BASIS OPTIMIZATION FOR SPECTRAL COMPRESSION OF DEEP LEARNING MATRICES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale inference pipelines are typically memory-bound, so smaller stored forms of their matrices reduce latency. Spectral compression trades a controlled error for a shorter description by discarding the least important directions of a matrix. This paper analyzes whether the basis in which the truncation occurs is an optimization target that yields savings beyond plain spectral compression. We (i) formalize basis-optimized spectral compression as a task scored by two costs, the parameters a representation stores and the values a product computed with it reads, (ii) introduce Basis-Native Disentanglement (BND), which adapts tensor disentanglement from quantum many-body physics by anchoring the basis search at the native coordinates and by building the rotation from a bounded number of planes to control its parameter cost, and (iii) run it on 1337 matrices from six families of deep learning matrices and two controls. At a relative truncation error of one percent, BND never stores more parameters than raw storage and stores fewer on 86% of matrices. Key-value caches and weights are the only matrix families that hold statistically significant structure beyond their spectrum that only a learned basis reveals. Saved parameters convert to load time at the rate of the channel beyond a fixed overhead. Our findings mark basis-optimized spectral compression as a real but narrow axis for compressing deep learning matrices.

1 INTRODUCTION

Inference in modern deep learning is often bound by memory bandwidth rather than arithmetic, since an accelerator must stream the stored parameters before it can compute with them. In these regimes, smaller stored forms yield measurable latency improvements, so practitioners are willing to trade controlled errors for a shorter description. Spectral compression treats a matrix as a linear operator, orders its rank one directions from the singular value decomposition (SVD) by magnitude and discards the tail to save parameters, so the error lands where the operator acts most weakly. This approach has been applied to deep learning with varying success (Denton et al., 2014; Wang et al., 2025).

Tensor disentanglement (TD) from quantum many-body (QMB) physics (Vidal, 2007; Evenbly & Vidal, 2009; Wei et al., 2025) performs a similar operation, yet learns an additional change of basis that exposes low-rank structure the original coordinates hide. Whether deep learning matrices hide such structure, and whether it can be used effectively for net compression, remains an open question. Our paper tries to answer it with three contributions (Appendix A lists what we do not claim).

Task formulation. We shift the task in QMB physics from rank truncation to optimizing two parameter counts that include every auxiliary matrix that cannot be absorbed otherwise, net storage (parameters persisted on disk) and net product (values read in every product during computation).

Basis-Native Disentanglement. We propose a method for this task that builds on traditional TD and tailors it to the requirements found in DL, (i) anchoring the search in the native basis for easy access to surface structure when present, (ii) building the rotation from a bounded number of planes to control the parameters the rotation can spend, and (iii) a CUDA kernel that computes products directly from the stored factors at over 80% of the memory bandwidth of the device.

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.

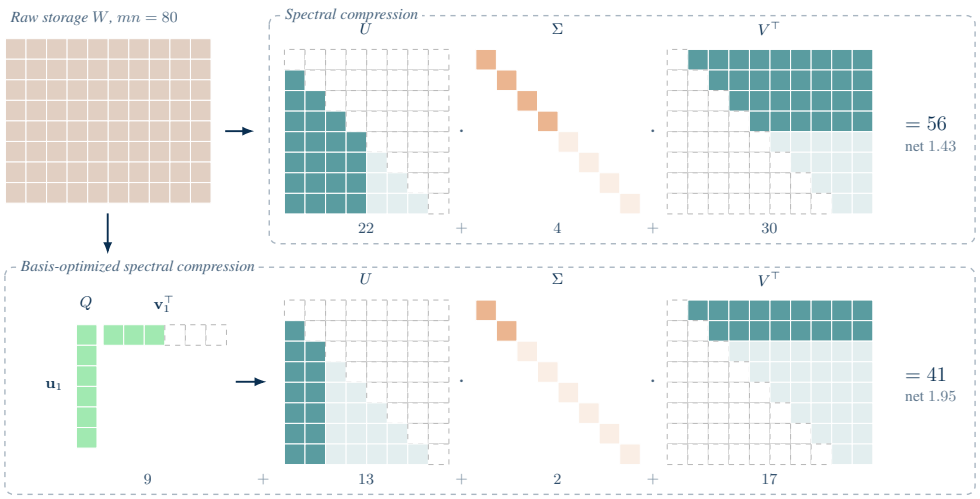

Figure 1: Two ways to store one matrix, drawn at schematic size with one cell per number. The count under each block is the number of free parameters it stores. Solid cells are stored and faint cells are cut. A plane pays for itself when the faint area it adds exceeds its own cells (Section 3).

Feasibility study. Running BND on 1337 matrices shows that an optimized basis can find hidden structure in deep learning matrices, which makes it a promising axis for compression. We find that matrices of one family behave alike, where caches and weights hold genuine structure a learned basis can exploit, activations point the same way without significance. Gradients and state spaces compress by more than an order of magnitude without rotation. Wherever parameters are saved they convert to time on the bandwidth bound channels beyond the fixed overhead of the device.

2 Related Work

Model compression techniques broadly reduce one of two quantities (Liang et al., 2021). Quantization methods reduce the bit precision assigned to each numerical entry (Frantar et al., 2022; Dettmers et al., 2022), whereas pruning (Han et al., 2015) and low-rank factorization (Denton et al., 2014) reduce the total count of parameters. Within this domain, low-rank approximation interprets network matrices as linear operators and truncates their trailing singular components. This principle originated in early work on parameter prediction and convolutional layer factorization (Denil et al., 2013; Denton et al., 2014; Jaderberg et al., 2014), and it remains prominent in large language models through activation-aware and truncation-aware variants (Yuan et al., 2023; Wang et al., 2025; Ashkboos et al., 2024a). Analogous low-rank assumptions underpin parameter-efficient fine-tuning (Hu et al., 2021), communication-efficient distributed training (Vogels et al., 2019; Zhao et al., 2024) and inference cache compression (Hooper et al., 2024; Liu et al., 2024). Across this entire body of work, however, truncation is computed strictly in the native coordinate basis in which the tensors are instantiated. None of these methods learns an alternate basis to expose latent rank structure.

Rotation prior to quantization to suppress activation outliers and fit weights into low-bit formats is a known concept (Chee et al., 2023; Ashkboos et al., 2024b; Liu et al., 2025). Those rotations act independently on rows or columns and preserve all singular values, which allows them to be absorbed into adjacent operations without adding stored parameters to the model. In contrast, any basis transformation capable of actually reducing algebraic rank cannot be absorbed into adjacent linear layers, which means the transformation itself must be explicitly stored, as proven in Appendix B.4.

This requirement connects our objective to tensor network disentanglement in quantum many-body physics, where unitary transformations are optimized to reveal latent low-rank structure prior to truncation (Vidal, 2007; Evenbly & Vidal, 2009; Hauru et al., 2020; Wei et al., 2025). There, however, the object being compressed is a quantum state whose number of amplitudes grows exponentially with the size of the system. A disentangler acts on a few neighbouring sites, costs a fixed number of

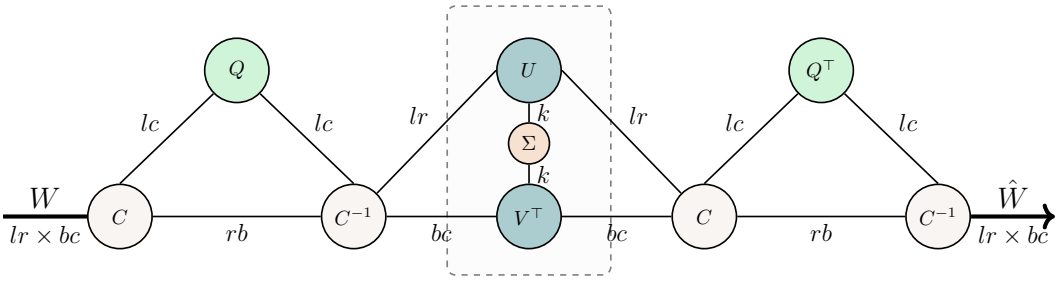

Figure 2: Basis-Native Disentanglement as a wire diagram, read left to right. The reshape C regroups the four indices of W so that l and c share one wire, the rotation Q acts on that wire alone, and C^{-1} returns to the native grouping, where the rank k truncation happens. The right half undoes the rotation and gives the reconstruction $\hat{W}$. The dashed box marks the stored factors, and the plane vectors of Q are the only other stored object.

parameters and is often shared across the whole system (Vidal, 2007; Evenbly & Vidal, 2009). Its cost is therefore negligible next to what it saves, so only the rank is optimized.

3 METHOD

The task. A deep learning matrix is not exponentially large, its disentangler costs a sizable fraction of the matrix itself, and it cannot be absorbed by other factors during rank truncation (Appendix B.4). Compression for deep learning is therefore only successful when the representation reduces the total count both in storage and in every product at inference.

Given a target matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, a representation R is any stored object from which a reconstruction $\hat{\boldsymbol{W}}(R)$ can be computed. We define a storage cost $\kappa_{\mathrm{s}}(R)$, the number of parameters that must be stored on disk for full reconstruction. Similarly, we define a product cost $\kappa_{\mathrm{p}}(R)$, the number of stored values a product $y = \hat{\boldsymbol{W}}(R)\, x$ reads. Each cost has a net compression factor $\mathrm{net}_*(R) = mn/\kappa_*(R)$ against the mn values of the matrix in its dense, quadratic form. As in any lossy compression, a representation is only valid if it reconstructs the matrix within a relative error budget ε, so the task is one of two maximizations,

$$\max_R \ \mathrm{net}_{\mathrm{s}}(R) \quad \text{or} \quad \max_R \ \mathrm{net}_{\mathrm{p}}(R), \qquad \text{each subject to} \qquad \mathrm{err}(R) \ = \ \frac{\|\boldsymbol{W} - \hat{\boldsymbol{W}}(R)\|_F}{\|\boldsymbol{W}\|_F} \ \leq \ \varepsilon. \tag{1}$$

Tensor disentanglement. We begin with the formulation of TD by Evenbly & Vidal (2009) and Wei et al. (2025). Their method builds on reshaping $\boldsymbol{W}$ into a tensor with four indices and grouping them differently, which gives a second matrix with the same entries but a different spectrum. An orthogonal $\boldsymbol{Q}$ acting on the rows of one of the two matrices moves entries between the rows and the columns of the other, so unlike a rotation of that matrix it changes its spectrum. The matrix can therefore be rotated until its spectrum decays fast.

Formally, choose factorizations $m = l\, r$ and $n = b\, c$, taking l and b as the largest divisors of m and n that do not exceed their square roots. The native matricization groups (l, r) against (b, c) and equals $\boldsymbol{W}$ itself, and the crossed matricization $C(\boldsymbol{W}) \in \mathbb{R}^{lc \times rb}$ groups (l, c) against (r, b). The disentangler $\boldsymbol{Q}$ is an orthogonal matrix acting on one grouping of indices, chosen to lower the rank across the other. It is the only learned component, and it is learned separately for every matrix (Section 3.3). In the formulation of Wei et al. (2025), $\boldsymbol{Q} \in \mathbb{R}^{lr \times lr}$ acts on the native row group and the truncation $\mathcal{T}_k$ is taken in the crossed cut, after which the rotation is undone,

$$\hat{\boldsymbol{W}}_{\mathrm{TD}} \ = \ \boldsymbol{Q}^\top C^{-1}\, \mathcal{T}_k\, C\, \boldsymbol{Q}\, \boldsymbol{W}, \tag{2}$$

which is designed to lower the rank the truncation needs to meet the budget. The reshapes are permutations and $\boldsymbol{Q}$ is orthogonal, so all error originates in the truncation, and a rotation acting on one grouping of indices leaves the singular values of that cut unchanged, which is why the disentangler cannot be absorbed into the factors and must be stored (Appendix B.4).

3.1 REPRESENTATION

The rotation. A dense rotation costs about as many parameters as the matrix it compresses, so we build $\boldsymbol{Q}$ from a small number h of planes. Any such $\boldsymbol{Q}$ is exactly h independent rotations, each turning within its own two-dimensional plane, and the planes themselves are learned, so the fit decides both where to rotate and by how much.

Formally, a dense rotation of the $p = lc$ rows of the crossed cut carries $p(p-1)/2$ free parameters, a cost that cannot be made smaller. We build $\boldsymbol{Q}$ from h planes instead,

$$\boldsymbol{Q} \;=\; (\boldsymbol{I}+\boldsymbol{A})^{-1}(\boldsymbol{I}-\boldsymbol{A}), \qquad \boldsymbol{A} \;=\; \boldsymbol{P}\boldsymbol{J}\boldsymbol{P}^{\top} \;=\; \sum_{i=1}^{h}\left(\boldsymbol{u}_i\boldsymbol{v}_i^{\top}-\boldsymbol{v}_i\boldsymbol{u}_i^{\top}\right), \tag{3}$$

where $\boldsymbol{P} = [\boldsymbol{u}_1\ \boldsymbol{v}_1\ \cdots\ \boldsymbol{u}_h\ \boldsymbol{v}_h] \in \mathbb{R}^{p\times 2h}$ holds the plane vectors, $\boldsymbol{J}$ is block diagonal with h copies of $\left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$, and the Cayley map keeps $\boldsymbol{Q}$ exactly orthogonal for arbitrary $\boldsymbol{P}$ (Helfrich et al., 2018; Lezcano-Casado & Martínez-Rubio, 2019). Whatever the vectors, a sum of h such generators can be rewritten as h generators on mutually orthogonal planes, so $\boldsymbol{Q}$ is always h independent plane rotations (Proposition 4). We use this form because its cost is simple to account for. A sum of h planes has rank at most $2h$, so the description is bounded by construction rather than by a penalty, and each plane carries three internal freedoms that need not be stored, so we charge $d(h) = 2hp-3h$ (Appendix B.1).

The native anchor. In physics, the structure a disentangler removes is always hidden in the basis the state is written in, and it is strong enough that the search finds it from any starting point. Many deep learning matrices instead carry structure in the basis they are stored in, so the starting point matters (Table 3). We start where that structure is, at no rotation, since by Eckart & Young (1936) the plain truncation of $\boldsymbol{W}$ is optimal in that cut and any structure $\boldsymbol{W}$ carries in its own coordinates is available there. We therefore return to the native cut before truncating and undo the rotation afterwards,

$$\hat{\boldsymbol{W}} \;=\; \underbrace{C^{-1}\,\boldsymbol{Q}^{\top}\,C}_{\text{undo the rotation}} \quad \underbrace{\mathcal{T}_k}_{\text{truncate, } lr\times bc} \quad \underbrace{C^{-1}\,\boldsymbol{Q}\,C}_{\text{rotate the crossed cut}} \quad \underbrace{\boldsymbol{W}}_{lr\times bc}. \tag{4}$$

The operation $C^{-1}\boldsymbol{Q}\,C$ allows the truncation to be performed in the native basis. At $\boldsymbol{Q} = \boldsymbol{I}$ it collapses and $\hat{\boldsymbol{W}} = \mathcal{T}_k\boldsymbol{W}$ is the plain SVD of $\boldsymbol{W}$. Stored packed, the truncation holds $k(m+n-k)$ free parameters (Appendix B.1), so the stored cost of the representation is $\kappa_{\mathrm{s}} = k(m+n-k)+d(h)$. Figure 2 draws the chain as a wire diagram.

