster. Compact: Compressed activations for memory-efficient llm training, 2025. URL https://arxiv.org/abs/2410.15352.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Yang Sui, Miao Yin, Yu Gong, Jinqi Xiao, Huy Phan, and Bo Yuan. Elrt: Efficient low-rank training for compact convolutional neural networks, 2024. URL https://arxiv.org/abs/2401.10341.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Xiuying Wei, Skander Moalla, Razvan Pascanu, and Caglar Gulcehre. Investigating low-rank training in transformer language models: Efficiency and scaling analysis, 2024. URL https://arxiv.org/abs/2407.09835.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick

von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL https://arxiv.org/abs/2203.03466.

Huanrui Yang, Minxue Tang, Wei Wen, Feng Yan, Daniel Hu, Ang Li, Hai Li, and Yiran Chen. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification, 2020. URL https://arxiv.org/abs/2004.09031.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL https://arxiv.org/abs/2403.03507.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2026. URL https://arxiv.org/abs/2303.18223.

A Why Orthonormality Helps

Consider a linear layer $W \in \mathbb{R}^{m\times n}$ parameterized in TSVD as $W = \frac{1}{\sqrt{r}} U\Sigma V^\top$ with $U^\top U = I_r$, $V^\top V = I_r$, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$, $\sigma_i > 0$. U and V are semi-orthogonal, i.e., points on the Stiefel manifolds $\mathrm{St}(m, r)$ and $\mathrm{St}(n, r)$ (App. B); all spectral scaling is carried by Σ, and $\|W\|_2 = \sigma_{\max}/\sqrt{r}$, where $\sigma_{\max} = \max_i \sigma_i$ and $\sigma_{\min} = \min_i \sigma_i$.

That constraining a linear map to be (semi-)orthogonal controls signal and gradient magnitude is a standard result, not specific to TSVD.

Proposition 1 (Signal and gradient stability in orthonormal TSVD)**.** *For any input $x \in \mathbb{R}^n$ and any back-propagated signal $\delta \in \mathbb{R}^m$,*

$$\|Wx\|_2 \le \frac{\sigma_{\max}}{\sqrt{r}}\|x\|_2, \qquad \|W^\top\delta\|_2 \le \frac{\sigma_{\max}}{\sqrt{r}}\|\delta\|_2,$$

with equality when $x \in \mathrm{span}(V)$ or $\delta \in \mathrm{span}(U)$, respectively, and a matching lower bound of $\sigma_{\min}/\sqrt{r}$ on that subspace.

This is a direct consequence of operator-norm submultiplicativity for (semi-)orthogonal maps. Arjovsky et al. (2016) proved the same bound for square orthogonal/unitary matrices, showing that it lets a recurrent network avoid vanishing and exploding gradients by construction rather than by clipping; Huang et al. (2017) extended the argument to rectangular, semi-orthogonal weight matrices in feedforward and convolutional networks, further showing that an orthogonal weight matrix preserves the mean and covariance of its input's distribution and, by fixing $r(r+1)/2$ degrees of freedom per factor, regularizes the network relative to an unconstrained factorization. The proof of Proposition 1 is identical in structure to Huang et al.'s proof (and omitted here). All amplification and attenuation in TSVD is therefore due to Σ alone; keeping U, V orthonormal for the full training run, rather than only at initialization, is what makes this bound hold throughout training rather than just at the start.

A common alternative is a soft penalty such as $\lambda\|W^\top W - I\|_F^2$ (Bansal et al., 2018), which only approximately enforces the constraint and trades off against the task loss. The exact alternative, Riemannian optimization on the Stiefel manifold, is stable but conventionally requires an SVD-based

retraction at every step (App. B). TSVD instead reparameterizes $U_k = \mathrm{qf}(A_{U_k})$ and $V_k = \mathrm{qf}(A_{V_k})$ via QR, the same reparameterize-rather-than-penalize strategy as Huang et al. (2017)'s orthogonal weight normalization, at the lower cost of a QR decomposition in place of an eigendecomposition or SVD.

