Only relative ranks matter in weight-clustered large language models

Anonymous authors
Paper under double-blind review

Abstract

Large language models (LLMs) contain billions of parameters, yet many exact values are not essential. We show that what matters most is the *relative rank* of weights—whether one connection is stronger or weaker than another—rather than precise magnitudes. To reduce the number of unique weight values, we apply weight clustering to pretrained models, replacing every weight matrix with K shared values from K-means. For Llama 3.1-8B-Instruct and SmolLM2-135M, reducing each matrix to only 16–64 distinct values preserves strong accuracy *without retraining*, providing a simple, training-free method to compress LLMs on disk. Optionally fine-tuning only the cluster means (centroids) recovers 30–40% of the remaining accuracy gap at minimal cost. We then systematically randomize cluster means while keeping assignments fixed. Scrambling the *relative ranks* of the clusters degrades quality sharply—perplexity can increase by orders of magnitude—even when global statistics such as mean and variance are preserved. In contrast, rank-preserving randomizations cause almost no loss at mid and late layers. On the other hand, when many layers are perturbed simultaneously, progressive layer-by-layer replacement reveals that *scale drift*—not rank distortion—is the dominant collapse mechanism; however, an affine correction $w' = aw + b$ with $a > 0$ (which preserves both rank order and overall weight distribution) can substantially delay this drift. This rank-based perspective offers a new lens on model compression and robustness.

1 Introduction

Large language models (LLMs) continue to improve rapidly across a wide range of language tasks (Brown et al., 2020; Touvron et al., 2023). Yet we still do not understand, in simple terms, how they work. A central question is: how tightly are a model's capabilities tied to the exact numerical values of its weights?

Modern compression methods already suggest the answer: not very tightly. Techniques such as pruning and quantization substantially modify model weights while largely preserving performance. Pruning sets a subset of weights to zero in an already trained model. Quantization reduces numerical precision, for example by converting weights from `float16` to `int8`. In many cases, these transformations cause little to no drop in accuracy. Sometimes light retraining after compression—or modest adjustments during pretraining—helps recover any small performance loss. These results indicate that precise weight values may be far less important than commonly assumed. At the same time, other findings point in the opposite direction: quantization methods must handle outlier channels at full precision to avoid collapse (Dettmers et al., 2022), and recent work has identified individual "superweights" whose removal alone destroys model function (Yu et al., 2024). Taken together, these observations suggest that *some* weight structure is essential — but which?

An even more aggressive form of compression is possible using a simple method: *weight clustering* with scalar K-means (Gong et al., 2014; Han et al., 2016). We group all entries of a pretrained weight matrix into K clusters and replace each value with its cluster representative, see Fig. 1. For example, with $K = 32$, a matrix containing 58 million distinct values is reduced to just 32 shared values—a reduction of nearly six

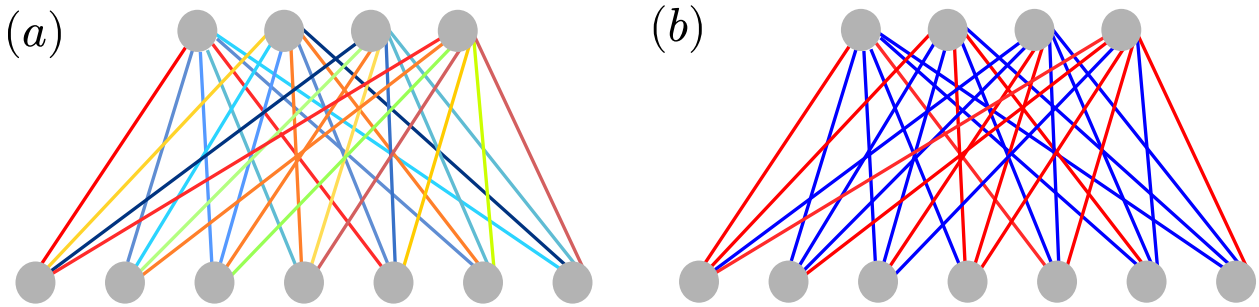

Figure 1: (a) A neural network layer showing connections between input neurons (top) and output neurons (bottom). Edge colors indicate the magnitude of the weights. (b) A clustered version of the same weight matrix with $K = 2$ clusters. In this paper, we demonstrate that when the clustered matrix remains performant, only the relative clustering of the weights (here, cold vs. hot) affects model accuracy. The absolute scale of each cluster can be changed by an affine transformation without substantially degrading model performance.

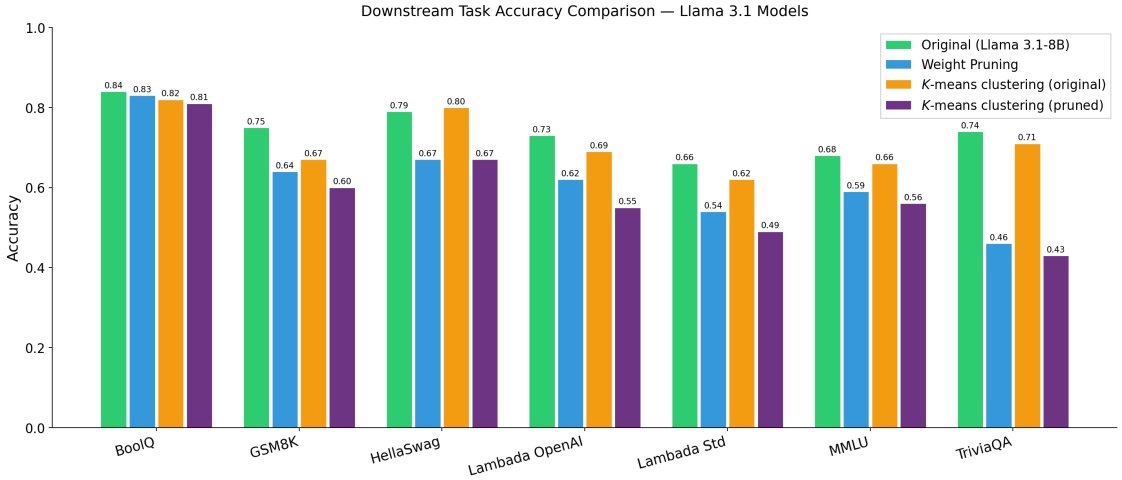

Figure 2: Downstream task accuracy comparison for clustered versions of Llama 3.1-8B-Instruct and a compressed Llama 3B model. We see that weight clustering largely preserves accuracy across all tasks without requiring any fine-tuning.

orders of magnitude. Remarkably, the model often continues to perform well despite this extreme averaging, and it does so without any retraining.

