EINSORT: SORTING IS ALL WE NEED FOR TENSORIZING LLM

Anonymous authors
Paper under double-blind review

ABSTRACT

Tensor networks provide efficient representations for compressing large neural networks, but identifying useful low-rank structure in foundation-model tensors remains challenging. We propose Einstein sorted sum (**EinSort**), an adaptive tensorization method that exposes low-rank structure through reversible index ordering before tensor decomposition. We prove that, under stated regularity assumptions, row-wise sorting of i.i.d. entries converges to a rank-one quantile template with relative Frobenius error $O_p(N^{-1/2})$; sliced or shared permutations then limit metadata overhead in practical settings. We evaluate EinSort in two deployment regimes: KV-cache compression for language models and weight compression for vision-language and vision-language-action models. Across WikiText-2 language modeling, GSM8K mathematical reasoning, TextVQA visual reasoning, and LIBERO robot manipulation, EinSort consistently improves downstream quality relative to unsorted tensor decomposition and the practical low-rank baselines evaluated in each setting.

1 INTRODUCTION

Large foundation models achieve remarkable performance across a variety of tasks (Katz et al., 2024). Nonetheless, their large parameter counts and memory requirements make deployment expensive (Schwartz et al., 2020). Numerous compression techniques have therefore been proposed (Xu & McAuley, 2023; Zhu et al., 2024; Bai et al., 2024a), including partial activation (Lin et al., 2024a), weight pruning (Bai et al., 2024b), quantization (Wang et al., 2024a), knowledge distillation (Hwang et al., 2024), and rank reduction (Yuan et al., 2023; Hwang et al., 2024; Saxena et al., 2024).

Among them, tensor decomposition methods (Lebedev et al., 2014; Sidiropoulos et al., 2017) provide a flexible framework for representing large neural networks with significantly fewer parameters (Novikov et al., 2015; Denil et al., 2013; Sainath et al., 2013; Jaderberg et al., 2014). Existing tensor network approaches (Roberts et al., 2019; Huggins et al., 2019) primarily focus on optimizing topology, contraction, and tensor ranks (Luo et al., 2024). However, one important degree of freedom has received relatively little attention: the ordering of tensor indices. In this work, we show that index ordering can substantially alter the effective rank structure of tensors. In particular, simple sorting operations can expose latent low-rank structure that is otherwise hidden in pretrained LLM weights and KV caches. Motivated by this observation, we propose Einstein sorted sum (**EinSort**), a tensorization framework that augments Einstein summation with reversible permutation operators.

Our contributions are summarized as follows: (1) We introduce EinSort, a tensor decomposition framework with adaptive index ordering for compression of foundation-model tensors. (2) We prove an asymptotic rank-one result for row-wise sorting and empirically show large reductions in effective tensor rank. (3) We propose practical low-overhead permutation schemes for tensor compression. (4) We demonstrate gains for LLM KV-cache compression and VLM/VLA weight compression across language modeling, mathematical reasoning, visual reasoning, and robot manipulation tasks.

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.

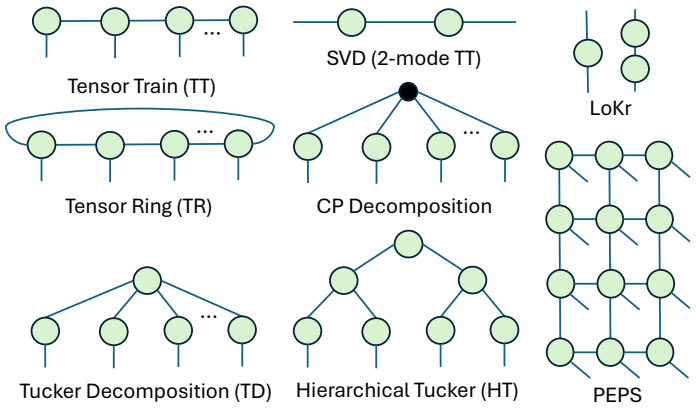


Figure 1: Various tensor networks in tensor diagrams.

2 TENSORIZING LLMS FOR COMPRESSION

2.1 EINSTEIN SUMMATION AS GENERAL TENSOR NETWORKS

Model decomposition is a widely used approach to compress large models, e.g., low-rank factorization via singular-value decomposition (SVD). More generally, model tensorization can be viewed through a tensor-network framework that includes tensor train (TT), tensor ring (TR), canonical polyadic (CP), Tucker decomposition (TD), hierarchical Tucker (HT), and projected entangled-pair states (PEPS), as depicted in Fig. 1. A broad class of tensor networks can be represented by Einstein summation (**einsum**), a compact notation for multilinear algebra. Given tensor cores with appropriate shapes, a tensor network can be expressed in einsum form; for example, a 4-mode TT is $X_{i,j,k,l} = \sum_{p,q,r} A_{i,p} B_{p,j,q} C_{q,k,r} D_{r,l}$, where $A_{i,p}$, $B_{p,j,q}$, $C_{q,k,r}$, and $D_{r,l}$ are factorized cores. The indices $\{i, j, k, l\}$ are physical site indices, whereas $\{p, q, r\}$ are bond indices whose ranges determine the tensor ranks. Reducing the tensor ranks can considerably reduce the memory footprint required to represent a tensor X. For example, if X has shape (d, d, d, d), the dense representation contains d^4 parameters, whereas a TT representation with maximum rank R uses $2d(R+1)R$ parameters. When $R = \rho d$ for $0 < \rho < 1$, this scales cubically as $2\rho d^2(\rho d + 1)$.

Several examples are listed in the pseudocode below:

```
TT = torch.einsum("ip,pjq,qkr,rl->ijkl", A, B, C, D)
TR = torch.einsum("mip,pjq,qkr,rlm->ijkl", A, B, C, D)
CP = torch.einsum("r,ri,rj,rk->ijk", A, B, C, D)
TD = torch.einsum("pqr,pi,qj,rk->ijk", A, B, C, D)
X = einops.rearrange("i p j q -> (i p) (j q)", TT) # unfolding 4-mode to
    2-mode
```

We use `einops` for folding and unfolding. See Appendix B for more examples. As shown above, einsum is a powerful tool for manipulating multilinear operations. It enables tensor-network topologies to be specified compactly through the einsum equation (a string such as `"ip,pj->ij"`) and appropriately shaped tensor cores.

Contraction-order optimization can also reduce computational cost. For example, for a 4-mode TT with site dimension $d = 32$ and rank $R = 20$, a naïve left-to-right contraction requires 3.4×10^{10} FLOPs, whereas an optimized contraction order reduces the cost to 4.4×10^7 FLOPs—nearly three orders of magnitude—using `opt_einsum`. Moreover, a factorized weight can be contracted directly with activations without first materializing the dense matrix; Appendix B gives representative contraction paths. These arithmetic savings do not by themselves imply wall-clock speedups, motivating the measured throughput study in Sec. 3 and the kernel limitations discussed in Sec. 3.5.

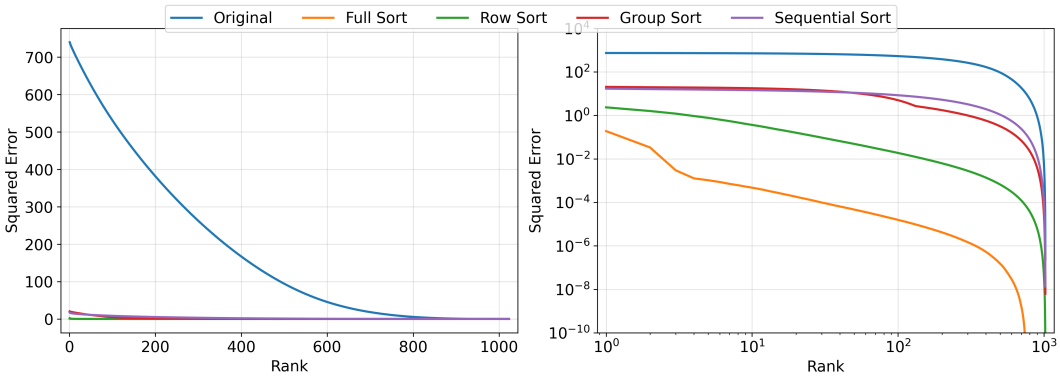


Figure 2: Tensorization error vs. rank for value-projection (`v_proj`) weights in the first layer of Qwen3-0.6B. Left: linear scale; right: log–log scale.

2.2 Gauge fixing

A large tensor can be factorized using many tensor-network structures, although exact solutions generally do not exist except in special cases such as matrix SVD. Because einsum is multilinear, gradient-based optimization can be used to minimize the tensorization error. For TT, recursive SVD over bond cuts provides a useful initialization, while alternating least squares (ALS) is often used to refine the estimate. We use `tensorly`.

The tensorization solution is not unique because it has **gauge** freedom (Evenbly, 2018). For example, consider factorizing X as $X \simeq AB$ for low-rank cores A and B. Truncated SVD provides optimal factors, with A derived from the left singular vectors and B from the right singular vectors of X. However, any full-rank junction matrix J can be inserted together with its inverse without changing the product, i.e., $AB = A(JJ^{-1})B = (AJ)(J^{-1}B) = A'B'$. Hence, there are infinitely many equivalent solutions for A and B. Fixing the gauge freedom, for example through canonicalization, can improve the numerical stability of tensor factorization (Phan et al., 2020).

Beyond numerical stability, choosing a suitable gauge at a junction can reduce the number of stored core parameters (Koike-Akino et al., 2026a). For instance, a junction can be chosen so that adjacent tensor cores take block-identity form. For a 4-mode TT, this can further reduce the parameter count to $2d(R+1)R - 2R^2$.

2.3 Index ordering

Using einsum, tensorization can adapt the contraction equation, core shapes, folding/unfolding operations, numerical solver, and gauge to a target tensor (e.g., pretrained weights). However, few tensor-network design methods optimize tensor indexing beyond a simple axis permutation such as `"ijkl->kjli"`. In this paper, we show that index ordering can play a critical role in exposing low-rank structure in tensors.

Consider first a toy example with a 2×2 tensor W_0 and its reordered version W_π:

$$W_0 = \begin{array}{c} \overset{j}{\longrightarrow} \\ i \downarrow \end{array} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \; W_\pi = \begin{array}{c} \overset{j \oplus i}{\longrightarrow} \\ i \downarrow \end{array} \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}. \tag{1}$$

(swap)

The first matrix has singular values $[3, 1]$, whereas the second has $[\sqrt{10}, 0]$ and rank 1. Here, W_π uses the index reordering $j \mapsto j \oplus i$ for indices (i, j). This index reordering is analogous to an **entangling** operation in a quantum tensor network: the example corresponds exactly to a controlled-NOT (CNOT) gate. Appendix L discusses this connection in more detail.

We next show empirical results suggesting that sorting can substantially reduce the required rank. Fig. 2 shows the squared reconstruction error $\|W - \hat{W}\|_{\mathrm{F}}^2$, where W is the original weight matrix and $\hat{W}$ is its reconstruction using a 2-mode TT. Specifically, we analyze the singular-value spectrum

of pretrained LLM weights. We illustrate full and row sorting in the pseudocode below; Fig. 2 additionally includes group and sequential sorting:

```
S = torch.linalg.svdvals(W)
loss = S.square().flip(-1).cumsum(-1)
Ws, _ = torch.sort(W.view(-1)) # full-sort
S = torch.linalg.svdvals(Ws.reshape_as(W))
Wr, _ = torch.sort(W, dim=-1) # row-sort
```

As shown in Fig. 2, increasing the rank improves reconstruction accuracy, but the improvement is slow when the original weights are decomposed by SVD without sorting. Notably, tensor sorting substantially reduces reconstruction error, enabling near-rank-one reconstruction. From the log–log plot, row-wise sorting has higher error than full sorting, but remains at least an order of magnitude lower than that without sorting. Therefore, even imperfect sorting can substantially improve reconstruction accuracy through index reordering.

2.4 Theoretical analysis

We next give a theoretical explanation that more closely matches the structured row-wise sorting used by EinSort.

Lemma 2.1 (Asymptotic rank-one structure of row-wise sorted matrices)**.** *Let $X \in \mathbb{R}^{M\times N}$ have i.i.d. entries from a continuous distribution with quantile function $Q = F^{-1}$, under the regularity assumptions stated in Appendix D.3. Let S be obtained by sorting each row in increasing order, define $q_j = Q((j+1)/(N+1))$, and let 1_M denote the length-M all-ones vector. Then*

$$\frac{\|S - 1_M q^\top\|_{\mathrm{F}}}{\|1_M q^\top\|_{\mathrm{F}}} = O_p(N^{-1/2}), \tag{2}$$

so S approaches the rank-one template $1_M q^\top$ in relative Frobenius norm as $N \to \infty$.

The result applies beyond the uniform model and directly explains the mechanism of row sorting: after ordering, each row concentrates around the same deterministic quantile curve q. Pretrained weights need not satisfy the i.i.d. assumptions, so Lemma 2.1 is a mechanism rather than a guarantee for every learned tensor; Fig. 2 provides the corresponding empirical evidence on LLM weights. Appendix D.3 gives the full assumptions and proof, while Appendix D.1 retains the earlier full-sort uniform special case, which gives an asymptotic rank-at-most-3 approximation.

Why coupled ordering matters. Ordinary independent axis permutations cannot create this rank reduction. For row and column permutation matrices P_{r} and P_{c}, $\sigma(X) = \sigma(P_{\mathrm{r}} X P_{\mathrm{c}})$ and hence the matrix rank is unchanged. Useful EinSort orderings therefore couple indices or depend on tensor values within a slice, as in row-wise sorting or the CNOT-like map $j \mapsto j \oplus i$ in Eq. (1), rather than merely relabeling rows and columns. Appendix D.4 provides the proof and discusses this distinction in more detail.

2.5 EinSort: Index-ordered tensor networks

These results motivate **EinSort**, a tensor-network framework that incorporates sorting into tensor decomposition. Conceptually, we use a reversible permutation operation $\pi[\cdot]$ inside tensor decomposition. Let $\mathcal{T}_\theta[\cdot]$ denote a tensor decomposition whose hyperparameters θ determine the einsum equation, shapes, topology, and related design choices. The sorted tensor decomposition is written as $\mathcal{T}_\theta^\pi := \pi^{-1}[\mathcal{T}_\theta[\pi[X]]]$. This formulation extends einsum-based tensorization to expose low-rank structure.

Although sorting may enable tensor cores with lower rank, the memory overhead of storing permutations is not negligible. In fact, sorting L values requires at most $\lceil \log_2(L!) \rceil$ bits using factoradic/Lehmer code (Lehmer, 1960). For example, full sorting for a tensor of shape $(1024, 1024)$ requires roughly 18.6 bits per entry, which often exceeds the memory footprint of the original tensors in FP16. (Row-wise sorting requires 8.6 bits.) Therefore, the method must balance sorting accuracy against tensor-rank reduction and permutation overhead.

We introduce a simple sorting method that reuses a shared permutation across multiple tensor slices. For a 4-mode tensor X with site dimension d, we may share a permutation across slices of the first

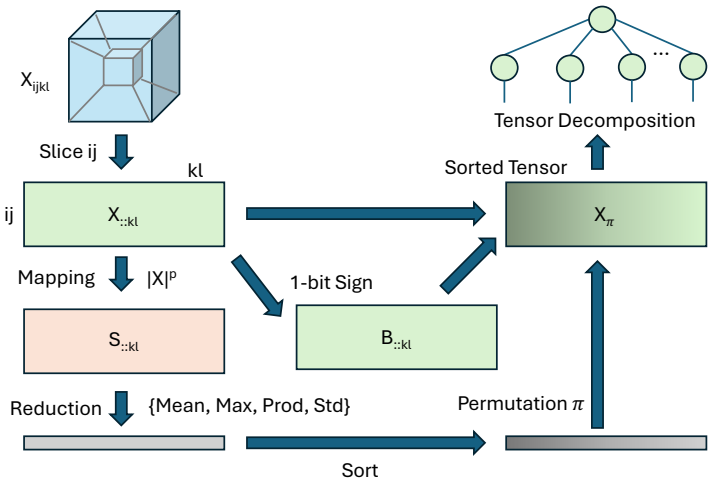


Figure 3: Sliced sorting with nonlinear score mapping and reduction. Non-negative tensor decomposition retains one bit of sign information.