An unconstrained low-rank factorization $W = AB^\top$ has no such guarantee: for any invertible c, $AB^\top = (cA)(c^{-1}B)^\top$ represents the identical W, so the individual factor gradients can grow and shrink without bound even while W's own gradient does not. Fixing $U^\top U = V^\top V = I_r$ removes this ambiguity, since the norm of each factor is fixed by construction.

Recent work on LLM pretraining specifically reaches the same conclusion: Qiu et al. (2025) show that reparameterizing weight matrices through orthogonal transformations gives provable control over their singular-value spectrum, tied to standard spectral-complexity generalization bounds. TSVD's use of orthonormal U and V is best read in this light: an established, well-conditioned parameterization whose bounded spectral norm ($\sigma_{\max}/\sqrt{r}$, Proposition 1) carries the same generalization benefit, now validated at full LLM-pretraining scale as well, combined with rank truncation for parameter savings.

B Comparison to Orthonormal Representations

Existing approaches to maintaining orthonormality fall into three categories: (1) initializing weight matrices as orthonormal, and not compensating for the inevitable deviation from orthonormality during unconstrained optimization (Fernandez-Lopez et al., 2024); (2) employing soft regularizers (Yang et al., 2020), which introduce additional hyperparameters for encouraging orthonormality; and (3) periodically projecting weights back onto the orthonormal manifold (Mo et al., 2025). The latter approach is most closely related to the TSVD method and is discussed in this subsection.

The LORO method (Mo et al., 2025) utilizes Riemannian optimization to maintain weight constraints. As the mathematical context, a manifold (Gallot et al., 1990) is a topological space that is locally flat but could be curved globally, such as a plane, sphere, torus, or saddle-shaped surface. A Riemannian manifold (Gallot et al., 1990) provides a way to measure distances, angles, etc. using a Riemannian metric. In Riemannian optimization (Absil et al., 2008; Boumal, 2023), variables are constrained to lie on a smooth manifold $\mathcal{M}$ (such as a sphere), rather than a flat Euclidean space $\mathbb{R}^n$. The core concepts for Riemannian optimization are thus:

- **The Manifold ($\mathcal{M}$):** The search space.
- **Tangent Space ($T_x\mathcal{M}$):** At any point $x \in \mathcal{M}$, the tangent space is a linear approximation of the manifold's local geometry. Gradients are computed in this space.
- **Riemannian Gradient:** The projection of the standard Euclidean gradient onto the tangent space, denoted as $\mathrm{grad}\, f(x) = \mathrm{P}_{T_x\mathcal{M}}(\nabla f(x))$.
- **Retraction:** A mapping $R_x : T_x\mathcal{M} \to \mathcal{M}$ that moves an element from the tangent space back onto the manifold.

LORO searches on the **low-rank matrix manifold**, defined as the set of matrices with a fixed rank r. Each matrix W has a *unique representation*, eliminating redundancy during the search process. However, the retraction operation requires an SVD operation, which is computationally expensive. The cost, in the number of FLOPs, for a matrix $W \in \mathbb{R}^{m\times n}$ is

$$mn^2 + n^3. \tag{5}$$

To manage this cost, LORO executes the SVD retraction sparingly, every $K = 500$ steps. In contrast, the QR decomposition in TSVD optimizes over the **Stiefel manifold**, i.e. the set of all matrices with orthonormal columns, denoted as

$$St(n, r) = \{X \in \mathbb{R}^{n\times r} \mid X^\top X = I_r\}. \tag{6}$$

This approach may result in *non-unique representations*, since different factors (U, Σ, V) can represent the same product W, which could be a limitation. However, the QR decomposition is significantly more efficient than the SVD operation (compare Eq. 3 and 5). Thus, the assumption of TSVD is that the computational efficiency of the projection onto the Stiefel manifold outweighs the