In this paper, we take a closer look at what structure truly matters inside weight-clustered models. We systematically modify the cluster means while keeping the cluster assignments fixed. Our results show that the exact numerical values of the cluster means are not what preserve accuracy. What matters most is their *relative rank*. Across all layer types we tested, rank-preserving changes consistently cause far less damage than rank-breaking ones. Rank preservation alone, however, is not always enough. Early layers are particularly sensitive to shifts in scale and offset, and extreme nonlinear rescalings can degrade performance even when rank order is maintained. Even so, relative rank is the main factor that separates safe modifications from destructive ones: while scale drift can be substantially delayed by affine correction, rank distortion has proven uncorrectable once introduced. This finding has a natural theoretical basis: for any linear layer $y = Wx$, an affine weight transform $w \to aw + b$ produces an output whose ranking across neurons is unchanged, and the resulting scale and offset are approximately absorbed by layer normalization.

A practically important finding is that simple weight clustering largely preserves downstream accuracy across a range of tasks (Fig. 2) *without* requiring any post-clustering training. Furthermore, optional fine-tuning ("healing") of only the cluster mean values recovers an additional 30–40% of the accuracy gap at minimal

cost, but is not required for strong results. Unless explicitly stated, all compression results and experiments in this paper use no post-compression training of any kind.

Relation to existing work. Weight clustering for neural networks dates back to the "Deep Compression" work of Han et al. (2016) and earlier results by Gong et al. (2014). They showed that simple K-means clustering can compress CNN weights by 16–24× with little loss in accuracy.

Modern LLM compression methods use more advanced techniques, including non-uniform or learned-codebook quantization (e.g., AQLM (Egiazarian et al., 2024), VPTQ (Liu et al., 2024), SqueezeLLM (Kim et al., 2024), GANQ (Zhao & Yuan, 2025)), low matrix or tensor rank decompositions (Rausch et al., 2025), and train-time clustering (Cho et al., 2024). Our contribution is different. We do not propose a new compression method. Instead, we use weight clustering as a tool to study implicit structure in LLM weights. Our goal is to identify which properties of LLM weights are strongly correlated with model performance.

Standard post-training quantization (PTQ) methods such as GPTQ (Frantar et al., 2023), AWQ (Lin et al., 2024), and SmoothQuant (Xiao et al., 2023) also tend to preserve the relative ordering of weights, although this property is rarely emphasized as central to their performance. Here, we show, in a simpler setting, that weight ranks are indeed strongly correlated with model accuracy. Weight clustering provides a clean framework for this analysis. Because cluster assignments and their representative values are stored separately, we can modify the representatives while keeping assignments fixed. This separation lets us directly test how changes in relative rank affect performance.

2 K-means clustering of LLM weights

By applying scalar K-means clustering to a weight matrix $W \in \mathbb{R}^{D\times O}$ we can assign all the weights in W to K clusters of distinct values. As a result, we obtain a (clustered) matrix $\hat{W} \in \mathbb{R}^{D\times O}$ given by:

$$\hat{W}[d, o] = c_{L[d,o]}, \tag{1}$$

where the *cluster label matrix* $L[d, o] \in \{1, \ldots, K\}^{D\times O}$ records the parent cluster for each individual weight $W[d, o]$, and c_k is the *centroid* (mean) of the kth cluster.[1]

The clustered weights $\hat{W}$ can be stored compactly on disk: Cluster labels are stored as packed $\lceil \log_2 K \rceil$-bit integers, while the typically short centroid vector adds negligible overhead (comprised of K float16 values). For instance, with $K = 32$, a 58.7M-weight MLP projection is represented by 32 shared values — a $1.8\times10^6\times$ reduction in distinct values (not end-to-end compression ratio) — achieving 2–4× storage reduction when labels are bit-packed and entropy-compressed.

3 Experiments: Randomizing weight-clustered LLMs

The central question of this paper is: In Eq. 1, if and how can we change the centroid values c inside a clustered model while keeping the cluster labels L of individual weights fixed, without significantly degrading the model? To study this, we performed experiments on three different models: (1) Llama 3.1-8B-Instruct, (2) SmolLM2-135M, and (3) a compressed Llama 3B variant. Models 1 and 2 are open-source, while Model 3 is a compressed version of Llama obtained by applying the CompactifAI algorithm (Tomut et al., 2024).

The first step in our experiments was to obtain a clustered version of all three models. For each model, we applied clustering to all attention — namely, the Query (Q), Key (K), Value (V), and Output (O) weights — and MLP projection matrices — namely, the gate, up, and down projections —, while embeddings, layer norms, and the LM head are left unchanged. We fixed the number of clusters, K, independently for all layers, including possibly a different K for the different components (attention $Q/K/V/O$ projections, MLP gate/up/down projections) within each transformer block, using a sensitivity criterion based on perplexity (a standard measure of text prediction quality; lower is better). Specifically, for any given layer, we chose a K that produced a clustered weight matrix that degraded the model perplexity by at most 0.5. This allows assigning a larger K to more sensitive layers.

[1] In Eq. 1, $[.,.]$ denotes matrix/vector indexing, e.g., $\hat{W}[d, o]$ denotes a weight located at row d and column o.

Table 1: Single-layer centroid replacement (layer 10 gate_proj, K=32). We replace the centroid values in a single MLP projection layer while preserving the cluster assignments throughout the model. "Rank-preserving" replacements preserve the rank ordering of the centroid values. "Random" replacements substitute centroid values from a distribution with the same mean and variance as the original values, but do not preserve centroid ranks. "Rand. perm." applies a random permutation to the centroid values, the most extreme form of rank-breaking. Rel. L_2 change measures how much the reconstructed weight matrix changed. μ and σ^2 are the mean and variance of the reconstructed weights.