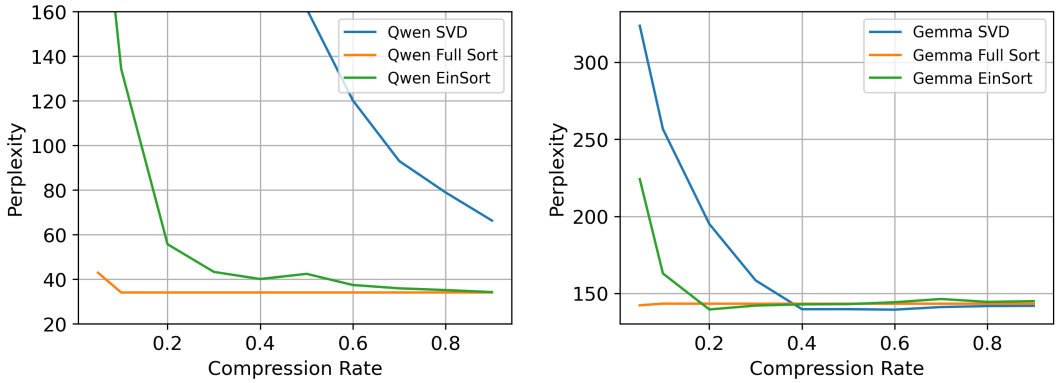


Figure 4: WT2 PPL under KV-cache compression for Qwen3-0.6B and Gemma3-4B.

mode, using statistics from $X_{:,j,k,l}$ to reorder the last three modes. This reduces permutation storage from $\lceil \log_2(d^4!) \rceil$ to $\lceil \log_2(d^3!) \rceil$ bits, i.e., by more than a factor of d. The slicing configuration therefore provides a direct way to control the permutation-memory overhead. For example, with $d = 32$, we have $\lceil \log_2(d^3!) \rceil / d^4 \simeq 0.4$ bits per weight. For the sorting metric, we consider several nonlinear mappings and reduction operators, including power scaling with mean, maximum, minimum, median, standard-deviation, and product reductions. An example is given in the pseudocode below.

```
S = X.abs().pow(p) # nonlinear map score
S = S.std(dim=[0], keepdim=True).expand_as(X) # std reduction at the
    first mode
perm = torch.argsort(S.flatten(1), dim=-1)
X = X.flatten(1).gather(-1,perm).view_as(S)
```

We search over folding, slicing modes, power exponents, and reduction operators to improve sorting accuracy. Besides the sorting metric, we use non-negative tensorization to further improve reconstruction accuracy. Specifically, we map the original tensor to non-negative values before decomposition. To recover negative values, we retain one sign bit per original element. This construction is illustrated in Fig. 3. Further details and pseudocode are provided in Appendix E.

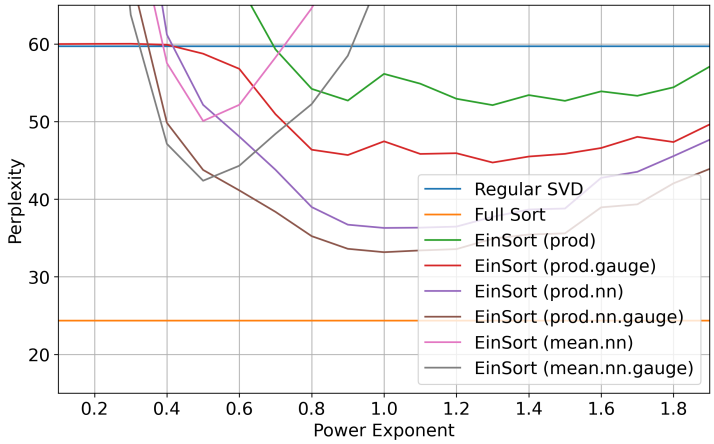


Figure 5: PPL vs. power exponent for Qwen3-1.7B at 50% compression.

Table 1: Reduction comparison at each operator's best power exponent.

Reduction	mean	max	min	median	std	prod
PPL	42.38	40.25	52.70	44.21	46.79	**33.17**
pow(p)	0.5	0.5	0.7	0.5	0.6	1.0

3 EXPERIMENTS

We evaluate EinSort in two complementary compression regimes: KV-cache compression for language models and model-weight compression for vision-language models (VLMs) and vision-language-action models (VLAs). The main paper reports evaluations on WikiText-2 (WT2) language modeling (Merity et al., 2016), GSM8K mathematical reasoning (Cobbe et al., 2021), TextVQA visual reasoning (Singh et al., 2019), and LIBERO robot manipulation (Liu et al., 2023a); Appendix R adds further LLM variants and detailed protocols. Model, dataset, and library details are provided in Appendices S, T, and U.

3.1 KV-CACHE COMPRESSION FOR LANGUAGE MODELING

We first evaluate Qwen3 (Yang et al., 2025) and Gemma3 (Gemma Team et al., 2025) on WT2. We partition the dataset into 64-token segments for prefilling and decompose the KV cache for decoding up to 64 new tokens. For sliced sorting, we use the KV-head axis as the reduction axis. For Qwen3 models with eight KV heads, the permutation-metadata cost can be at least eight times smaller than with full sorting. Under the 64-vector stacking protocol used here, the sliced permutation costs about 1.44 bits per KV-cache element for Qwen3-0.6B/1.7B and 3.14 bits for Gemma3-4B. The Phi-3.5-mini setting in Sec. 3.2 uses only 0.35 bits per KV-cache element. Full sorting is therefore used as a diagnostic upper bound rather than a storage-efficient deployment configuration.

Fig. 4 shows perplexity (PPL) for Qwen3-0.6B and Gemma3-4B-it when the KV cache is compressed. Full sorting approaches uncompressed performance. PPL degrades rapidly under regular SVD without sorting, whereas EinSort maintains substantially better PPL. Gemma3 is more tolerant to compression, although its uncompressed PPL is higher than Qwen3's.

Fig. 5 studies nonlinear power scaling. An exponent near 1 performs best for product reduction, suggesting that absolute-value mapping is sufficient without additional power scaling, whereas mean reduction performs best around 0.5. Gauge fixing improves accuracy by eliminating parameter redundancy, and non-negative tensorization further improves performance at the cost of one sign bit per element. Table 1 compares reduction rules at their best-performing power exponents; product reduction performs best in this setting, whereas the other reductions favor exponents around 0.5–0.7.

Table 2: KV-cache tensor-network topology comparison for Qwen3-0.6B.

Compression	SVD	TT	TR	CP	TD
90%	47.38	27.04	36.94	**26.88**	27.04
80%	53.24	29.81	36.95	**27.99**	29.81

Table 3: Qwen3-0.6B decoding throughput relative to the original model without KV-cache decomposition.

Device	SVD	TT	TR	CP	TD
CPU	78.7%	62.0%	62.7%	3.3%	32.5%
GPU	79.6%	61.9%	55.6%	3.7%	28.6%

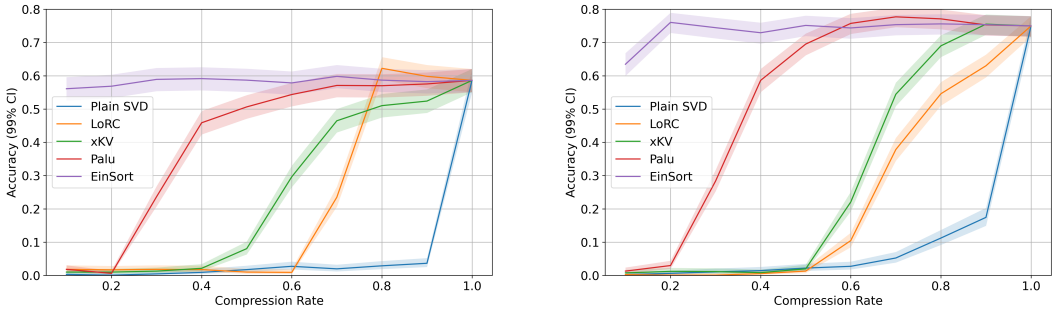
Table 2 compares tensor-network topologies with square-root mapping, product reduction, non-negative factorization, and gauge fixing. Except for SVD, we fold the KV cache into a three-mode tensor. CP gives the lowest PPL in this setting, whereas TR performs worst. Table 3 reports decoding throughput for the same setting. On both an Apple M1 CPU and an NVIDIA A40 GPU, TT/TR reduce throughput by roughly 40% relative to the uncompressed baseline, whereas TD reduces throughput by about 70% despite achieving PPL close to TT. CP is substantially slower because it requires iterative Khatri–Rao products across the cores. The current implementation prioritizes validating the compression principle; optimized inference kernels are left for future work.

3.2 KV-CACHE COMPRESSION FOR MATHEMATICAL REASONING

Perplexity alone does not establish whether compressed KV caches preserve multi-step reasoning, so we additionally evaluate GSM8K (Cobbe et al., 2021) with Phi-3.5-mini (Haider et al., 2024) and Phi-4-mini (Abouelenin et al., 2025). We allow up to 256 generated tokens with the prompt "Solve the following math problem. Show your reasoning, then give the final answer." We compare EinSort with plain SVD and three practical KV-cache baselines: LoRC (Zhang et al., 2025), xKV (Chang et al., 2025), and Palu (Chang et al., 2024). Fig. 6 reports exact-answer accuracy with Wilson 99% confidence intervals. EinSort preserves high reasoning accuracy under aggressive compression: for both models, its accuracy remains close to the uncompressed endpoint throughout much of the sweep, while the competing methods degrade sharply at higher compression. The advantage is especially pronounced in the severe-compression regime, complementing the WT2 perplexity results with a downstream generation task and stronger practical baselines.

Cross-model transfer and storage tradeoff. The main-body spectrum in Fig. 2 uses Qwen3-0.6B for clarity, while Appendix C repeats the analysis for Qwen3-1.7B, Gemma3-4B, and Phi-3.5-mini. Across these models, structured row-wise, group, and sequential sorting consistently reduce reconstruction error relative to unsorted decomposition. For Qwen3-0.6B, full, row, group, and sequential sorting require approximately 18.6, 8.6, 3.9, and 5.0 permutation bits per weight; the restricted variants still reduce reconstruction error by more than an order of magnitude relative to the unsorted case in the reported spectrum study. Thus practical EinSort spans a continuum from low-metadata structured ordering to the full-sort diagnostic ceiling.

Appendix R also adds WT2 sweeps for Qwen3-1.7B and Phi-3.5-mini. The former follows the Qwen3-0.6B trend; for Phi-3.5-mini, slicing reduces permutation metadata from about 11.1 to 0.35 bits per KV-cache element while moving PPL toward the full-sort result. Together with the VLM/VLA results below, this supports transfer across model size, architecture family, and tensor type.

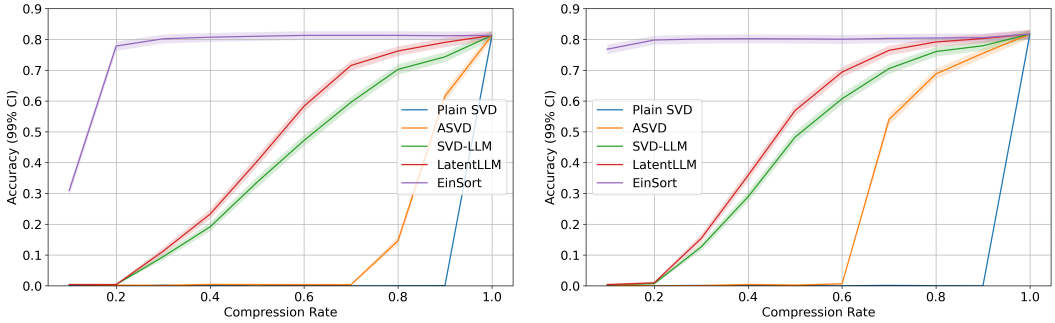


(a) Phi-3.5-mini-instruct (b) Phi-4-mini-instruct

Figure 6: GSM8K exact-answer accuracy under KV-cache compression. Shaded regions show Wilson 99% confidence intervals.

3.3 Weight compression in vision-language models

We next test whether the same index-ordering principle transfers from KV caches to model weights in multimodal models. We evaluate Qwen3-VL (Bai et al., 2025) on TextVQA and compress the weights of linear modules in the VLM language backbone. We compare plain SVD, ASVD (Yuan et al., 2023), SVD-LLM (Wang et al., 2024b), and LatentLLM (Koike-Akino et al., 2026a). All methods except plain SVD use activation-aware preconditioning computed from the same 64 calibration samples from the TextVQA training split. As shown in Fig. 7, EinSort retains high accuracy on both Qwen3-VL-4B and Qwen3-VL-8B even at the 20% compression setting, while the compared low-rank baselines degrade sharply. This experiment directly validates EinSort on persistent model weights rather than only transient KV-cache tensors.



(a) Qwen3-VL-4B-Instruct (b) Qwen3-VL-8B-Instruct

Figure 7: TextVQA accuracy under VLM weight compression. Shaded regions show 99% confidence intervals.

3.4 Weight compression in vision-language-action models

Finally, we evaluate whether weight compression preserves closed-loop embodied performance. We compress all linear modules in the attention and multilayer perceptron blocks of X-VLA (Zheng et al., 2025), $\pi_{0.5}$ (Intelligence et al., 2025), and VLA-JEPA (Sun et al., 2026) and evaluate 800 rollouts over the LIBERO Spatial, Object, Goal, and Long suites. Their VLM backbones are Florence-2 (Xiao et al., 2024a), PaliGemma (Beyer et al., 2024), and Qwen3-VL, respectively. We compare the same weight-decomposition baselines as above; except for plain SVD, all methods use 64 LIBERO-Spatial episodes for activation calibration.

Table 4 summarizes average success rates; Appendix R.4 provides the per-suite results, Wilson 95% confidence intervals, and qualitative rollout snapshots. At the reported compression rates, EinSort achieves 92.38%, 97.13%, and 97.75% average success for X-VLA, $\pi_{0.5}$, and VLA-JEPA,

respectively, compared with 93.63%, 97.75%, and 97.50% for the uncompressed models. The strongest compared decomposition baseline reaches 26.25%, 57.38%, and 59.75%, respectively. Cases in which EinSort numerically exceeds the uncompressed model fall within the reported confidence intervals and are therefore not interpreted as significant gains.

Table 4: Average LIBERO success rate (↑) across four task suites. Parentheses give the reported compression rates.

Method	X-VLA (0.1)	$\pi_{0.5}$ (0.4)	VLA-JEPA (0.8)
Original	93.63%	97.75%	97.50%
Plain SVD	2.63%	17.75%	0.00%
ASVD	17.88%	4.25%	52.00%
SVD-LLM	25.50%	49.50%	13.50%
LatentLLM	26.25%	57.38%	59.75%
EinSort	**92.38%**	**97.13%**	**97.75%**

3.5 Practical accounting and limitations

The full-sort curves are best interpreted as an algorithmic ceiling: storing an arbitrary permutation can require more memory than the original FP16 tensor, as discussed in Sec. 2.5. The practical EinSort configurations therefore use sliced or shared permutations. Under the KV-cache protocol above, permutation metadata costs 1.44, 3.14, and 0.35 bits per KV-cache element for Qwen3-0.6B/1.7B, Gemma3-4B, and Phi-3.5-mini, respectively. Non-negative tensorization adds one sign bit per element, while gauge fixing can reduce tensor-core storage. Consequently, deployment memory is the sum of tensor cores, permutation metadata, and any optional sign information rather than a function of rank alone; the slicing configuration provides a direct knob for trading metadata cost against sorting fidelity.

The present study establishes the algorithmic value of index ordering but is not yet an end-to-end optimized systems implementation. Table 3 shows decode-throughput overhead without fused contraction/reordering kernels, and searching over folding, slicing, reduction, and topology choices adds construction cost. We also evaluate KV-cache and model-weight compression separately rather than jointly. The WT2 protocol uses 64-token prefilling and up to 64 decoded tokens, while GSM8K allows up to 256 generated tokens; substantially longer-context deployment, joint weight-and-KV-cache compression, dedicated peak-memory accounting, and optimized kernels remain important future work. These limitations bound the systems claims while leaving the within-protocol comparisons unchanged.

4 Conclusion

We introduced EinSort, a tensorization framework that uses reversible index ordering to expose implicit low-rank structure before tensor decomposition. The method is supported by an asymptotic rank-one result for row-wise sorting and uses sliced or shared permutations to trade sorting fidelity for metadata cost. Empirically, EinSort improves KV-cache compression for language modeling and mathematical reasoning and transfers to model-weight compression in VLMs and VLAs, preserving visual-reasoning accuracy and closed-loop robot-manipulation success under aggressive compression. Together, these results support index ordering as an additional design axis alongside tensor rank, topology, and activation-aware preconditioning. Practical deployment will require end-to-end memory accounting, longer-context validation, and optimized contraction/reordering kernels.

AI use statement

In this work, we used OpenAI ChatGPT as an assistive tool for language editing, proofreading, LaTeX/template conversion, manuscript organization, explanatory-prose drafting and revision, and theoretical derivation, formulation, and proof development. We did not use generative AI to produce experimental measurements, fabricate data, or replace author evaluation of the results. All experiments were conducted by the authors, mostly using A40 GPUs. The authors reviewed and verified all

AI-assisted manuscript and theory-development content and take responsibility for the final text, claims, derivations, proofs, and other artifacts produced with the aid of generative AI.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen Bach, Jianmin Bao, Alon Benhaim, Martin Cai, Vishrav Chaudhary, Congcong Chen, et al. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-LoRAs. *arXiv preprint arXiv:2503.01743*, 2025.

Vadim Abronin, Aleksei Naumov, Denis Mazur, Dmitriy Bystrov, Katerina Tsarova, Artem Melnikov, Sergey Dolgov, Reuben Brasher, and Michael Perelshein. TQCompressor: improving tensor decomposition methods in neural networks via permutations. In *2024 IEEE 7th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp. 503–506. IEEE, 2024.

Jyoti Aneja, Michael Harrison, Neel Joshi, Tyler LaBonte, John Langford, and Eduardo Salinas. Phi-4-reasoning-vision-15B technical report. *arXiv preprint arXiv:2603.03975*, 2026.

Yamato Arai and Yuma Ichikawa. Quantization error propagation: Revisiting layer-wise post-training quantization. *arXiv preprint arXiv:2504.09629*, 2025.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, et al. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*, 2024a.