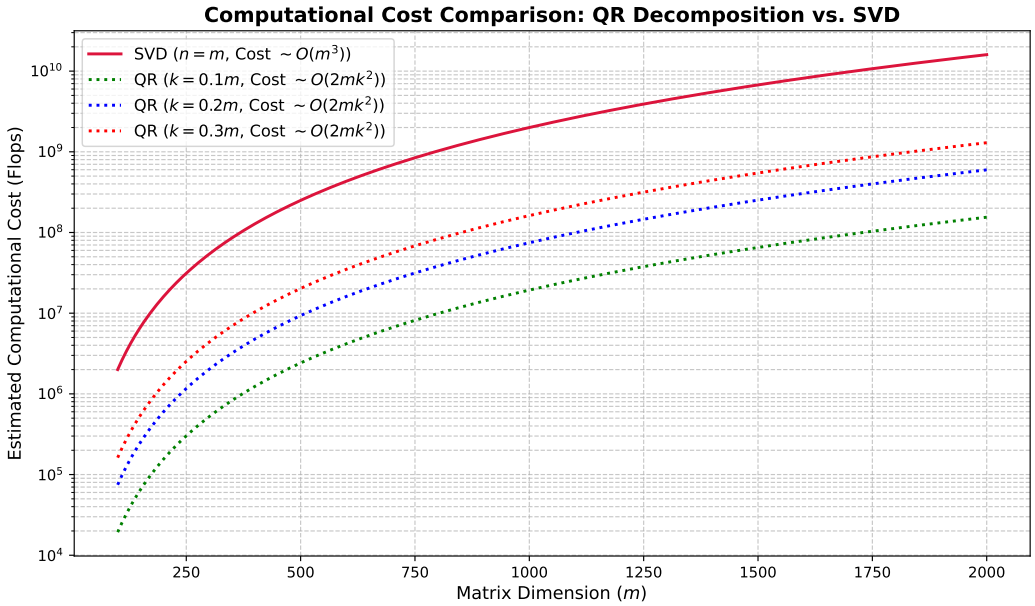


Figure 3: Comparison of the computational cost for QR decomposition of $W_q \in \mathbb{R}^{m\times k}$ matrix with various k values and SVD for $W_s \in \mathbb{R}^{m\times n}$ matrix with $m = n$. The plot illustrates that QR decomposition requires significantly fewer FLOPs.

theoretical benefits of unique representation on the fixed-rank manifold. Indeed, the results in this paper suggest that, given the massive scale of current and future LLMs, the significant reduction in retraction overhead justifies the inherent redundancy in factorized representation, making TSVD a more practical approach for large-scale optimization than other current methods.

C Full-Parameter vs Rank-k Memory and Compute Comparison

This section compares the memory and compute requirements of full-parameter weight matrices and rank-k representation as used in TSVD. It shows that the TSVD representation reduces both the number of FLOPs and the memory requirements. For reference, the formulas for memory and compute comparison of full rank vs rank-k are summarized in Tab. 6.

Memory comparison. For a full-parameter weight matrix $W \in \mathbb{R}^{m\times n}$, mn elements are stored, where each element can take a few bytes depending on their format (4 bytes for FP32, or 2 bytes for FP16, etc.).

For $W \approx U_k\Sigma_k V_k^\top$ with $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and $V_k \in \mathbb{R}^{n\times k}$, mk, k, and nk elements are stored for unconstrained parameters, i.e., $k(m+n+1) \approx k(m+n)$, and the same amount for constrained parameters (i.e. QR decomposition on U_k and V_k, and softplus on Σ). Hence, the total memory requirement is $2k(m+n)$.