	Llama 3.1-8B-Instruct				**Compressed Llama 3B**				**SmolLM2-135M**			
Replacement mode	**PPL**	L_2 **rel.**	$\boldsymbol{\mu}$ ($\times 10^{-4}$)	$\boldsymbol{\sigma^2}$ ($\times 10^{-4}$)	**PPL**	L_2 **rel.**	$\boldsymbol{\mu}$ ($\times 10^{-4}$)	$\boldsymbol{\sigma^2}$ ($\times 10^{-4}$)	**PPL**	L_2 **rel.**	$\boldsymbol{\mu}$ ($\times 10^{-4}$)	$\boldsymbol{\sigma^2}$ ($\times 10^{-4}$)
Baseline (orig.)	9.08	—	−0.2	2.0	12.32	—	−0.3	2.5	27.52	—	−1.1	324.4
Ord-pres. (sorted)	9.17	0.52	−16.8	0.6	12.46	0.62	−6.3	4.5	28.35	0.52	−234.4	94.5
Ord-pres. (wt. freq.)	9.17	0.53	−3.0	0.8	12.48	0.54	−1.8	0.6	28.28	0.54	−48.6	109.8
Ord-pres., sign-scr.	9.13	0.51	−73.4	2.0	12.33	0.31	+16.8	2.5	27.53	0.09	−157.7	324.5
Sign-pres., ord-brk.	9.56	1.43	−25.3	6.7	13.37	1.70	−30.3	27.3	36.58	1.43	−289.9	1,069.0
Rand. Gauss. (μ, σ)	9.76	1.42	+25.5	2.4	20.98	3.10	+3.1	2.4	43.92	1.41	+257.4	384.9
Rand. perm. (full)	20.67	2.69	+0.0	10.9	460.2	1.99	−19.4	7.2	3,151	2.88	−133.3	2,126.7

After determining optimal layerwise values of K as described above, we applied K-means clustering to the three models (1)–(3) listed previously, without any subsequent fine-tuning. For SmolLM2-135M, weight-clustering all attention and MLP layers reduces ∼106M distinct values to ∼10.5k centroids in total; for Llama 3.1-8B-Instruct, the total centroid budget across all clustered layers is 8,192. Applied to Llama 3.1-8B-Instruct with the adaptive per-layer K as described above, weight clustering (without fine-tuning) increases WikiText-2 perplexity from 8.64 to 9.08 (internal harness; see Appendix B). Finally, clustering the 3B compressed variant (Tomut et al., 2024) of Llama 3.1-8B-Instruct (PPL 11.88) yields a clustered model with PPL 12.32.

3.1 Randomizing single layers

To begin, we randomized the centroid/mean values (stored in vector c in Eq. 1) inside a single selected layer within the clustered baseline LLM to probe for three distinct effects. First, is scrambling cluster rank more degrading than preserving cluster ranks? Second, should we pay attention to the sign of the cluster means, even when preserving the cluster ranks? And third, how does the impact of these perturbations vary with depth of the layer within the model.

All experiments are carried out on three weight-clustered models — Llama 3.1-8B-Instruct (baseline PPL 9.08), a pre-compressed Llama 3B (PPL 12.32), and SmolLM2-135M (PPL 27.52) — by randomizing the centroid values in some way and reconstructing modified weight matrices using Eq. (1). The Llama 3.1-8B-Instruct and SmolLM2-135M models are publicly available, making these results directly replicable.

Table 1 summarises the results of the experiments, which we explain in detail in the following subsections. Unless the layer is listed explicitly, all single-layer experiments are performed on `model.layers.10.mlp.gate_proj`.[2]

3.1.1 Scrambling Centroid Ranks

We randomize the $K = 32$ centroids of the above mentioned layer, drawing random values from a Gaussian distribution with the same mean (μ) and variance (σ^2) as the original clustered weight matrix.

First, we replace original centroids with the drawn random values, which generally scrambles the cluster ranks. Across all three models, this causes substantial degradation: PPL rises by +7% on Llama-Instruct

[2]We selected this mid-depth layer (layer 10 of 32) as a representative layer where the contrast between rank-preserving and rank-breaking perturbations is clear and not dominated by the extreme sensitivity of early layers. In Sec. 3.1.4 we probe the impact of layer depth on these results.

Table 2: Monotone centroid transformations on `layers.10.mlp.gate_proj` (labels fixed). All transforms preserve centroid rank order. μ and σ^2 are the mean and variance of the reconstructed weight matrix $\hat{W}$ after each transform. The mean stays near zero throughout (the centroid distribution is symmetric); variance is the informative predictor of degradation.

	Llama 3.1-8B-Instruct			**Compressed Llama 3B**			**SmolLM2-135M** †		
Transform	**PPL**	$\boldsymbol{\mu}$ ($\times 10^{-4}$)	$\boldsymbol{\sigma^2}$ ($\times 10^{-4}$)	**PPL**	$\boldsymbol{\mu}$ ($\times 10^{-4}$)	$\boldsymbol{\sigma^2}$ ($\times 10^{-4}$)	**PPL**	$\boldsymbol{\mu}$ ($\times 10^{-2}$)	$\boldsymbol{\sigma^2}$ ($\times 10^{-2}$)
Identity	9.08	−0.2	2.0	12.32	−0.3	2.5	27.52	−1.1	3.2
Affine $w \mapsto 0.5w$	9.18	−0.1	0.5	12.46	−0.1	0.6	28.25	−0.5	0.8
Affine $w \mapsto 2.0w$	9.59	−0.3	8.2	13.10	−0.6	9.9	30.58	−2.2	13.0
$\tanh(\alpha w)$, $\alpha = 1$	9.08	−0.2	2.0	12.32	−0.3	2.5	27.55	−1.1	3.0
$\tanh(\alpha w)$, $\alpha = 2$	9.59	−0.3	8.1	13.09	−0.6	9.9	29.36	−2.3	10.3
$\tanh(\alpha w)$, $\alpha = 3$	11.33	−0.5	18.3	15.89	−0.8	22.2	33.90	−3.3	19.0
$\mathrm{sign}(w)\lvert w\rvert^\gamma$, $\gamma = 1.5$	9.37	−0.03	0.05	12.66	−0.05	0.07	28.09	−0.3	1.0
$\mathrm{sign}(w)\lvert w\rvert^\gamma$, $\gamma = 0.5$	**1,159.52**	**−3.6**	**111.8**	**1,510.1**	**1.2**	**122.3**	31.51	−13.0	14.2

† SmolLM2-135M has larger weight magnitudes; μ and σ^2 are scaled by 10^{-2} rather than 10^{-4}.

(9.08 → 9.76), +70% on the pre-compressed model (12.32 → 20.98), and +60% on SmolLM2 (27.52 → 43.92).