3.2 EXECUTION

At inference the stored object is used only through products $y = \hat{\boldsymbol{W}}x$. Forming $\hat{\boldsymbol{W}}$ would write and then read all mn values, exactly the dense cost the representation exists to avoid, so it is never formed. Write $\boldsymbol{M}_k = \mathcal{T}_k\,C^{-1}\boldsymbol{Q}\,C\,\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$ for the stored truncation, so that the reconstruction of Section 3.1 reads $\hat{\boldsymbol{W}} = C^{-1}(\boldsymbol{Q}^{\top}C(\boldsymbol{M}_k))$. By the Woodbury identity (Woodbury, 1950) the Cayley map takes the form $\boldsymbol{Q}^{\top} = \boldsymbol{I} + \boldsymbol{P}\boldsymbol{G}_c\boldsymbol{P}^{\top}$ with $\boldsymbol{G}_c = -2\,(\boldsymbol{P}^{\top}\boldsymbol{P}+\boldsymbol{J})^{-1} \in \mathbb{R}^{2h\times 2h}$, derived in Appendix B.2, and since C is linear the reconstruction is the plain truncation plus a correction and the product splits accordingly,

$$y \;=\; \hat{\boldsymbol{W}}x \;=\; \underbrace{\boldsymbol{M}_k\,x}_{y_1} + \underbrace{\boldsymbol{\Delta}\,x}_{y_2}, \qquad \boldsymbol{\Delta} \;=\; C^{-1}\big(\boldsymbol{P}\boldsymbol{G}_c\boldsymbol{P}^{\top}C(\boldsymbol{M}_k)\big)\,. \tag{5}$$

The two terms share the input and nothing else, so they are computed separately and aggregated. The first term is the standard factored product (Denton et al., 2014), two matrix vector products that stream $\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$ and $\boldsymbol{U}$ once, $k(m+n)$ numbers, at the bandwidth roofline of the device. At $h = 0$ the second term vanishes, the rotation lane is skipped and the product is that of plain SVD.

The rotation lane. We never form the correction $\boldsymbol{\Delta}$, since it has rank $2h$ in the crossed cut but is in general of full rank in the native cut, where each of its terms is a Kronecker product. Instead of regrouping the matrix we regroup the input and the output between the two cuts. Beyond $\boldsymbol{M}_k$ the lane only loads the plane matrix $\boldsymbol{P}$, and in the crossed cut the correction factors as

$$C(\boldsymbol{\Delta}) \;=\; \widetilde{\boldsymbol{U}}\,\widetilde{\boldsymbol{W}}^{\top}, \qquad \widetilde{\boldsymbol{U}} = \boldsymbol{P}\boldsymbol{G}_c \in \mathbb{R}^{lc\times 2h}, \qquad \widetilde{\boldsymbol{W}} = C(\boldsymbol{M}_k)^{\top}\boldsymbol{P} \in \mathbb{R}^{rb\times 2h}, \tag{6}$$

of which only the two factors $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{W}}$ are kept. Each of its $2h$ rank one terms is a Kronecker product in the native cut (Van Loan & Pitsianis, 1993), so it acts on a vector through two small matrix products instead of one large one (Van Loan, 2000). With x regrouped as $c \times b$, $\tilde{\boldsymbol{u}}_j$ as $l \times c$ and $\widetilde{\boldsymbol{w}}_j$ as $r \times b$,

$$y_2 \;=\; \boldsymbol{\Delta}\, x \;=\; \sum_{j=1}^{2h} \operatorname{vec}\!\big(\tilde{\boldsymbol{u}}_j\, x\, \widetilde{\boldsymbol{w}}_j^{\top}\big)\,. \tag{7}$$

The lane therefore reads only the $2h(lc + rb)$ numbers of the two factors, few enough to stay in the on-chip cache from one product to the next while the identity lane streams its $k(m + n)$ numbers from device memory, and it never touches an object of size $m \times n$. The kernel reads the stored values of both lanes on every call, so the product cost of Section 3 is

$$\kappa_{\mathrm{p}} \;=\; k(m+n) \;+\; 2h(lc+rb). \tag{8}$$

3.3 Training

Training searches for the smallest rank that meets the budget, and keeps a rotation only if the rank it removes is worth more than the planes it costs. Tensor disentanglement fits $\boldsymbol{Q}$ to the normalized spectral tail of the crossed view of the rotated tensor beyond a rank k,

$$\mathcal{L}_k^{\mathrm{TD}}(\boldsymbol{Q}) \;=\; \frac{\sum_{i>k} \sigma_i^2(C(\boldsymbol{Q}\boldsymbol{W}))}{\|\boldsymbol{W}\|_F^2}, \tag{9}$$

the squared relative error of its best rank k approximation by Proposition 2 of Appendix B. The search has two loops and one constraint. Given a candidate rank k, the inner loop minimizes $\mathcal{L}_k^{\mathrm{TD}}$ over $\boldsymbol{Q}$, by Riemannian descent in the original, and the rank is feasible when the optimized tail satisfies $\mathcal{L}_k^{\mathrm{TD}}(\boldsymbol{Q}_k) \le \varepsilon$ for the budget ε, a check made on the singular values of the rotated tensor and not on any value the optimizer reports. For a fixed rotation the tail is nonincreasing in k, so feasibility is monotone in k up to optimizer noise. The outer loop bisects k to the smallest feasible rank, each probe being one inner fit at the midpoint. The budget is consulted only at that check and never inside the fit.

Finding a plane count. The procedure above is taken entirely from Wei et al. (2025). What we add is a loop over the plane count, since this parameter does not exist in the original formulation. For each $h = 0, 1, 2, \ldots$ the inherited bisection runs with $\boldsymbol{Q}$ built from h planes and returns the smallest rank $k^*(h)$ that meets the budget. Since BND truncates in the native cut, the tail is taken there, $\sum_{i>k} \sigma_i^2(C^{-1}(\boldsymbol{Q}\, C(\boldsymbol{W})))/\|\boldsymbol{W}\|_F^2$, and since $\boldsymbol{Q}$ is now built from planes, its $2h$ plane vectors are fitted jointly by L-BFGS from a small random start at which $\boldsymbol{Q}$ is close to the identity, so the fit finds which planes to rotate in as it finds the angles. The loop returns the best certified pair,

$$\mathcal{C} \;=\; \{(k^*(h), h) : h = 0, \ldots, h_{\max}\}, \qquad (k, h) \;=\; \arg\max_{(k,h)\in\mathcal{C}} \frac{mn}{\kappa_{\mathrm{s}}(k,h)}. \tag{10}$$

The pair at $h = 0$ is never removed, so the result is never worse than plain SVD in stored cost. Since further plane counts only add candidates, the search can run as long as compute allows, and the result only improves. We are aware that looping over candidate h is an inefficient procedure, chosen for exactness rather than efficiency, and Appendix C.3 states the cap and the patience rule that end it in practice.

4 Experimental Setup

The dataset. The dataset contains a total of 1337 matrices. 1037 of them are real deep learning matrices, transformer (256) and convolutional (37) weights, gradients (100), activations (75), key-value caches (368), and state space matrices (128), including 73 weights extracted from mixtures of experts at current frontier scale. We add 300 synthetic matrices that are either entirely incompressible, Gaussian noise (150), or carry known hidden structure, planted controls (150). Shapes range from 96×48 to 32768×8192. Appendix D lists the sources of every family, provenance and exclusions, and Appendix H collects the terms of the paper.

Table 1: Results per family at $\varepsilon = 0.01$, means over the matrices of each family. Net is the stored or product net, and % the share of matrices whose net is above one. The product net is reported as served, one wherever the load time probe runs the dense product. Inf. is the measured inference speedup, of a host to device copy in the storage block and of one product in the product block (Section 4). The twin block gives the mean excess of a matrix's gain over plain SVD above that of its spectrum twin, tests it with a Wilcoxon signed rank test under Holm correction across families, and gives the mean search time of the twin runs, which share shape and protocol with their sources. Appendix F.1 gives bootstrap 95% intervals.

	Storage			**Product**			**Twin**		
Family	**Net** ↑	**%** ↑	**Inf.** (×) ↑	**Net** ↑	**%** ↑	**Inf.** (×) ↑	**Excess** ↑	**Test**	**Time** (s) ↓
Compressible without learning									
State spaces	23.73	100	3.83	23.51	100	0.97	0	ns	266
Gradients	19.15	100	12.49	18.95	100	1.97	0	ns	1 018
Structure recovered by the learned rotation									
Planted controls	8.86	100	1.20	5.89	98	1.01	7.64	***	242
Caches	1.18	86	1.03	1.12	19	1.00	1.0e−2	**	135
Weights	1.14	94	1.01	1.12	1	1.01	5.3e−4	*	820
Activations	1.05	81	1.00	1.02	21	1.07	3.2e−3	ns	57
No structure beyond the spectrum									
Frontier weights	1.00	82	1.00	1.00	0	1.00	−7.0e−4	ns	17 331
Gaussian noise	1.00	44	1.00	1.00	0	1.00	1.6e−4	ns	313

Base setup. We run the search once on every matrix in our dataset under a frozen protocol, whose hyperparameters Appendix E lists for full reproducibility. Kernel latencies are measured on a single consumer device with paired interleaved trials and captured graphs for small products. We report the measured inference speedup (Inf.) as dense over compressed wall clock time, under the storage cost for a host to device copy of the packed form and under the product cost for one product with the form the load time probe serves. Both include the fixed overhead every copy and every kernel launch pays, and we deliberately keep it, so that the results show when gains saturate on real hardware rather than what the byte counts alone would promise. We score every result by the storage cost κ_{s} and the product cost κ_{p} defined in Section 3, and report means per family.

Error threshold. The primary budget is $\varepsilon = 0.01$, one budget for all families, selected to establish what gains are possible. It lies within the tolerance of most families, while language model weights call for a tighter budget of about 0.005 (Appendix G). We see a single budget as preferable to tuning ε per task, since it makes the gains of different families directly comparable. Appendices F.5 and G give an early estimate of how family-specific budgets shift the net compression achieved, which requires further investigation and is deliberately out of scope for this work.

Baselines. Raw storage and truncated SVD are the primary baselines, the latter being optimal in the fixed basis we initialize the search from by Eckart & Young (1936). Four further formats cover the classes beyond it, a tensor train and a Tucker decomposition on the same four index reshape, an interpolative CUR and a sparse plus low rank hybrid, described in Appendix F.3. Since the rotation of the original TD is dense and cannot pay for itself (Section 3.1), it is not added as a baseline.

Ablation on our method. We re-evaluate our method on a common subset of one hundred matrices, stratified over families. Each run replaces one of the adjustments we propose in Section 3 with the choice of the original TD setting or another alternative. The anchor group compares starting in the native basis against starting at a random rotation. The rotation family group compares the plane construction against Givens chains, butterfly factors (Dao et al., 2019) and Monarch-style products (Dao et al., 2022) at matched description charge. The protocol is kept identical throughout, and every run is scored as in the main run, by its stored net at the same budget, reported as the mean per family together with the search time.

Ablation on hidden structure. We run an additional experiment to probe the hypothesis of hidden structure in families of deep learning matrices. A subset of thirty matrices per family, and ten

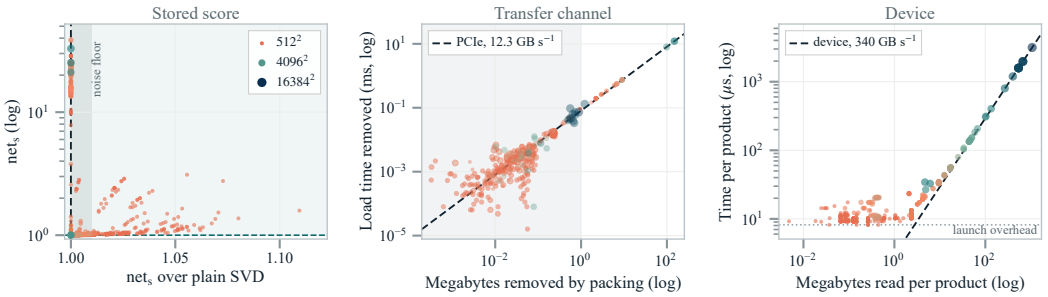

Figure 3: Left, every real matrix, its stored net against its ratio to plain SVD. The grey band is the gain the search also finds on pure noise. Middle, every benchmark pattern, megabytes removed by packing against load time removed, with the PCIe rate dashed. Right, megabytes read per product against time per product for the form the probe serves, with the device bandwidth dashed and the fixed launch overhead dotted. Color marks matrix size.

for frontier weights, is drawn evenly across each family's gain distribution. Each is paired with a *spectrum twin*, a matrix with identical singular values in a randomized basis, $\widetilde{\boldsymbol{W}} = \boldsymbol{Q}_1 \boldsymbol{\Sigma} \boldsymbol{Q}_2^\top$ with $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ drawn uniformly from the orthogonal group (Appendix D.3 gives the exact construction). Since the twin carries just the spectrum, a real matrix that gains more than its twin can only owe the excess to its basis, which is the hidden structure this paper is looking for. We report the Excess, the mean over matrices of the real gain over plain SVD minus the gain of its twin, and test it with a Wilcoxon signed rank test under Holm correction.

5 Results

Where matrices carry structure. Across all 1337 matrices the method never produced a representation that stores more than SVD or raw storage at the budget. Table 1 shows the outcome per family under both costs.

Gradients and state space matrices compress heavily on average with no rotation, $19\times$ and $24\times$ in storage and $19\times$ and $24\times$ in product net, by far the largest product nets among the real families. However, these gains are attributable to their native spectrum, not to the basis rotation, so plain SVD recovers them as well, and both match their spectrum twins exactly. Caches ($1.18\times$) and activations ($1.05\times$) gain modestly in storage, but the twin analysis shows that caches and weights hold structure that basis rotations can exploit. Among the caches where the rotation earns anything, a third exceed their twin by more than a factor of two, one by two orders of magnitude, and none falls below it ($p < 0.01$ after correction), for a mean excess of 0.010 against 0.00016 on noise, while activations point the same way without significance. Weights reach a mean net of 1.14 for storage and beat raw storage on 94% of matrices, and their mean excess over the twins of 0.00053 is significant, a smaller effect than in caches. The budget itself explains most gains near one. On Gaussian noise the method beats raw storage 44% of the time, which only the error budget makes possible, since a one percent budget lets the truncation drop the last directions of even a flat spectrum. The twins show that none of it comes from the basis.

Execution speed. Figure 3 and the two Inf. columns of Table 1 show the time saved during execution. Improving net storage yields measurable gains at load time, which tracks the packed count to three decimals, from a tenth of a percent on the largest frontier matrices to thirty three fold on a vocabulary gradient. Gradients load 12.5 times faster on average, state spaces 3.8 times, and the other real families load at most three percent faster.