As a result, rank-k needs less memory than full rank when

$$2k(m+n) < mn, \tag{7}$$

that is,

$$k < \frac{mn}{2(m+n)}. \tag{8}$$

Example. To compare the memory requirements of full-parameter and rank-k weight matrices, suppose $k = 128$, and $n = m = 1024$. Full-parameter weight matrix requires memory for $m * n = 1,048,576$ floating point values. Rank-k matrix requires memory for $2k(m+n) = 2*128*(1024+1024) = 524,288$ floating point values, which is a 50% reduction in memory requirements.

Table 6: Resource comparison for weight matrix $W \in \mathbb{R}^{m\times n}$ and $W \approx U_k\Sigma_k V_k^\top$ with $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and $V_k \in \mathbb{R}^{n\times k}$, where B is the batch size, S is the sequence length, and t is the number of gradient accumulation steps.

	Full Rank	**Rank-*k***
Memory	mn	$2k(m+n)$
Flops	$6tBSmn$	$tBS(6k(m+n)) + 2k^2(m+n)$

Compute Comparison. For an input $x \in \mathbb{R}^n$ and a weight matrix $W \in \mathbb{R}^{m\times n}$, computing $y = Wx$ in the forward pass requires mn multiply-add operations thus $2mn$ FLOPs. During backpropagation, computing both the weight gradient ∂W and the input gradient ∂x requires approximately $4mn$ FLOPs. Therefore, the total cost of one forward pass and one backward pass through the layer is approximately $6mn$ FLOPs.

For $W \approx U_k\Sigma_k V_k^\top$ with $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and $V_k \in \mathbb{R}^{n\times k}$, computing $y = Wx$ in the forward pass costs

$$2kn \text{ (for } z = V_k^\top x) \;+\; k \text{ (for } z' = \Sigma_k z) \;+\; 2mk \text{ (for } y = U_k z'). \tag{9}$$

We ignore the FLOPs for Σ_k, both in the forward and backward pass, as it is negligible due to its efficient computation as element-wise vector product. The total cost for the forward pass is therefore $\approx 2k(n+m)$ FLOPs. Computing the QR decomposition in the forward pass costs $\approx 2nk^2$ FLOPs for an n by k matrix, but it only happens once per update step as the calculated value is cached and re-used. During backpropagation, computing both the weight gradient ∂W and the input gradient ∂x requires approximately $4nk$ FLOPs for V_k and $4mk$ for U_k, totaling $4k(n+m)$. Therefore, the total cost of one forward pass and one backward pass through the layer is approximately $6k(m+n) + 2k^2(m+n)$ FLOPs.

In LLM pretraining, gradient accumulation is a standard practice. This further reduces the influence of QR decomposition on the overall FLOPs. Given a batch size of B, sequence length of S, and number of accumulation steps of t, the number of FLOPs for a weight matrix $W \in \mathbb{R}^{m\times n}$ is

$$6tBSmn, \tag{10}$$

and the number of FLOPs for $W \approx U_k\Sigma_k V_k^\top$ with $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and $V_k \in \mathbb{R}^{n\times k}$ is

$$tBS(6k(m+n)) + 2k^2(m+n), \tag{11}$$

where the FLOPs for Σ_k are ignored (because it is a negligible cost). The second term is the QR decomposition FLOPs for V_k and U_k.

Example. To compare the number of FLOPs for of full rank and rank-*k*, suppose $B = 512$, $S = 256$, $k = 128$, and $n = m = 1024$. We calculate the number of FLOPs for three numbers of accumulation steps: $t = 1, 8$, and 16.

The *full-parameter* weight matrix incurs

$$824.63 \times 10^9, \quad 6.59 \times 10^{12}, \quad 13.19 \times 10^{12} \tag{12}$$

FLOPs for $t = 1$, $t = 8$ and $t = 16$ respectively.