Guangji Bai, Yijiang Li, Chen Ling, Kibaek Kim, and Liang Zhao. SparseLLM: Towards global pruning for pre-trained language models. *arXiv preprint arXiv:2402.17946*, 2024b.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Daniel Bershatsky, Daria Cherniuk, Talgat Daulbaev, Aleksandr Mikhalev, and Ivan Oseledets. LoTR: Low tensor rank weight adaptation. *arXiv preprint arXiv:2402.01376*, 2024.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. PaliGemma: A versatile 3B VLM for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Robert K Brayton, Gary D Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media, 1984.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayi7qezU87.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing KV-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

Chi-Chih Chang, Chien-Yu Lin, Yash Akhauri, Wei-Cheng Lin, Kai-Chiang Wu, Luis Ceze, and Mohamed S Abdelfattah. xKV: Cross-layer SVD for KV-cache compression. *arXiv preprint arXiv:2503.18893*, 2025.

Arnav Chavan, Zhuang Liu, Deepak Gupta, Eric Xing, and Zhiqiang Shen. One-for-all: Generalized LoRA for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*, 2023.

Xiangyu Chen, Jing Liu, Ye Wang, Pu Wang, Matthew Brand, Guanghui Wang, and Toshiaki Koike-Akino. SuperLoRA: Parameter-efficient unified adaptation for large vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8050–8055, 2024a.

Zhuo Chen, Rumen Dangovski, Charlotte Loh, Owen Dugan, Di Luo, and Marin Soljačić. QuanTA: Efficient high-rank fine-tuning of LLMs with quantum-informed tensor adaptation. *Advances in Neural Information Processing Systems*, 37:92210–92245, 2024b.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Herbert A David and Haikady N Nagaraja. *Order statistics*. John Wiley & Sons, 2004.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Advances in neural information processing systems*, 26, 2013.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. DaViT: Dual attention vision transformers. In *European conference on computer vision*, pp. 74–92. Springer, 2022.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in neural information processing systems*, 30, 2017.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. KronA: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022.

Glen Evenbly. Gauge fixing, canonical forms, and optimal truncations in tensor networks with closed loops. *Physical Review B*, 98(8):085155, 2018.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Dan Fu, Simran Arora, Jessica Grogan, Isys Johnson, Evan Sabri Eyuboglu, Armin Thomas, Benjamin Spector, Michael Poli, Atri Rudra, and Christopher Ré. Monarch mixer: A simple sub-quadratic GEMM-based architecture. *Advances in Neural Information Processing Systems*, 36:77546–77603, 2023.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019.

Emman Haider, Daniel Perez-Becker, Thomas Portet, Piyush Madan, Amit Garg, Atabak Ashfaq, David Majercak, Wen Wen, Dongwoo Kim, Ziyi Yang, et al. Phi-3 safety post-training: Aligning language models with a "break-fix" cycle. *arXiv preprint arXiv:2407.13833*, 2024.

Babak Hassibi, David Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. *Advances in neural information processing systems*, 6, 1993.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning*, 2024.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

William Huggins, Piyush Patil, Bradley Mitchell, K Birgitta Whaley, and E Miles Stoudenmire. Towards quantum machine learning with tensor networks. *Quantum Science and technology*, 4(2): 024001, 2019.

Injoon Hwang, Haewon Park, Youngwan Lee, Jooyoung Yang, and SunJae Maeng. PC-LoRA: Low-rank adaptation for progressive model compression with knowledge distillation. *arXiv preprint arXiv:2406.09117*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. GPT-4 passes the bar exam. *Philosophical Transactions of the Royal Society A*, 382(2270):20230254, 2024.

Toshiaki Koike-Akino, Jing Liu, and Ye Wang. μ-moe: Test-time pruning as micro-grained mixture-of-experts. *arXiv preprint arXiv:2505.18451*, 2025a.

Toshiaki Koike-Akino, Francesco Tonin, Yongtao Wu, Frank Zhengqing Wu, Leyla Naz Candogan, and Volkan Cevher. Quantum-PEFT: Ultra parameter-efficient fine-tuning. *arXiv preprint arXiv:2503.05431*, 2025b.

Toshiaki Koike-Akino, Xiangyu Chen, Jing Liu, Ye Wang, Pu Perry Wang, and Matthew Brand. LatentLLM: Activation-aware transform to multi-head latent attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 22644–22652, 2026a.

Toshiaki Koike-Akino, Jing Liu, and Ye Wang. TTQ: Activation-aware test-time quantization to accelerate LLM inference on the fly. *arXiv preprint arXiv:2603.19296*, 2026b.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Derrick H Lehmer. Teaching combinatorial tricks to a computer. In *Proceedings of Symposia in Applied Mathematics*, pp. 179–193. American Mathematical Society, 1960.

Ming Li, John Tromp, and Paul Vitányi. Reversible simulation of irreversible computation. *Physica D: Nonlinear Phenomena*, 120(1-2):168–176, 1998.

Yuhang Li, Donghyun Lee, Ruokai Yin, and Priyadarshini Panda. Optimal brain decomposition for accurate LLM low-rank approximation. *arXiv preprint arXiv:2604.00821*, 2026.

Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Junwu Zhang, Munan Ning, and Li Yuan. MoE-LlaVa: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*, 2024a.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100, 2024b.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023a.

Jing Liu, Toshiaki Koike-Akino, Pu Wang, Matthew Brand, Ye Wang, and Kieran Parsons. LoDA: Low-dimensional adaptation of large language models. In *NeurIPS'23 Workshop on on Efficient Natural Language and Speech Processing*, 2023b.

Jing Liu, Toshiaki Koike-Akino, Ye Wang, Hassan Mansour, and Matthew Brand. AWP: Activation-aware weight pruning and quantization with projected gradient descent. *arXiv preprint arXiv:2506.10205*, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/liu24bz.html.

Shiyi Luo, Mingshuo Liu, Yifeng Yu, Shangping Ren, and Yu Bai. An adaptive tensor-train decomposition approach for efficient deep neural network compression. *arXiv preprint arXiv:2408.01534*, 2024.

Edward J McCluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35(6): 1417–1444, 1956.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. *Advances in neural information processing systems*, 28, 2015.

Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of physics*, 349:117–158, 2014.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavskỳ, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. Stable low-rank tensor decomposition for compression of convolutional neural network. In *European Conference on Computer Vision*, pp. 522–539. Springer, 2020.

Sebastian Prillo and Julian Eisenschlos. SoftSort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning*, pp. 7793–7802. PMLR, 2020.

Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. TensorNetwork: A library for physics and machine learning. *arXiv preprint arXiv:1905.01330*, 2019.

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea Goldsmith, and Mert Pilanci. Compressing large language models using low rank and low precision decomposition. *Advances in Neural Information Processing Systems*, 37:88981–89018, 2024.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6655–6659. IEEE, 2013.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression. *arXiv preprint arXiv:2408.05646*, 2024.

Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of physics*, 326(1):96–192, 2011.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on signal processing*, 65(13):3551–3582, 2017.

Amanpreet Singh, Vivek Natarjan, Meet Shah, Yu Jiang, Xinlei Chen, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8317–8326, 2019.

Atul Kumar Sinha and François Fleuret. AA-SVD: Anchored and adaptive SVD for large language model compression. *arXiv preprint arXiv:2604.02119*, 2026.

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. VLA-JEPA: Enhancing vision-language-action model with latent world model. *arXiv preprint arXiv:2602.10098*, 2026.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Oscar Y Takeshita. Permutation polynomial interleavers: An algebraic-geometric perspective. *IEEE Transactions on Information Theory*, 53(6):2116–2132, 2007.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Guifré Vidal. Class of quantum many-body states that can be efficiently simulated. *Physical review letters*, 101(11):110501, 2008.

Changyuan Wang, Ziwei Wang, Xiuwei Xu, Yansong Tang, Jie Zhou, and Jiwen Lu. Q-VLM: Post-training quantization for large vision-language models. *arXiv preprint arXiv:2410.08119*, 2024a.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*, 2024b.

Miles Williams and Nikolaos Aletras. On the impact of calibration data in post-training quantization and pruning. *arXiv preprint arXiv:2311.09755*, 2023.

T Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4818–4829, 2024a.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=NG7sS51zVF.

Canwen Xu and Julian McAuley. A survey on model compression and acceleration for pretrained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 10566–10575, 2023.

Xianglong Yan, Zhiteng Li, Tianao Zhang, Haotong Qin, Linghe Kong, Yulun Zhang, and Xiaokang Yang. Recalkv: Low-rank kv cache compression via head reordering and offline calibration, 2025. URL https://arxiv.org/abs/2505.24357.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shin-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard B W Yang, Giyeong Oh, and Yanmin Gong. Navigating text-to-image customization: From lyCORIS fine-tuning to model evaluation. In *International Conference on Learning Representations*, 2024.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. LLM inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Amir Zandieh, Majid Daliri, and Insu Han. QJL: 1-bit quantized JL transform for KV cache quantization with zero overhead, 2024. URL https://openreview.net/forum?id=xHPVGmLXjd.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

Rongzhi Zhang, Kuan Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and yelong shen. LoRC: Low-rank compression for LLMs KV cache with a progressive compression strategy, 2025. URL https://openreview.net/forum?id=NI8AUSAc4i.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024.

Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. Delta-LoRa: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*, 2023.

A Related work

A.1 Model compression

Model compression for LLMs aims to mitigate substantial computational and memory requirements Zhu et al. (2024); Yuan et al. (2024). Such methods primarily fall into four categories: weight quantization Lin et al. (2024b); Frantar et al. (2022); Wang et al. (2024a), network pruning LeCun et al. (1989); Hassibi et al. (1993); Frantar & Alistarh (2023); Bai et al. (2024b), knowledge distillation Hsieh et al. (2023); Hwang et al. (2024), and rank reduction Yuan et al. (2023); Liu et al. (2024a); Hwang et al. (2024); Saxena et al. (2024); Saha et al. (2024).

A.2 Low-rank compression

Low-rank compression for neural networks builds on the empirical observation that trained weight matrices and convolutional kernels are often highly redundant and admit accurate low-rank approximations. Early work by Denil et al. (2013) showed that a large fraction of network parameters can be predicted from a small subset, motivating structured factorization approaches. Sainath et al. (2013) and Jaderberg et al. (2014) popularized the use of SVD and related factorization techniques, achieving substantial speedups with limited accuracy loss. This line was extended by Denton et al. (2014), who applied low-rank approximations to convolutional filters, and Lebedev et al. (2014); Sidiropoulos et al. (2017), who introduced tensor decomposition methods such as Tucker and CP decompositions, though care is required to avoid instability or accuracy degradation Phan et al. (2020).

A.3 KV-cache reduction

KV-cache memory is a major bottleneck in long-context LLM inference, growing linearly with sequence length, batch size, number of layers, and KV hidden dimension. Existing approaches to reduce KV-cache memory can be broadly categorized into token eviction, quantization, and representation compression.

Token eviction methods reduce memory by selectively discarding cached tokens. StreamingLLM (Xiao et al., 2024b) preserves initial attention-sink tokens together with a sliding window of recent tokens for stable streaming decoding, while H2O (Zhang et al., 2023) retains a balance of recent tokens and heavy-hitter tokens that contribute significantly to attention scores. PyramidKV (Cai et al., 2025) further improves eviction-based compression by allocating nonuniform cache budgets across layers, retaining more KV entries in lower layers and fewer in higher layers. Although effective, eviction-based approaches may discard information needed for long-range reasoning.

Quantization methods compress KV tensors without removing tokens. KIVI (Liu et al., 2024b) proposes tuning-free asymmetric 2-bit KV-cache quantization, using per-channel quantization for keys and per-token quantization for values. QJL (Zandieh et al., 2024) applies a Johnson-Lindenstrauss transform followed by sign-bit quantization, eliminating the need to store scale and zero-point quantization metadata while providing an unbiased, low-distortion estimator for query-key inner products. TurboQuant Zandieh et al. (2026) extends this direction with online vector quantization, combining an MSE-oriented quantizer with QJL-based residual correction to improve inner-product estimation.

Low-rank compression reduces KV-cache memory by exploiting redundancy in hidden dimensions or KV projections. LoRC (Zhang et al., 2025) applies low-rank approximation to KV weight matrices with layer-wise sensitivity and progressive compression. Palu Chang et al. (2024) decomposes KV projections via activation-aware SVD for KV-cache reduction, while xKV Chang et al. (2025) uses cross-layer SVD. ReCalKV (Yan et al., 2025) introduces a post-training low-rank KV-cache compression method with tailored strategies for keys and values: head-wise similarity-aware reordering clusters structurally similar heads and applies grouped SVD to the key projection matrix, while offline calibration and matrix fusion calibrate the low-rank value projection using a small calibration dataset.

A.4 Low-rank adaptation

LoRA (Hu et al., 2021) updates the pretrained weight matrix through the addition of a product of two low-rank matrices and has been widely adopted (Zi et al., 2023; Chavan et al., 2023; Hayou et al., 2024). Variants include CP decomposition with LoTR (Bershatsky et al., 2024), Tucker decomposition with SuperLoRA (Chen et al., 2024a), nonlinear low-rank mappings such as LoDA (Liu et al., 2023b), Hadamard- and Kronecker-product parameterizations (Yeh et al., 2024; Edalati et al., 2022), and quantum tensor networks Chen et al. (2024b); Koike-Akino et al. (2025b).

A.5 Activation-aware compression

Decomposition ASVD (Yuan et al., 2023) improves low-rank decomposition by incorporating activation statistics. It has been applied to SVD-LLM (Wang et al., 2024b) and Palu (Chang et al., 2024). AA-SVD (Sinha & Fleuret, 2026) incorporates an error-propagation factor motivated by QEP Arai & Ichikawa (2025). OBD-LLM Li et al. (2026) further uses gradient information as an approximate Hessian to improve ASVD.

Quantization HAWQ Dong et al. (2019) uses layer-wise quantization based on optimal brain pruning LeCun et al. (1989). GPTQ Frantar et al. (2022) extends this approach using zero-shot calibration, while activation-aware quantization (AWQ) Lin et al. (2023) uses activation-dependent scaling. AWP Liu et al. (2025) uses projected gradient descent for activation-aware quantization and pruning within a compressed-sensing framework. QEP Arai & Ichikawa (2025) mitigates error propagation across layers.

Pruning Activation-aware pruning methods Williams & Aletras (2023) include SparseGPT Frantar & Alistarh (2023), which uses layer-wise optimal brain surgeon Dong et al. (2017); Hassibi et al. (1993); LeCun et al. (1989). SparseLLM Bai et al. (2024b) extends this idea to joint module compression. Wanda Sun et al. (2023) greatly simplifies the pruning mechanism, and μ-MoE Koike-Akino et al. (2025a) applies this idea to online pruning.

A.6 Permutation/sorting networks

Permutation and sorting neural networks are architectures designed to learn or process orderings over sets and sequences. The Deep Sets framework of Zaheer et al. (2017) establishes that any permutation-invariant function over a set can be decomposed into a sum of elementwise embeddings followed by a global function, providing a theoretical foundation for set-based learning. In parallel, differentiable sorting and ranking networks have emerged to learn orderings explicitly, such as NeuralSort (Grover et al., 2019) and SoftSort (Prillo & Eisenschlos, 2020), which provide continuous relaxations of sorting operators to enable gradient-based optimization. A closely related approach is the Gumbel–Sinkhorn network Mena et al. (2018), which leverages entropic optimal transport and Gumbel noise to produce differentiable approximations of permutation matrices via Sinkhorn normalization.

A.7 Tensor networks

Tensor networks (Roberts et al., 2019) represent multidimensional arrays by factorizing them into networks of lower-dimensional tensors. Common tensor-network structures include matrix product states (MPS) and tree tensor networks (TTNs) (Huggins et al., 2019), based on tensor train (TT) decomposition and hierarchical Tucker (HT) decomposition, respectively. More sophisticated tensor-network structures used in quantum machine learning (QML) include the multiscale entanglement renormalization ansatz (MERA) (Vidal, 2008) and projected entangled-pair states (PEPS) (Orús, 2014). Tensorization provides efficient parameterizations of deep neural network architectures (Novikov et al., 2015).