Every copy pays a fixed overhead of about twenty microseconds, which reduces gains primarily on smaller matrices and explains the disparity between net costs and their impact on loading times. When correcting for this overhead, the load time removed by packing equals the bytes removed at the PCIe rate of 12.3 GB/s with $R^2 = 0.9999$ over 458 patterns (Figure 5 in Appendix F.7).

Table 2: Stored net per family at $\varepsilon = 0.01$, means over matrices, best in bold and second underlined. Every format looks for its own smallest representation that meets the budget, and scores one if it finds nothing below raw storage. We compare stored parameters only, since the product cost depends on an execution scheme the other formats do not come with.

Format	Reference	Grad	SSM	Planted	Act	Cache	Weight	Frontier	Noise
matrices		100	120	150	75	368	301	73	150
BND (ours)	this paper	19.15	23.73	**8.86**	**1.05**	**1.18**	<u>1.14</u>	1.00	1.00
Truncated SVD	Eckart & Young (1936)	19.15	23.73	<u>1.03</u>	<u>1.04</u>	<u>1.16</u>	1.13	1.00	1.00
Tensor train	Oseledets (2011)	<u>22.40</u>	**51.30**	1.02	1.03	1.12	1.13	1.00	1.00
Tucker	Kim et al. (2016)	5.18	22.20	1.00	1.01	1.02	1.00	1.00	1.00
Sparse + low rank	Candès et al. (2011)	**42.71**	<u>24.02</u>	<u>1.03</u>	<u>1.04</u>	<u>1.16</u>	**1.14**	1.00	1.00
CUR	Mahoney & Drineas (2009)	15.33	1.00	1.00	1.00	1.01	1.08	1.00	1.00

Above the fixed overhead, the factored kernel streams at 295 GB/s against 341 GB/s for the dense product it is compared to. That dense product is a strong reference in its own right. On the same matrices it matches or beats the vendor library cuBLAS at every size we measured except highly oblong ones, running 2.26 times faster at 1024^2, 1.16 times at 4096^2 and 1.01 times at 16384^2, and 2% slower at 768×50257.

Baseline comparisons. Against the other formats (Table 2), our method has the highest mean on planted controls, activations and caches, and is second on weights, where the sparse plus low rank hybrid is ahead by 0.3 percent. Frontier weights and noise separate no format. The planted controls separate a net of 8.86 from the 1.00 to 1.03 of every alternative at $p < 0.001$. On the two families that are low rank without any rotation other formats lead. A tensor train doubles our net on state spaces, winning on all 120 of them, and the hybrid averages 42.7 on gradients against our 19.2, carried by three matrices it compresses more than five hundred fold.

Components. Table 3 shows that only planes anchored in the native basis recover the planted controls, a mean net of 8.08 against at most 1.09 for every other rotation family. On the real families the alternatives come close, and Givens chains store slightly less than our method on activations and weights, 1.062 against 1.052 on activations, as do Monarch products on weights. Their advantage is as large on Gaussian noise, however, so it reflects the budget rather than structure (Appendix F). The foreign anchor falls below plain SVD on every real family, most on weights with 1.007 against 1.893, and the coarse plane grid gives up a quarter of the planted net at less than half the search time.

6 Discussion, Limitations and Future Work

Result recap. Our results confirm the expected conclusion that different matrix families show different latent structure a basis rotation can exploit. They suggest that caches and, more weakly, weights hold deeper structure that can be found by optimizing towards a different basis, while activations point the same way without significance. BND consistently finds representations more efficient than raw storage on all other matrices too, though these are either natively low-rank like state spaces and gradients, confirming what prior work on low-rank adapters (Hu et al., 2021), the low intrinsic dimension of fine-tuning (Aghajanyan et al., 2021), and low-rank gradients (Zhao et al., 2024) established, or expose surface level structure that can be exploited within the given error threshold. We cannot exclude that part of this picture reflects the limits of our optimizer rather than of the matrices, and we would welcome gains we could not find.

Separating storage and product cost. Above all, our results demonstrate that storage and product cost tell different stories about the effectiveness of a compression method. While improvements in storage cost can be achieved across all matrix families including Gaussian noise, improvements in product cost, which are arguably more important for matrices that are reused at every inference step, are more difficult to achieve. Both costs have easy fallbacks that can be switched interchangeably if necessary, which is why strict optimality on both is not necessary for a method to be useful. Keeping both in mind is preferable though, as it pinpoints where a method excels and where it does not.

Table 3: Components of our method, each swapped out alone on a common subset of 100 matrices, 20 planted controls, 15 activations, 18 caches, 32 weights and 15 noise matrices. Cells give the mean stored net per family as in Table 2, with plain SVD as reference, and Time the mean search time relative to the full method. The best value of a family column is bold and the second underlined. Noise carries no mark, since a higher net there shows how far a configuration exploits the budget, not structure. The weight column is lifted by two low rank vision convolutions at nets of 20.6 and 9.8. Table 9 holds the other swaps.

Configuration	**Planted** ↑	**Act** ↑	**Cache** ↑	**Weight** ↑	**Noise** ∘	**Time** ↓
Full method	**8.076**	<u>1.052</u>	**1.226**	1.899	1.005	1.0×
Plain SVD	1.025	1.031	1.203	1.893	1.001	—
Rotation family, at matched charge						
Dense with penalty	1.025	1.031	1.203	1.893	1.001	0.6×
Givens chain	1.062	**1.062**	<u>1.223</u>	**1.904**	1.017	3.9×
Butterfly	1.067	1.048	1.218	1.898	1.009	1.4×
Monarch	1.093	1.049	1.212	<u>1.903</u>	1.013	0.8×
Anchor and search						
Foreign anchor	1.013	1.028	1.119	1.007	1.005	1.2×
Coarse plane grid	<u>6.199</u>	1.031	1.203	1.893	1.001	0.4×

Tuning ε. Further specifying the error threshold ε for a given matrix family, or at an even smaller scope on a per-model basis, will reveal the true magnitudes at which compression can occur through spectral compression with basis optimization. Preliminary work on permissible error thresholds (Appendix G) shows that much larger budgets are not unlikely for many use cases, which can meaningfully shift net storage *and* product gains beyond what is presented here. The cache of Qwen2.5 3B keeps its closed book recall at $\varepsilon = 0.2$, where plain truncation alone stores 1.54 times fewer values, and gradients train unchanged at $\varepsilon = 0.3$ with 4.3 to 5.7 times fewer.

Caches hold structure. Since we find that caches hold genuine structure, tuning an algorithm like BND to work well with this family is a natural extension. BND is intended as a general-purpose matrix disentangler, so an application- and matrix-specific competitor will likely outperform our method by some margin on the same task. What is important to note is that caches, similar to gradients or activations, are typically fleeting structures, for which learning the basis must occur very quickly to remain useful. The current training method (to put it plainly) is almost hilariously inadequate for these use cases. We genuinely see this as feasible, and Appendix F shows some principled angles for how this can happen, yet fully testing this while maintaining the strict optimality guarantees this paper promises is beyond this paper's scope.

Combining approaches. Possibly the most promising avenue of future work we see is the synergy of basis-optimized spectral compression with compression methods that rely on rotations for their own purposes anyway (Chee et al., 2023; Ashkboos et al., 2024b; Liu et al., 2025). Using this method in isolation from other compression methods was an artificial constraint for the sake of understanding basis optimization for deep learning matrices more deeply.

7 Conclusion

We transferred basis optimization for spectral compression from physics to deep learning and evaluated it under an accounting that charges every stored parameter. The method never loses against SVD or raw storage in stored parameters, and where it wins the saved bytes convert to load time at a measured one to one rate. The structure a learned basis finds beyond the spectrum sits in key value caches and, to a lesser degree, in weights, while gradients and state spaces compress far more through their spectrum alone. We release the benchmark, the accounting and the method for others to build on.

Ethics Statement

This work involves no human subjects, collects no personal data, and uses only public models and synthetic matrices. We do not see a direct risk of harm from the method. One indirect risk deserves study. Spectral truncation discards the directions a matrix uses least, and behavior that serves rare inputs may live in exactly those directions, so compressing a network this way could degrade it more for underrepresented groups than its average error suggests. Compression has been shown to affect underrepresented data disproportionately (Hooker et al., 2019; 2020), but whether basis optimized spectral compression does so, and whether a learned basis makes it better or worse than plain truncation, is a hypothesis we have not tested and leave to future work.

Reproducibility Statement

All results derive from one frozen implementation with fixed hyperparameters, released together with the benchmark matrices, the null constructions, the result caches and the kernel. Appendix E lists every setting, and every number in the paper regenerates from the released caches by script.

AI Use Statement

We used a generative AI assistant throughout this work, and we describe its part in detail so that the authors' contribution is scoped clearly against it. The short version is that the ideation and the creative decisions are the authors', the core mechanism was conceived and first implemented by the authors, and the assistant served as an implementer, an auditor and a drafting aid under the authors' direction, with every output checked by the authors before it entered the paper.

Ideation and research direction. The question of whether a learned basis exposes structure in deep learning matrices, the transfer from tensor network disentanglement, the native anchor, the plane construction, the free parameter accounting and the choice of families and controls originate with the authors. The assistant took part in discussions of these ideas, raised objections and alternatives, and helped settle design decisions such as the twin study and the stopping rule, and the authors made every decision.

Method and code. The authors wrote the core mechanism, the rotation, its accounting and the certified search, from ideation through a first implementation. Starting from that origin, the assistant was used to set up the experiment scripts, the benchmark harness, the statistical tests, the figure and table generators and the execution kernel, and to find and fix bugs in the training and search pipelines. The authors reviewed, ran and verified all of this code.

Experiments and results. All experiments were designed by the authors and run on the authors' hardware. The assistant launched runs at the authors' request, monitored them, and aggregated their outputs from the result caches. It did not produce, alter or select any measurement. Which results to report was decided by the authors.

Synthetic data. The planted controls and the Gaussian noise of the benchmark are generated by scripts. The constructions were specified by the authors and the scripts were written with the assistant's help. No real matrix of the benchmark was produced or modified by generative AI.

Mathematical claims. The propositions of Appendix B and the derivations of the parameter count and the Woodbury form were drafted with the assistant's help, then audited, modified and finalized by the authors, and checked once more with the assistant's help. The authors take responsibility for their correctness.

Retrieval and discovery. Literature search was initiated by the assistant and audited, completed and reviewed by the authors, who read every cited work and wrote every bibliography entry. Suggestions of related work that the authors could not verify were discarded.

Writing. The manuscript was written by the authors. The assistant drafted passages from the authors' outlines and results, above all in the appendix, where it migrated results from logs and caches into tables and text, and the authors rewrote and finalized every section. The assistant also reviewed the text for errors, inconsistencies and unsupported claims, proposed structure and cuts, and polished wording, and the authors accepted or rejected each change.

Figures and tables. All figures and tables are generated by scripts from the result caches. The scripts were written with the assistant's help to the authors' specification and their outputs were checked by the authors against the caches.

Not used. We did not use generative AI to produce or alter any measurement, to decide which results to report, to write the bibliography, or for translation.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Saleh Ashkboos, Maximilian Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, volume 2024, pp. 11682–11701, 2024a.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024b.

Wouter Baert and NICK VANNIEUWENHOVEN. Atc: an advanced tucker compression library for multidimensional data. 2023.

Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36:4396–4429, 2023.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *International conference on machine learning*, pp. 1517–1527. PMLR, 2019.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022.

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 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.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998. doi: 10.1137/S0895479895290954.

Glen Evenbly and Guifré Vidal. Algorithms for entanglement renormalization. *Physical Review B—Condensed Matter and Materials Physics*, 79(14):144108, 2009.

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.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Joe Harris. *Algebraic geometry: a first course*. Springer Science & Business Media, 2013.

Markus Hauru, Maarten Van Damme, and Jutho Haegeman. Riemannian optimization of isometric tensor networks. *arXiv preprint arXiv:2007.03638*, 2020.

Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning (ICML)*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248*, 2019.

Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. Characterising bias in compressed models. *arXiv preprint arXiv:2010.03058*, 2020.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.

Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning (ICML)*, 2019.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

Jun Li, Li Fuxin, and Sinisa Todorovic. Efficient riemannian optimization on the stiefel manifold via the cayley transform. *arXiv preprint arXiv:2002.01113*, 2020.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025.

Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

Ivan V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286.

Charles F Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

Charles F Van Loan and Nikos Pitsianis. Approximation with kronecker products. In *Linear algebra for large scale and real-time applications*, pp. 293–314. Springer, 1993.

Guifre Vidal. Entanglement renormalization. *Physical review letters*, 99(22):220405, 2007.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations*, volume 2025, pp. 19299–19319, 2025.

Julia Wei, Alec Dektor, Chungen Shen, Zaiwen Wen, and Chao Yang. Numerical optimization for tensor disentanglement. *arXiv preprint arXiv:2508.19409*, 2025.

Max A Woodbury. *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

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

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

A Expectation Management

This section states what each of the three contributions listed in the introduction does and does not deliver. It is meant to be read before the results.

What the guarantee covers. The method never returns a representation with more parameters than plain SVD or raw storage at the stated error budget. The guarantee rests on two facts. The identity candidate is never removed from the search, and every candidate is certified on the exact singular values of its reconstruction. It therefore holds for any inner optimizer and any setting of the frozen hyperparameters. It says nothing about how far above the floor a matrix lands, and the benchmark shows that most weights land barely above it. The guarantee bounds the downside at zero. The size of the upside must be read from the tables.

What the task measures. The task is defined per matrix under a certified reconstruction error. The budget is an input. The paper does not say which budget a given model tolerates, since that depends on the layer, the task and the application. Choosing the budget belongs to the use case, not to the method. Downstream task quality is measured only in Appendix G, mostly for plain truncation and for the rotation on one small model. What the paper does show is the cost of any budget in stored parameters on a fixed set of matrices, so the trade can be made with numbers.

What kind of paper this is. The study is a census first and a comparison second. It asks which kinds of matrices admit learned spectral compression at all, and only then how well one method does on them. Table 2 gives a leaderboard. The finding we consider durable is in Table 1. The gains sort by the origin of a matrix, and the structure a learned basis can find lives most clearly in key value caches and, to a lesser degree, in weights. We expect better methods to beat the floors we set. We expect the map of where structure lives to hold.

What the method costs. The search runs once, before inference, and its compute is a one time cost. That cost is not small. Every probe of the search is a second order fit with an exact singular value decomposition at each step, and the frontier matrices in the benchmark take hours each on a consumer device. The method therefore suits matrices that are compressed once and served many times, and settings with idle compute. It does not suit anything that must happen at request time. Section 6 names a faster variant whose certification is unfinished.

What the method is not. The method reduces the number of parameters, not the bits per parameter. It therefore combines with quantization rather than competing with it. The relevant comparison is with other low rank formats, which Table 2 provides. The method is also not an end to end system. Wall clock products, load transfers and a selection that rules out losses are measured separately, and the space this saves goes to the component analysis of the method. A demonstration on a running language model would be welcome and is not in this paper.

B Formal Results

This section collects the statements that the main text uses without proof. They mark which parts of the method are settled by argument and which rest on measurement. In short, every deterministic step is optimal for its own cost. All approximation sits in one step, the search over rotations, and in the worst case that step loses nothing against plain SVD.