However, next we *sort* the drawn values in ascending order, and replace the original centroids in order of their value, thus, preserving the ranks of the clusters — the cluster that originally represented "strongly positive" weights is still the most positive cluster, the "near-zero" cluster remains the least extreme, and so on. In this case, we find that this rank-preserving randomization increases perplexity by only +1% to +3% across all three models (e.g., 9.08 → 9.17 on Llama-Instruct). Frequency-weighted monotone replacements (spacing centroids in proportion to their cluster sizes) yield similarly small degradations.

We conclude that at least for a mid-network layer (we explore depth dependence in Sec. 3.1.4), all three models are largely indifferent to the exact centroid values as long as the corresponding cluster ranks are preserved.

3.1.2 Scrambling Centroid Signs

Next, we probed if the numerical sign of the centroid values impacts the randomization even when the cluster ranks are preserved.

We find that when the randomization preserved the numerical sign of the centroids but otherwise scrambled their ranks, perplexity degraded in all three models (e.g., PPL 9.08 → 9.56 on Llama-Instruct, 12.32 → 13.37 on the pre-compressed model, 27.52 → 36.58 on SmolLM2). However, when the centroid ranks are strictly preserved but some centroids change sign[3], perplexity remained near baseline in all cases (e.g., PPL 9.13 on Llama-Instruct, 12.33 on the pre-compressed model, 27.53 on SmolLM2).

We conclude that the global rank order matters more than sign structure. The model maintains the ranks of connection strengths — not merely their sign — and this is what must be preserved for the model to function correctly.

3.1.3 Which Rank-Preserving Randomizations Are Safe?

The results so far establish that rank-breaking is harmful. A separate question is whether *all* rank-preserving transforms are safe, or whether other properties of the transform also matter? We compared the impact of a family of strictly rank-preserving transforms on the cluster means of `layers.10.mlp.gate_proj` keeping cluster assignments fixed, summarized in Table 2.

[3]This can be achieved by applying an affine shift $w' = w + b$, where $b = -(c_{\min} + c_{\max})/2$

We find that most of these rank-preserving transforms keep PPL near baseline across all three models. The main outlier is power compression with $\gamma = 0.5$, which causes catastrophic degradation on Llama-Instruct (PPL 1,160) and the pre-compressed model (PPL 1,510), though notably only mild degradation on SmolLM2 (PPL 31.51, 1.14× baseline). Tanh $\alpha = 3$ also degrades all three models. Power $\gamma = 0.5$ expands the tails while Tanh $\alpha = 3$ compresses them (large centroids are squeezed toward ± 1); both distort the weight variance (evident from Table 2) despite preserving rank order, which we believe drives the degradation.

These results suggest that *preserving ordering alone is not sufficient* — the overall weight distribution must also be preserved. Therefore, a broad class of "safe" transformations of the cluster means, namely, maps $f : c_k \mapsto f(c_k)$ are precisely those that preserve both ordering and the mean (μ) and variance (σ^2) of the reconstructed weight distribution:

$$\sum_k p_k f(c_k) = \sum_k p_k c_k \quad \text{(preserve } \mu\text{)}, \tag{2}$$

$$\sum_k p_k [f(c_k) - \bar{c}]^2 = \sum_k p_k [c_k - \bar{c}]^2 \quad \text{(preserve } \sigma^2\text{)}. \tag{3}$$

A natural choice for f are the *affine maps*:

$$f(w) = aw + b, \quad a > 0. \tag{4}$$

To gain some intuition as to why such affine transformation can preserve performance in some situations, consider any linear layer W inside the network. If every entry of W undergoes the affine transform $w \to aw+b$ with $a > 0$ the output activations $y = Wx$ of the layer transform as

$$y_o \;\to\; a\, y_o \;+\; b \sum_d x_d\,, \tag{5}$$

where the offset $b \sum_d x_d$ is identical for every output neuron o. Because the offset is the same across all outputs, the ranks of the activations $y_1, y_2, \ldots, y_O$ are preserved for any $a > 0$. In other words, the ordering of neurons — those that fire most strongly, those near zero, and those most negative — remains unchanged. Subsequent monotone nonlinearities, including SiLU in SwiGLU blocks, therefore preserve this ordering as well, while propagating the scale and offset through the network until encountering a **LayerNorm**. The latter subtracts the mean of the activation vector (removing the constant offset $b \sum_d x_d$ exactly) and divides by the standard deviation (absorbing the scale factor a). Therefore, post-normalization activations are approximately unchanged, making *single* weight affine transformations a near *gauge symmetry* of the parameterization. We verified this numerically: after applying $w \mapsto 0.5w+0.01$ to a single layer's weights, the following post-LayerNorm activations have cosine similarity >0.99 with those in the untransformed model.

The residual stream breaks this symmetry only partially: because each block adds its output to the residual, the affine change is diluted but not fully cancelled, which is why early layers — sitting at the front of a long residual chain — are more sensitive (Section 3.1.4).

When the same affine map is applied to *all* layers simultaneously, these residual contributions accumulate and LayerNorm can no longer absorb the shift exactly. Nevertheless, the global results in Table 2 ($w \mapsto 0.5w$, $w \mapsto 2w$) show that the residual leakage remains small in practice, consistent with each intermediate LayerNorm partially re-centering and re-scaling the activations before the next block.

Note that transforms listed in Table 2 that distort higher-order moments (beyond mean and variance) of the weight distribution — such as the kurtosis expansion from $\gamma = 0.5$ or the tail compression from large-α tanh — create activation distributions that LayerNorm cannot repair, thus, explaining why variance of the reconstructed weights is the informative predictor of degradation in Table 2.

