CLAWS: Calibration-Aware Activation Sparsity for Instruction-Tuned LLMs

Noah Cylich [1] **Karen Mosoyan** [1] **Henry Ndubuaku** [1] **Roman Shemet** [1]

Abstract

Modern LLMs contain gated MLPs whose per-token activations are highly skewed, a natural axis for dynamic top-K sparsification. Practical sparsity needs both accurate routing and a mask the inference kernel can execute efficiently. CATS (Lee et al., 2024) gives the hardware-friendly row-wise mask but routes on local gate information alone, while LaRoSA (Liu et al., 2025b) improves routing through rotating the basis with calibration data, but induces a column-sparse pattern poorly aligned with quantized row-major layouts. We introduce CLAWS, a gradient-saliency calibration method for gated-MLP sparsification that keeps the CATS row-sparse execution pattern and multiplies the runtime gate score by a static per-neuron saliency constant, so the top-K mask still maps directly to quantized kernels. We also explain why this multiplier reorders the native gate score but barely moves LaRoSA's rotated-input score, isolating where calibration helps. On Gemma-4-E2B-IT at 50% FFN density, CLAWS recovers dense-level Avg5 and surpasses matched-recipe CATS with LoRA by 6.8pp MMLU (49.6 vs. 42.8), and a custom ARM kernel delivers a $\times 1.24$ wall-clock speedup on the full MLP block.

1. Introduction

Modern LLMs spend a substantial fraction of inference compute in a gated MLP at the center of every transformer block. The MLP applies a gate projection and an up projection to the input, multiplies the two elementwise, and produces the block output via a down projection (Shazeer, 2020). This pointwise gating naturally generates highly skewed per-token activations (Liu et al., 2023). Inference cost is proportionally reduced by skipping the down-projection rows of the unselected neurons under a top-K mask of density $d = K/D_{\text{FFN}}$.

The challenge is to identify which K neurons to retain at each token. The contribution of neuron j to the MLP output is the rank-1 term $\phi((W_g x)_j) \cdot (W_u x)_j \cdot W_d[:, j]$, but routing quality also depends on how sensitive the downstream loss is to that term. CATS (Lee et al., 2024) takes the simplest route and scores each neuron by its gate magnitude $|\phi((W_g x)_j)|$. This needs no preparation and no extra inference work, but it ignores the up activation, the structural contribution of W_d, and how much the loss actually depends on each neuron. GRIFFIN (Dong et al., 2024) scores neurons by the richer product $|\text{gate}(x) \cdot \text{up}(x)|$, yet computes the mask once at prefill and reuses it across all decode steps, so its routing is not truly per-token. LaRoSA (Liu et al., 2025b) instead improves the basis in which magnitudes are compared. It estimates an orthogonal rotation Q from the covariance of layer inputs over a small calibration set, fuses Q into W_g and W_u offline, and takes a magnitude top-K over the rotated input dimensions. The rotated basis routes well, but it induces a column-sparse access pattern that aligns poorly with the row-major quantized layouts of standard toolchains (Section 4.3). Each prior method thus secures one of the two requirements for practical activation sparsity, accurate routing or an executable mask, but not both.

We introduce **CLAWS** (Calibration-Aware Activation Sparsity), a gradient-saliency calibration method for gated-MLP sparsification. CLAWS applies a calibration-time first-order saliency signal in the native FFN-neuron basis, producing row-sparse masks that map directly onto quantized CPU/ARM kernels. CLAWS can be combined with rank-128 LoRA adapters trained against a τ-annealed mask. The adapters fuse at deployment, leaving the runtime path as a single saliency-calibrated scalar multiply followed by the same top-K selection as CATS. We additionally use LaRoSA-style rotated-input sparsity as a mechanistic diagnostic. It tests whether the same static saliency idea can improve a strong rotated magnitude score, and explains why the answer is largely no. Our study centers on the instruction-tuned Gemma-4-E2B-IT, with cross-model replication on the Llama-3-8B base model.

Claims:

[1]Cactus Compute. Correspondence to: Noah Cylich <noahcylich@cactuscompute.com, noahcylich@gmail.com>, Karen Mosoyan <karen@cactuscompute.com>, Henry Ndubuaku <henry@cactuscompute.com>, Roman Shemet <roman@cactuscompute.com>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

1. CLAWS recovers dense-level Avg5 on Gemma-4-E2B-IT at d=0.5 while surpassing matched-recipe CATS+LoRA by **6.8pp** MMLU (49.6 vs. 42.8), showing that the saliency-calibrated routing remains useful under the native row-sparse constraint.
2. Static saliency multipliers have much less leverage in LaRoSA-style rotated-input sparsity. We trace this to multiplicative score geometry. The rotated score $|Q^\top x|$ is broader than its saliency modifier, while GELU compresses the native gate score enough for CLAWS's modifier to alter rankings, the same leverage gap that decides whether any static multiplier can reorder a magnitude-based router. Section 5 develops this, with the full derivation in Appendix G.
3. A custom NEON scatter-gather kernel delivers a $\times 1.24$ wall-clock speedup from sparsity on the full Gemma-4-E2B-IT MLP block at d=0.5 on Apple M-series ARM (cold-cache 35-layer cycling) on the Cactus mobile inference kernel. The same density yields $\times 0.62$ on H100 BF16 because tensor cores dominate scatter-GEMV at the relevant matmul sizes. ARM is a natural deployment target for activation sparsity at this density.