B.1 Counting Free Parameters

The stored cost κ_{s} of Section 3 charges a representation for the free parameters it holds, not for the entries of its arrays. This subsection derives the count for a truncated factorization, shows that the count can be achieved in transfer, and explains why the naive count fails. None of this is new. The dimension count is classical, and the packing is standard numerical practice, used for the same purpose by tensor compression libraries (Baert & VANNIEUWENHOVEN, 2023). We spell it out because every number in the paper rests on it.

The count. A rank k truncation $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}$ of an $m \times n$ matrix is stored as three arrays with $mk+k+nk$ entries. Not all of them are free. The columns of $\boldsymbol{U}$ are orthonormal, which imposes $k(k+1)/2$ constraints, and the same holds for $\boldsymbol{V}$, so the factors together carry

$$\left(mk - \tfrac{k(k+1)}{2}\right) + k + \left(nk - \tfrac{k(k+1)}{2}\right) \;=\; k\,(m+n-k) \tag{11}$$

free parameters. The two bracketed terms are the dimensions of the Stiefel manifolds of orthonormal k frames in $\mathbb{R}^m$ and $\mathbb{R}^n$ (Edelman et al., 1998). The total is the dimension of the variety of $m \times n$ matrices of rank at most k (Harris, 2013). The count is therefore a property of the object and not of the factorization chosen to store it. For distinct singular values the decomposition is unique up to k sign flips, a discrete ambiguity that costs no parameters.

Why the count is attainable. A count that could not be shipped would be meaningless, so the paper uses formats that move exactly $k(m+n-k)$ numbers. An orthonormal $m \times k$ factor is the first k columns of the product of k Householder reflections. A reflection with its leading entries fixed to the canonical form is described by the entries below its diagonal. The factor is therefore recoverable from $mk - k(k+1)/2$ numbers. The receiving device rebuilds the dense columns by applying the reflections, which costs compute but no bytes. The kernel of Section 3.2 prepares execution directly from this packed form, and the transfer measurements of Section 5 confirm that the bytes moved match the count to three decimals.

Consequences. Three facts follow that the main text uses throughout. First, at full rank the count equals mn. An untruncated factorization costs exactly as much as the matrix itself, so the identity candidate of the search is never charged for being a factorization. Second, a truncation to rank k saves $(m-k)(n-k)$ parameters, which is positive for every k below $\min(m,n)$. The plain SVD score is therefore at least one, and the floor of Proposition 3 holds without a raw storage fallback. Third, a matrix with a flat spectrum, Gaussian noise being the extreme case, needs full rank at any tight budget and therefore scores one. In other words, noise is incompressible.

Why the naive count fails. Counting the $k(m+n)$ array entries instead rewards a factorization for the k^2 numbers that orthonormality already fixes. Under that count a full rank factorization appears to cost $k(m+n) > mn$ for $k > mn/(m+n)$. Raw storage must then be kept as a fallback for every near full rank matrix, while a truncation slightly below full rank appears to save far more than it does. Section 5 reports the effect on the benchmark. Array counting declares most noise matrices compressible and gives weight matrices double digit advantages that the free parameter count reduces to one. The array count is still the right price for execution, because a product on a device reads the rebuilt dense factors. It is the product cost κ_{p} of Section 3.2.

The rotation. The learned rotation of Section 3.1 is charged by the same rule. Its h plane pairs are stored as the $2hp$ entries of $\boldsymbol{P} \in \mathbb{R}^{p\times 2h}$, and not all of them are free. Any pair $(\boldsymbol{u}_i, \boldsymbol{v}_i)$ can be replaced by $[\boldsymbol{u}_i\ \boldsymbol{v}_i]\,\boldsymbol{R}$ for a matrix $\boldsymbol{R} \in \mathbb{R}^{2\times 2}$ of unit determinant without changing the generator, since $\boldsymbol{R}\left(\begin{smallmatrix}0 & 1\\ -1 & 0\end{smallmatrix}\right)\boldsymbol{R}^{\top} = \det(\boldsymbol{R})\left(\begin{smallmatrix}0 & 1\\ -1 & 0\end{smallmatrix}\right)$. These matrices form a three dimensional group, so each plane carries three internal freedoms, and we charge $d(h) = 2hp - 3h$. The charge is read off the fitted generator, so a plane whose vectors the fit drives to zero, or into the span of another plane, lowers the rank of $\boldsymbol{A}$ and is not charged.

The exact count. For $h > 1$ the planes share further freedoms. The generator $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{J}\boldsymbol{P}^{\top}$ is unchanged under $\boldsymbol{P} \mapsto \boldsymbol{P}\boldsymbol{M}$ for every $\boldsymbol{M}$ with $\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\top} = \boldsymbol{J}$, the symplectic group of dimension $h(2h+1)$, so $\boldsymbol{A}$ has $2hp - h(2h+1)$ free parameters. Storage attains this count. A row permutation brings a basis of the span of $\boldsymbol{P}$ to the form $[\boldsymbol{I};\ \boldsymbol{B}]$ with $\boldsymbol{B} \in \mathbb{R}^{(p-2h)\times 2h}$, and $\boldsymbol{A}$ is fixed by $\boldsymbol{B}$ and its skew form on that span, $2h(p-2h) + h(2h-1) = 2hp - h(2h+1)$ numbers. Our charge differs from this count in two small ways. It deducts $3h$ rather than $h(2h+1)$ shared freedoms, and it deducts them for the requested rather than the retained planes. Charging every rotated census matrix the exact count of the planes it keeps changes its net by between -1.2 and $+3.8$ percent and turns no gain over plain SVD into a loss.

B.2 The Woodbury Form of the Cayley Map

The execution of Section 3.2 writes $\boldsymbol{Q}^{\top}$ as the identity plus a rank $2h$ correction. The coefficient matrix $\boldsymbol{G}_c$ of that correction follows from the Woodbury identity applied to the Cayley map. With $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{J}\boldsymbol{P}^{\top}$ and $\boldsymbol{J}^2 = -\boldsymbol{I}$, so that $\boldsymbol{J}^{-1} = -\boldsymbol{J}$, the identity $(\boldsymbol{I} + \boldsymbol{P}\boldsymbol{J}\boldsymbol{P}^{\top})^{-1} = \boldsymbol{I} - \boldsymbol{P}(\boldsymbol{J}^{-1} + \boldsymbol{P}^{\top}\boldsymbol{P})^{-1}\boldsymbol{P}^{\top}$ (Woodbury, 1950) gives

$$\boldsymbol{Q} \;=\; (\boldsymbol{I}+\boldsymbol{A})^{-1}(\boldsymbol{I}-\boldsymbol{A}) \;=\; 2(\boldsymbol{I}+\boldsymbol{A})^{-1} - \boldsymbol{I} \;=\; \boldsymbol{I} - 2\,\boldsymbol{P}\,(\boldsymbol{P}^{\top}\boldsymbol{P} - \boldsymbol{J})^{-1}\boldsymbol{P}^{\top}, \tag{12}$$

and transposing, with $(\boldsymbol{P}^{\top}\boldsymbol{P} - \boldsymbol{J})^{\top} = \boldsymbol{P}^{\top}\boldsymbol{P} + \boldsymbol{J}$ because $\boldsymbol{J}$ is skew,

$$\boldsymbol{Q}^{\top} \;=\; \boldsymbol{I} + \boldsymbol{P}\boldsymbol{G}_c\boldsymbol{P}^{\top}, \qquad \boldsymbol{G}_c \;=\; -2\,(\boldsymbol{P}^{\top}\boldsymbol{P} + \boldsymbol{J})^{-1} \in \mathbb{R}^{2h\times 2h}. \tag{13}$$

The only inverse is of a $2h \times 2h$ matrix, computed once at pack time. The product $\boldsymbol{P}\boldsymbol{G}_c$ is the left factor of the correction that the rotation lane keeps in memory. The same form makes Cayley parameterizations of large orthogonal matrices affordable in Riemannian optimization (Li et al., 2020).

B.3 Accounting and Guarantees

The guarantee of Section 3.3 rests on four short statements.

Proposition 1, exactness. The reshapes C and C^{-1} permute matrix entries. Permutations preserve the Frobenius norm, and so does multiplication by an orthogonal matrix. Consequently, for every rotation $\boldsymbol{Q}$ and every matrix $\boldsymbol{M}$ of the native shape,

$$\left\|\boldsymbol{W} - C^{-1}\big(\boldsymbol{Q}^{\top}C(\boldsymbol{M})\big)\right\|_F \;=\; \|\boldsymbol{W}_{\boldsymbol{Q}} - \boldsymbol{M}\|_F, \qquad \boldsymbol{W}_{\boldsymbol{Q}} = C^{-1}\big(\boldsymbol{Q}\,C(\boldsymbol{W})\big)\,. \tag{14}$$

Proof. Permuting entries leaves the sum of squares unchanged, and $\|\boldsymbol{Q}\boldsymbol{X}\|_F^2 = \mathrm{tr}(\boldsymbol{X}^{\top}\boldsymbol{Q}^{\top}\boldsymbol{Q}\boldsymbol{X}) = \|\boldsymbol{X}\|_F^2$. Apply C and then $\boldsymbol{Q}$ to the difference inside the norm. □

All reconstruction error therefore comes from the truncation, and the untruncated pipeline is lossless.

Proposition 2, conditional optimality. Fix the rotation. Among all rank k matrices stored in the native cut of the gauged matrix, the truncated SVD of $\boldsymbol{W_Q}$ minimizes the reconstruction error of $\hat{\boldsymbol{W}}$, and the minimum equals the spectral tail $(\sum_{i>k} \sigma_i^2(\boldsymbol{W_Q}))^{1/2}$. *Proof.* By Proposition 1 the reconstruction error equals $\|\boldsymbol{W_Q} - \boldsymbol{M}\|_F$ over rank k candidates $\boldsymbol{M}$, which Eckart & Young (1936) show is minimized by the truncated SVD, with the stated value. □

Every certified candidate is therefore error optimal at its own cost, given its rotation. The only step that can be improved is the choice of $\boldsymbol{Q}$.

Proposition 3, floor and anytime. The search always retains the identity candidate at the naive rank k_0, decides feasibility on exact singular values, and returns the certified candidate of maximal net. Hence the reported error satisfies its budget, the final stored score satisfies $\mathrm{net_s} \geq \mathrm{net_s^{SVD}} \geq 1$, and the result is nondecreasing in additional search time. *Proof.* The first claim is the definition of certification. The second follows since the identity candidate is never removed and selection is a maximum over a set containing it, while $mn - k(m+n-k) = (m-k)(n-k) \geq 0$ gives $\mathrm{net_s^{SVD}} \geq 1$ for every $k \leq \min(m,n)$. The third holds because added compute only adds candidates to the maximum. □

Under the free parameter accounting the raw fallback is never needed, since plain SVD stores at most mn free parameters at every rank.

Proposition 4, expressivity. Every real skew symmetric matrix of rank $2h$ is a sum of h plane generators, so the parameterization of Section 3.1 realizes exactly the generators of rank at most $2h$, and at $h = \lfloor p/2 \rfloor$ it realizes all of them. The Cayley map is a bijection between skew matrices and the rotations without eigenvalue -1, a dense subset of the rotation group. As h grows, the parameterization therefore ranges from plain SVD to the closure of the full one sided rotation group, and no rotation is excluded in the limit. *Proof sketch.* The real canonical form of a skew matrix is a direct sum of 2×2 rotation generators, each of which is $\boldsymbol{u}\boldsymbol{v}^\top - \boldsymbol{v}\boldsymbol{u}^\top$ for an orthogonal pair, giving the first claim. Bijectivity of the Cayley map onto its stated range is classical, and rotations with eigenvalue -1 lie in the closure of those without. □

We do not claim that the search over rotations reaches a global optimum. The landscape is not convex, as the entry barrier of Appendix C shows, and an approximate optimizer can miss gains. The four statements confine all approximation to that single step, and by Proposition 3 that step in the worst case loses nothing against plain SVD.

Remark, orbit invariants and the role of the anchor. Write $C(\boldsymbol{W}) = \boldsymbol{U}_c \boldsymbol{\Sigma}_c \boldsymbol{V}_c^\top$ in the crossed cut. Every matrix the method can reach is $\boldsymbol{Q}\, C(\boldsymbol{W}) = (\boldsymbol{Q}\boldsymbol{U}_c)\boldsymbol{\Sigma}_c \boldsymbol{V}_c^\top$, so the crossed singular values and the crossed right factor are the same under every rotation. The reachable matrices form an orbit of dimension at most $p(p-1)/2$ inside the mn dimensional space of matrices. A rotation can therefore change the native spectrum only within the limits these invariants set, and it can never turn one matrix into an arbitrary other. The loss landscape over the full rotation group is also fixed by the matrix alone, and the anchor changes neither its minima nor its optimum. What the anchor does determine is which rotations are reachable at a fixed plane count. With h planes the search reaches exactly the rotations whose generator has rank at most $2h$, which are the rotations close to the anchor in description length. The native anchor puts plain SVD among them for every h, and this is all the floor rests on. A foreign anchor removes it, and the anchor ablation of Table 3 measures the effect.

B.4 Separability and Absorption

Readers who know the quantization literature will ask the following question. Several methods rotate a matrix and pay nothing for the rotation, because the rotation is absorbed into a neighboring layer. Why can our rotation not be absorbed in the same way? The answer is that rotations that can be absorbed and rotations that reduce rank are disjoint categories. This subsection makes that precise.

Call a transformation of $\boldsymbol{W}$ *separable* when it acts as $\boldsymbol{W} \mapsto \boldsymbol{A}\boldsymbol{W}\boldsymbol{B}^\top$ for orthogonal $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, that is, when it mixes rows among rows and columns among columns without coupling the two.

Proposition 5, absorbable implies spectrum preserving. A separable transformation leaves every singular value of $\boldsymbol{W}$ unchanged, so it cannot reduce the rank needed at any error budget. *Proof.* If $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ then $\boldsymbol{A}\boldsymbol{W}\boldsymbol{B}^\top = (\boldsymbol{A}\boldsymbol{U})\boldsymbol{\Sigma}(\boldsymbol{B}\boldsymbol{V})^\top$, and $\boldsymbol{A}\boldsymbol{U}$ and $\boldsymbol{B}\boldsymbol{V}$ have orthonormal columns, so the right hand side is a singular value decomposition of the transformed matrix with the same $\boldsymbol{\Sigma}$. □

Proposition 6, the converse direction. A transformation that is absorbable into the adjacent linear maps of a network, or into the stored factors themselves, is separable. *Proof sketch.* Absorption into the adjacent maps means replacing the layer $y = \boldsymbol{W}x$ by $y = (\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{W}(\boldsymbol{B}^\top\boldsymbol{B})x$ and pushing $\boldsymbol{A}$ into the following layer and $\boldsymbol{B}$ into the preceding one, which requires the transformation to factor as a map on the output index times a map on the input index, which is separability. Absorption into the stored factors is the same statement with $\boldsymbol{A}$ folded into $\boldsymbol{U}$ and $\boldsymbol{B}$ into $\boldsymbol{V}$. □

Together the two statements split rotations into two kinds. A rotation that comes for free preserves the spectrum and therefore buys no rank. A rotation that changes the spectrum couples the row and column indices, cannot be pushed into either neighbor, and must be stored and charged. The crossed matricization $C(\boldsymbol{W})$ produces the second kind by construction. It groups part of the row index with part of the column index, so a rotation acting there is non separable. This is the formal basis of the statement in Section 3.1 that the crossed cut is the only place a rotation can help.