B EINSTEIN SUMMATION

Einstein summation (einsum) is a powerful notation for representing a diverse set of tensor networks. Several examples, including TT, TR, CP, TD, HT, PEPS, LoHa, and LoKr (Yeh et al., 2024), are listed in the pseudocode below:

```
#TT: A(di,dp), B(dp,dj,dq), C(dq,dk,dr), D(dr,dl)
X = torch.einsum("ip,pjq,qkr,rl->ijkl", A, B, C, D)

#TR: A(dm,di,dp), B(dp,dj,dq), C(dq,dk,dr), D(dr,dl,dm)
X = torch.einsum("mip,pjq,qkr,rlm->ijkl", A, B, C, D)

#CP: A(dr), B(dr,di), C(dr,dj), D(dr,dk)
X = torch.einsum("r,ri,rj,rk->ijk", A, B, C, D)

#TD: A(dp,dq,dr), B(dp,di), C(dq,dj), D(dr,dk)
X = torch.einsum("pqr,pi,qj,rk->ijk", A, B, C, D)

#HT: A(di,dp), B(dj,dq), C(dk,dr), D(dl,ds), E(dp,dq,dt), F(dr,ds,du), G(
    dt,du)
X = torch.einsum("ip,jq,kr,ls,pqt,rsu,tu->ijkl", A, B, C, D, E, F, G)

#PEPS: A(di,dr,dp), B(dj,dp,ds,dq), C(dk,dq,dt), D(dl,dr,du), E(dm,du,ds,
    dv), F(dl,dv,dt)
X = torch.einsum("irp,jpsq,kqt,lru,musv,lvt->ijklmn", A, B, C, D, E, F)

#LoHa: A(di,dp), B(dp,dj), C(di,dq), D(dq,dj)
X = torch.einsum("ip,pj,iq,qj->ij", A, B, C, D)

#LoKr: A(di,dj), B(dp,dr), C(dr,dq)
X = torch.einsum("ij,pr,rq->ipjq", A, B, C)

X = einops.rearrange("i p j q -> (i p) (j q)", X) # unfolding 4-mode to
    2-mode
```

We use `einops`[1] for folding and unfolding. It enables tensor-network topologies to be specified compactly through the einsum equation (a string such as `"ip,pj->ij"`) and appropriately shaped tensor cores. A Monarch mixer Fu et al. (2023) can also be represented by a series of einsum operations without explicitly writing permutation operators. The Monarch mixer uses dilated mixing with a gradually expanding receptive field:

$$W = P_{m+1}\mathrm{diag}[W_m]\cdots\mathrm{diag}[W_2]P_2\mathrm{diag}[W_1]P_1\mathrm{diag}[W_0]P_0, \tag{3}$$

where the P_k are permutations with a Cooley–Tukey-like butterfly structure and $\mathrm{diag}[W_k]$ are block-diagonal weights. This construction generalizes many structured matrices, such as the Fourier and Walsh–Hadamard transforms, while retaining low complexity and high parameter efficiency. It can be realized compactly by the following einsum chain:

```
# 3-stage dilated Monarch mixer
# Assume input activation X (..., d, d, d), Mixer W (3, d, d, d, d)
X = torch.einsum("...ij,...i->...j", W[0], X) # the first local mixing
X = torch.einsum("...iaj,...ia->...ja", W[1], X) # the second dilated
    mixing
X = torch.einsum("iabj,...iab->...jab", W[2], X) # the last dilated
    mixing
```

As the example shows, einsum can construct a broad range of tensor networks by specifying the contraction equation and tensor shapes. Pseudocode for a tensor-network module with an arbitrary einsum equation and shapes is shown below:

```
import opt_einsum as oe

class TensorNet(torch.nn.Module):
```

[1] https://einops.rocks/

```python
    def __init__(self, shapes, equation, device=None):
        super().__init__()
        self.shapes = shapes
        self.equation = equation
        self.device = device
        self.cores = torch.nn.ParameterList(
            [
                torch.nn.Parameter(torch.empty(*shape, device=self.device
                    ))
                for shape in self.shapes
            ]
        )

        # contraction order optimization
        self.expression = oe.contract_expression(self.equation, *self.
            shapes)

        self.reset_parameters() # initialize cores

    def contract(self, *args, **kwargs):
        # reconstruct a tensor from cores with an optimized contraction
            expression
        return self.expression(*self.cores, *args, **kwargs)

    def fit_als(self, target, **kwargs):
        # fit cores to a target tensor with ALS
        with torch.inference_mode():
            cores = als(self.equation, self.cores, target, **kwargs)
            for k, core in enumerate(cores):
                self.cores[k].copy_(core)

    def fit_grad(self, target, **kwargs):
        # gradient optimization
        lr = kwargs.get("lr", 1e-3)
        opt = torch.optim.AdamW(self.cores, lr=lr)
        for k in range(kwargs.get("steps", 1000)):
            opt.zero_grad()
            output = self.contract()
            loss = torch.nn.functional.mse_loss(output, target)
            loss.backward()
            opt.step()
```

Here, we use `opt_einsum`[2] to optimize the contraction order. We use `tensorly`[3] to initialize cores, and we can fine-tune them with gradient optimization.

In addition to reducing parameter count while retaining flexible tensor-network expressivity, contraction-order optimization can reduce computational cost. For example, for a 4-mode TT with site dimension $d = 32$ and rank $R = 20$, a naïve left-to-right contraction requires 3.4×10^{10} FLOPs, whereas an optimized contraction order reduces the cost to 4.4×10^{7} FLOPs—nearly three orders of magnitude—using `opt_einsum`. The following pseudocode demonstrates this reduction:

```python
# tensor reconstruction can be faster with contraction optimization
shape = [(32,20),(20,32,20),(20,32,20),(20,32)]
_, info = opt_einsum.contract_path("ip,pjq,qkr,rl->ijkl", *shape, shapes=
    True, optimize="optimal")
```

Furthermore, when the weight is decomposed in such a tensor network, we often do not need to explicitly reconstruct the weight before multiplying with the input activation. For example, if the batched input activation has shape $(64, 1024)$, reconstructing a $(1024, 1024)$ weight matrix and multiplying it by the input requires 2.7×10^{12} FLOPs in total. An optimized direct contraction path reduces the estimated cost to 6.9×10^{6} FLOPs—more than five orders of magnitude lower and

[2] https://optimized-einsum.readthedocs.io/en/stable/

[3] https://tensorly.org/stable/index.html

below the estimated cost of materializing the weight alone. The following pseudocode illustrates this benefit:

```
# weight-activation multiplication can be even faster with contraction
    optimization
# Weight: W (1024, 1024) -> (32, 32, 32, 32); Input X (64, 1024) -> (64,
    32, 32)
shape = [(32,20),(20,32,20),(20,32,20),(20,32),(64,32,32)]
_, info = opt_einsum.contract_path("ip,pjq,qkr,rl,...ij->..kl", *shape,
    shapes=True, optimize="optimal")
```

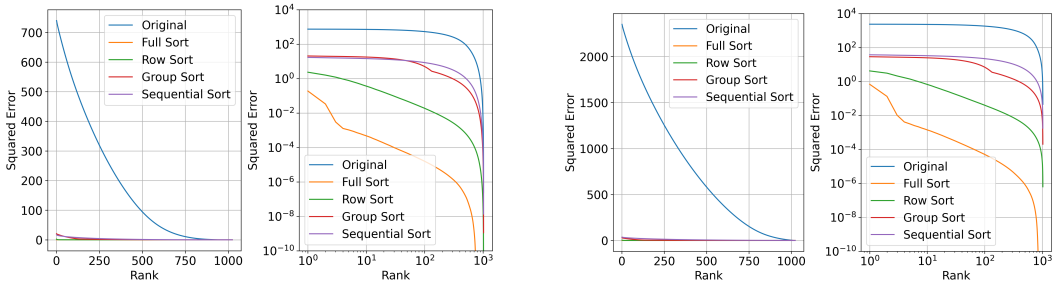
C RANK REDUCTION WITH TENSOR SORTING

We provide additional empirical results showing that sorting can substantially reduce the required rank. Fig. 8 shows the squared reconstruction error $\|W - \hat{W}\|_{\mathrm{F}}^2$, where W is the original weight matrix and $\hat{W}$ is its reconstruction using a 2-mode TT (i.e., SVD). Specifically, we analyze the singular-value spectra of pretrained weights from Qwen3-0.6B, Qwen3-1.7B, Gemma3-4B, and Phi-3.5-mini. We also evaluate several sorting options, as shown in the pseudocode:

```
# Original singular-value spectrum
S = torch.linalg.svdvals(W)
loss = S.square().flip(-1).cumsum(-1)

# Full sorting
Wf, _ = torch.sort(W.view(-1))
Wf = Wf.reshape_as(W)

# Row-wise sorting
Wr, _ = torch.sort(W, dim=-1)

# Row-wise group sorting
_, perm = torch.sort(W, dim=-1) # row-wise sort
perm, _ = torch.sort(perm.view(-1, groupsize), dim=-1) # revert
    permutation within group
Wg = torch.gather(W, dim=-1, index=perm.view_as(W))

# Row-wise sequential axis sorting
primes = sympy.factorint(W.shape[1]) # prime factorization
shape = [-1] + [p for p, e in primes.items() for _ in range(e)] # like [
    dout, 2,2,...,2]
Ws = W.view(*shape)
for axis in range(1, len(shape)):
    Ws, _ = torch.sort(Ws, dim=axis)
Ws = Ws.view_as(W)
```

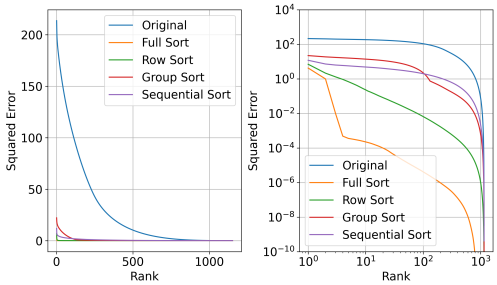
Group sorting orders elements row-wise while ignoring the ordering within each group of size g. For example, with eight elements and $g = 4$, we preserve which elements belong to the lower and upper groups while ignoring the ordering within each group. In this case, the required storage is reduced from $\lceil \log_2(n!) \rceil / n$ to $\lceil \log_2\big(n!/(g!)^{n/g}\big) \rceil / n$ bits per element. Sequential sorting performs round-robin sorting across axes. For an m-mode tensor with site dimension d, the required permutation memory is $\lceil \log_2(d!) \rceil m/d$ bits.

As shown in Fig. 8, increasing the rank improves reconstruction accuracy, but the improvement is slow when the original weights are decomposed by SVD without sorting. Notably, tensor sorting substantially reduces reconstruction error, enabling near-rank-one reconstruction. The log–log plots show that row-wise sorting, group sorting (with group size $g = 64$), and sequential axis sorting have higher errors than full sorting but remain at least an order of magnitude lower than the unsorted case. For Qwen3-0.6B, full sorting requires $\lceil \log_2(2^{20}!) \rceil / 2^{20} \simeq 18.6$ bits per weight, whereas row sorting requires $\lceil \log_2(2^{10}!) \rceil / 2^{10} \simeq 8.6$ bits. Group sorting requires $\lceil \log_2(2^{10}!/(g!)^{2^{10}/g}) \rceil / 2^{10} \simeq 3.9$ bits per weight, and sequential sorting requires 5 bits per weight. Therefore, even approximate sorting schemes that reduce permutation memory can substantially improve reconstruction accuracy through index ordering. We observe a similar trend across all LLMs considered.

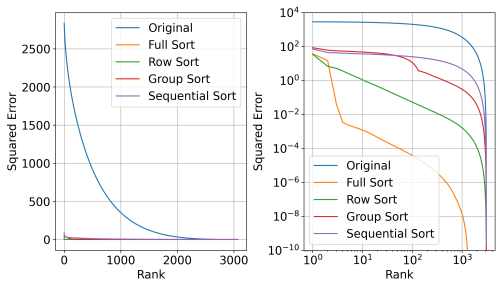


(a) Qwen3-0.6B (layers.0.self_attn.v_proj)

(b) Qwen3-1.7B (layers.0.self_attn.v_proj)



(c) Gemma3-4B (layers.0.self_attn.v_proj)



(d) Phi-3.5-mini (layers.0.self_attn.o_proj)

Figure 8: Tensorization error vs. rank for representative pretrained LLM weights. Sorting can significantly reduce the required rank. Left: linear scale; right: log–log scale.

D Theoretical Analysis

D.1 Uniform full-sort special case

For completeness, we record a complementary full-sort result for the uniform case.

Lemma D.1 (Uniform full-sort special case)**.** *Let $X_0, \dots, X_{L-1}$ be i.i.d. random variables uniformly distributed on $[a, b]$, sort them as $X_{(0)} \le \cdots \le X_{(L-1)}$, and fold the sequence into $X \in \mathbb{R}^{M \times N}$ with $X_{i,j} = X_{(Ni+j)}$ and $L = MN$. Then X is asymptotically approximated in probability by a matrix of rank at most 3, with entrywise stochastic error $O_p(L^{-1/2})$.*

Proof. For $k \in \{0, \dots, L-1\}$, the order statistic $X_{(k)}$ has the same distribution as

$$a + (b - a)Y_{(k)}, \tag{4}$$

where $Y_{(k)}$ is the $(k+1)$-st order statistic of L i.i.d. uniform random variables on $[0, 1]$. It is well known (David & Nagaraja, 2004) that $Y_{(k)}$ follows the Beta distribution:

$$Y_{(k)} \sim \mathrm{Beta}(k + 1, L - k). \tag{5}$$

Hence, we have

$$\mathbb{E}[X_{(k)}] = a + (b - a)\mathbb{E}[Y_{(k)}] = a + (b - a)\frac{k+1}{L+1}, \tag{6}$$

and

$$\mathrm{Var}[X_{(k)}] = (b - a)^2 \frac{(k+1)(L-k)}{(L+1)^2(L+2)}. \tag{7}$$

Since

$$(k+1)(L-k) \le \frac{(L+1)^2}{4}, \tag{8}$$

we have

$$\mathrm{Var}[X_{(k)}] \le \frac{(b-a)^2}{4(L+2)} = \mathcal{O}(L^{-1}). \tag{9}$$

Therefore, by Chebyshev's inequality,

$$X_{(k)} = a + (b-a)\frac{k+1}{L+1} + \mathcal{O}_p(L^{-1/2}). \tag{10}$$

Now setting $k = Ni + j$, we can write the matrix entry at (i, j) as

$$X_{i,j} = a + (b-a)\frac{(Ni+j)+1}{L+1} + \mathcal{O}_p(L^{-1/2}) \tag{11}$$

$$= N\frac{b-a}{L+1}i + \frac{b-a}{L+1}j + \Big(a + \frac{b-a}{L+1}\Big) + \mathcal{O}_p(L^{-1/2}). \tag{12}$$

Thus, in matrix form, we can write

$$X = N\frac{b-a}{L+1}\ell_M 1_N^\top + \frac{b-a}{L+1}1_M\ell_N^\top + \Big(a + \frac{b-a}{L+1}\Big)1_M 1_N^\top + \mathcal{E}, \tag{13}$$

where every entry of $\mathcal{E}$ is $\mathcal{O}_p(L^{-1/2})$.

The first three terms are rank-one matrices. Hence their sum has rank at most 3. Therefore X is asymptotically approximated by a rank-at-most-3 matrix, with entrywise stochastic error $\mathcal{O}_p(L^{-1/2})$. □

The bias term in the third component vanishes when $a = -b/L$, since $a + (b-a)/(L+1) = 0$.

D.2 Random projection for approximate uniformity

The preceding lemma motivates a heuristic extension beyond the uniform case. Indeed, many random vectors become approximately Gaussian after an orthogonal mixing transform due to the central limit effect. Since a Gaussian random variable can be converted into a uniform random variable through its cumulative distribution function, the sorted-matrix structure above approximately applies to a broader class of distributions after an appropriate mixing transform.

More specifically, let

$$x = [x_0, \ldots, x_{L-1}]^\top \tag{14}$$

be a random vector with independent entries having finite variance, not necessarily uniformly distributed. Let $U \in \mathbb{R}^{L\times L}$ be a fixed orthogonal mixing matrix whose entries are sufficiently delocalized, such as the normalized Walsh–Hadamard matrix, and define

$$y = Ux. \tag{15}$$

Each component

$$y_i = \sum_{k=0}^{L-1} U_{ik}x_k \tag{16}$$

is then a weighted sum of many independent random variables. Under standard Lindeberg-type conditions, the central limit theorem implies that, as $L \to \infty$, each y_i converges in distribution toward a Gaussian random variable:

$$y_i \xrightarrow{d} \mathcal{N}(\mu_i, \sigma_i^2). \tag{17}$$

After normalization,

$$z_i = \Phi\left(\frac{y_i - \mu_i}{\sigma_i}\right), \tag{18}$$

where Φ denotes the standard Gaussian cumulative distribution function, we obtain approximately uniform random variables

$$z_i \approx \mathcal{U}(0,1). \tag{19}$$

Consequently, after a sufficiently mixing orthogonal transform and marginal Gaussianization, a broader class of random matrices can be related approximately to the uniform setting analyzed in Lemma D.1. This heuristic suggests that similar low-rank structure may arise beyond the exact uniform model.

This argument is heuristic because the transformed variables are generally not exactly independent, and finite-dimensional convergence from the central limit theorem does not directly imply joint uniformity. Nevertheless, for highly mixing transforms such as Walsh–Hadamard matrices, the approximation can become accurate in high dimensions and is widely exploited in randomized numerical linear algebra and signal processing.

D.3 Detailed assumptions and proof of Lemma 2.1

Here we state the regularity conditions suppressed in the concise main-text version. Let $\{X_{i,j}\}_{i\in\mathbb{Z}_M,\, j\in\mathbb{Z}_N}$ be i.i.d. random variables with continuous distribution function F and density f. Assume there exists $c > 0$ such that $f(x) \geq c$ on the support of F, and that the quantile function $Q(u) = F^{-1}(u)$ satisfies

$$\int_0^1 Q(u)^2 \, \mathrm{d}u < \infty. \tag{20}$$

For each row i, let $X_{i,(1)} \leq \cdots \leq X_{i,(N)}$ be its order statistics and define $S_{i,j} = X_{i,(j+1)}$ and $q_j = Q((j+1)/(N+1))$. Under these assumptions, Lemma 2.1 states that

$$\frac{\|S - 1_M q^\top\|_{\mathrm{F}}}{\|1_M q^\top\|_{\mathrm{F}}} = O_p(N^{-1/2}). \tag{21}$$

Proof. Define

$$U_{i,j} = F(X_{i,j}). \tag{22}$$

By the probability integral transform, the variables $U_{i,j}$ are i.i.d. uniform on $[0,1]$. Since $Q = F^{-1}$,

$$X_{i,(j+1)} = Q(U_{i,(j+1)}), \tag{23}$$

where $U_{i,(j+1)}$ is the $(j+1)$-st order statistic of N i.i.d. uniform random variables.