3.1.4 Dependence of Randomization on Layer Depth

Next, we probed the impact of randomizing cluster means inside layers at varying depths in the model. For each model we selected an early, mid, and late MLP projection as well as an attention projection; the exact layer indices differ across models because the architectures have different depths (e.g., layer 26 is late for the

Table 3: Single-layer centroid replacement at varying depths. Each cell perturbs only the named layer; all other layers are unmodified. Rank-breaking conditions (kp sign/brk ord, gauss rand) are consistently more damaging than order-preserving ones (kp ord/brk sign, gauss mono) at every layer.

	kp sign brk ord	**kp ord brk sign**	**gauss rand**	**gauss mono**
Llama 3.1-8B-Instruct (PPL = 9.08)				
l1 (early)	9.64	117.39	2,260	11.30
l10 (mid)	9.56	9.13	9.76	9.17
l30 (late)	13.18	9.08	47.30	10.00
l10 (q-proj)	10.17	9.08	9.83	9.48
Compressed Llama 3B (PPL = 12.32)				
l1 (early)	78,847	24.39	28,315	102.6
l10 (mid)	13.37	12.32	13.22	12.47
l26 (late)	19.56	12.33	293.7	13.35
l10 (q-proj)	15.69	12.37	13.53	13.01
SmolLM2-135M (PPL = 27.52)				
l1 (early)	36.58	27.72	42.60	28.49
l10 (mid)	36.58	27.53	43.92	28.35
l28 (late)	665,943	28.96	69.98	39.40
l10 (q-proj)	32.74	27.58	31.24	28.27

compressed Llama 3B but mid-network for the 32-layer Llama-Instruct, where we use layer 30 instead). See Table 3 for the specific layers used.

The results are listed in Table 3. First, the qualitative pattern is consistent across all four layers and all three models: in every row, the two rank-breaking columns are more damaging than the two rank-preserving columns. Thus, this pattern is not an artefact of a single layer or a single model.

Second, while rank preservation is necessary it is not always sufficient to preserve clustering accuracy. For the early layer (layer 1), even the rank-preserving randomization gives a considerable hit to perplexity. In the Instruct model, the sign-scramble (which uses an affine shift) pushes PPL to 117.39, indicating the sign ladder is load-bearing at early layers. In the pre-compressed model, the sorted-Gaussian replacement gives PPL 102.6. This is a cumulative-drift mechanism, as we explore further in Section 3.2: early layers sit at the front of the propagation chain, so any scale or offset shift amplifies across the remaining depth. By contrast, the mid-network and late layers tolerate order-preserving distortions gracefully (PPL within 1.1× baseline).

Third, rank-breaking at early layers is catastrophic: in the pre-compressed model, PPL reaches the tens of thousands; in the Instruct model, Gaussian random replacement at layer 1 collapses to 2,260; and in SmolLM2, order-breaking at the late layer (L28) causes an extreme collapse to 665,943.

Together, these observations confirm the main thesis while sharply qualifying it: rank-breaking is the main failure mode at every depth; rank-preserving replacements can still hurt via cumulative scale drift, and the penalty grows as the randomized layer sits earlier in the network.

3.2 Randomizing Multiple Layers

Generally, we expect that small distortions introduced at each layer can compound — each layer's output becomes a slightly wrong input for the next, and eventually the signal degrades beyond recovery. To test how errors propagate, we progressively randomized centroids in a rank-preserving way, block-by-block across all transformer blocks of each model, tracking perplexity after every two blocks (replacing from deepest to shallowest, since early layers are known to be more sensitive).

The results are plotted in Fig. 3. Without scale correction (thin lines), even two blocks of monotone-random centroid replacement cause immediate catastrophic collapse ($> 10^5\times$ baseline) in all three models, showing that per-layer scale and offset shifts propagate destructively through the remaining depth.

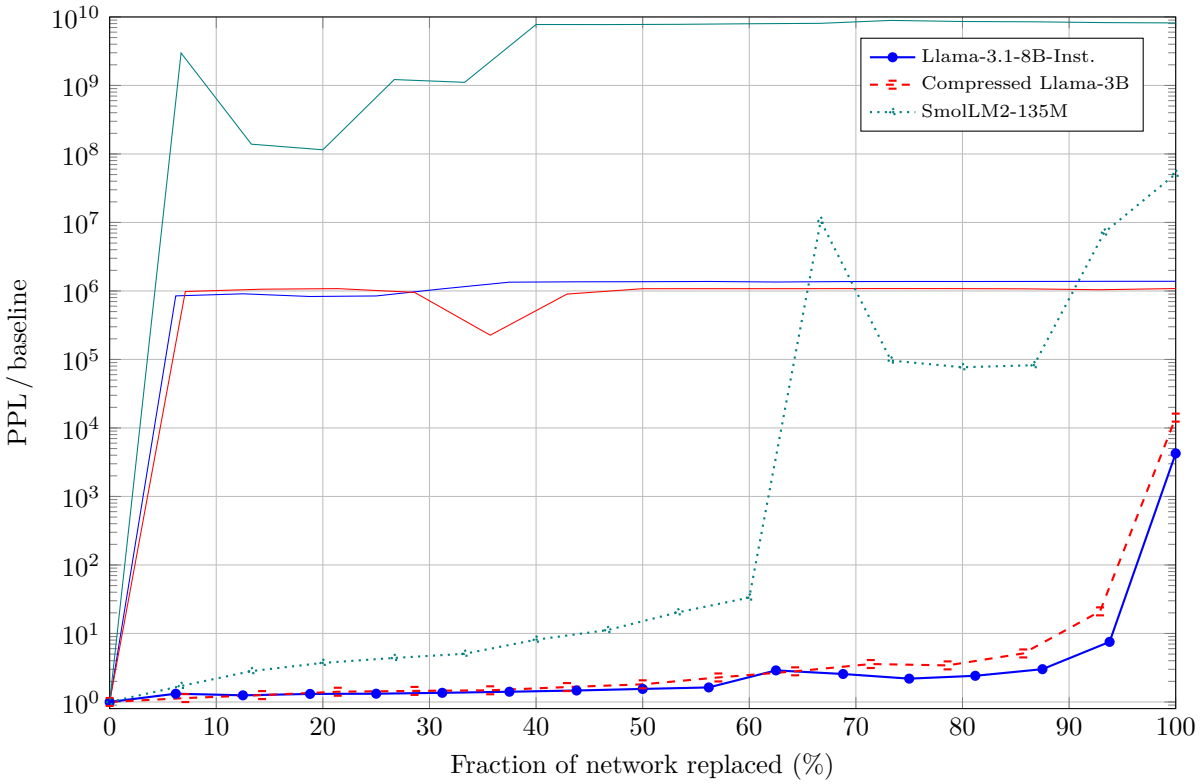

Figure 3: Normalized perplexity (PPL / baseline) as centroids are progressively replaced with rank-preserving-random values, one transformer block at a time from deepest to shallowest. Thick lines with markers show affine-corrected replacement (mean and variance of the reconstructed weights are preserved); thin lines of matching color show uncorrected replacement. Each curve normalizes by its own baseline (Llama 3.1-8B-Instruct: 9.08; compressed Llama 3B: 12.32; SmolLM2-135M: 27.52).