A practical corollary follows. Because our rotation is non separable, it combines with the separable rotations of quantization rather than competing with them. The two act on different index groupings and are charged differently. One is free and the other is paid.

C The Shape of the Search

The main text states the task in two lines. This section describes what the search actually optimizes over. That object is less smooth than the continuous language of the paper suggests, and the places where intuition fails explain several design decisions of the method. Nothing here is needed to follow the results. Every claim can be checked against the measurements in Figure 4.

C.1 Discrete and Continuous Variables

Four variables define the task. The error budget ε and the rotation $\boldsymbol{Q}$ are continuous. The truncation rank k and the plane count h are integers. The score is therefore a function on an integer grid, and the continuous variables enter it only through a feasibility decision. That decision asks whether the tail of the rotated spectrum at rank k fits under ε. Two consequences follow that are easy to miss if one thinks of compression as a smooth trade.

First, ranks and planes are priced differently. Under the packed count, removing the kth spectral direction saves

$$\kappa_{\mathrm{s}}(k) - \kappa_{\mathrm{s}}(k-1) \;=\; k(m+n-k) - (k-1)(m+n-k+1) \;=\; m+n-2k+1 \qquad (15)$$

parameters. This value falls as k grows. Directions near the naive floor are nearly worthless, and directions deep in the spectrum are worth almost $m+n$. A plane, by contrast, costs a fixed $2p-3$ regardless of what it buys. A plane therefore pays for itself only where it buys at least $(2p-3)/(m+n-2k+1)$ ranks at the current k. The numerator is fixed while the denominator shrinks toward the floor, so a plane that is worthless at k_0 can be worth a great deal a few ranks deeper. A profit requires reaching ranks well below k_0, which is why the search targets a candidate rank rather than the whole spectrum.

Second, the score moves in steps and can decrease. A plane that buys no rank still costs $2p-3$. The net saved as a function of h is therefore a staircase that rises where a plane unlocks ranks and falls where it does not. The right panel of Figure 4 shows this on all four of its matrices. Feasibility is also a threshold on a continuous quantity, so a fit that lands just above the budget loses a whole rank. The boundary is therefore sensitive to optimizer luck, and the boundary ablation of Appendix F puts this effect at up to three percent of net.

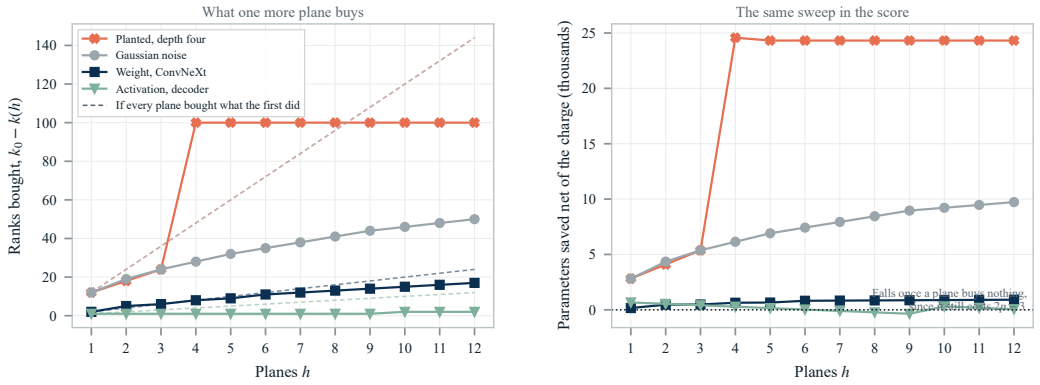

Figure 4: What one more plane buys, measured on one matrix from each regime under the frozen inner fit at $\varepsilon = 0.01$. Every plane count is fitted jointly with three restarts. Left, the ranks the best h plane rotation buys, against the dashed line that additivity would predict from the first plane alone. The planted control, built with four planes, buys twelve, eighteen and twenty four ranks at one to three planes and then one hundred at four. This is the entry barrier followed by a collapse in rank. Noise buys less with every plane. The activation saturates at once. Right, the same sweep in the score, parameters saved net of the plane charge. The score is a staircase and falls wherever a plane buys nothing, so the search must select the best certified point rather than the last one.

C.2 A Simple Model and Why It Fails

Suppose each principal direction beyond the kept rank had a price, the number of planes needed to rotate it out of the tail. Suppose also that the directions were independent, so that buying one changed nothing about the others. Sorting the directions by price would then give the whole trajectory of the search. The ranks bought by h planes, $g(h) = k_0 - k(h)$, would be the number of directions priced at h or less, which can only grow with h. In the simplest version, where each plane retires one direction, it would be the straight line $g(h) = h$. The dashed lines in the left panel of Figure 4 show the closest version of this model for each matrix, the line through what its first plane actually bought.

The measured curves break this model in both directions, and the following statement shows that no version of it can hold.

Proposition 7, plane purchases are not additive. Let $g(h)$ be the ranks bought by the best rotation on h planes, with $g(0) = 0$. For any matrix on which some rotation buys a rank, g is not additive in h. *Proof.* Additivity with $g(0) = 0$ forces $g(h) = h \cdot g(1)$, a line through the origin. By Proposition 4 every rotation is reachable at $h = \lfloor p/2 \rfloor$, so g is bounded above by $k_0 - k_{\min}$ and attains that bound at a finite h. A line through the origin that is bounded is identically zero, which contradicts $g(1) \geq 1$ or $g(h^*) \geq 1$ for the h^* at which a rank is first bought. □

The proof says only that the curve must bend. The data show which way, and the answer depends on the regime. On the planted control the first three planes buy 12, 18 and 24 ranks and the fourth buys 100, where additivity would give 48. The purchase is superadditive at the planted depth. This is the entry barrier. The generator of the planted rotation has rank eight, and by Proposition 4 no rotation on fewer than four planes can represent it. Nothing guarantees partial credit below the true depth, and the measurement shows that the partial credit that exists is small. On noise the twelfth plane has bought 50 ranks where the rate of the first plane would give 144, more than the matrix has. The purchase is subadditive, with diminishing returns. On the activation nothing beyond the first rank is bought at any h.

No additive model can work, for a structural reason. Planes combine through the Cayley map, $\boldsymbol{Q} = \mathrm{Cayley}(\boldsymbol{A}_1 + \boldsymbol{A}_2)$, which is not the product $\mathrm{Cayley}(\boldsymbol{A}_1)\,\mathrm{Cayley}(\boldsymbol{A}_2)$ unless the generators commute. The singular values of $C^{-1}(\boldsymbol{Q}\,C(\boldsymbol{W}))$ are also not linear in $\boldsymbol{Q}$. A set of planes therefore buys what it buys as a set. The same fact appears in the component analysis as the failure of greedy plane scheduling. It is also direct evidence that the landscape is not convex, since partial progress

toward the planted optimum is unavailable below the barrier. Similar difficulty is documented for disentangling optimization in tensor networks (Hauru et al., 2020). The method responds in three ways. Planes are fitted jointly. The outer search keeps every certified candidate and returns the best, rather than stopping at the first plane that fails to pay. The patience rule waits for a second miss before giving up, since the plane that pays can follow several that did not.

C.3 When the Loop over Planes Stops

The loop over the plane count in Section 3.1 can run as long as compute allows and can only improve the result, since each further plane count adds a candidate to a maximum and removes none. Two things end it in practice. A hard cap needs no fit, and a patience rule reads the fits as they arrive.

The cap. Write $\mathrm{net}_{\mathrm{s}}^{*}(h) = mn/\big(k^{*}(h)(m+n-k^{*}(h)) + d(h)\big)$ for the certified score at plane count h. Even a rotation that drove the rank to one would score at most $mn/\big((m+n-1)+d(h)\big)$, so no plane count with

$$d(h) \;\geq\; k_0\,(m+n-k_0) \;-\; (m+n-1) \tag{16}$$

can beat the plain SVD candidate at $h = 0$, and the loop never needs to go there. With $d(h) = h(2p-3)$ this is a bound on h that costs nothing to evaluate. The frozen protocol adds a fixed ceiling $h_{\max} = 8$, which Appendix E lists and which the planted controls never reach.

The patience rule. Within the cap the loop stops based on the fits, not on exhaustion. Let $B(h) = \max_{h' \leq h} \mathrm{net}_{\mathrm{s}}^{*}(h')$ be the best certified score up to h. Say that a gain has appeared at h once $B(h) > 1.02\,\mathrm{net}_{\mathrm{s}}^{*}(0)$, and call plane count h a miss when $\mathrm{net}_{\mathrm{s}}^{*}(h) \leq B(h-1)$. The loop stops at the first h at which a gain has appeared and both $h-1$ and h are misses. It returns the pair that attains $B(h)$. Two features are deliberate. First, misses are not counted before a gain has appeared. Proposition 7 and Figure 4 show that the plane that pays can follow several that do not, which is the entry barrier. A loop that stopped at the first miss would never reach that plane. Second, the two percent margin in the definition of a gain keeps sculpting on noise, which earns about one percent at small shapes, from triggering the rule. On a matrix with no structure the loop therefore runs to the cap and returns plain SVD. In short, the loop continues until the planes start to pay, then stops once two plane counts in a row fail to improve on the best.

What this is not. The rule is a stopping heuristic, not a proof of optimality. A gain can in principle hide beyond two misses, and the coarse grid ablation of Section 5 measures the cost of searching less. The rule is also the simplest loop that keeps the accounting exact, since every candidate it compares is certified on exact singular values. We chose it for that reason, not for speed.

D The Benchmark

D.1 Why a Benchmark at All

A single matrix cannot show whether learned spectral compression is worthwhile, because the answer varies by more than an order of magnitude across the kinds of matrices deep learning produces. The benchmark measures this variation instead of assuming it. It also lets a later method be compared with ours on identical inputs under identical accounting. We collected matrices of every kind we could extract from public models, and added two families of synthetic controls whose correct answers are known in advance.

D.2 Composition and Provenance

Table 4 lists the benchmark. All real matrices come from publicly available checkpoints on the Hugging Face Hub, extracted with the scripts released with the paper. The released manifest lists every matrix with its repository, revision and SHA-256 checksum. The frontier extracts come from `zai-org/GLM-5.3`, `zai-org/GLM-5.3-Flash`, `deepseek-ai/DeepSeek-V4-Flash-0731`, `deepseek-ai/DeepSeek-V4-Pro-0813` and `Qwen/Qwen3.8-2.4T-A95B`. Weights are taken directly from the checkpoints. Activations and caches are recorded by running each

Table 4: The benchmark. Counts are matrices, not models. Shapes give the smallest and largest matrix in each group. The three synthetic groups at the bottom have known correct answers and bound the method from both sides.

Group	Source	Count	Shapes
Real matrices, 1037 in total			
Weights, text encoders	10 encoders, BERT to MPNet	80	256^2 to 3072×768
Weights, decoders	12 decoders, GPT-2 to Qwen2.5	96	192×576 to 50304×512
Weights, vision	6 transformers, 5 convolutional	85	96×48 to 3072×768
Weights, audio	HuBERT, wav2vec2, Whisper	24	384^2 to 3072×768
Weights, other	CLIP, Mamba	16	1536×48 to 3072×768
Caches, Pythia	12 layers, key and value	288	495×64
Caches, Llama	5 layers, key and value	80	4096×128
Activations	5 models, several layers	75	26×768 to 32×768
Gradients	5 models, two prompts	100	512^2 to 50257×768
State spaces	Mamba-130m decay banks	120	256×512
Frontier mixtures	GLM-5.3, DeepSeek-V4, Qwen3.8	73	128×8192 to 32768×8192
Controls, 300 in total			
Planted, shallow	known rotation, depth 1 and 2	76	128^2 and 256^2
Planted, deep	known rotation, depth 4 and 8	74	128^2 and 256^2
Gaussian noise	independent normal entries	150	128^2 to 1024×256

model once on a fixed short prompt and saving the intermediate tensors. Gradients come from one backward pass per prompt on two prompts in five models, taken in training mode, so their exact values depend on the dropout state of that run. The frontier sample is extracted shard by shard from mixture of experts checkpoints too large to hold in memory at once.

D.3 The Three Controls

Each control answers a different question. Together they bound the method from both sides.

Planted matrices, the false negative floor. A planted matrix is built backwards. We start from a low rank matrix of known rank r, view it in the crossed cut, and apply a random rotation of known depth D. The rotation scrambles the matrix so that it appears close to full rank in its native basis. The correct answer is therefore known before any search runs, and a method that cannot recover rank r at depth D is too weak. We use $r \in \{4, 8, 16\}$ and $D \in \{1, 2, 4, 8\}$ at sizes 128^2 and 256^2, so recovery can be read as a function of how deeply the structure is hidden.

Gaussian noise, the false positive floor. A matrix of independent normal entries contains nothing to find. Any gain reported on it is an artifact of the accounting, not a discovery. Noise matrices are included at several shapes because the gain from sculpting a noise spectrum shrinks as the matrix grows, roughly one percent at 128^2 and a tenth of that at 1024^2. A claimed gain must therefore clear this floor at its own shape.

Spectrum matched twins, the decisive control. The two floors above cannot separate the two reasons a real matrix might compress. A cache and a weight matrix may both gain one percent, one because its basis holds structure and the other because its spectrum happens to decay. The twin separates these cases. Given a real matrix $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ we draw fresh random orthogonal $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ and build $\widetilde{\boldsymbol{W}} = \boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_2^\top$, which has the exact singular values of the source and nothing else. The two bases are Haar distributed, that is uniform over the orthogonal group. They are the orthonormal factors of the reduced QR decompositions of two Gaussian matrices of sizes $m \times r$ and $n \times r$ with $r = \min(m, n)$. This gives the same distribution as slicing a full random rotation and is affordable on the frontier shapes. The twin is searched under the same frozen protocol as its source, and its net and its net over plain SVD are recorded next to the source's. Any gain the real matrix achieves beyond its own twin can only come from the basis. We pair thirty matrices of every family with a twin, and ten for the frontier weights, since a twin of a frontier matrix costs hours on the device. The matrices are sampled evenly across each family's gain distribution rather than from its best

Table 5: The spectrum twin control. Every matrix is paired with a twin that has its exact singular values in a random orthogonal basis, and matrices are drawn evenly across each family's gain distribution rather than from its best cases. Active counts matrices where the rotation gains more than half a percent, and the next two columns count those whose gain exceeds their twin's by the stated factor. Below counts matrices whose gain falls under their twin's, which a family without basis structure should also show. The test is a Wilcoxon signed rank test on the paired excess, with Holm correction across families.