The uniform order statistics satisfy

$$\mathbb{E}[U_{i,(j+1)}] = \frac{j+1}{N+1}, \tag{24}$$

$$\mathrm{Var}(U_{i,(j+1)}) = \frac{(j+1)(N-j)}{(N+1)^2(N+2)} \leq \frac{1}{4(N+2)} = O(N^{-1}). \tag{25}$$

Since $f(x) \geq c > 0$, the quantile function $Q(u)$ is Lipschitz. Indeed,

$$Q'(u) = \frac{1}{f(Q(u))}, \tag{26}$$

hence

$$|Q'(u)| \leq \frac{1}{c}. \tag{27}$$

Therefore, for some constant $C > 0$,

$$|Q(u) - Q(v)| \leq C|u - v|. \tag{28}$$

Using this Lipschitz property,

$$\begin{aligned}\mathbb{E}\left[\left(X_{i,(j+1)} - q_j\right)^2\right] &= \mathbb{E}\left[\left(Q(U_{i,(j+1)}) - Q\left(\frac{j+1}{N+1}\right)\right)^2\right] \\ &\leq C^2 \operatorname{Var}(U_{i,(j+1)}) \\ &= O(N^{-1}).\end{aligned} \tag{29}$$

Summing over j,

$$\sum_{j=0}^{N-1} \mathbb{E}\left[\left(X_{i,(j+1)} - q_j\right)^2\right] = O(1).$$

Now define the error matrix

$$\mathcal{E}_{i,j} = X_{i,(j+1)} - q_j. \tag{30}$$

Then

$$\mathbb{E}\|\mathcal{E}\|_{\mathrm{F}}^2 = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \mathbb{E}[\mathcal{E}_{i,j}^2] \tag{31}$$

$$= M \cdot O(1) \tag{32}$$

$$= O(M). \tag{33}$$

Hence,

$$\|\mathcal{E}\|_{\mathrm{F}} = O_p(\sqrt{M}). \tag{34}$$

Next,

$$\|1_M q^\top\|_{\mathrm{F}}^2 = M\|q\|_2^2 = M\sum_{j=0}^{N-1} q_j^2. \tag{35}$$

Since $q_j = Q((j+1)/(N+1))$,

$$\frac{1}{N}\sum_{j=0}^{N-1} q_j^2 \to \int_0^1 Q(u)^2 \,\mathrm{d}u. \tag{36}$$

Therefore,

$$\sum_{j=0}^{N-1} q_j^2 = \Theta(N), \tag{37}$$

which implies

$$\|1_M q^\top\|_{\mathrm{F}} = \Theta(\sqrt{MN}). \tag{38}$$

Finally,

$$\frac{\|\mathcal{E}\|_{\mathrm{F}}}{\|1_M q^\top\|_{\mathrm{F}}} = \frac{O_p(\sqrt{M})}{\Theta(\sqrt{MN})} = O_p(N^{-1/2}). \tag{39}$$

Since

$$S = 1_M q^\top + \mathcal{E}, \tag{40}$$

the matrix S converges to the rank-one matrix $1_M q^\top$ in relative Frobenius norm. □

D.4 INDEPENDENT PERMUTATION

Row-wise sorting can induce rank-one structure as in Lemma 2.1. In contrast, independent permutations of individual tensor axes do not change matrix singular values or rank.

Proposition D.2 (Invariance of singular values and rank under independent permutations)**.** *Let $X \in \mathbb{R}^{M\times N}$ be a matrix, and let $P_{\mathrm{r}} \in \mathbb{R}^{M\times M}$ and $P_{\mathrm{c}} \in \mathbb{R}^{N\times N}$ be arbitrary row and column permutation matrices, respectively. Then,*

$$\sigma(X) = \sigma(P_{\mathrm{r}} X P_{\mathrm{c}}), \tag{41}$$

where $\sigma(\cdot)$ denotes the set of singular values. Consequently,

$$\operatorname{rank}(X) = \operatorname{rank}(P_{\mathrm{r}} X P_{\mathrm{c}}). \tag{42}$$

Proof. Let the SVD of X be

$$X = U\Sigma V^{\top}, \tag{43}$$

where $U \in \mathbb{R}^{M\times M}$ and $V \in \mathbb{R}^{N\times N}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{M\times N}$ is rectangular diagonal, with the singular values of X on its diagonal.

Because permutation matrices are orthogonal, $P_{\mathrm{r}}U$ and $P_{\mathrm{c}}^{\top}V$ are also orthogonal. Therefore,

$$P_{\mathrm{r}} X P_{\mathrm{c}} = P_{\mathrm{r}} U \Sigma V^{\top} P_{\mathrm{c}} \tag{44}$$

$$= (P_{\mathrm{r}} U)\Sigma(P_{\mathrm{c}}^{\top} V)^{\top}. \tag{45}$$

This is an SVD of $P_{\mathrm{r}} X P_{\mathrm{c}}$ with the same diagonal matrix Σ. Hence, $P_{\mathrm{r}} X P_{\mathrm{c}}$ and X have the same singular values. Since matrix rank equals the number of nonzero singular values, the rank is also invariant under independent row and column permutations. □

Together with Lemma 2.1, this observation motivates axis-dependent permutations such as CNOT-like entangling operations: the permutation of column index j should depend on row index i. From this perspective, independent row and column permutations, such as those considered in TQCompressor Abronin et al. (2024), cannot by themselves change matrix singular values or rank.

D.5 REDUCTION IMPACT

Sliced sorting with reduction decreases permutation-memory requirements, but it can also degrade sorting accuracy. Under a simple uniform model, we can characterize how the signal-to-variation margin changes with the reduction size.

Proposition D.3 (Scaling law for mean and product reduction on uniform random variables)**.** *Let $U_{i,j}$ be i.i.d. random variables uniformly distributed on $[-1, 1]$, with $i = 1, \ldots, N$ and $j = 1, \ldots, M$, and define $A_{i,j} = |U_{i,j}|^p$ for $p > 0$. Define the mean and product reductions across index j as $Y_i = M^{-1}\sum_{j=1}^{M} A_{i,j}$ and $Z_i = \prod_{j=1}^{M} A_{i,j}$. Then, conditioned on a target factor $A_{i,j}$,*

$$\mathbb{E}[Y_i \mid A_{i,j}] = \frac{A_{i,j}}{M} + \frac{M-1}{M(p+1)}, \tag{46}$$

$$\operatorname{Var}[Y_i \mid A_{i,j}] = \frac{M-1}{M^2}\Big(\frac{1}{2p+1} - \frac{1}{(p+1)^2}\Big), \tag{47}$$

$$\mathbb{E}[Z_i \mid A_{i,j}] = \frac{A_{i,j}}{(p+1)^{M-1}}, \tag{48}$$

$$\operatorname{Var}[Z_i \mid A_{i,j}] = A_{i,j}^2\Big(\frac{1}{(2p+1)^{M-1}} - \frac{1}{(p+1)^{2(M-1)}}\Big). \tag{49}$$

Hence, the target-dependent margin for mean reduction and the corresponding margin for product reduction scale as

$$\frac{\mathbb{E}[Y_i \mid A_{i,j}] - (M-1)/[M(p+1)]}{\mathrm{Var}[Y_i \mid A_{i,j}]^{1/2}} = \frac{A_{i,j}}{\sqrt{M-1}}\Big(\frac{1}{2p+1} - \frac{1}{(p+1)^2}\Big)^{-1/2} = A_{i,j}\,\mathcal{O}(M^{-1/2}), \tag{50}$$

$$\frac{\mathbb{E}[Z_i \mid A_{i,j}]}{\mathrm{Var}[Z_i \mid A_{i,j}]^{1/2}} = \Big(\frac{(p+1)^{2(M-1)}}{(2p+1)^{M-1}} - 1\Big)^{-1/2} = \mathcal{O}\left(\left[\frac{2p+1}{(p+1)^2}\right]^{M/2}\right). \tag{51}$$

Thus, the mean-reduction margin decays polynomially with M, whereas the product-reduction margin decays exponentially with M.

Proof. Let $A = |U|^p$ for $U \sim \mathcal{U}(-1,1)$. Its CDF is

$$F_A(a) = \mathbb{P}(|U|^p < a) = a^{1/p}, \qquad 0 \le a \le 1, \tag{52}$$

and its density is

$$f_A(a) = \frac{1}{p} a^{1/p-1}, \tag{53}$$

i.e., $A \sim \mathrm{Beta}(1/p, 1)$. Its moments satisfy

$$\mathbb{E}[A^k] = \mathbb{E}[|U|^{kp}] = \frac{1}{pk+1}. \tag{54}$$

For the mean reduction, condition on A_1 and write

$$Y \mid A_1 = \frac{1}{M}\left(A_1 + \sum_{i=2}^{M} A_i\right). \tag{55}$$

Then

$$\mathbb{E}[Y \mid A_1] = \frac{A_1}{M} + \frac{M-1}{M(p+1)}, \tag{56}$$

$$\mathrm{Var}[Y \mid A_1] = \frac{M-1}{M^2}\left(\frac{1}{2p+1} - \frac{1}{(p+1)^2}\right). \tag{57}$$

For the product reduction,

$$Z \mid A_1 = A_1 \prod_{i=2}^{M} A_i, \tag{58}$$

so

$$\mathbb{E}[Z \mid A_1] = A_1\left(\frac{1}{p+1}\right)^{M-1}, \tag{59}$$

$$\mathbb{E}[Z^2 \mid A_1] = A_1^2\left(\frac{1}{2p+1}\right)^{M-1}. \tag{60}$$

Subtracting the squared mean yields

$$\mathrm{Var}[Z \mid A_1] = A_1^2\left[\left(\frac{1}{2p+1}\right)^{M-1} - \left(\frac{1}{p+1}\right)^{2(M-1)}\right], \tag{61}$$

which gives the stated margins. □

A similar qualitative trade-off can arise for other distributions. The mean reduction has a target-dependent margin that increases with $A_{i,j}$, so stochastic variation is more severe for small target magnitudes. The product-reduction margin is independent of $A_{i,j}$, which can make sliced sorting more uniform across target magnitudes, although its margin decays more rapidly with the reduction size M. For example, with $p = 1$, $M = 8$, and $|U_{i,j}| = 0.1$, the margins are approximately 0.13 for mean reduction and 0.41 for product reduction.

E EINSORT

These results motivate sorting as a design variable for tensor networks. We formulate the **EinSort** framework around an Einstein sorted sum. Conceptually, we use a reversible permutation operation $\pi[\cdot]$ inside tensor decomposition. Let $\mathcal{T}_\theta[\cdot]$ denote a tensor decomposition whose hyperparameters θ determine the einsum equation, shapes, topology, and related design choices. The sorted tensor decomposition is written as $\mathcal{T}_\theta^\pi := \pi^{-1}[\mathcal{T}_\theta[\pi[X]]]$. This formulation extends einsum-based tensorization to expose low-rank structure.

Although sorting may enable tensor cores with lower rank, the memory overhead of storing permutations is not negligible. Specifically, sorting L values requires at most $\lceil \log_2(L!) \rceil$ bits using factoradic/Lehmer code (Lehmer, 1960). For example, full sorting for a tensor of shape $(1024, 1024)$ requires roughly 18.6 bits per entry, which often exceeds the memory footprint of the original tensor in FP16. Therefore, the method must balance sorting accuracy against tensor-rank reduction and permutation overhead.

We consider several simple sorting methods in this paper, while many other options could be explored. One of the simplest approaches is to reuse a shared permutation across multiple tensor slices. This requires a sorting metric that can define a useful shared permutation across slices. For a 4-mode tensor X of shape (d, d, d, d), we may share a permutation across slices of the first mode and reorder the remaining modes $\{j, k, l\}$ using statistics from $X_{:,j,k,l}$. This reduces permutation storage from $\lceil \log_2(d^4!) \rceil$ to $\lceil \log_2(d^3!) \rceil$ bits, i.e., by more than a factor of d. By adjusting the slicing dimension, the permutation footprint can be reduced to below one bit per weight in some settings. For example, with $d = 32$, we have $\lceil \log_2(d^3!) \rceil / d^4 \simeq 0.4$ bits per weight. For the sorting metric, we consider several nonlinear mappings and reduction operators, including power scaling with mean, maximum, minimum, median, standard-deviation, and product reductions. An example using power scaling and standard-deviation reduction is shown in the pseudocode below.

```
# X (1024, 1024)
X = X.reshape([2] * 20) # folding to 20-mode tensor of shape (2,2,...,2)
S = X.abs().pow(p) # scoring with p-th power mapping
axis = [0,1,2] # slicing modes (the first 3 modes for simplicity)
S = S.std(dim=axis, keepdim=True).expand_as(X) # std reduction
perm = torch.argsort(S.view(2,2,2,-1), dim=-1) # shared permutation
X = X.view(2,2,2,-1).gather(-1, perm).view([2] * 20) # reorder
```

This construction allows flexible choices of slicing modes, power exponent, and reduction operator. We specifically consider six reduction operators:

- Mean reduction: `torch.mean()`
- Max reduction: `torch.max()`
- Min reduction: `torch.min()`
- Median reduction: `torch.median()`
- Standard-deviation reduction: `torch.std()`
- Product reduction: `torch.prod()`

In addition, we consider several nonlinear mappings before reduction:

- Linear: no mapping
- Power: `x.pow(p)`
- Exponential: `x.mul(p).clamp(-10,10).exp()`
- Additive logarithm: `x.add(p).clamp(4.4e-5,2.2e4).log()`
- Multiplicative logarithm: `x.mul(p).clamp(4.4e-5,2.2e4).log()`

We search over folding strategies and modes to improve sorting accuracy while controlling memory cost. Appendix F compares different mappings and reductions.

In addition to the sorting metric, we employ non-negative tensorization to improve reconstruction accuracy. Specifically, we map the original tensor to non-negative values before tensor decomposition.

To recover negative values, we retain the original sign information, which requires one additional bit per element. This construction is illustrated in Fig. 3.

Pseudocode for non-negative tensorization is shown below.

```
# X (1024, 1024)
X = X.reshape([2] * 20) # folding to 20-mode tensor of shape (2,2,...,2)
S = X.abs().pow(p) # scoring with p-th power mapping

if nn_flag: # non-negative tensorization flag
    X, B = S, X.sign() # X is now non-negative part, and B keeps its sign
        information

axis = [0,1,2] # slicing modes (the first 3 modes for simplicity)
S = S.std(dim=axis, keepdim=True).expand_as(X) # std reduction
perm = torch.argsort(S.view(2,2,2,-1), dim=-1) # shared permutation
X = X.view(2,2,2,-1).gather(-1, perm).view([2] * 20) # reorder
```

The resulting sorted tensor contains only non-negative elements for tensorization. After reconstruction, the retained sign information restores the signs of the original elements. The reconstruction process is shown below:

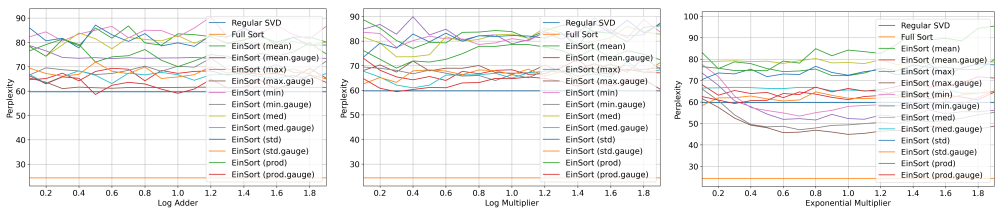
```
# tensor decomposition for sorted X(2,2,...,2)
factors = tensor_decomp(X, gauge=True) # gauge fixing to save memory

# tensor reconstruction: X is approx sorted non-negative tensor
X = tensor_reconst(factors, gauge=True)

# inverse permutation
perm_inv = torch.argsort(perm, dim=-1)
X = X.view(2,2,2,-1).gather(-1, perm_inv).view([2] * 20)

if nn_flag:
    X = X.clamp(0).pow(1/p) # approx. inverse nonlinear mapping
    X = X * B # inverse sign

# unfolding back to original tensor shape
X = X.reshape(1024, 1024)
```

Because the permutation is reversible, no information is discarded prior to tensor decomposition. Note that certain reversible permutations used in EinSort can be interpreted as classical analogues of entangling operations in tensor-network-based quantum circuits as discussed in Appendix L.

F NONLINEAR MAPPING AND REDUCTION

Beyond the power scaling in Fig. 5, Fig. 9 compares several nonlinear mappings. Most alternatives perform poorly, except for several reductions under exponential mapping. Overall, power scaling yields lower PPL than exponential mapping in these experiments. Table 5 lists the best PPL for each mapping/reduction combination, with and without gauge fixing. Product reduction performs best in most cases, whereas exponential scaling favors max reduction. The max and mean reductions with power scaling give the third- and fourth-best PPL values, 40.25 and 42.38, respectively. Appendix D provides intuition for these differences: product reduction is insensitive to the target magnitude, whereas mean reduction can be less reliable in low-magnitude regimes. In all cases, gauge fixing improves accuracy.

Fig. 10 shows results for Phi-3.5-mini at 50% and 80% compression. The trend differs from the Qwen3 case in Fig. 5. Specifically, the best power exponent is around 0.4 with non-negative tensorization, whereas values above 2 can perform best without it. The reduction operators are also comparable and attain their best scores at similar exponents. Nonetheless, gauge fixing and non-negative tensorization substantially narrow the gap to full sorting. For the Phi-3.5-mini case, the shared permutation requires 0.35 bits per KV-cache element.