Rank-k matrices incus the following FLOPs at accumulation step $t = 1$, where the second term is the QR decomposition FLOPs:

$$206.15 \times 10^9 + 67.1 \times 10^6. \tag{13}$$

For accumulation step $t = 8$, it is

$$1.64 \times 10^{12} + 67.1 \times 10^6. \tag{14}$$

And, for accumulation step $t = 16$, it is

$$32.9 \times 10^{12} + 67.1 \times 10^6. \tag{15}$$

Table 7: Major architectural parameters of open-weights model families. Llama-2, Llama-3.1 and OLMo 3 models of different sizes are compared with respect to their embedding dimension d_1, their hidden MLP dimension d_2, the number of layer n_{layers}, number of query heads $n_{\text{heads,q}}$ and number of key and value heads $n_{\text{heads,kv}}$. Grey numbers denote multiples compared to the smallest model within the same model family. Except for Llama-2 7B to 13B, parameters change non-proportionally. Most notable, $n_{\text{heads,kv}}$ often decrease due to GQA. Fig. 8 illustrates how normalized compression ranks are influenced by these parameters, demonstrating a consistent scaling trend that enables their prediction in larger models from smaller ones, with the exception of key and value projection ranks, which decrease due to GQA.

Parameter	Llama-2					Llama-3.1			OLMo 3		
num_parameters	7B	13B	× 7B	70B	× 7B	8B	70B	× 8B	7B	32B	× 7B
d_1	4096	5120	1.25	8192	2	4096	8192	2	4096	5120	1.25
d_2	11008	13824	1.26	28672	2.6	14336	28672	2	11008	27648	2.5
n_{layers}	32	40	1.25	80	2.5	32	80	2.5	32	64	2
$n_{\text{heads,q}}$	32	40	1.25	64	2	32	64	2	32	40	1.25
$n_{\text{heads,kv}}$	32	40	1.25	8	0.25	8	8	1	32	8	0.25

Conclusion. Thus, (1) Even for accumulation step $t = 1$, the QR decomposition FLOPs are a small proportion of the overall FLOPs, and (2) As the number of accumulation steps increases to 8 or 16, the QR decomposition FLOPs become an even smaller proportion of the overall FLOPs. Rank-k weight matrices + QR decomposition as used in the TSVD framework thus offer a strong profile in terms of memory and compute for sufficiently small k.

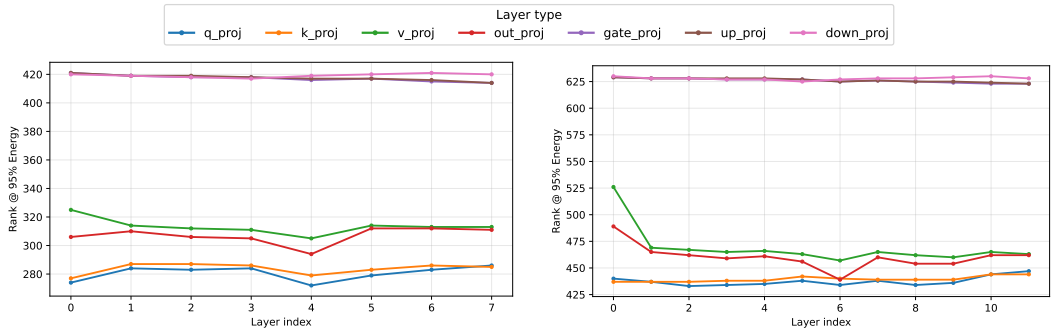
D Compression Rank Analysis for Open-Weight Model Families

Sec. 4 in the main paper analyzed the compression ranks (Eq. 4) of individual linear projections (query, key, value, out for the attention layer; gate, up, down for the MLP layer) within individual transformer layers for the full-parameters models trained for benchmarking in Sec. 5.

To assess the generality of the findings and highlight potential deviations, the following analysis extends to large, publicly available open-weight model families: Llama-2 (Touvron et al., 2023b), Llama-3.1 (Dubey et al., 2024), and Olmo 3 (Olmo et al., 2026). The architectural parameters of these models are listed in Tab. 7. The query heads and key/value heads were treated separately, because many modern architectures employ grouped query attention (GQA) (Ainslie et al., 2023).