2. Background

Gated FFN. For input $x \in \mathbb{R}^{D_{\text{model}}}$, the gated feed-forward block (Shazeer, 2020) uses a gate projection W_g and up projection W_u, both in $\mathbb{R}^{D_{\text{FFN}} \times D_{\text{model}}}$, and a down projection $W_d \in \mathbb{R}^{D_{\text{model}} \times D_{\text{FFN}}}$, computing

$$h = \phi(W_g x) \odot W_u x, \quad y = W_d h, \quad (1)$$

where ϕ is the gating non-linearity. Common instantiations are $\phi = \text{SiLU}$ (the SwiGLU variant, used in e.g. Llama-3) and $\phi = \text{GELU}$ (the GeGLU variant, used in e.g. Gemma-3 and Gemma-4). Each dimension $j \in [D_{\text{FFN}}]$ of h is a *neuron* with activation $h_j = \phi((W_g x)_j) \cdot (W_u x)_j$.

Top-K activation sparsity. Given a per-neuron routing score $r_j(x)$, the top-K mask $m_j(x) \in \{0, 1\}$ retains the K neurons with highest scores and zeros the rest. The sparsified output is $y = W_d(m(x) \odot h)$. Density $d = K/D_{\text{FFN}}$ controls the active fraction. We focus on $d = 0.5$. A routing score may incorporate statistics precomputed offline from a small calibration set $\mathcal{D}_{\text{calib}}$ of held-out sequences matched to the deployment distribution, at no inference cost.

CATS baseline. $r_j^{\text{CATS}}(x) = |\phi((W_g x)_j)|$. No calibration data, no auxiliary network, no inference overhead beyond an argpartition. CATS serves as the routing baseline that proposed methods must improve upon at matching cost.

Table 1 places the routing signals discussed in this paper, including the GRIFFIN and LaRoSA baselines and the scores we introduce, on the inference-cost axis. Empirical comparison appears in Section 4.

Table 1. Routing signals and their inference cost. CLAWS (Section 3.2) and the rotated diagnostic (Section 5) are ours. The remaining rows are prior work (Lee et al., 2024; Dong et al., 2024; Liu et al., 2025b). Each score is the per-neuron or per-rotated-coordinate scalar ranked by top-K, with h_j the neuron activation of Eq. (1). "Cost beyond CATS" is the per-token, per-layer overhead of forming the score. GRIFFIN amortizes it once per sequence at prefill, and LaRoSA's rotation fuses into the weights offline, leaving only the residual adapter.

Method	Score	Cost beyond CATS
CATS	$\lvert\phi((W_g x)_j)\rvert$	none
GRIFFIN	$\lvert h_j \rvert$	$O(D_{\text{FFN}})$, per sequence
CLAWS	$\lvert\phi((W_g x)_j)\rvert \cdot c_j$	$O(D_{\text{FFN}})$
LaRoSA	$\lvert(Q^\top x)_k\rvert$	D^2_{model} adapter
Rotated diag.	$\lvert(Q^\top x)_k\rvert \cdot c_k^{\text{rot}}$	LaRoSA + $O(D_{\text{model}})$
Oracle	$\lvert h_j \rvert$	$O(D_{\text{FFN}})$

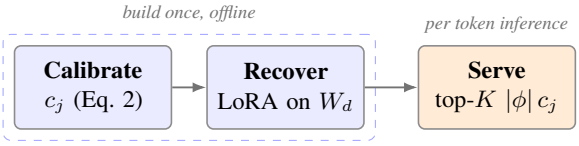

Figure 1. The CLAWS workflow. We compute the static saliency buffer c_j once on the self-distilled calibration set (Eq. 2), recover quality with rank-128 LoRA on the down projection W_d, and at inference scale the CATS gate score by c_j before the usual top-K selection. Each retained neuron selects a full row of W_d.

The rotated diagnostic tests whether saliency calibration has the same leverage in a strong LaRoSA input basis. CLAWS tests whether that principle remains useful when the mask is constrained to native FFN rows, the sparsity pattern required by the quantized ARM/CPU deployment path. Both depend on a calibration distribution that matches deployment, a choice whose consequences we return to in Section 3.3.