(a) Additive logarithm (b) Multiplicative logarithm (c) Exponentiation

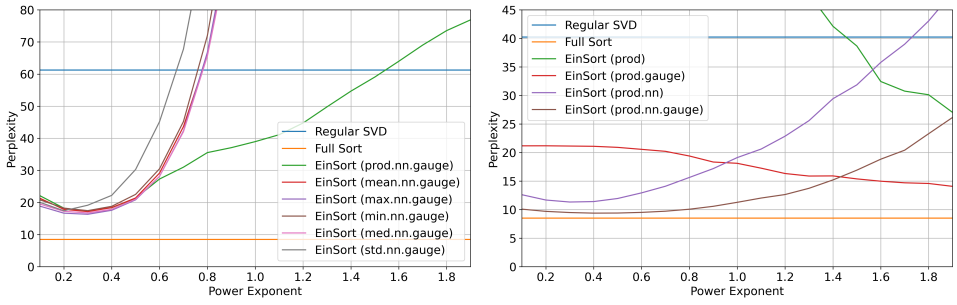
Figure 9: Different nonlinear mappings for Qwen3-1.7B at 50% compression: additive logarithm, multiplicative logarithm, and exponentiation.

Table 5: PPL with different nonlinear mappings and reductions for Qwen3-1.7B at 50% compression. The four best scores are highlighted. Regular SVD without sorting has 60.0 PPL.

Reduction	mean	max	min	median	std	prod
linear w/o gauge fixing	77.13	73.55	78.12	79.07	73.82	**59.56**
linear w/ gauge fixing	62.79	61.49	66.04	66.34	61.19	**50.84**
pow w/o gauge fixing	77.46	73.54	78.08	79.05	72.46	**59.43**
pow w/ gauge fixing	63.60	61.44	66.03	66.30	57.42	**50.56**
abs.pow w/o gauge fixing	63.75	75.25	81.11	65.47	69.91	**52.12**
abs.pow w/ gauge fixing	55.26	63.25	67.52	56.22	60.78	**44.72**
pow.nn w/o gauge fixing	50.08	46.56	68.21	55.66	51.37	**36.29**
pow.nn w/ gauge fixing	42.38	40.25	52.70	44.21	46.79	**33.17**
exp w/o gauge fixing	72.31	**51.60**	53.45	77.44	70.48	74.60
exp w/ gauge fixing	61.07	**44.90**	47.06	58.37	63.07	59.29
add.log w/o gauge fixing	76.52	73.30	80.50	77.72	74.31	**70.04**
add.log w/ gauge fixing	64.20	61.03	66.62	64.99	63.07	**58.61**
mul.log w/o gauge fixing	76.99	78.88	77.42	73.48	73.58	**69.01**
mul.log w/ gauge fixing	63.98	66.91	63.96	62.60	60.97	**59.41**

G GAUGE FIXING

We next describe the gauge-fixing procedure used in our experiments. Consider a 4-mode TT for a tensor X decomposed by four cores A, B, C, and D as depicted in Fig. 11. Because the representation has gauge freedom, the tensor cores representing X are not unique. Specifically, any full-rank junction matrix can be inserted together with its inverse between tensor cuts A–B, B–C, and C–D.



(a) 50% compression (b) 80% compression

Figure 10: Different reduction operators and nonlinear mappings for Phi-3.5-mini at 50% and 80% compression.

A standard canonicalization approach uses SVD or QR decomposition to make all cores orthogonal except at one orthogonality center. For example, we employ an r-truncated SVD for the product AB as

$$\mathsf{svd}_r[AB] = U_A S_B V_B, \tag{62}$$

where U_A and V_B contain the left and right singular vectors, S_B is a diagonal singular-value matrix, and r is the bond rank. We then update the tensor cores A and B as follows:

$$A \leftarrow U_A, \tag{63}$$
$$B \leftarrow S_B V_B. \tag{64}$$

We repeat the procedure for the updated product BC, writing $\mathsf{svd}_r[BC] = U_B S_C V_C$, and assign $U_B \to B$ and $S_C V_C \to C$. Similarly, we decompose $CD = U_C S_D V_D$ and assign $U_C \to C$ and $V_D \to D$. The diagonal tensor S_D can be merged into either C or D, or retained as the orthogonality center. This makes all cores except the orthogonality center orthogonal. This SVD-based gauge fixing is illustrated in Fig. 12(a).

This canonicalization fixes the gauge freedom and improves numerical stability during tensor decomposition. Beyond numerical stability, Koike-Akino et al. (2025b) discussed reducing the storage required for orthogonal matrices through Stiefel-manifold parameterizations based on the exponential map, Cayley transform, Householder reflections, Givens rotations, Taylor series, or Neumann series. Such Stiefel-manifold parameterizations can reduce the required number of parameters from dr to $dr - r(r+1)/2$ for a matrix of shape $d \times r$ with $d \geq r$.

However, Stiefel-manifold parameterizations generally require nonlinear mappings to construct orthogonal matrices. Koike-Akino et al. (2026a) proposed a simpler method to fix the gauge freedom by using block identity form. For example, decompose A by LU factorization as follows:

$$A = P_A L_A U_A, \tag{65}$$

where $P_A \in \mathbb{R}^{d\times d}$ is an optional pivoting permutation, $L_A \in \mathbb{R}^{d\times r}$ is a lower-triangular matrix, and $U_A \in \mathbb{R}^{r\times r}$ is an upper-triangular matrix. Because the permutation can be represented by an index transform using $\lceil \log_2(d!) \rceil$ bits with factoradic coding, it need not be stored as a full $d \times d$ matrix. We split the lower-triangular factor into an upper square block and a lower dense block:

$$L_A = \begin{bmatrix} \bar{L}_A \\ \underline{L}_A \end{bmatrix}, \qquad \bar{L}_A \in \mathbb{R}^{r\times r}, \qquad \underline{L}_A \in \mathbb{R}^{(d-r)\times r}. \tag{66}$$

Then L_A can be written in block-identity form as follows:

$$L_A = \underbrace{\begin{bmatrix} I_r \\ \underline{L}_A \bar{L}_A^{-1} \end{bmatrix}}_{L'_A \in \mathbb{R}^{d\times r}} \bar{L}_A. \tag{67}$$

The permutation P_A is optional, but without appropriate pivoting $\bar{L}_A$ can be singular. We then use the permuted block-identity factor as the new A and merge the remaining factor into B:

$$A \leftarrow P_A L'_A, \tag{68}$$
$$B \leftarrow \bar{L}_A U_A B. \tag{69}$$

Because the upper identity block of L'_A need not be stored, only the lower block $\underline{L}_A \bar{L}_A^{-1}$ is retained, reducing the parameter count from dr to $dr - r^2$. By repeating the LU procedure, the cores A, B, and C can be converted to block-identity form. Consequently, the total TT parameter count is reduced from $2dr(1+r)$ to $2dr(1+r) - 3r^2$. Fig. 12(b) shows LU-based gauge fixing.

The pseudocode is shown below:

```
def gauge_fixing(A, B): # left core A and right core B with bond rank r
    d, r = A.shape # assuming d > r
    assert B.shape[0] == r # assuming B is in (r, ...)

    # LU decomposition: P @ L @ U = A
    P, L, U = torch.linalg.lu(A)
```

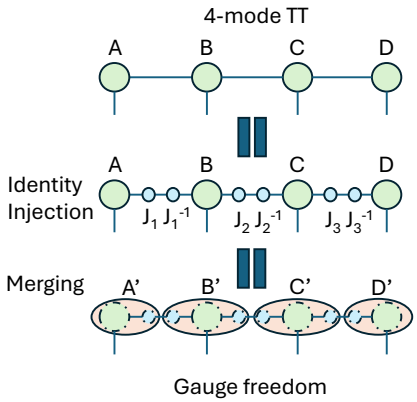


Figure 11: Gauge freedom: tensor cores are equivalent under insertion of any full-rank transform and its inverse at each cut.

```
    # permutation index from matrix
    perm = P.argmax(-1)
    assert torch.allclose(A, (L @ U)[perm]) # A = (L @ U)[perm]

    # partition
    L0 = L[:r] # upper part, which is unitriangular
    L1 = L[r:] # lower part

    # block identity's lower part: A = L1 @ L0.inverse()
    A = torch.linalg.solve_triangular(L0, L1, upper=False, left=False,
        unitriangular=True)

    # New A tensor can be constructed from reduced-parameter A and perm
        if needed
    # A = torch.cat((torch.eye(r), A))[perm]

    # New B tensor merging: (L0 @ U) @ B
    B = torch.einsum("ij,j...->i...", (L0 @ U), B)
    return A, perm, B
```

QR decomposition can be used as an alternative to LU decomposition. However, the current implementation of `torch.linalg.qr` can be numerically unstable when the columns are linearly dependent; pivoting, as in the LU procedure above, can mitigate this issue.

H ACTIVATION-AWARE TENSORIZATION

We note that EinSort can be integrated with activation-aware decomposition methods such as ASVD Yuan et al. (2023), AA-SVD Sinha & Fleuret (2026), and OBD-SVD Li et al. (2026). The core idea of these methods is to use input and/or output preconditioning. ASVD uses the following objective function:

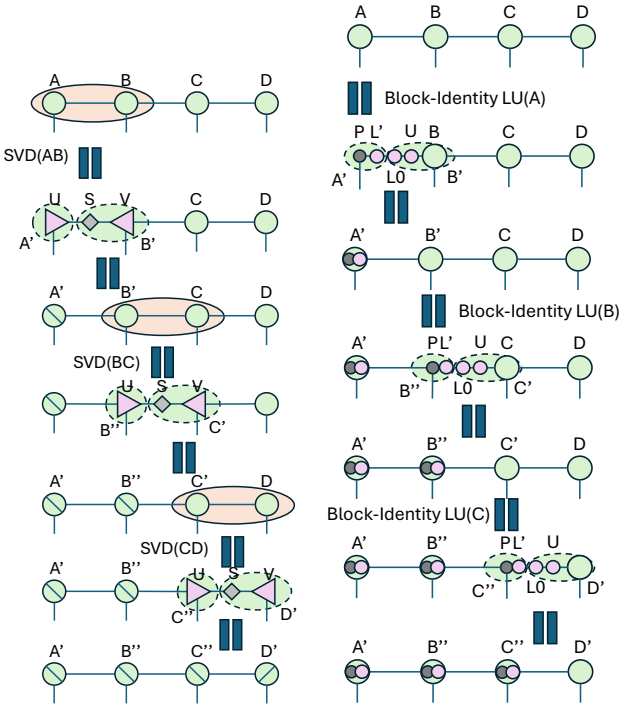
$$\hat{W} = \arg\min_{W'} \|WX - W'X\|^2 + \lambda\|W - W'\|^2, \tag{70}$$

where X contains input activations from calibration data and λ weights the activation-unaware reconstruction term. The solution has the form

$$\hat{W} = \mathsf{svd}_r[WC^{1/2}]C^{-1/2}, \tag{71}$$

where $C = XX^\top + \lambda I$ and $C^{1/2}$ is the corresponding preconditioner. AA-SVD extends this objective to account for error propagation as follows:

$$\hat{W} = \arg\min_{W'} \alpha\|WX - W'X'\|^2 + \beta\|WX - W'X\|^2 + \lambda\|W - W'\|^2, \tag{72}$$



(a) SVD canonicalization (b) LU canonicalization

Figure 12: Gauge fixing with SVD and LU canonicalization. SVD: Tensor cores can be left- or right-unitary around the orthogonality center, and Stiefel-manifold parameterization can represent orthogonal cores with fewer parameters (Koike-Akino et al., 2025b). LU: Tensor cores can be placed in block-identity form, which can likewise be represented with fewer parameters (Koike-Akino et al., 2026a).

where X' is the modified input activation after decomposing preceding layers, and α and β weight the two activation terms. The solution has the form

$$\hat{W} = \mathsf{svd}_r[WC_1C_2^{-1/2}]C_2^{-1/2}, \tag{73}$$

$$C_1 = \alpha XX'^{\top} + \beta XX^{\top} + \lambda I, \tag{74}$$

$$C_2 = \alpha X'X'^{\top} + \beta XX^{\top} + \lambda I. \tag{75}$$

OBD-SVD extends ASVD to use output preconditioning to minimize:

$$\hat{W} = \arg\min_{W'} \|G^{1/2}(W - W')C^{1/2}\|^2, \tag{76}$$

where G represents gradient-derived information used to approximate the Hessian. The solution has the form

$$\hat{W} = G^{-1/2}\mathsf{svd}_r[G^{1/2}WC^{1/2}]C^{-1/2}. \tag{77}$$

These cases, including ASVD, AA-SVD, and OBD-SVD, can be expressed using suitable preconditioners G, C_1, and C_2:

$$\hat{W} = G^{-1/2}\mathsf{svd}_r[G^{1/2}WC_1C_2^{-1/2}]C_2^{-1/2}. \tag{78}$$

EinSort can incorporate the same activation and gradient statistics as follows:

$$\hat{W} = G^{-1/2}\mathcal{T}_\theta^{\pi}[G^{1/2}WC_1C_2^{-1/2}]C_2^{-1/2}. \tag{79}$$

I Quantization-aware tensorization

The tensor cores can also be quantized or pruned to further reduce the memory footprint of low-rank tensor networks. Because einsum contractions are differentiable, quantization and pruning can be incorporated into gradient optimization, for example through straight-through estimation. Specifically, the tensor cores are quantized (or pruned) as $\hat{W} = \mathcal{Q}[W]$ where $\mathcal{Q}[\cdot]$ denotes the quantization operation (and/or pruning), which itself may be nondifferentiable. The quantized tensors are then passed to the einsum contraction using a straight-through estimator: $W' = \hat{W} + W - \mathrm{stopgrad}(W)$, where $\mathrm{stopgrad}$ denotes stopping the autograd path. The pseudocode is shown below.

```
def quantize_contract(self, target, *args, **kwargs):
    # quantize cores
    cores = self.quantize(self.cores, **kwargs)

    # straight-through estimator
    for k in range(len(cores)):
        cores[k] += self.cores[k] - self.cores[k].detach()

    # quantization-aware contraction to use gradient optimization
    return self.expression(*cores, *args, **kwargs)
```

J Test-time adaptation

Tensor ranks can be optimized across layers and modules (Luo et al., 2024). However, most work uses static rank selections that do not adapt during inference. Inspired by test-time quantization (TTQ) (Koike-Akino et al., 2026b), we can consider test-time rank adaptation. TTQ provides theoretical motivation for adapting a test-time weight approximation to each input prompt rather than using a static offline approximation.

Consider a simple toy example in which the weight matrix W is approximated by two rank-one factors, $\hat{W} = a_1b_1^{\top} + a_2b_2^{\top}$, and every column of the input activation X is orthogonal to both b_1 and b_2. Then, $\hat{W}X = 0$, which implies that both rank-one factors contribute nothing for such tokens, indicating that different rank components are needed. Conversely, if X is orthogonal to b_2 but aligned with b_1, e.g., $X = b_1c^{\top}$, then $\hat{W}X = |b_1|^2a_1c^{\top}$, so the second factor is redundant and can be

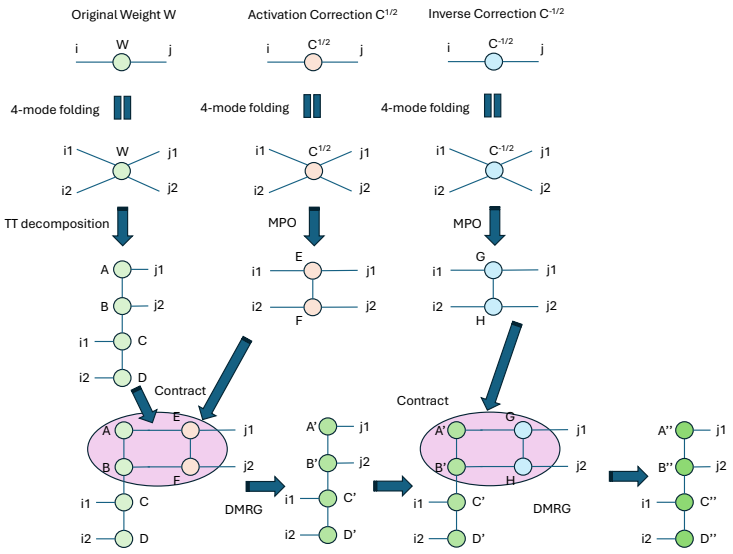


Figure 13: Test-time tensor train adaptation based on online activation preconditioner C.

omitted. This toy example illustrates why a weight decomposition can benefit from adapting to the input tokens at inference time.

We may use a simple scoring network to determine the bond ranks at test time. Specifically, a small neural network can take the online activation X as input and predict tensor-network ranks. The network can be trained on synthetic data or calibration data held out from test samples. This enables test-time rank adaptation for tensor networks.

Another alternative is to update a preconditioner at test time and adapt the tensor cores online. For example, let W of shape (d^2, d^2) use a 4-mode TT of shape (d, d, d, d) with a moderately high rank R. The online token correlation $C = XX^\top + \lambda I_{d^2}$ can be decomposed as a 2-mode matrix product operator (MPO) with site dimension d and bond rank R' using the density matrix renormalization group (DMRG) algorithm Schollwöck (2011). We can then update the 4-mode TT cores locally and sequentially by contracting with the MPO and retaining dominant eigenspaces subject to bond ranks below R. Fig. 13 illustrates this test-time adaptation framework, in which the TT is updated using the online preconditioner C.