Results. Compression ranks per transformer layer and projection type are shown in Fig. 5 - 7 for completeness. Except for the very first layer, they are relatively consistent per projection type. Following the analysis in Sec. 4 normalized compression ranks (averaged over all transformer layers) are reported in Fig. 8, which are discussed in detail in the following. For Llama-2 7B and 13B, the same constant scaling behavior as in the analysis on the benchmark models in Sec. 4 is observed. This result was expected given that all major architectural parameters are scaled proportionally. In contrast, the 70B variant adopts GQA, reducing the number of key and value heads by a factor of four. This change is directly reflected in the compression ranks of key and value projections, which decrease relative to the query and output projections. Additionally, the MLP intermediate dimension is scaled disproportionately, resulting in higher compression ranks for the gate, up, and down projections. For Llama-3.1, the number of key and value heads remain constant while most other parameters double from 8B to 70B. Correspondingly, the normalized compression ranks of key and value heads are approximately halved. The transition from OLMo 3 7B to 32B exhibits a similar pattern to that observed between Llama-2 13B and 70B.

Conclusion. Overall, we find that the findings of Sec. 4 generalize to common open-weight families. However, upon changing critical architectural parameters disproportionally when scaling to larger models, e.g. through introducing GQA, rank selection needs to be adapted accordingly.



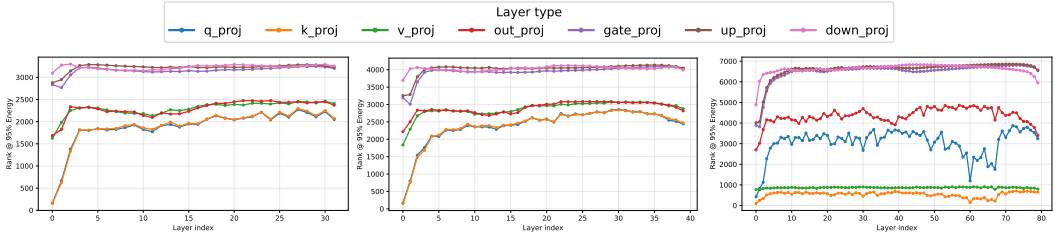
(a) 60M (8 layers)

(b) 130M (12 layers)

(c) 350M (24 layers)

(d) 1B (24 layers)

Figure 4: Rank required to capture 95% of the spectral energy for each weight type (query, key, value, out, gate, up, and down projections), plotted across transformer layers, for four models from the benchmark of Sec. 5.2.1: (a) 60M (8 layers), (b) 130M (12 layers), (c) 350M (24 layers), and (d) 1B (24 layers) parameters. Within every model, the required rank is nearly constant across layer index once the first layer is excluded, and the relative ordering of weight types (e.g., gate/up/down projections consistently needing higher rank than query/key/value/out) is preserved across scales. This uniformity is what justifies assigning a single compression rank per weight type, rather than tuning a separate rank for every individual layer, without sacrificing the accuracy of the energy-based rank selection.

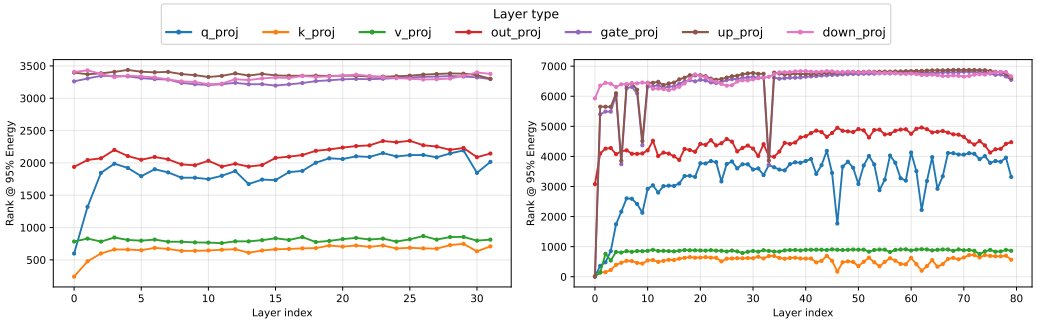


(a) 7B (32 layers)