3. CLAWS

3.1. Overview

CLAWS adds one calibrated constant to the CATS router and changes nothing else about how a token is served. Figure 1 traces the path. We compute the per-neuron saliency constant c_j once on a calibration set (Eq. 2), scale the ordinary CATS gate score by it at inference, and keep CATS's top-K selection over the down-projection rows. An optional round of LoRA fine-tuning recovers most of the quality lost to sparsification and fuses into W_d, so the deployed model keeps the dense parameter count. Section 3.2 defines the constant and Section 3.3 the recovery training.

3.2. Routing Score

The contribution of neuron j to the MLP output is the rank-1 term $\phi((W_g x)_j) \cdot (W_u x)_j \cdot W_d[:, j]$. CLAWS keeps the native FFN-neuron basis so that selecting neuron j corresponds to retaining a row-sparse access pattern in the down projection. It then uses first-order activation saliency to estimate which native neurons are most important under the deployment distribution. We define the per-neuron weight

$$c_j = \frac{\mathbb{E}_{\mathcal{D}_{\text{calib}}}\big[|h_j(x)| \cdot \|W_d[:, j]\|_2 \cdot |g_j(x)|\big]}{\mathbb{E}_{\mathcal{D}_{\text{calib}}}[|\phi((W_g x)_j)|]}. \quad (2)$$

where $g_j(x) = \partial L / \partial h_j(x)$ is the calibration-time gradient of the loss with respect to the native FFN activation. Eq. (2) is a first-order saliency criterion. Zeroing neuron j perturbs the hidden state by $-h_j(x)$, so to first order the induced loss change satisfies

$$|\Delta L_j(x)| \approx |h_j(x)\, g_j(x)| = |\phi((W_g x)_j)(W_u x)_j| \cdot |g_j(x)|,$$

the standard Taylor/SNIP saliency. The numerator of Eq. (2) weights this saliency by $\|W_d[:, j]\|_2$ to express it on the scale of the neuron's contribution to the MLP output, and the denominator divides out the expected gate magnitude already measured by the runtime score, leaving c_j as a token-independent multiplier on the runtime gate signal. The **CLAWS routing score** is

$$r_j^{\text{CLAWS}}(x) = |\phi((W_g x)_j)| \cdot c_j. \quad (3)$$

Setting $c_j = 1$ for all j recovers CATS as the special case in which calibration data provides no information, so the behavioral difference between the two routers reduces to how far the multiplier moves the gate-magnitude ranking. The constant c_j aggregates the expected up-projection magnitude at neuron j, the down-projection column norm $\|W_d[:, j]\|_2$, the gating non-linearity ϕ, and calibration-time loss sensitivity into a single scalar computed once over a few hundred calibration sequences. This modification is very cheap at runtime, adding only a per-channel scalar multiplication over CATS. Later, Section 4.2 and Appendix B quantify how much the ranking moves in accordance with these multipliers.

3.3. Training

CLAWS pairs the routing score of Section 3.2 with LoRA fine-tuning (Hu et al., 2022) around the dynamic mask, using a τ-annealed soft mask in the spirit of Concrete and Gumbel-softmax relaxations (Maddison et al., 2017; Jang et al., 2017) and distillation against the unsparsified dense model (Hinton et al., 2015).

Calibration and training data. The calibration constants in Section 3.2 are expectations over a calibration distribution $\mathcal{D}_{\text{calib}}$, and the multipliers and the LaRoSA covariance are useful only when that distribution matches deployment. For Gemma-4-E2B-IT we self-calibrate on 500 instruction-formatted chat sequences rendered with the Gemma-4-IT chat template. For Llama-3-8B-base we use 500 raw C4 sequences, since C4 matches the base-model deployment distribution. Distillation runs on the same distribution as calibration so that c_j remains valid as the LoRA adapters update. The teacher is the unsparsified dense model and the loss is $\mathrm{KL}(\text{student} \| \text{teacher}) + 0.2 \cdot \mathrm{CE}$ on teacher-forced ground-truth tokens. Section 4.4 tests this distribution-matching prediction.

Saliency calibration. The gradients in Eq. (2) are computed once on the calibration set before LoRA training, using the same teacher-forced distillation loss as above. For each layer, we accumulate token-level products of activation magnitude, down-projection column norm, and absolute activation gradient, then normalize by the expected runtime magnitude to obtain the static CLAWS multipliers c_j. For the rotated diagnostic in Section 5, we separately accumulate analogous coordinatewise gradient statistics with respect to z_k. The multipliers are fixed during training and deployment. At inference, CLAWS only adds one multiply by the precomputed c_j before the standard top-K selection.