A key question is whether this collapse is driven specifically by *scale drift* (the reconstructed weight distribution shifting layer by layer) or by some other aspect of the randomized monotone distortion of the centroids. To isolate this, we repeat the block-by-block randomization while also preserving the weight distribution (mean and variance) by applying affine maps (Eq. 4) that preserve the mean and variance of the reconstructed weights (Eqs. 2–3). Under this scale-corrected randomization, the perplexity collapse is substantially delayed and reduced in magnitude (Fig. 3). The effect is model-size dependent: Llama 3.1-8B-Instruct stays within 1.6× baseline through 56% of its network; the pre-compressed 3B model stays within 2× through ∼50%; while SmolLM2-135M degrades faster (5× at one-third of the network) and collapses entirely at ∼67%. This confirms that *scale drift* is the dominant mechanism of cumulative failure when many layers are perturbed: rank distortions that also shift the reconstructed weight distribution propagate destructively through layer normalisations, while distortions that preserve that distribution are far more benign.

Crucially, however, the two degradation mechanisms — scale drift and rank distortion — differ in their correctability. Scale drift can be substantially delayed by affine correction, whereas rank distortion, once introduced, has no known post-hoc remedy (Section 3.1.1). The greater robustness of larger models might stem from the fact that their layers are wider and residual streams deeper, which could provide more corrective capacity to absorb per-layer perturbations. In terms of the LayerNorm absorption argument (Section 3.1.3), affine correction removes the first-order distributional mismatch (mean and variance) that LayerNorm would otherwise need to absorb at each step, which is why it delays collapse so dramatically. The residual collapse in the corrected curves reflects higher-order shape distortions — changes in skewness,

Table 4: Representative compression approaches and their execution paths.

Method	Representation	Optimisation	Runtime path
GPTQ, AWQ	Uniform / low-bit	Hessian / activ.-aware	Dense GEMM
AQLM	Additive codebooks	MRF / beam search	Dequant+GEMM or fused
VPTQ, GANQ	Non-uniform / LUT	Vector / MIQP-style	Rebuild or fused
Ours	Scalar K-means	K-means (+ healing)	Rebuild-to-dense

kurtosis, and tail structure introduced by the nonlinear centroid replacement — that LayerNorm cannot correct, and which compound multiplicatively through depth.

3.3 Centroids have a near-normal distribution

For a typical MLP gate projection in the pre-compressed model ($7168 \times 2816 = 20.2$M values, $K = 32$), centroids form a roughly Gaussian distribution centred near zero. Thus, the assignment of weights to clusters is highly *imbalanced* — 19 of the 32 centroids cover 90% of weights, with the two or three near-zero centroids each assigned to over 1.2M weights. Only a handful of "tail" centroids handle extreme values. This imbalance helps explain why monotone replacement works at mid-network layers: the model's behaviour is dominated by the densely-populated near-zero centroids, and as long as the most extreme centroids remain the most extreme (in both directions), each weight position receives a qualitatively correct signal.

Recent work has identified individual "superweights" whose removal alone destroys model function (Yu et al., 2024), which may seem at odds with the finding that exact magnitudes are dispensable. However, what makes a weight critical need not be its raw magnitude but its *rank* — being the strongest or most extreme connection. If superweights are present, clustering tends to isolate them into sparsely-populated tail clusters, preserving their rank amongst the weights.

4 Relation to Quantization and Codebook Methods

Post-training quantization (PTQ) methods such as GPTQ (Frantar et al., 2023), AWQ (Lin et al., 2024), and INT4 (Jacob et al., 2018) also preserve weight ranks, because their quantization grids are monotone: a weight larger than another before quantization will be at least as large after. Thus, these methods implicitly preserve rank structure, and the rank dependence we observe is likely not specific to K-means clustering but shared by any bin-based compression scheme.

Weight clustering provides a cleaner setting for studying this dependence than PTQ for two reasons. First, it stores the cluster labels L and centroid values c as explicit separate objects, so either can be perturbed independently; PTQ does not readily admit this separation. Second, K-means clustering places clusters where the data are densest (near-zero weights), while uniform PTQ grids must span the full range and suffer an outlier penalty; K-means is unaffected because outliers form their own sparsely-populated tail groups. Indeed, randomly permuting the centroid values while keeping cluster assignments fixed — the most extreme form of rank-breaking — causes catastrophic degradation across all three models (see "Rand. perm." row in Table 1: PPL rises to 20.67 on Llama-Instruct, 460.2 on the pre-compressed model, and 3,151 on SmolLM2). This is even worse than rank-breaking Gaussian replacement, which at least preserves mean and variance.

Table 4 places our approach among related methods. Codebook-based methods, e.g., AQLM (Egiazarian et al., 2024), VPTQ (Liu et al., 2024), SqueezeLLM (Kim et al., 2024), GPTVQ (Van Baalen et al., 2024), can achieve higher compression but typically require specialised kernels; train-time methods (eDKM (Cho et al., 2024), LCD (Liu et al., 2025a)) optimise during training rather than post hoc. In contrast, our approach is applied post-training, weights are reconstructed from compact storage via Eq. 1, and no custom inference kernels are employed.

Table 5 summarises the accuracy–storage trade-off on Llama 3.1-8B-Instruct (WikiText-2 perplexity, vLLM harness; see Appendix B), including when quantization is added on top of compression and clustering. Weight clustering applied to the original Llama 3.1-8B-Instruct model reaches PPL 9.32 at 6.9 GB disk without any

Table 5: Accuracy and storage on Llama 3.1-8B-Instruct (WikiText-2 PPL, vLLM harness; see Appendix B). Clustered models use rebuild-to-dense execution, so GPU RAM matches the dense baseline. The compressed + clustered model appears as PPL 12.32 in Section 3 and PPL 13.36 here due to different evaluation libraries (Appendix B).