Family	**Pairs**	**Active**	$> \mathbf{2}\times$	$> \mathbf{5}\times$	**Below**	**Mean gain** real / twin	**Test**
State spaces	30	0	0	0	0	0.0000 / 0.0000	ns
Gradients	30	0	0	0	0	0.0000 / 0.0000	ns
Planted controls	30	30	29	29	0	7.6598 / 0.0195	***
Activations	30	17	3	2	5	0.0151 / 0.0119	ns
Caches	30	21	7	5	0	0.0197 / 0.0096	**
Weights	30	23	0	0	0	0.0082 / 0.0077	*
Frontier weights	10	0	0	0	4	0.0002 / 0.0009	ns
Gaussian noise	30	12	0	0	3	0.0041 / 0.0039	ns

cases. This matters because an earlier spot check that paired only the highest gain matrix per family reported margins two to five times larger than the family actually shows. Three families have an outcome known by construction and serve as a check on the instrument. The random basis destroys a planted rotation, so a planted twin must sit at the noise floor while its source does not. Gradients and state spaces leave nothing for a rotation to earn on either side, so source and twin must agree at zero. Table 1 carries the compact block, and Table 5 reports all eight families in full, with the stricter factor of five and the mean gains of source and twin, under one Holm correction.

D.4 Exclusions

Four details decide which matrices appear and how they were run. Matrices with a dimension below sixteen are skipped as degenerate. Matrices whose larger dimension exceeds eight hundred are routed to a separate pass, since the search cost grows quickly with the rotation space. The frontier sample is handled by that pass throughout. Eleven frontier matrices with a smallest dimension of 8192 exceeded the capacity of the device solver. They were screened with a host side singular value decomposition only, which fixes their rank but not their rotation. They appear in the census as SVD results and are marked as such in the released cache. A further thirty expert matrices were extracted for the cross expert study of Section 5 but not kept on disk, so they are excluded from every count in this paper.

E The Frozen Protocol

E.1 Why Freeze

Every number in this paper comes from one implementation with one set of inner hyperparameters. These were chosen before the census ran and not adjusted afterwards. The method searches over a discrete pair of rank and plane count, and a tuned search would make the reported floor a property of the tuning rather than of the method. Freezing may leave gain on the table. We did not measure how much, but the component analysis shows that search choices matter, a coarser plane grid reaching a stored net about seven percent lower on the matrices where the method gains. In return, no result was selected after the fact.

E.2 Settings

Table 6 lists every setting required to reproduce a single matrix result.

Table 6: The frozen protocol. These values produced every result in the paper and were not varied per matrix or per family.

Stage	Setting	Value
Task	error measure	relative Frobenius, $\|\boldsymbol{W} - \hat{\boldsymbol{W}}\|_F / \|\boldsymbol{W}\|_F$
	primary budget	$\varepsilon = 0.01$, swept to 0.10 in Appendix F
	accounting	$k(m + n - k) + \max(0,\ 2hp - 3h)$
Reshape	index rule	largest divisor at most $\sqrt{d}$, applied to m and to n
	rotation space	$p = lc$, acting on the crossed cut only
Inner fit	optimizer	L-BFGS, strong Wolfe line search
	iterations	250 maximum per fit
	tolerances	gradient 10^{-12}, change 10^{-16}
	initialization	plane vectors from $0.05 \cdot \mathcal{N}(0, 1)$
	selection within a fit	lowest loss seen, not the last iterate
Outer search	plane counts	$h = 0, 1, \ldots, 8$, with $h = 0$ the identity
	rank search	binary search for the smallest feasible k in $[1, k_0]$
	early stop	two consecutive misses, counted only after a real gain
	charge read off	skew rank of the learned generator times p, less $3h$
	skew rank tolerance	singular values above 0.01 of the leading one
Decisions	feasibility	exact singular values of the reconstructed matrix
	final selection	highest net among certified candidates

E.3 Reproducing One Result

To reproduce a single number, follow these steps. Load the matrix and compute k_0, the smallest rank whose truncation error meets the budget. This is the SVD baseline and the first candidate. For each plane count from one to eight, fit the plane vectors with the settings above at a candidate rank, find the smallest feasible rank by binary search, read the charge off the learned generator, and record the net. Stop early once two consecutive plane counts fail to improve on the best net seen, provided at least one real gain has already appeared. Return the candidate of highest net. The identity candidate is never discarded, so the returned net is at least the SVD net by construction.

Two details are easy to overlook. The feasibility decision uses the exact singular values of the reconstructed matrix, not the optimizer's own loss. An optimizer that converges poorly therefore reports a worse result, never an unsupported one. The charge is read off the generator that was actually learned, not off the requested plane count, so a fit that effectively uses fewer planes is charged less.

F Extended Results

F.1 Intervals for the Means of Table 1

Table 7 gives percentile bootstrap intervals for every mean of Table 1. The intervals treat the matrices of a family as the sample, so they describe how much each value would move under a different draw of matrices from the same sources, not the variation of a single run, since the search is deterministic under the frozen protocol. Shares and test verdicts carry no interval. The twin excess of caches and weights stays above zero across the interval, that of activations and noise touches zero, and that of frontier weights stays below it.

F.2 The Design Space

Nearly every element of the method has a reasonable alternative. Table 8 lists them, grouped into decisions that define the task and decisions that define our implementation of it. Evidence for each component on a common matrix set is in Section 5 and in the rest of this appendix. One entry is easy to get wrong. An objective that treats all ranks at once spreads effort over the whole spectrum and misses tight budgets. Targeting the candidate rank recovers the needed precision.

Table 7: Bootstrap 95% intervals for Table 1, resampling the matrices of each family 2000 times. All values are means, the twin excess over the twinned pairs.

Family	Storage net	Product net	Storage Inf.	Product Inf.	Twin excess	Twin time (s)
State spaces	23.73 [23.34, 24.13]	23.51 [23.09, 23.93]	3.83 [3.76, 3.88]	0.97 [0.97, 0.98]	0 [0, 0]	266 [256, 279]
Gradients	19.15 [18.15, 20.18]	18.95 [17.96, 20.05]	12.49 [11.62, 13.33]	1.97 [1.58, 2.43]	0 [0, 0]	1018 [942, 1080]
Planted controls	8.86 [7.95, 9.83]	5.89 [5.27, 6.53]	1.20 [1.16, 1.24]	1.01 [0.99, 1.02]	7.64 [5.70, 9.68]	242 [210, 274]
Caches	1.18 [1.14, 1.22]	1.12 [1.09, 1.16]	1.03 [1.02, 1.03]	1.00 [1.00, 1.00]	1.0e−2 [3.5e−3, 1.8e−2]	135 [98, 180]
Weights	1.14 [1.02, 1.31]	1.12 [1.01, 1.29]	1.01 [1.00, 1.03]	1.01 [1.00, 1.01]	5.3e−4 [1.4e−4, 1.1e−3]	820 [627, 1024]
Activations	1.05 [1.04, 1.07]	1.02 [1.01, 1.03]	1.00 [1.00, 1.01]	1.07 [1.04, 1.10]	3.2e−3 [−1.4e−4, 7.4e−3]	57 [46, 66]
Frontier weights	1.00 [1.00, 1.00]	1.00 [1.00, 1.00]	1.00 [1.00, 1.00]	1.00 [1.00, 1.00]	−7.0e−4 [−1.5e−3, −1.4e−4]	17331 [9566, 26366]
Gaussian noise	1.00 [1.00, 1.00]	1.00 [1.00, 1.00]	1.00 [0.99, 1.00]	1.00 [1.00, 1.01]	1.6e−4 [−1.2e−5, 3.7e−4]	313 [239, 399]

Table 8: The design space, grouped by layer. The middle columns give our choice and where its evidence is. The last column names what a different choice would require or offer.

Axis	Our Choice	Evidence	Open Alternatives
Benchmark and task			
Error measure	relative Frobenius	§3	spectral norm, activation aware
Error budget	0.01 primary	App. E	looser budgets, swept
Accounting	free parameters	§3	bits, entropy coding
Null design	floors and spectrum control	§4	task specific nulls
Method, representation			
Anchor	native basis	Tab. 3	arbitrary factorizations
Reshape rule	near square indices	App. E	searched or learned indices
Rotation family	explicit planes	Tab. 3	learned coordinate pairs, deeper products
Parameterization	Cayley map	App. F	exponential, Householder
Sidedness	one sided	Tab. 9	two sided for symmetric structure
Stored precision	full, eight bit tested	App. F	four bit destroys structure
Method, optimization and search			
Inner optimizer	second order	Tab. 9	faster first order variants
Objective	targeted tail	Tab. 9	rank agnostic, penalized
Certification	exact spectra	§3.1	proxy verdicts, uncertified
Outer search	nested binary	Tab. 9	parallel multisection, priors
Boundary policy	single fit	App. F	repeated fits with voting
Execution			
Factor layout	contiguous reads	App. F	
Thin rank path	split, then dispatch	App. F	fused single pass

Several gaps remain. Activation aware error needs calibration data and breaks the single matrix scope, so we leave it out on purpose, not for lack of merit. The rotation families we compare in Section 5 all use fixed coordinate structure. A family that learns where to rotate, and not only how much, remains open. The parallel search is designed but not shipped. Each gap is a place where the benchmark can be beaten.

F.3 The Baseline Formats

The four formats of Table 2 use our accounting. They are data free, certified against the exact reconstruction, and charged in free parameters. A tensor train and a Tucker decomposition on the same four index reshape cover the multilinear formats, and for each we take the better of the native and crossed index orders. An interpolative CUR stores actual rows and columns without any orthonormality discount. A sparse plus low rank hybrid stores a rank k factorization plus the largest residual entries at two numbers each. Every baseline searches for its own smallest certified representation at the same budget. A format that would rather spend error than parameters must still meet the budget first. Appendix F.5 sweeps all formats across looser budgets.

Table 9: The component analysis in full. Each row swaps one component of the frozen method, and checkmarks mark the kept components. Cells give the mean stored net per family as in Table 2, and Time the mean search time relative to the frozen method. The best value of a family column is bold and the second underlined, and noise carries no mark.

	Kept From Frozen											
Configuration	Anchor	Planes	Loss	Optimizer	One side	Search	**Planted ↑**	**Act ↑**	**Cache ↑**	**Weight ↑**	**Noise ○**	**Time ↓**
Reference												
Full method	✓	✓	✓	✓	✓	✓	<u>8.076</u>	1.052	**1.226**	1.899	1.005	1.0×
Plain SVD	✓		no rotation				1.025	1.031	1.203	1.893	1.001	—
Rotation family, at matched charge												
Dense with penalty	✓	✗	✓	✓	✓	✓	1.025	1.031	1.203	1.893	1.001	0.6×
Givens chain	✓	✗	✓	✓	✓	✓	1.062	**1.062**	1.223	**1.904**	1.017	3.9×
Butterfly	✓	✗	✓	✓	✓	✓	1.067	1.048	1.218	1.898	1.009	1.4×
Monarch	✓	✗	✓	✓	✓	✓	1.093	1.049	1.212	<u>1.903</u>	1.013	0.8×
Anchor												
Foreign anchor	✗	✓	✓	✓	✓	✓	1.013	1.028	1.119	1.007	1.005	1.2×
Objective												
All ranks at once	✓	✓	✗	✓	✓	✓	6.175	1.031	1.203	1.893	1.001	0.2×
Cost penalty added	✓	✓	✗	✓	✓	✓	5.420	1.039	1.216	1.896	1.003	1.4×
Tracked subspace	✓	✓	✗	✓	✓	✓	5.040	1.049	1.211	1.899	1.005	0.4×
Optimization												
First order	✓	✓	✓	✗	✓	✓	**8.097**	<u>1.053</u>	1.224	1.899	1.005	2.1×
First order, then polish	✓	✓	✓	✗	✓	✓	7.989	1.051	<u>1.225</u>	1.899	1.005	1.4×
Sidedness and search												
Two sided	✓	✓	✓	✓	✗	✓	6.116	1.046	1.215	1.902	1.007	1.4×
Coarse grid	✓	✓	✓	✓	✓	✗	6.199	1.031	1.203	1.893	1.001	0.4×

F.4 Component Analysis in Full

Table 9 lists all fourteen configurations over one hundred matrices. Table 3 in the main text shows the rotation family, anchor and search rows. This subsection gives the composition of that subset and a per family breakdown of the two results that are easiest to misread.

The subset was stratified to cover the range of behaviors, not to represent the benchmark proportionally. A proportional sample would be dominated by caches and would contain too few planted controls to test recovery. It holds 20 planted controls, split evenly between shallow and deep, 18 Pythia caches, 15 activations, 15 noise matrices, and 32 weight matrices from audio, decoder, encoder, state space and vision sources. Sizes are capped so that fourteen configurations on one hundred matrices fit in a day of compute. The full study took 13.9 hours and produced 1400 certified runs with no failures after five retries of a transient solver error.

The first result that is easy to misread is the Givens chain. It stores the least of all configurations on activations and weights and beats the frozen method on nearly two thirds of all matrices, which would be odd for a component we rejected. The per family breakdown explains it. Givens chains win on 15 of 15 noise matrices, 28 of 32 weights, 12 of 18 caches and 9 of 15 activations, and on none of the 20 planted controls. These are the families where most of the gain is sculpting, and where the landscape is a single flat basin. There, many cheap angles sculpt a smooth spectrum slightly better than a few expensive planes. The margins are around one percent and the cost is more than three times that of the frozen method. Where structure exists the same construction recovers nothing. Rotation families therefore trade sculpting against discovery, and a method meant to find structure should be judged on discovery.

The second is the first order optimizer. It keeps the full gain on average but recovers only 15 of 20 planted controls, against 20 for the frozen method. Its wins are scattered, 2 of 15 activations, 2

Table 10: Mean net compression ↑ as the error budget is loosened, on the census matrices of four families. Scoring follows the main table, so a format that certifies nothing below raw storage scores one. Beyond the primary budget a few matrices that one format compresses by orders of magnitude dominate the mean, so the ordering changes with the budget.

Family	ε ∘	**SVD** ↑	**TT** ↑	**Tucker** ↑	**S+LR** ↑	**CUR** ↑
Gradients	0.01	19.15	22.40	5.18	42.71	15.33
$n = 100$	0.02	20.54	23.77	5.72	52.60	15.34
	0.05	25.66	27.61	7.73	98.17	16.73
	0.10	37.75	37.87	19.59	740.61	23.57
Caches	0.01	1.16	1.12	1.02	1.16	1.01
$n = 368$	0.02	1.39	1.25	1.07	1.38	1.06
	0.05	2.82	3.13	4.14	3.09	1.16
	0.10	8.45	24.97	29.33	8.80	2.37
Activations	0.01	1.04	1.03	1.01	1.04	1.00
$n = 75$	0.02	1.14	1.08	1.06	1.14	1.00
	0.05	3.40	6.63	7.98	164.74	2.74
	0.10	6.81	43.28	43.25	1563.62	5.61
Weights	0.01	1.13	1.13	1.00	1.14	1.08
$n = 301$	0.02	1.18	1.17	1.00	1.17	1.12
	0.05	1.23	1.21	1.01	1.19	1.12
	0.10	1.31	1.31	1.02	1.23	1.13

of 18 caches, 3 of 32 weights and 4 of 20 planted. This is the pattern of a search that sometimes lands in a different local optimum, not of one that is systematically better or worse. Together with the landscape measurements below, this is the clearest evidence that precision pays exactly where structure is hidden.