(b) 13B (40 layers)

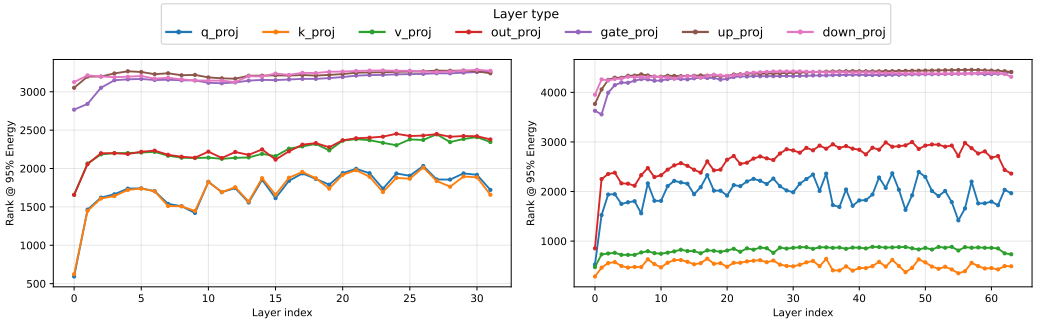
(c) 70B (80 layers)

Figure 5: Compression ranks for different weight types (query, key, value, out, gate, up and down projections) across transformer layers for different model sizes of the Llama-2 model family. Except for the first layers, compression ranks are relatively uniform.



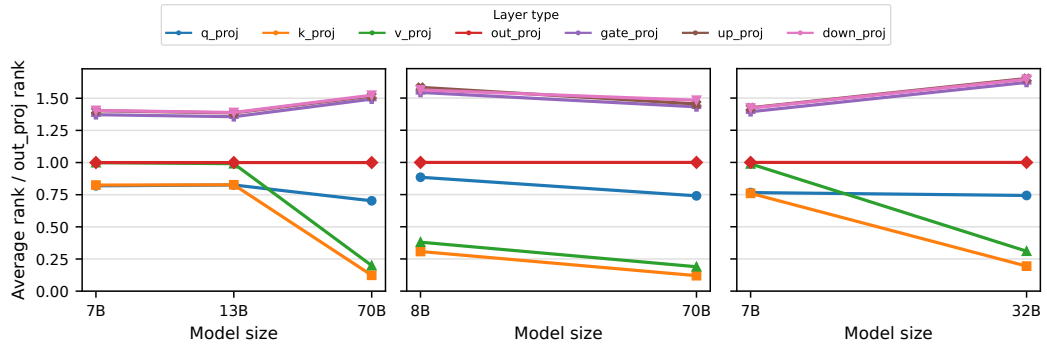
(a) 8B (32 layers) (b) 70B (80 layers)

Figure 6: Compression ranks for different weight types (query, key, value, out, gate, up and down projections) across transformer layers for different model sizes of the Llama-3.1 model family. Except for the first layers, compression ranks are relatively uniform.



(a) 7B (32 layers) (b) 32B (64 layers)

Figure 7: Compression ranks for different weight types (query, key, value, out, gate, up and down projections) across transformer layers for different model sizes of the OLMo-3 model family. Except for the first layers, compression ranks are relatively uniform.



(a) Llama 2 (b) Llama 3.1 (c) OLMo 3

Figure 8: Normalized compression ranks (averaged over layers) of different model sizes within different model families. Ranks remain similar except that key and value projection ranks often decrease due to GQA.