Model	**Method**	**Disk (GB)**	**GPU (GB)**	**PPL**
Llama 3.1-8B-Instruct: Baseline (FP16)	Full precision	15.0	14.96	8.87
FP8	Uniform quant.	9.1	8.46	9.04
AWQ INT4	Uniform quant.	5.7	5.3	9.35
GPTQ INT4	Uniform quant.	5.7	5.3	9.30
GGUF Q5_K_S	Uniform quant.	5.3	5.2	8.99
AQLM 2-bit	Codebook	4.1	3.80	11.77
Clustered (Orig) — no training	K-means	6.9	14.96	**9.32**
Compressed 3B-Llama	CompactifAI	6.1	6.01	12.62
Compressed + Clustered	CompactifAI + K-means	3.2	6.01	13.36
Compressed + Clustered + Fine-tuned	CompactifAI + K-means	3.2	6.01	13.05
Compressed + Clustered + GPTQ	CompactifAI + K-means + INT4	**2.6**	**2.56**	14.21
Compressed + Clustered + AWQ	CompactifAI + K-means + INT4	**2.6**	**2.55**	13.86

post-compression training. This is comparable in accuracy to INT4 quantization (GPTQ/AWQ), with GPU RAM matching the dense baseline under rebuild-to-dense execution. Applying clustering on top of the pre-compressed model achieves 3.2 GB disk (4.7× reduction from baseline), and an additional INT4 pass pushes storage to **2.6 GB** at **2.55 GB** GPU RAM (Pre-comp. + clustering + AWQ, PPL 13.86) — the most compact configuration. Optional centroid-only fine-tuning ("healing") recovers 30–40% of the accuracy gap; the label structure $\mathbf{L}$ is the primary remaining bottleneck. See also Appendix C for an execution-path analysis.

5 Conclusions

We have used weight clustering as a lens to study which properties of pretrained weight matrices truly matter for performance. Our central finding, demonstrated across three models of different scales and architectures (Llama 3.1-8B-Instruct, a pre-compressed Llama 3B, and SmolLM2-135M), is that exact weight values are not the key factor. Instead, the *relative ranks* of the cluster centroids matter most. Rank-breaking perturbations consistently cause large degradation, while rank-preserving replacements leave perplexity nearly unchanged at mid and late layers. We identify the class of "safe" centroid transformations as affine maps $f(w) = aw + b$ ($a > 0$), which preserve both rank order and the mean and variance of the reconstructed weight distribution. This rank dependence is likely not unique to clustering: any bin-based compression method (including PTQ) implicitly preserves rank structure, and weight clustering is especially suited for studying this because cluster labels and representative values are stored separately.

Our experiments also reveal important qualifications. Early layers are far more sensitive than mid or late layers: even rank-preserving perturbations can degrade performance substantially at layer 1, due to cumulative scale drift through the network. Progressive layer-by-layer replacement reveals that when many layers are perturbed simultaneously, *scale drift* becomes the dominant collapse mechanism; however, an affine correction $w' = aw + b$ with $a > 0$ can substantially delay this drift. The two degradation mechanisms are fundamentally different in character: scale drift is correctable, while rank distortion is not — no post-hoc remedy has been found to recover accuracy once centroid ranks are scrambled.

As a practical compression method, weight clustering applied to the original Llama 3.1-8B-Instruct achieves PPL 9.32 at 6.9 GB disk without any fine-tuning. Combined with pre-compression and INT4 quantization, storage reaches 2.6 GB at 2.55 GB GPU RAM. Optional centroid-only fine-tuning recovers 30–40% of the remaining accuracy gap at minimal cost. However, without specialised inference kernels, clustered models require reconstructing dense weight matrices at load time, limiting runtime memory savings.

Future work includes: (1) characterising the safe region for simultaneous centroid transformations across layers; (2) differentiable optimisation of cluster assignments to close the remaining accuracy gap without full fine-tuning (e.g., building on (Cho et al., 2024; Liu et al., 2025a)); and (3) formalising the conjecture that centroid rank acts as an approximate sufficient statistic for model performance — in particular, whether the approximation quality can be bounded as a function of layer depth and the higher moments of the centroid transform.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Minsik Cho, Keivan A Vahid, Qichen Fu, Saurabh Adya, Carlo C Del Mundo, Mohammad Rastegari, Devang Naik, and Peter Zatloukal. eDKM: An efficient and accurate train-time weight clustering for large language models. *IEEE Computer Architecture Letters*, 23(1):37–40, 2024.

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.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *International Conference on Machine Learning (ICML)*, volume 235, pp. 12284–12303. PMLR, 2024.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024. URL https://zenodo.org/records/10256836. Version 0.4.5.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization, 2014.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *International Conference on Machine Learning (ICML)*, volume 235, pp. 23901–23923. PMLR, 2024.

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

Fangxin Liu, Ning Yang, Junping Zhao, Tao Yang, Haibing Guan, and Li Jiang. LCD: Advancing extreme low-bit clustering for large language models via knowledge distillation, 2025a.

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. VPTQ: Extreme low-bit vector post-training quantization for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8181–8196, 2024.

Zihan Liu, Xinhao Luo, Junxian Guo, Wentao Ni, Yangjie Zhou, Yue Guan, Cong Guo, Weihao Cui, Yu Feng, Minyi Guo, et al. VQ-LLM: High-performance code generation for vector quantization augmented LLM inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1496–1509. IEEE, 2025b. doi: 10.1109/HPCA61900.2025.00112.

Roman Rausch, David Jansen, Sukhbinder Singh, and Román Orús. Globally optimized SVD compression of LLMs via Fermi-function-based rank selection and gauge fixing, 2025.

Andrei Tomut, Saeed S. Jahromi, Abhijoy Sarkar, Uygar Kurt, Sukhbinder Singh, Faysal Ishtiaq, Cesar Muñoz, Prabdeep Singh Bajaj, Ali Elborady, Gianni del Bimbo, Mehrazin Alizadeh, David Montero, Pablo Martin-Ramiro, Muhammad Ibrahim, Oussama Tahiri Alaoui, John Malcolm, Samuel Mugel, and Roman Orus. Compactifai: Extreme compression of large language models using quantum-inspired tensor networks, 2024.

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

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. GPTVQ: The blessing of dimensionality for LLM quantization, 2024.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, pp. 38087–38099. PMLR, 2023.

Mengxia Yu, De Wang, Qi Shan, Colorado Reed, and Alvin Wan. The super weight in large language models, 2024.

Pengxiang Zhao and Xiaoming Yuan. GANQ: GPU-adaptive non-uniform quantization for large language models. In *International Conference on Machine Learning (ICML)*, 2025.