F.5 Baselines Across Error Budgets

The main text compares against four other formats at the primary budget. A reviewer may ask whether the ordering depends on that budget, since a format that spends error differently could overtake at a looser one. Table 10 sweeps all formats from one to ten percent error on the four families where anything is to be found. The convention of the main table applies, so a format that certifies nothing below raw storage scores one.

Two observations follow. The ordering depends on the budget. The sparse plus low rank hybrid leads on gradients at every budget and on activations from $\varepsilon = 0.05$, where a few matrices reach several hundred fold, and the Tucker decomposition leads on caches from the same budget. On weights plain SVD and the tensor train stay level throughout. The gains on weights grow slowly, plain SVD reaching 1.31 at a tenfold budget, while caches and activations grow far faster once the budget passes five percent. This is a property of those spectra, not of any method.

F.6 The Search Landscape

The component analysis shows that some swaps matter only in some families. To see why, we measured the geometry of the inner optimization directly, on a stratified subset of eighteen matrices, each at its own census operating point.

We use two measures, both standard in the literature on flat and sharp minima (Hochreiter & Schmidhuber, 1997; Keskar et al., 2016; Li et al., 2018). Eight independent restarts from fresh initializations give the spread of the certified final errors. This shows how many distinct optima the landscape has and how reliably the good one is found. Loss profiles along random directions of the plane parameter space, scaled relative to the parameter norm, give the basin width. We define it as the relative perturbation radius at which the loss doubles. Our width inverts the usual sharpness. It is a radius for a fixed loss increase rather than an increase at a fixed radius.

Table 11: The search landscape by family, means over the probe subset. Restart spread is the relative range of certified errors across eight restarts, so a small value means one basin. Basin width is the relative parameter perturbation that doubles the loss, so a small value means a sharp optimum. The two regimes differ by at least a factor of four in restart spread and by more than an order of magnitude in basin width.

Family	**Restart spread** ↓	**Basin width** ↑	**Reading** ○
Planted, deep	0.94	0.003	many sharp optima
Planted, shallow	0.99	0.005	many sharp optima
Caches	0.06	0.40	one broad basin
Activations	0.24	0.70	one flat basin, one rough outlier
Weights	0.06	0.17	one broad basin
Gaussian noise	0.07	0.10	one broad basin

Table 12: Cost of removing one kernel feature, as a ratio to the time of the full path ↑. A value above one means the feature helps. Paired interleaved trials on a single consumer device, medians of several hundred pairs. The last column runs both streaming stages through the vendor library instead of our kernels.

n ○	k ○	**No float4** ↑	**Shared memory** ↑	**Unfolded** Σ ↑	**Vendor library** ↑
8192	225	1.03	1.45	1.01	1.01
8192	1474	1.01	1.03	1.01	0.99
8192	4055	1.01	1.01	1.00	0.98
16384	451	1.00	1.26	1.01	0.88
16384	2949	1.00	1.01	1.00	0.99
16384	8110	1.01	1.00	1.00	0.99

Table 11 splits the families into two regimes. Where the gain comes from sculpting, any optimizer from any initialization reaches the same broad flat basin. This is why the optimizer swaps of Table 9 tie there and why a cheaper search would suffice. Where basis structure is hidden, the landscape has many sharp optima. Restarts differ by more than an order of magnitude in final error, and moving the parameters by a fraction of a percent doubles the loss. Restarts and second order precision therefore pay exactly where structure exists.

Two caveats apply to these numbers. Random directions underestimate sharpness compared with a worst case ascent direction. The plane parameterization also has internal freedoms along which the loss is exactly flat, which inflates every measured width. Both biases work against the sharpness reading, and the gap between regimes is far larger than either can explain. One real activation in the probe shows the planted pattern rather than that of its own family. This suggests that landscape roughness could serve as a cheap detector of basis structure, which we have not pursued.

F.7 The Execution Layer

Section 3.2 claims that saved parameters become saved time. That claim holds only if the kernel reads only the bytes the accounting charges. This subsection reports what each part of the kernel contributes and where the claim stops holding.

Which features pay. Each feature of the shipped kernel was disabled in turn, and the product was timed against the full path in paired, interleaved trials on square matrices at three ranks. Table 12 reports the cost of removing each one. Warp shuffle reductions are the only feature that matters substantially, and only where reductions are frequent. The vectorized loads and the folding of the singular values into the stored factor are each worth a few percent. An earlier version also shipped a split reduction meant to fill the device at thin ranks. It measured as a net loss at every shape and was removed. The regime it targeted is the one where the work goes to the vendor library anyway.

Where the byte law stops. Two regimes fall outside the law, and selection handles both. At small sizes every execution form sits on a fixed launch overhead of tens of microseconds, so byte savings buy nothing. When the product is captured as a single graph at load, which is ordinary

Table 13: Both channels by matrix size over every distinct benchmark pattern, before selection, so including every pattern whose compressed product reads more values than the dense matrix. Each channel gives the mean number of values the compressed form moves relative to dense, the values a product reads, $k(m+n)+2h(lc+rb)$, and the values a copy ships, κ_{s}. The time columns give the mean time of the compressed form relative to dense, for the product on the device in graph mode and for the host to device copy, and the share columns count patterns where the compressed form was faster. Small products sit on the fixed launch overhead, and larger ones read more values than dense on average and are slower for it. The load time selection serves the dense product wherever this happens, while the copy follows the bytes at every size.

Entries mn	**Patterns**	**Product**			**Copy**		
		values ↓	time ↓	faster	values ↓	time ↓	faster
under 10^5	244	0.834	1.601	12%	0.681	0.942	70%
10^5 to 10^6	138	1.198	2.115	14%	0.826	0.846	69%
10^6 to 10^7	101	1.036	1.671	25%	0.761	0.767	76%
10^7 to 10^8	20	1.164	1.225	15%	0.855	0.856	45%
above 10^8	12	1.440	1.472	0%	0.999	0.999	50%

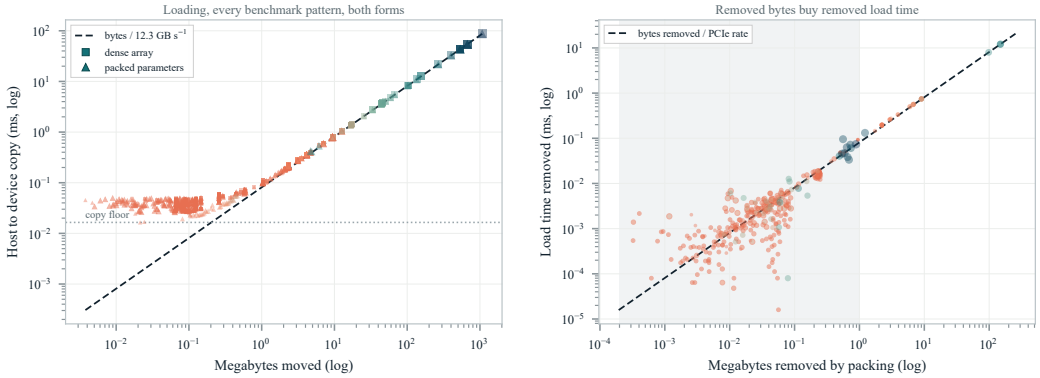

Figure 5: The byte law on the transfer channel, medians of five independent sweeps of host to device copies from pinned memory. Left, every benchmark pattern in both forms lies on one PCIe line above a fixed copy overhead. Right, the same data as differences, the bytes a pattern removes against the load time it removes, which cancels the overhead. The shaded region marks savings near the few microsecond resolution of the instrument. Patterns with zero bytes to remove or unresolved savings are omitted. No measured pattern loses time, and even the smallest gains are realized, with the labeled giants loading a tenth of a percent faster.

serving practice, the overhead lifts and measured gains appear already at census sizes. The rotation lane is governed by a cache side cost rather than by the byte count. A matrix whose rotation saves parameters but only a marginal number of bytes therefore runs more slowly in its compressed form. At a byte ratio of 0.99 the plain factored product runs at 0.96 of dense while the rotation corrected product runs at 1.04, which is a loss. The load time selection compares the available forms once and runs the cheapest, so the loss is never executed. This is a property of the selection and not of the representation, whose product cost stays above the dense one.

The two channels by size. Table 13 bins every distinct benchmark pattern by matrix size and reports the time of the compressed form relative to dense on both channels, before selection. The product column loses in every bin. The loss is largest for small matrices, where the fixed launch overhead adds to the factor byte count, and for the giants above 10^8 entries, where the fixed launch overhead is gone and the factor count alone explains a 47 percent loss. The copy column gains in every bin where the instrument can resolve a difference. Figure 5 shows the copy channel in full, with both forms of every pattern on one PCIe line and the removed bytes against the removed load time.

Table 14: Where the time goes, for square matrices at the byte ratio of the gradients, medians of paired trials in microseconds. Dense and factored products are each timed as plain launches and as one captured graph. Submission is the plain minus the graph time of the dense product, the host side cost that capture removes. Streaming is the dense bytes over the bandwidth measured at the largest size, and the fixed overhead is the rest of the graph time, the device side cost no compression can reduce. The last column, factored over dense graph time, approaches the byte ratio of 0.055 as the fixed costs become small.

		Dense		Factored		Dense time split			
Side	**k**	**Plain**	**Graph**	**Plain**	**Graph**	**Submit**	**Static**	**Stream**	**Ratio ↓**
512	14	23.6	12.9	34.3	14.0	10.7	9.8	3.1	1.085
1024	28	37.3	26.1	47.2	20.0	11.2	13.7	12.4	0.765
2048	56	60.9	60.3	19.5	18.4	0.6	10.7	49.6	0.305
4096	113	201.7	201.7	33.3	31.7	0.0	3.1	198.6	0.157
8192	225	791.6	791.6	71.7	69.6	0.0	0.0	794.4	0.088
16384	451	3178.2	3177.5	209.4	207.9	0.8	0.0	3177.5	0.065

Table 15: Measured latency gain at the census sizes under three execution modes, one paired measurement per matrix, with the faster form selected so that no gain falls below one. Each block gives the share of matrices gaining more than five percent and the mean gain. Plain launches leave almost everything on the fixed launch overhead. Capturing the product as one graph at load removes the host side submission cost, and gains appear, most clearly for gradients. Pipelining 64 products spreads submission over both forms and brings the overhead back, which is why we compare graphs rather than subtract launch cost from one side only.

		Plain launches		One graph		Pipelined	
Family	**Count**	> 1.05	**Mean**	> 1.05	**Mean**	> 1.05	**Mean**
State spaces	120	0%	1.00	0%	1.00	0%	1.00
Gradients	100	42%	1.82	81%	1.90	39%	1.25
Planted controls	150	0%	1.00	5%	1.01	0%	1.00
Activations	75	0%	1.00	13%	1.05	0%	1.00
Caches	368	1%	1.00	2%	1.00	0%	1.00
Weights	301	2%	1.00	4%	1.01	0%	1.00
Frontier weights	73	3%	1.00	0%	1.00	0%	1.00
Gaussian noise	150	0%	1.00	11%	1.02	0%	1.00
All	1337	4%	1.06	10%	1.08	3%	1.02

Where the time goes. Table 14 splits the measured time of a product into the part no compression can reduce and the part that scales with bytes, at six square sizes and the byte ratio of the gradient class. The submission cost that graph capture removes is about eleven microseconds at small sizes and disappears once a product is long enough to hide it. The fixed overhead on the device is about ten microseconds and is gone by the largest sizes. The streaming part spans three orders of magnitude. The last column shows the result. At 512 the factored product loses, since its two launches cost more than the bytes it saves. At 1024 it gains a quarter. From 4096 upward the time ratio tracks the byte ratio to within the static remainder, reaching 0.065 against a byte ratio of 0.055 at 16384. The remaining gap there is the two launches of the factored product against the one of the dense product.

Execution modes at census sizes. Table 15 reports the census matrices at their own sizes under three execution modes, which shows the fixed launch overhead and its removal. With plain launches almost every matrix sits on the overhead, and gains above five percent appear on four percent of the benchmark. Capturing the product as one graph lifts the overhead, and 81 percent of the gradient family then gains more than five percent, at a mean gain of 1.90. The other families stay at the overhead because their products are too short for a byte saving to register. Pipelining 64 products amortizes submission for both forms and restores the overhead, and the gradient share falls back to 39 percent. The fair comparison therefore uses graphs rather than arithmetic that subtracts launch cost from one side only.

F.8 Ablations Outside the Single Swap Format

Four further questions do not fit the one component at a time format of Table 9. They are reported here.

Stored precision of the rotation. All reported results store the plane vectors at full precision. We measured how far that precision can be reduced. The rotation survives eight bit integers at our budget and collapses at four. The collapse is useful. A rotation that survived four bits would not have been carrying much information, so the precision ladder is a crude second test of whether a learned rotation is real.

Plane scheduling. Fitting planes one at a time and freezing each before adding the next is much cheaper than fitting them jointly, and it fails. Plane savings do not add up, because the rank a set of planes unlocks is a property of the set, not the sum of what each plane unlocks alone. This is the same non convexity that the entry barrier of Appendix C shows, seen from the optimization side. Proposition 7 there is the formal statement.

Boundary robustness. The binary search over the rank makes a feasibility decision at each step, and a single unlucky fit near the boundary carries through to the result. Repeating the fit at the boundary and taking the best changes the final net by up to three percent on matrices where the rotation is active. We did not adopt repeated fits, since the frozen protocol fixes one fit per decision. We report the size of the effect so that a later method can decide differently.

Fast objective variants. The targeted objective needs a singular value computation per step, which dominates the cost. A variant that tracks the leading subspace instead runs roughly three times faster and reaches 82 percent of the frozen net on average. Its verdicts are no longer certified by construction, so adopting it would require a separate certification pass. This is the most promising direction for making the search cheap enough to run during inference rather than before it.

G Downstream Effect of the Error Budget

The task certifies a per matrix reconstruction error and says nothing about what a budget costs a whole model. This section measures that cost on weights, caches and a vision backbone, so the budget of the main text can be read against the quality a deployment would see. Every number can be regenerated from the released caches by script.

Setup. All models are public checkpoints from the Hugging Face Hub, `Qwen/Qwen2.5-0.5B`, `-1.5B` and `-3B`, `Qwen/Qwen3-1.7B` and `-4B`, `TinyLlama/TinyLlama_v1.1`, `HuggingFaceTB/SmolLM2-1.7B`, `distilgpt2` and `EleutherAI/pythia-70m` to `-2.8b`. For weights, every block matrix of a model is replaced by its certified truncation at one uniform budget ε, which is the $h = 0$ candidate of the search. Embeddings, heads and norms are left untouched, and the whole model is evaluated. We measure perplexity on WikiText-2 over 64 windows of 1024 tokens, and the mean KL per token and the top one agreement of the compressed model with the uncompressed one over 8192 tokens. We score three ranked tasks, LAMBADA, HellaSwag and ARC-Easy, by log likelihood. We decode two open ended tasks greedily, GSM8K with eight shot chain of thought and closed book TriviaQA. Twenty language models from 19M to 3.6B parameters take part, including one model at four points of its training. Models above one billion parameters run in bfloat16. For caches, the eight shot prompt is prefilled, the keys and values of every head are replaced by their certified truncation at ε, and the answer is generated from the compressed cache with the weights untouched. For vision, every convolution and linear weight of ConvNeXt tiny and DeiT small is truncated, and top one accuracy is measured on the Imagenette validation set.