K Activation Compression

In addition to LLM weights and the KV cache, EinSort can compress intermediate activations required for gradient updates. For example, consider a linear module with forward pass $Y = WX$. The weight update under backpropagation is then:

$$W \leftarrow W - \eta \underbrace{\frac{\partial \mathcal{L}}{\partial Y}}_{G} X^\top, \tag{80}$$

where η is the learning rate, $\mathcal{L}$ is the loss, and G is the gradient with respect to the output Y. This highlights the memory cost of LLM gradient updates, which require retaining intermediate activations X across layers. EinSort can therefore reduce the memory required for such intermediate activation tensors.

L Connection with Quantum Gates

Tensor index permutations in EinSort are closely related to entangling operations in quantum tensor networks. In particular, reversible Boolean permutations can be interpreted as classical analogues of quantum logic gates, including controlled-NOT (CNOT) and Toffoli gates. For binary indices, as in

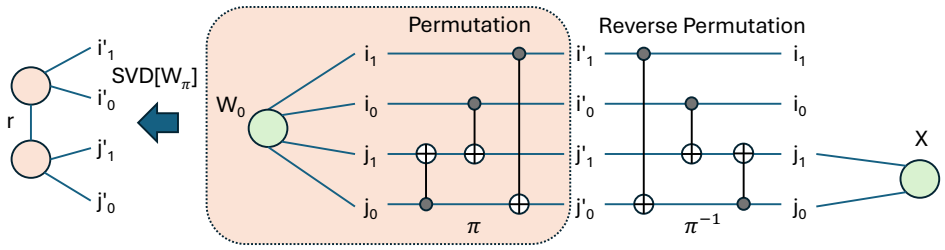


Figure 14: CNOT chain realizing the row-wise sorting π for W_0 in Eq. (82). The sorted matrix W_π can then be decomposed into low-rank tensor factors. The input X is contracted with the decomposed W_π through the inverse permutation π^{-1}, implemented by the reverse CNOT chain.

Eq. 1, the transformation $(i, j) \mapsto (i, i \oplus j)$ corresponds exactly to the action of a CNOT gate, where one index acts as a control bit and the other is conditionally flipped. Indeed, CNOT corresponds to a permutation matrix:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{81}$$

As another example, consider applying row-wise sorting to a 4×4 matrix W_0:

$$W_0 = \begin{matrix} \\ i \downarrow \end{matrix} \overset{j \rightarrow}{\begin{bmatrix} 4.0 & 1.0 & 2.0 & 3.0 \\ 2.1 & 3.1 & 4.1 & 1.1 \\ 1.2 & 3.2 & 4.2 & 2.2 \\ 3.3 & 2.3 & 1.3 & 4.3 \end{bmatrix}}, \pi = \begin{matrix} \\ i \downarrow \end{matrix} \overset{j_1 \oplus j_0 \oplus i_0,\ j_0 \oplus i_1 \rightarrow}{\begin{bmatrix} 0 & 3 & 2 & 1 \\ 2 & 1 & 0 & 3 \\ 2 & 1 & 3 & 0 \\ 3 & 0 & 1 & 2 \end{bmatrix}}, \quad W_\pi = \begin{bmatrix} 4.0 & 3.0 & 2.0 & 1.0 \\ 4.1 & 3.1 & 2.1 & 1.1 \\ 4.2 & 3.2 & 2.2 & 1.2 \\ 4.3 & 3.3 & 2.3 & 1.3 \end{bmatrix}, \tag{82}$$

where the row index $i \in \mathbb{Z}_4$ can be represented with two binary indices $i_1 \in \mathbb{Z}_2$ and $i_0 \in \mathbb{Z}_2$ as $i = 2i_1 + i_0$. Similarly, the column index $j \in \mathbb{Z}_4$ has two binary indices, $j_1, j_0 \in \mathbb{Z}_2$, such that $j = 2j_1 + j_0$. Then the row-wise permutation π can be expressed as: $j_1 \leftarrow j_1 + j_0 + i_0 \pmod 2$, $j_0 \leftarrow j_0 + i_1 \pmod 2$. The column index j now depends on the row index i, which is the key property enabling rank reduction discussed in Appendix D. This is an entangling operation realized by the CNOT chain in Fig. 14. The singular values of the unsorted matrix W_0 are approximately $[10.6, 4.1, 1.6, 0.6]$, whereas those of W_π are approximately $[11.5, 0.2, 0.0, 0.0]$, making W_π nearly rank 2. Hence, the sorted matrix can be approximated well by a lower-rank tensor decomposition. The CNOT chain is reversible, so the inverse chain can be used to contract the reconstructed tensor with an activation X, as shown in the figure.

However, CNOT and NOT gates alone realize only affine permutations. More generally, an arbitrary reversible permutation can be realized by CNOT plus Toffoli gates, e.g., $(i, j, k) \mapsto (i, j, k \oplus ij)$, where two control bits jointly determine the target transformation. From the tensor-network perspective, these reversible index transformations may convert highly entangled tensors into representations with lower effective bond dimensions, thereby exposing latent low-rank structure prior to tensor decomposition. This connection suggests broader links between tensor compression, reversible computation, and quantum-inspired representations, where index permutations play a role analogous to basis transformations that simplify entanglement patterns.

Motivated by the interpretation of permutations as compositions of Toffoli-like reversible gates, we can further reduce the memory footprint required to represent sorting permutations. Specifically, after obtaining a target permutation vector, we convert the permutation into a sequence of Toffoli-gate operations, thereby inducing an equivalent Boolean-function representation. Using ancilla bits can further reduce reversible-circuit complexity through pebble-game techniques Li et al. (1998). The resulting Boolean function can then be compressed into a compact sum-of-products form using standard logic synthesis techniques such as the Quine–McCluskey algorithm (McCluskey, 1956)

or Espresso (Brayton et al., 1984). Using PyEDA[4], we empirically confirmed that the number of permutation gates can be substantially reduced, particularly when introducing don't-care sets corresponding to low-score permutations that can be ignored with minimal impact on sorting quality. For example, the total number of gates was reduced from $1{,}976$ to 744 when allowing approximately 50% permutation cancellation through don't-care optimization. We also use RevKit[5] to simplify reversible permutation circuits.

M Algebraic polynomial permutation

CNOT gates act on binary indices, but the idea extends to arbitrary dimensions through modular additions such as $(i, j) \mapsto (i, j + i \bmod d_j)$ when the second dimension has size d_j. A related way to parameterize permutations is through algebraic permutation polynomials. For example, a quadratic permutation polynomial (QPP) Takeshita (2007) can be defined by the following mapping:

$$\pi(i) = a + b\,i + c\,i^2 \bmod n, \tag{83}$$

where $i \in \mathbb{Z}_n$ is an index modulo n. The bijectivity conditions are $\gcd(b, n) = 1$ and $\mathrm{rad}(n) \mid c$, where $\gcd$ denotes the greatest common divisor and rad denotes the radical. We may optimize the polynomial coefficients to improve the tensorization accuracy. However, a polynomial over a finite ring does not represent all permutations in general. For suitable finite-field representations, algebraic polynomials over a Galois field can instead be used to represent reversible permutations. For example, Lagrange interpolation over a Galois field can construct a polynomial $\pi(x) = \sum_i y_i \prod_{j \neq i} (x - x_j)/(x_i - x_j)$ satisfying selected index mappings $x_i \mapsto y_i$. To construct such polynomials over a Galois field, we use `galois.lagrange_poly`[6].

N Learned permutation operations

In this paper, we focus on practical permutations based on sliced sorting with nonlinear mappings and several reduction operators. Although sorting has theoretical motivation, as discussed in Appendix D, it is only one heuristic for choosing permutations. In more general settings, the permutations and nonlinear mappings could instead be learned jointly. For example, permutation operations can be optimized through gradient updates using NeuralSort (Grover et al., 2019), SoftSort Prillo & Eisenschlos (2020), or Gumbel–Sinkhorn (Mena et al., 2018). Our objective could be formulated as follows:

$$\alpha \|W - \mathcal{T}_\theta^\pi[W]\|^2 + \lambda |[\theta, \pi]|, \tag{84}$$

where $|[\theta, \pi]|$ denotes the total memory cost for tensorization hyperparameters θ and permutation π. The first term can also incorporate input and output preconditioners, as discussed in Appendix K.

We can also include nonlinear mappings in θ for both sorting and tensorization. Specifically, we can apply a nonlinear mapping such as exponentiation to W and use its inverse after decomposition for reconstruction. More specifically, we can write

$$\mathcal{T}_\theta^\pi[W] = \phi^{-1}[\pi^{-1}[\mathcal{T}_\theta[\pi[\phi[W]]]]], \tag{85}$$

where $\phi[\cdot]$ denotes reversible nonlinear mappings such as exponentiation, and $\phi[\cdot]^{-1}$ is its inverse. A noninvertible mapping such as the absolute value $|\cdot|$ can be made reversible by retaining sign information; we refer to this construction as non-negative tensorization. For nondifferentiable hyperparameters, reinforcement learning could instead be used to design permutations and nonlinear mappings together with tensor ranks, shapes, and topologies.

O Folding and ordering

Consider a four-mode tensor X of shape $[d_1, d_2, d_3, d_4]$, where, for example, d_1 indexes layers, d_2 distinguishes keys from values, d_3 indexes heads, and d_4 is the head dimension. Axis ordering, folding, and unfolding can be varied in several ways. For example, we may use:

[4] `https://github.com/cjdrake/pyeda`
[5] `https://github.com/msoeken/revkit`
[6] `https://github.com/mhostetter/galois`

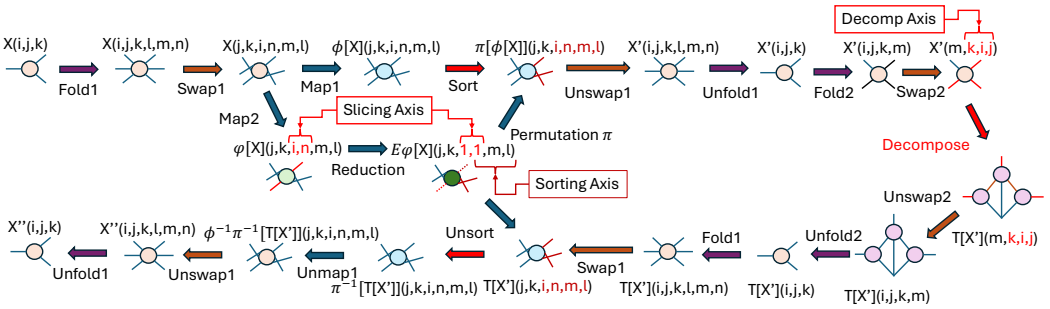


Figure 15: EinSort pipeline with generalized folding/unfolding, ordering, slicing, sorting, and mapping.

- a 4-mode TT for X as is;
- a 4-mode TT for an axis-swapped version of X, such as `X.permute(3,1,0,2)`;
- treating the second axis as a batch dimension and applying a 3-mode TT to the remaining three axes;
- unfolding to $[d_1 \times d_2, d_3, d_4]$ and applying a 3-mode TT: `X.reshape(-1, d3, d4)`;
- folding X into a 6-mode tensor such as `X.reshape(a,b,c,d,e,f)` and applying a 5-mode TT to the last five axes.

Because tensor decomposition depends on mode ordering, the axis order and folding shape can be adjusted to minimize reconstruction error. Similarly, before decomposition, we can choose which axes to sort. The sorting and decomposition axes need not be identical or even fully overlap. We can independently choose the axes used for slice reduction. Fig. 15 shows the EinSort framework using different folding/unfolding and axis ordering for sorting, reduction, and decomposition. The schematic also depicts different nonlinear mappings for sorting (ϕ) and decomposition (φ).

In addition, sorting can be applied sequentially across axes, as discussed in Appendix C; for example, axes $[3, 5]$ can be sorted first and $[2, 4]$ later rather than sorting all axes $[2, 3, 4, 5]$ jointly. If all axes have size d, jointly sorting the last four modes requires $\lceil \log_2(d^4!) \rceil / d^4$ bits per element, whereas the sequential scheme requires $2\lceil \log_2(d^2!) \rceil / d^2$ bits per element. For example, with $d = 32$, this reduces the requirement from 18.6 to 17.1 bits per element. Repeating the per-axis sequential sorting four times reduces this estimate further to 14.7 bits.

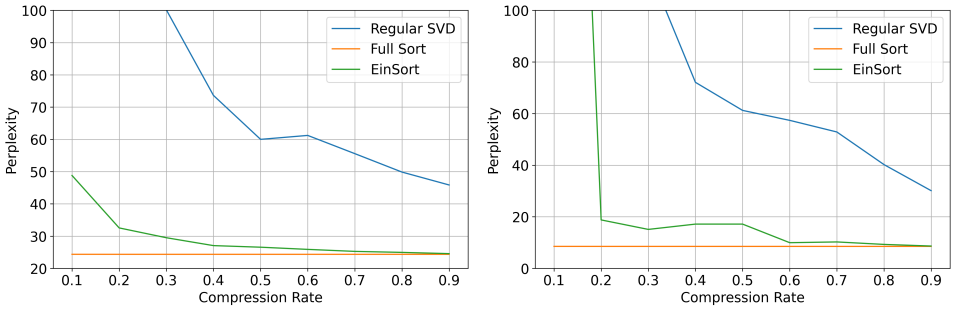
P Benefits and Limitations

EinSort has several benefits:

- Theoretical analysis suggests that sorting can reduce effective tensor ranks under specified assumptions.
- Sorting has a useful connection to entangling operations in quantum tensor networks.
- The einsum-based formulation supports a wide range of tensor-network topologies.
- EinSort provides substantial flexibility over hyperparameters, including slicing order, mode selection, folding combinations, nonlinear mappings, and reduction operations.
- The framework can be applied to several tensor types, including LLM weights, KV caches, and intermediate activations used for gradient computation.

Nonetheless, it has some drawbacks and limitations.

- Finding effective tensor topologies, folding shapes, and index orderings is not straightforward.
- Factorizing large tensors can be computationally expensive.



(a) Qwen3-1.7B model.

(b) Phi-3.5-mini (3.8B) model.

Figure 16: PPL versus compression rate for Qwen3-1.7B and Phi-3.5-mini (3.8B) models.

- Permutation metadata can remain a substantial source of memory overhead.
- The current implementation lacks high-performance CUDA kernels jointly optimized for contraction and index reordering.

Q Experimental setup

We conducted experiments on several foundation-model benchmarks to evaluate EinSort. Our experimental setup follows SparseLLM Bai et al. (2024b) and builds on its public codebase[7]. For weight-compression experiments, following existing work Sun et al. (2023), we decomposed all linear layers in the transformer backbone. We implemented EinSort in PyTorch Paszke et al. (2019) and used the Hugging Face Transformers library Wolf (2019) to handle models and datasets. All experiments were conducted on NVIDIA A40 or A100 GPUs.

For LLM experiments, we considered Qwen3 Yang et al. (2025), Gemma3 Gemma Team et al. (2025), Phi-3/Phi-3.5 Haider et al. (2024), and Phi-4 Abouelenin et al. (2025); Aneja et al. (2026) models. We evaluated multiple model sizes to characterize EinSort across scales. For language modeling, we measured perplexity on the raw WikiText-2 (WT2) benchmark Merity et al. (2016). Details of the models and datasets are provided in Appendices S and T.

For KV-cache experiments, we reduce along the head axis and sort over a stack of 64 head-dimension vectors. The resulting sliced permutation requires $\lceil \log_2((64 \times d)!) \rceil / (64 \times d \times h)$ bits per KV-cache element, where d is the head dimension and h is the number of KV heads. For Qwen3-0.6B/1.7B, sliced sorting requires $\lceil \log_2((64 \times 128)!) \rceil / (64 \times 128 \times 8) \simeq 1.44$ bits per KV-cache element. For Gemma3-4B, the corresponding cost is $\lceil \log_2((64 \times 256)!) \rceil / (64 \times 256 \times 4) \simeq 3.14$ bits per KV-cache element. For Phi-3.5-mini, the permutation requires $\lceil \log_2((64 \times 96)!) \rceil / (64 \times 96 \times 32) \simeq 0.35$ bits per KV-cache element. The slicing configuration can be adjusted to control this memory overhead.

R Additional benchmark results

R.1 LLM model variants

In addition to Qwen3-0.6B and Gemma3-4B, we report WT2 perplexity for Qwen3-1.7B and Phi-3.5-mini in Fig. 16. Fig. 16a shows PPL for Qwen3-1.7B when its KV cache is compressed by regular SVD, full-sort SVD, and EinSort. The trend is similar to that of Qwen3-0.6B in Fig. 4.

Fig. 16b shows the corresponding PPL for Phi-3.5-mini. EinSort approaches the full-sort result while using only 0.35 bits per KV-cache element for permutation metadata, compared with approximately 11.1 bits for full sorting in this setting.

[7] https://github.com/BaiTheBest/SparseLLM

Table 6: Success rate (↑) for X-VLA with different compression methods on the LIBERO robot-manipulation benchmark at a compression rate of 0.1. Boldface and underlining denote the best and second-best results, respectively. An asterisk "*" indicates performance comparable to the original uncompressed VLA.