A Additional Compression Results

A.1 Internal Throughput (Gen TPS)

Table 6 reports inference throughput under an internal `model.generate` harness (H200, batch=1, prompt=50, generation=50, greedy decoding). This is a different measurement stack from the vLLM serving benchmarks in Appendix C; the two should not be compared directly. Weight clustering on the *original* model runs at the same throughput as the FP16 baseline: under rebuild-to-dense execution, tensors are reconstructed to bf16 at load time and runtime shapes are identical. The $\sim$3$\times$ gain for pre-compressed variants comes from the pre-compression stage reducing parameter count; clustering adds storage reduction at negligible throughput impact.

B Evaluation Harness Details

We use two evaluation stacks, which are *not directly comparable* in absolute perplexity:

- **Internal harness**: WikiText-2 perplexity via EleutherAI's `lm-eval` harness (Gao et al., 2024) (`lm_eval.evaluator.simple_evaluate`, `batch_size="auto"`). Used for the controlled experiments (Section 3).

Table 6: Inference throughput on Llama 3.1-8B-Instruct (H200, internal harness: batch=1, prompt=50, generation=50, greedy decoding). Gen TPS = generated tokens/second. *Not directly comparable* to vLLM serving numbers in Appendix C.

Model	**Method**	**Gen TPS**
Baseline (FP16)	Full precision	2194
FP8	Uniform quant.	3316
AWQ INT4	Uniform quant.	4606
GPTQ INT4	Uniform quant.	1812
AQLM 2-bit	Codebook	1023
Clustering (Orig) — no training	K-means (rebuild-to-dense)	≈2194
Pre-compressed only	Pre-compression	6991
Pre-comp. + clustering	Pre-comp. + K-means (rebuild-to-dense)	6872

Table 7: vLLM interactive serving benchmark on Llama 3.1-8B-Instruct (512→128 tokens, concurrency=1). TTFT = time to first token; TPOT = time per output token.

Model	**Tok/s**	**TTFT (ms)**	**TPOT (ms)**	**Power (W)**	**TFLOPs**	**Disk (GB)**	**Note**
Original	155	25.1	6.30	266	0.50	15.0	Baseline
Orig. clust. rebuild	157	25.6	6.21	232	0.51	6.9	Same speed, less disk
Pre-compressed	259	18.6	3.74	188	0.83	6.1	Best throughput
Pre-comp. clust. rebuild	269	17.0	3.61	186	0.87	3.1	Best trade-off
Orig. LUT	14	373	67.0	177	0.05	6.5	Kernel-bound
Pre-comp. LUT	34	135	28.2	166	0.11	3.0	Kernel-bound

- **vLLM harness**: Perplexity and throughput via vLLM 0.7.3. Used for Table 5 (accuracy/storage) and Appendix C (serving benchmarks).

Trends are consistent across harnesses; only within-harness comparisons are meaningful for absolute values.

C Inference Speed and Execution Paths

Weight clustering can be deployed via two execution paths: *rebuild-to-dense* (weights reconstructed once at load time, then standard matrix multiplication) and *indexed LUT* (direct operation on labels and centroids without materialising the dense weight matrix). Table 7 compares both paths on the vLLM interactive serving benchmark (Llama 3.1-8B-Instruct, 512→128 tokens, concurrency=1).

Throughput gains for the pre-compressed models arise from their reduced parameter count, not from weight clustering itself; clustering primarily converts this gain into additional *storage* reduction, with minor throughput impact under rebuild-to-dense execution. The LUT rows ("clustering LUT" and "Pre-comp. LUT") show that direct indexed execution is substantially slower — 11–14× lower throughput than rebuild-to-dense — due to kernel bottlenecks explained below.

Why LUT execution is slow. In principle, the grouped matrix-vector product can be rewritten as:

$$y_o = \sum_{d=1}^{D} x_d \cdot c_{L_{do}} = \sum_{k=1}^{K} c_k \cdot \underbrace{\sum_{d:L_{do}=k} x_d}_{\text{partial sum for } k} \tag{6}$$

reducing arithmetic from $O(DO)$ to $O(DK + KO)$ when $K \ll D$. In practice this saving is realised only when label patterns are structured enough to allow contiguous reductions. Table 8 shows that on CPU, all true LUT strategies are 4–6× *slower* than dense computation.

Table 8: CPU inference time under different execution strategies (average over 3 prompts, 32 generated tokens). "Precompute" reconstructs weights once at load time and uses dense matrix multiplication; all other rows perform per-step LUT/gather execution.

Strategy	**Avg. Time (s)**	**vs. Dense**
Dense baseline	0.65	1.00×
Precompute (FP32)	0.66	1.02×
LUT fallback	2.94	4.52× slower
Gather-matmul	3.01	4.63× slower
Chunked reconstruct	4.08	6.28× slower
Vectorized scalar	4.25	6.53× slower

The root cause is that with row-level partitioning (chunk size $D_m = 128$, varying local K per chunk), the effective codebook size per output is $K_{\text{eff}} \approx O$: each centroid is used approximately once per output computation, eliminating the reuse that LUT acceleration depends on. Irregular memory accesses (gather) then dominate, causing cache misses and pipeline stalls. LUT inference can be competitive when: (i) $K \ll O$ so that each centroid is reused many times; (ii) label patterns have high regularity within computation tiles; and (iii) a fused kernel keeps the centroid table in fast cache. These conditions are exploited by methods such as AQLM (Egiazarian et al., 2024) (additive codebooks) and VQ-LLM (Liu et al., 2025b) (codebook caching across the memory hierarchy), but are not met by plain scalar K-means with row-level partitioning.

The practical recommendation is therefore to use the rebuild-to-dense path, which is reliable, immediately deployable, and matches the throughput of the original model.

D Per-Layer Cluster Budget Distribution

Table 9 reports the number of MLP and attention layers assigned to each cluster budget K under the adaptive per-layer selection criterion (Section 2).

K=32 dominates across all three models. The pre-compressed Llama variant requires more K=64 layers (50 vs. 32) than the original Llama 3.1-8B-Instruct, which is expected: the pre-compression stage already introduces distortion, so more centroids are needed to stay within the 0.5 PPL tolerance. Only SmolLM2-135M has any K=16 layers (10 total, mostly in attention projections).

Table 9: Number of MLP and attention projection layers assigned to each cluster budget K across the three models used in this work.

Model		K=16	K=32	K=64	**Total**
Pre-comp. Llama	MLP	0	55	29	84
	Attn	0	91	21	112
Llama 3.1-8B-Inst.	MLP	0	78	18	96
	Attn	0	114	14	128
SmolLM2-135M	MLP	1	66	23	90
	Attn	9	84	27	120