Weights. Table 16 gives the closed form measurements and Table 17 the tasks. At $\varepsilon = 0.01$ most models disagree with their uncompressed version on a fifth of all next tokens. The effect on a task depends on what the task asks for. Ranking four fixed continuations costs one to two points. One exact token costs five to seven. Eighty correct tokens of arithmetic cost five to twenty two,

and GSM8K on Qwen3 4B falls from 78 to 56 percent. For every model, the budget at which it still agrees with itself on 95 percent of tokens lies between 0.002 and 0.01. The stored net of the truncated weights at that budget is 1.000 to 1.001. There is therefore no operating point at which plain truncation of language model weights is both harmless and worthwhile in size. Two further facts stand out. The admissible budget does not depend on model size, with a correlation of 0.04 between log parameters and log ε_{95} across seventeen models. It also shrinks with training. Across the four checkpoints of Pythia 410m it tightens six fold from step 10k to the end of training, at constant architecture and data. At $\varepsilon = 0.01$ the greedy continuation of the compressed model departs from that of the uncompressed model after six to nine tokens, while solutions keep their length. The model writes fluent text that differs almost everywhere and is wrong where a task depends on the exact path. Table 19 repeats the comparison of the main text on a whole model. On distilgpt2, the learned rotation of every block matrix gives lower perplexity and lower KL than plain SVD at the same certified budget, on both budgets tested, while storing about as many parameters, 0.2 to 0.4 percent fewer. At equal size and equal certified error, the rotated model stays closer to the original.

Caches. Table 18 shows the opposite regime. At the budget where GSM8K has lost ten points, the weights of a model give up 0.1 to 1.2 percent of their parameters, while its cache gives up 2.2 to 9.3 percent. That is an order of magnitude more at the same cost in quality. Recall of stored facts is almost unaffected. TriviaQA on Qwen2.5 3B moves by 1.5 points at $\varepsilon = 0.2$, where the cache is 35 percent smaller. The spread between models is large. The cache of Qwen2.5 tolerates twice the removal of Pythia and four times that of Qwen3 before TriviaQA loses a tenth of its own accuracy. The spread does not follow the number of query heads that share a cached head, so we do not attribute it to attention layout. Vision backbones sit between the two. ConvNeXt loses 0.6 points of top one at $\varepsilon = 0.05$ and 1.2 at $\varepsilon = 0.1$, and DeiT loses 1.0 and 2.3, at stored nets of 1.02 to 1.08.

Gradients. For a gradient, the downstream use is the optimizer step, so the test is training. We fine tune distilgpt2 and Pythia 160m on WikiText-2 for 400 steps at batch 8 by 512 tokens. Every block weight gradient is replaced by its certified truncation at ε before the AdamW step, with no error feedback and the same data order and seed for every budget. Up to $\varepsilon = 0.3$ the final validation loss changes by at most 0.005 nats while the gradients store 4.3 to 5.7 times fewer parameters. At $\varepsilon = 0.5$ it changes by 0.022 nats at 12 to 25 fold. Gradients therefore tolerate a budget thirty times the one the census uses,

Activations. The residual stream entering every block, truncated per sequence, is the one family where the budget itself fails. From the eighth block of Qwen2.5 0.5B on, the hidden state of the first token has a norm 115 times that of the median token. A relative Frobenius budget of one percent therefore reproduces that row to four decimals and leaves the other tokens at a median error of seven percent on an 86 token sequence. The ranked tasks collapse, while perplexity, measured on long windows, is unchanged. A budget for activations has to exempt or reweight this attention sink. The activation family of the census is certified under the same budget and carries the same caveat.

What this means for the budget. The budget a deployment can afford depends on the matrix family and the task, not on the method. One percent is at the edge of harmless for language model weights and well inside it for caches and vision weights. The compression a budget buys differs by a further order of magnitude between those families. A single budget therefore does not represent the savings an implementation would see, and the per family scores of Table 1 should be read with each family's own admissible budget in mind. A looser budget also gives a learned basis more room to pay for itself, since the charge of a rotation is fixed while the rank it can remove grows with the budget. The census at one percent is therefore a lower bound on what the method finds for every family that tolerates more.

Table 16: Weights of every model, all block matrices truncated at one budget. Params counts the truncated weights, and ε_{95} is the budget at which the compressed model still picks the same next token as the uncompressed one 95 percent of the time. The other columns are measured at $\varepsilon = 0.01$, the KL per token against the uncompressed model, the top one agreement, the mean accuracy change over LAMBADA, HellaSwag and ARC-Easy with 500 examples each, and the stored net. The bottom block shows one model at four points in training.

Model	**Params**	**PPL**	$\boldsymbol{\varepsilon_{95}}$	**KL**	**Agree**	**Tasks**	$\mathbf{net_s}$
pythia-70m	19M	52.1	0.0020	0.282	0.70	-3.1	1.004
distilgpt2	42M	47.0	0.0065	0.014	0.92	+0.9	1.004
pythia-160m	85M	31.5	0.0020	0.155	0.77	+0.4	1.003
pythia-410m	302M	18.8	0.0025	0.189	0.78	-8.2	1.002
Qwen2.5-0.5B	358M	14.9	0.0054	0.214	0.76	+0.1	1.002
pythia-1b	805M	15.4	0.0033	0.144	0.79	-0.1	1.002
TinyLlama 1.1B	969M	9.4	0.0106	0.004	0.97	+0.1	1.003
OLMo-2-0425-1B	1074M	10.2	0.0026	0.201	0.78	-10.2	1.003
pythia-1.4b	1208M	13.8	0.0021	0.127	0.83	-1.4	1.002
Qwen2.5-1.5B	1310M	10.5	0.0058	0.083	0.86	+0.1	1.002
Qwen3-1.7B	1409M	18.8	0.0037	1.582	0.57	-25.1	1.003
SmolLM2-1.7B	1611M	9.2	0.0024	0.488	0.69	-10.5	1.003
Falcon3-3B-Base	2422M	9.1	0.0020	0.092	0.85	-2.6	1.002
pythia-2.8b	2517M	11.9	0.0020	0.163	0.80	-5.9	1.001
phi-2	2517M	11.2	0.0027	0.032	0.91	+0.2	1.004
Qwen2.5-3B	2775M	9.0	0.0055	0.071	0.88	-3.2	1.010
Qwen3-4B	3633M	15.4	0.0041	0.234	0.80	-3.9	1.002
Pythia 410m across training, steps of 2M tokens							
step 10k	302M	27.0	0.0144	0.001	0.98	+0.0	1.001
step 36k	302M	22.0	0.0072	0.024	0.91	+0.0	1.001
step 72k	302M	19.8	0.0034	0.051	0.87	-0.9	1.002
step 143k	302M	18.8	0.0025	0.189	0.78	-8.2	1.002

Table 17: Weights truncated at one budget, change in accuracy points against the uncompressed model. GSM8K uses eight shot chain of thought and exact match on the final number over 200 questions, TriviaQA closed book exact match over 400 questions, and Ranked the mean of LAMBADA, HellaSwag and ARC-Easy over 2000 examples each where measured. Drift is the number of greedy tokens before the output first differs from the uncompressed model's continuation of a WikiText prompt. TinyLlama solves only three percent of GSM8K, so its GSM8K column says little.

Model	ε	$\mathbf{net_s}$	GSM8K	TriviaQA	Ranked	Drift
Qwen2.5-1.5B, GSM8K 0.62, TriviaQA 0.35 at full precision						
	0.002	1.000	-1.0	+0.3		62
	0.005	1.001	+0.5	+0.5		37
	0.01	1.002	-8.0	-1.0		8
	0.02	1.006	-28.0	-6.7		3
	0.05	1.021	-61.0	-28.0		1
	0.1	1.055	-60.5	-34.2		0
Qwen2.5-3B, GSM8K 0.76, TriviaQA 0.41 at full precision						
	0.002	1.006	-0.5	-0.7	-0.2	60
	0.005	1.008	-1.0	-0.7	-0.5	35
	0.01	1.010	-5.5	-2.5	-2.8	9
	0.02	1.015	-17.5	-7.2	-10.0	3
	0.05	1.032	-64.5	-12.2		2
	0.1	1.070	-73.5	-26.2		1
Qwen3-4B, GSM8K 0.78, TriviaQA 0.36 at full precision						
	0.002	1.000	+2.5	-0.8	-0.3	40
	0.005	1.001	-2.5	-3.2	-0.7	19
	0.01	1.002	-22.0	-7.0	-3.7	6
	0.02	1.005	-47.5	-15.7	-9.6	3
	0.05	1.021	-67.0	-26.0		1
	0.1	1.058	-77.5	-31.2		1
TinyLlama 1.1B, GSM8K 0.03, TriviaQA 0.33 at full precision						
	0.002	1.000	+0.5	+0.3		71
	0.005	1.001	-0.5	+0.0	+0.1	85
	0.01	1.003	+0.0	-0.8	-0.3	45
	0.02	1.008	+0.0	-8.0	-6.5	7
	0.05	1.026	-2.0	-14.3		2
	0.1	1.066	-0.5	-29.5		1

Table 18: Key value caches truncated per head at one budget during the benchmark, weights untouched. We prefill the eight shot prompt of 1349 tokens, truncate the keys and values of every head, and generate the answer from the truncated cache. Cache net is the stored net of the cache, the task columns are accuracy changes in points, and share is the number of query heads that read each cached head. Pythia solves under two percent of GSM8K, so only its TriviaQA column is informative.

Model	ε	Cache net	GSM8K	TriviaQA	Drift
Qwen2.5-1.5B, share 6, GSM8K 0.62, TriviaQA 0.35 at full precision					
	0.01	1.017	-1.0	+0.0	68
	0.05	1.085	-11.5	+0.3	23
	0.1	1.203	-10.5	+0.5	15
	0.2	1.571	-17.0	-1.7	11
	0.3	2.179	-37.5	-12.7	6
Qwen2.5-3B, share 8, GSM8K 0.73, TriviaQA 0.41 at full precision					
	0.01	1.007	+3.0	-0.5	31
	0.05	1.070	-8.0	+0.2	16
	0.1	1.186	-15.0	-1.3	10
	0.2	1.538	-25.0	-1.5	8
	0.3	2.116	-52.0	-11.8	5
Qwen3-4B, share 4, GSM8K 0.71, TriviaQA 0.35 at full precision					
	0.01	1.013	+2.0	-1.2	47
	0.05	1.073	-8.5	-2.5	16
	0.1	1.186	-12.5	-5.2	11
	0.2	1.544	-54.0	-9.0	5
	0.3	2.142	-69.5	-18.0	1
pythia-1.4b, share 1, GSM8K 0.01, TriviaQA 0.19 at full precision					
	0.01	1.058	-1.5	-0.3	37
	0.05	1.287	+0.0	-1.7	6
	0.1	1.507	+0.0	-4.0	3
	0.2	2.005	-1.0	-8.8	2
	0.3	2.737	-1.5	-13.0	1

Table 19: Learned rotation against plain SVD on all block matrices of distilgpt2, both certified at the same budget. The rotation comes from the search of Section 3.3 with at most two planes per matrix. Net is the stored net of the weights, and the other columns are measured on WikiText-2. The uncompressed model has a perplexity of 47.0.

ε	Form	$\mathbf{net_s}$	PPL	KL	Agree	Rotated
0.05	plain SVD	1.026	90.4	0.685	0.554	0/24
0.05	rotation	1.028	81.5	0.594	0.576	24/24
0.1	plain SVD	1.071	223.1	1.564	0.372	0/24
0.1	rotation	1.075	174.3	1.341	0.400	23/24

H GLOSSARY

Every term below is used in one sense throughout the paper.

Term	Meaning
The task	
Budget ε	The largest relative Frobenius error a representation may have, $\|\boldsymbol{W} - \hat{\boldsymbol{W}}\|_F / \|\boldsymbol{W}\|_F$. One percent unless stated.
Certified	Checked on the exact singular values of the reconstruction, never on the value an optimizer reports.
Stored cost κ_{s}, stored net $\mathrm{net}_{\mathrm{s}}$	The free parameters a representation holds, and mn/κ_{s}. What a checkpoint keeps and a bus moves.
Product cost κ_{p}, product net $\mathrm{net}_{\mathrm{p}}$	The stored values a product with the representation reads, and mn/κ_{p}. What a memory bound device pays per call.
Share (%)	In Table 1, the share of matrices whose stored, respectively product, net exceeds one, beating raw storage or the dense product.
Inf.	Measured speedup of dense over compressed time, for a copy under storage and a product under product cost.
Plain SVD	Truncated SVD of the matrix in the basis it is stored in, the optimum of Eckart & Young (1936), and the candidate of the search without a rotation.
The benchmark	
Family	A kind of matrix by what it represents, weights, frontier weights, gradients, activations, key value caches and state spaces, plus two control families.
Planted control	A matrix built with low rank structure hidden behind a known rotation of a few planes. Recovered when the search reaches within five percent of its planted net.
Gaussian noise	The control family without any structure, whose net is the noise floor.
Noise floor	The gain over plain SVD that the search extracts from Gaussian noise of the same shape, the level below which a gain is no evidence of structure.
Spectrum twin	A matrix with the same singular values as a real one in a uniformly random basis, so that a gain beyond the twin's comes from the basis alone.
Census	The run of the frozen method on every benchmark matrix, from which every table of the main text reads.
The method	
Indices l, r, b, c	The factors of $m = lr$ and $n = bc$ that turn a matrix into a tensor with four indices.
Native cut, crossed cut	The grouping of indices (l, r) against (b, c), which is the matrix itself, and (l, c) against (r, b).
Rotation $\boldsymbol{Q}$, disentangler	The learned orthogonal matrix on the crossed row group. Disentangler is the name in the physics literature.
Plane, plane count h	A rotation within the span of two vectors, and the number of them in $\boldsymbol{Q}$.
Charge $d(h)$	The free parameters of a rotation of h planes, $2hp - 3h$.
Native anchor	Starting the search at $\boldsymbol{Q} = \boldsymbol{I}$, so that plain SVD is always a candidate. The foreign anchor starts at a random rotation instead.
Identity lane, rotation lane	The two terms of the product, the plain factored product and the correction the rotation adds.
Sculpting	Lowering the certified rank of a smooth spectrum by a few directions without any hidden structure, the gain noise achieves.
Entry barrier	The plane count below which a rotation buys nothing although more planes would, Appendix C.
Frozen method, frozen protocol	The method with the fixed settings of Appendix E, used for every reported result.
The measurements	
Common subset	The 100 matrices of the component study, stratified over families.
Load time selection	Measuring the compressed and the dense product once at load and running the faster, so that execution is never slower than dense.
ε_{95}	In Appendix G, the budget at which a compressed model still agrees with the uncompressed one on 95 percent of next tokens.