Method	Spatial	Object	Goal	Long	Avg	(95% CI)
Original	95.0%	98.5%	93.5%	87.5%	93.63%	(91.72–95.12%)
Plain SVD	7.5%	1.0%	0.0%	2.0%	2.63%	(1.72–3.98%)
ASVD	32.5%	21.0%	<u>5.0</u>%	13.0%	17.88%	(15.38–20.68%)
SVD-LLM	41.0%	<u>26.0</u>%	4.0%	31.0%	25.50%	(22.60–28.63%)
LatentLLM	<u>43.0</u>%	<u>26.0</u>%	3.5%	<u>32.5</u>%	<u>26.25</u>%	(23.32–29.41%)
EinSort	**93.5**%	**97.5**%	**90.5**%	***88.0**%	**92.38**%	(90.33–94.02%)

R.2 GSM8K PROTOCOL NOTE

The main paper reports the GSM8K KV-cache compression results in Fig. 6. Dataset details and licensing information are provided in Appendix T.

R.3 VLM VISUAL REASONING: PROTOCOL NOTE

The main paper reports the Qwen3-VL-4B and Qwen3-VL-8B TextVQA weight-compression results in Fig. 7. All activation-aware baselines and EinSort use the same 64 calibration samples from the TextVQA training split.

R.4 VLA ROBOT MANIPULATION: PER-SUITE RESULTS

Table 4 in the main paper summarizes the average LIBERO success rates. Here, we provide the corresponding per-suite results and qualitative rollouts for several vision-language-action (VLA) models. Specifically, we evaluate X-VLA Zheng et al. (2025), $\pi_{0.5}$ Intelligence et al. (2025), and VLA-JEPA Sun et al. (2026) on the LIBERO benchmark Liu et al. (2023a). Their VLM backbones are Florence-2 Xiao et al. (2024a), PaliGemma Beyer et al. (2024), and Qwen3-VL Bai et al. (2025), respectively. Here, we focus on model-weight compression with EinSort rather than KV-cache reduction. We compress all linear modules in the attention and multilayer perceptron (MLP) blocks of each VLA language model.

We compare plain SVD, ASVD Yuan et al. (2023), SVD-LLM Wang et al. (2024b), and LatentLLM Koike-Akino et al. (2026a) as baselines. Except for plain SVD, all methods use 64 episodes from the LIBERO-Spatial suite for activation calibration and preconditioning. We evaluate success rates over 800 rollouts on four LIBERO benchmarks: Spatial, Object, Goal, and Long. Tables 6, 7, and 8 show the success rates for X-VLA, $\pi_{0.5}$, and VLA-JEPA, respectively. Across these LIBERO evaluations, EinSort substantially outperforms the compared weight-decomposition baselines. EinSort occasionally exceeds the uncompressed VLA baseline numerically, but these differences lie within the Wilson confidence intervals and are not interpreted as significant improvements. Fig. 17 shows example video snapshots from the compressed VLA-JEPA model on the LIBERO benchmark suites.

S LARGE FOUNDATION MODELS

Qwen3 We use Qwen3 (Yang et al., 2025) dense models, a family of decoder-only transformers built on a consistent architecture with RMSNorm, SwiGLU feed-forward layers, and rotary positional embeddings. All variants employ grouped-query attention (GQA) with a fixed small number of key–value heads while scaling the number of query heads with model width, reducing KV-cache cost. Importantly, the hidden size is decoupled from the attention projection width, providing additional flexibility. Parameters are listed in Table 9. The family is released under the Apache-2.0 license.

Table 7: Success rate (↑) for the $\pi_{0.5}$ VLA model with different compression methods on the LIBERO robot-manipulation benchmark at a compression rate of 0.4. Boldface and underlining denote the best and second-best results, respectively. An asterisk "*" indicates performance comparable to the original uncompressed VLA.

Method	Spatial	Object	Goal	Long	Avg	(95% CI)
Original	97.5%	100.0%	97.0%	96.5%	97.75%	(96.47–98.57%)
Plain SVD	27.0%	34.5%	8.5%	1.0%	17.75%	(15.26–20.55%)
ASVD	4.0%	5.5%	7.5%	0.0%	4.25%	(3.06–5.88%)
SVD-LLM	60.0%	79.0%	32.0%	27.0%	49.50%	(45.05–52.96%)
LatentLLM	66.5%	88.5%	37.0%	37.5%	57.38%	(53.92–60.76%)
EinSort	**96.0%**	**99.5%**	***97.5%**	**95.5%**	**97.13%**	(95.72–98.08%)

Table 8: Success rate (↑) for VLA-JEPA with different compression methods on the LIBERO robot-manipulation benchmark at a compression rate of 0.8. Boldface and underlining denote the best and second-best results, respectively. An asterisk "*" indicates performance comparable to the original uncompressed VLA.

Method	Spatial	Object	Goal	Long	Avg	(95% CI)
Original	97.5%	100.0%	99.0%	93.5%	97.50%	(96.17–98.38%)
Plain SVD	0.0%	0.0%	0.0%	0.0%	0.00%	(0.00–0.48%)
ASVD	62.0%	75.5%	49.5%	21.0%	52.00%	(48.54–55.44%)
SVD-LLM	18.5%	16.0%	19.0%	0.5%	13.50%	(11.31–16.04%)
LatentLLM	87.0%	64.5%	59.0%	28.5%	59.75%	(56.31–63.09%)
EinSort	***98.0%**	***100.0%**	**98.0%**	***95.0%**	***97.75%**	(96.47–98.57%)

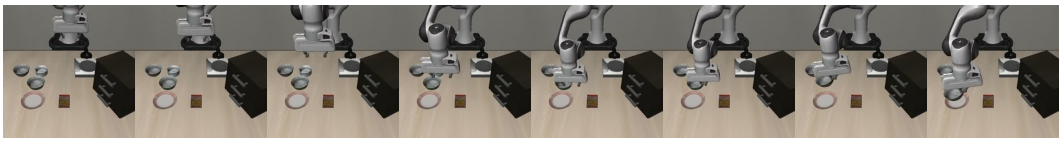
Gemma3 Gemma3 models (Gemma Team et al., 2025) are decoder-only transformer architectures released across a wide range of scales, from 270M to 27B parameters, and include both text-only and multimodal variants. Similar to Qwen3, all Gemma3 models adopt RMSNorm, SwiGLU feed-forward networks, and rotary positional embeddings, as well as grouped-query attention (GQA). The per-head dimension is fixed at 256 across model sizes. Parameters are listed in Table 10. The family is licensed under the Gemma terms of use.

Phi-3 and Phi-3.5 Phi-3 and Phi-3.5 models (Haider et al., 2024) are language models developed for efficient on-device and edge AI. These models target strong reasoning and coding performance through high-quality training data and optimized transformer architectures. Parameters are listed in Table 11. The family is released under the MIT license.

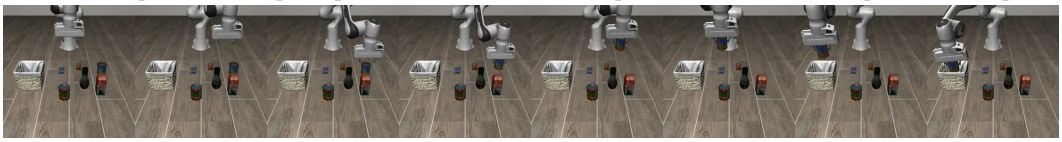
Phi-4 The Phi-4 family (Abdin et al., 2024) comprises compact language and multimodal foundation models for efficient reasoning, coding, and on-device AI. The family includes text-only, reasoning-specialized Aneja et al. (2026), and multimodal variants Abouelenin et al. (2025) supporting language, vision, and audio inputs, while maintaining strong efficiency relative to model size. Parameters are listed in Table 12. The family is released under the MIT license.

Qwen3-VL Qwen3-VL models Bai et al. (2025) extend the Qwen3 transformer with vision–language capabilities. The models integrate a SigLIP-based vision encoder Zhai et al. (2023) with the Qwen3 language backbone Yang et al. (2025) through a projection module, enabling joint reasoning over images, videos, and text for tasks such as visual question answering, captioning, and document understanding. Parameters are listed in Table 13. The family is licensed under Apache-2.0.

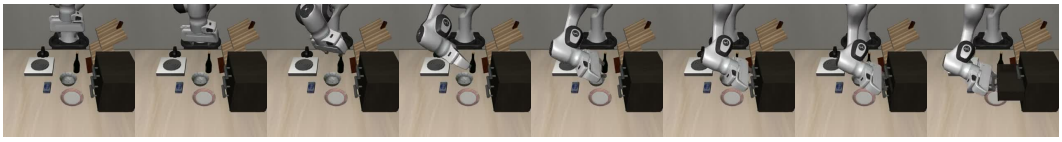
X-VLA X-VLA (Cross-modal Vision–Language–Action) Zheng et al. (2025) is a unified framework that jointly models visual perception, language understanding, and action generation within a shared representation space. Leveraging multimodal pretraining, X-VLA captures compositional and temporal structure, allowing robust generalization across tasks and environments for embodied AI systems. It uses Florence-2 Xiao et al. (2024a) as its VLM backbone, with a DaViT-B vision



(a) LIBERO-Spatial task 0: “pick up the black bowl between the plate and the ramekin and place it on the plate”



(b) LIBERO-Object task 0: “pick up the alphabet soup and place it in the basket”



(c) LIBERO-Goal task 0: “open the middle drawer of the cabinet”



(d) LIBERO-Long task 0: “put both the alphabet soup and the tomato sauce in the basket”

Figure 17: Robot-manipulation video snapshots for the compressed VLA-JEPA model on the LIBERO benchmark suites.

Table 9: Architecture parameters of Qwen3 dense models Yang et al. (2025)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
0.6B	28	16	8	1024	128	3072	Qwen/Qwen3-0.6B
1.7B	28	16	8	2048	128	6144	Qwen/Qwen3-1.7B
4B	36	32	8	2560	128	9728	Qwen/Qwen3-4B
8B	36	32	8	4096	128	12288	Qwen/Qwen3-8B
14B	40	40	8	5120	128	17408	Qwen/Qwen3-14B
32B	64	64	8	5120	128	25600	Qwen/Qwen3-32B

encoder Ding et al. (2022). The model parameters are listed in Table 14. The model is released under the Apache-2.0 license.

$\pi_{0.5}$ The $\pi_{0.5}$ model Intelligence et al. (2025) is a 2.3B-parameter VLA transformer that maps visual observations and language instructions to continuous robot actions through a flow-matching diffusion policy. It uses PaliGemma Beyer et al. (2024) as the VLM backbone and Gemma Team et al. (2024) in the flow-matching diffusion policy. Trained via distillation from a larger foundation model on diverse robot interaction data, it achieves strong zero-shot generalization while remaining lightweight and deployable for real-world manipulation tasks. Parameters are listed in Table 15. The model is licensed under the Gemma terms of use.

VLA-JEPA VLA-JEPA (Vision–Language–Action with Joint Embedding Predictive Architecture) Sun et al. (2026) augments VLA policies with a latent predictive world model based on V-JEPA2 Assran et al. (2025), enabling the agent to learn action-relevant future representations without reconstructing pixels. Built on a Qwen3-VL-2B VLM backbone Bai et al. (2025) and a V-JEPA2 latent encoder based on ViT-L, it predicts future latent states and generates robot actions using a diffusion policy based on DiT-B Peebles & Xie (2023). By jointly learning perception, language grounding, and latent dynamics, VLA-JEPA achieves strong generalization and robustness across

Table 10: Gemma3 instruction-tuned text transformer parameters Gemma Team et al. (2025)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
270M	18	4	1	640	256	2048	google/gemma-3-270m-it
1B	26	4	1	1152	256	6912	google/gemma-3-1b-it
4B	34	8	4	2560	256	10240	google/gemma-3-4b-it
12B	48	16	8	3840	256	15360	google/gemma-3-12b-it
27B	62	32	16	5376	256	21504	google/gemma-3-27b-it

Table 11: Phi-3/Phi-3.5 instruction-tuned text transformer parameters Haider et al. (2024)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
mini (3.8B)	32	32	32	3072	96	8192	`microsoft/Phi-3.5-mini-instruct`
small (7B)	32	32	8	4096	128	—	`microsoft/Phi-3-small-8k-instruct`
medium (14B)	40	40	10	5120	128	17920	`microsoft/Phi-3-medium-4k-instruct`

manipulation tasks while improving data efficiency and transfer to unseen environments. Parameters are listed in Table 16. The model is released under the Apache-2.0 license.

T DATASETS

WikiText-2 (WT2) WikiText-2 (WT2) (Merity et al., 2016) is a language-modeling benchmark derived from verified good and featured articles on Wikipedia. The dataset is available under the CC BY-SA-4.0 license. The `wikitext-2-raw-v1` configuration contains 36,718, 3,760, and 4,358 samples in the train, validation, and test splits, respectively. We use `https://huggingface.co/datasets/mindchain/wikitext2`.

GSM8K The Grade School Math 8K (GSM8K) dataset (Cobbe et al., 2021) is a benchmark for evaluating mathematical reasoning in LLMs. It contains grade-school-level word problems written by human annotators, each paired with a detailed natural-language solution and a final numeric answer. The dataset is designed to assess multi-step arithmetic reasoning rather than factual recall, requiring models to generate intermediate reasoning steps to solve problems correctly. GSM8K is released under the MIT license and contains 7,473 training samples and 1,319 test samples. We use `https://huggingface.co/datasets/openai/gsm8k`.

TextVQA TextVQA Singh et al. (2019) evaluates a model's ability to read and reason about text in images when answering visual questions. Specifically, models must recognize text in the images and reason over that text to answer the questions. TextVQA contains 45,336 questions over 28,408 images from the OpenImages dataset. We use `https://huggingface.co/datasets/lmms-lab/textvqa`, licensed under CC-BY-4.0.

LIBERO LIBERO Liu et al. (2023a) is a benchmark for robotic vision–language–action (VLA) learning that evaluates long-horizon, compositional manipulation in simulated household environments. It provides five benchmark suites: Spatial, Object, Goal, Long, and Short. The Short suite contains 90 task variants, while each of the other suites contains 10 tasks. Each task includes multimodal data—video observations, language instructions, and low-level control trajectories—enabling end-to-end learning from vision and language to robot actions. We use `https://huggingface.co/datasets/lerobot/libero`, licensed under Apache-2.0.

Table 12: Phi-4 instruction-tuned text transformer parameters Abouelenin et al. (2025); Aneja et al. (2026)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
mini (3.8B)	32	24	8	3072	128	8192	`microsoft/Phi-4-mini-instruct`
reasoning (14B)	40	40	10	5120	128	17920	`microsoft/Phi-4-reasoning`

Table 13: Qwen3-VL transformer parameters Bai et al. (2025)

Model	layers	heads	KV heads	hidden size	head dim	MLP dim	Hugging Face ID
2B	24	16	8	2048	128	11008	Qwen/Qwen3-VL-2B-Instruct
4B	32	32	8	4096	128	22016	Qwen/Qwen3-VL-4B-Instruct
8B	36	32	8	4096	128	22016	Qwen/Qwen3-VL-8B-Instruct
32B	64	40	8	5120	128	27392	Qwen/Qwen3-VL-32B-Instruct

U LIBRARIES

We use the following public libraries:

- PyTorch `https://github.com/pytorch/pytorch`; version 2.9.1; BSD-3-Clause license
- opt_einsum `https://github.com/dgasmith/opt_einsum`; version 3.3.0; MIT license
- einops `https://github.com/arogozhnikov/einops`; version 0.8.2; MIT license
- TensorLy `https://github.com/tensorly/tensorly`; version 0.9.0; BSD-3-Clause license
- transformers `https://github.com/huggingface/transformers`; version 5.8.0; Apache-2.0 license
- datasets `https://github.com/huggingface/datasets`; version 4.85; Apache-2.0 license
- SciPy `https://github.com/scipy/scipy`; version 1.17.1; BSD-3-Clause license
- PyEDA `https://github.com/cjdrake/pyeda`; version 0.29.0; BSD-2-Clause license
- RevKit `https://github.com/msoeken/revkit`; MIT license
- Galois `https://github.com/mhostetter/galois`; version 0.4.11; MIT license
- LeRobot `https://github.com/huggingface/lerobot`; version 0.5.1; Apache-2.0 license

Table 14: X-VLA transformer parameters Zheng et al. (2025): Hugging Face ID lerobot/xvla-libero

Module	layers	heads	KV heads	hidden size	head dim	MLP dim
VLM: Florence-2-Large	24	16	16	1024	64	4096
Vision Encoder: DaViT	12	64	64	2048	32	8192

Table 15: $\pi_{0.5}$ VLA transformer parameters Intelligence et al. (2025): Hugging Face ID lerobot/pi05_libero_finetuned

Module	layers	heads	KV heads	hidden size	head dim	MLP dim
LLM: Gemma-2B	18	8	1	2048	256	16384
Vision Encoder: SigLIP ViT-L	24	16	16	1024	64	4096
Diffusion Policy: Gemma-300M	18	8	1	1024	128	4096

Table 16: VLA-JEPA transformer parameters Sun et al. (2026): Hugging Face ID lerobot/VLA-JEPA-LIBERO

Module	layers	heads	KV heads	hidden size	head dim	MLP dim
VLM: Qwen3-VL-2B	36	16	8	1536	96	8960
World model: V-JEPA2 ViT-L	24	16	16	1024	64	4096
Diffusion Policy: DiT-B	12	12	12	768	64	3072