LoRA and mask schedule. Rank-128 LoRA adapters are added to W_d. The gate projection, up projection, layer norms, and embeddings are frozen so that the routing signal stays calibrated. The hard top-K mask is non-differentiable. During training it is replaced by a soft mask $\tilde{m}_j = \sigma(\ell_j / \tau)$, where ℓ_j is a logit derived from the rank of r_j^{CLAWS}. τ anneals linearly from 1.0 to 0.01 over the first 60% of steps and is held at 0.01 for the remaining 40%, sharpening the soft mask toward binary top-K. At inference the soft mask collapses to hard top-K and the LoRA adapters fuse into W_d, leaving the deployed parameter count unchanged from the dense baseline. Full hyperparameters in Appendix F.

Matched LoRA control. We re-run every "+LoRA" row in Table 2 under the same 2500-step MMLU-aux + chat distillation recipe, so routing is the controlled variable. Under that protocol, CLAWS reaches MMLU 49.6 while CATS+LoRA reaches 42.8, and the Dense+LoRA control reaches 44.4, showing that the gain is not explained by the LoRA recipe alone. Full hyperparameters are in Appendix F.

Table 2. Main comparison at d=0.5 on Gemma-4-E2B-IT, $n = 500$ per task, 5-shot MMLU. The top block routes with no fine-tuning. The bottom block applies the same rank-128 LoRA recipe (∼80M tokens) to every method, so routing is the only variable within it. LoRA adapters fuse into W_d at inference, leaving the deployed parameter count unchanged from dense. ΔMMLU is relative to dense.

Method	PPL	MMLU	Avg5	ΔMMLU
Dense Gemma-4-E2B-IT	1.27	46.2	51.44	0
No fine-tuning				
CATS	3.56	29.2	43.04	−17.0
GRIFFIN-prefill	3.94	25.4	46.16	−20.8
LaRoSA	2.65	34.0	48.60	−12.2
CLAWS signal only	2.17	35.4	45.72	−10.8
Matched-recipe LoRA (∼80M tokens)				
Dense + LoRA	1.25	44.4	50.72	−1.8
CATS + LoRA	1.89	42.8	52.60	−3.4
LaRoSA + LoRA	1.54	46.6	50.00	+0.4
CLAWS (ours)	**1.70**	**49.6**	**53.36**	**+3.4**

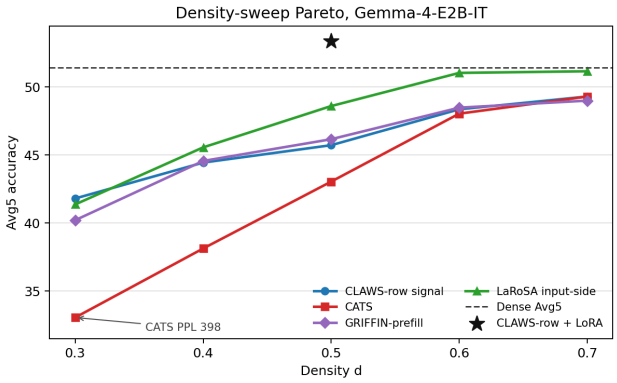

Figure 2. Density-sweep Pareto on Gemma-4-E2B-IT. Avg5 (mean of MMLU/ARC-C/ARC-E/BoolQ/HellaSwag) vs density d. Four no-training methods are plotted as curves: CATS, GRIFFIN-prefill, input-side LaRoSA, and the CLAWS signal (no FT). The full CLAWS method (with LoRA) is plotted as a single star at d=0.5, recovering dense Avg5 and exceeding every LaRoSA point. The CLAWS signal is consistently above CATS at matched density, CATS PPL increases sharply at d=0.3 (annotated). Numerical table in Appendix A.

4. Experiments

4.1. Setup

We evaluate on Gemma-4-E2B-IT (Google, 2025) (4.65B parameters, 35 layers, $D_{\text{FFN}} = 6144$) for the main results, and Llama-3-8B-base (Dubey et al., 2024) for cross-model transfer. Evaluation uses $n = 500$ items per task, 5-shot MMLU and zero/few-shot ARC-C, ARC-E, BoolQ, and HellaSwag. PPL is measured on a held-out instruction-formatted JSONL eval set. Avg5 is the unweighted mean of the five benchmarks. Calibration uses 500 instruction-formatted sequences (MMLU-aux + chat) for Gemma-4-IT and raw C4 for Llama-3-base. Baselines are CATS, GRIFFIN-prefill (a variant of (Dong et al., 2024) that applies the sequence-level mask during prefill scoring, detailed in Appendix C), and LaRoSA, all using upstream code where available. The two models use different gating non-linearities in the FFN of Eq. (1). Gemma-4-E2B-IT is GeGLU ($\phi = \text{GELU}_{\text{tanh}}$) and Llama-3-8B-base is SwiGLU ($\phi = \text{SiLU}$). Equation (2) substitutes the model's actual ϕ when computing c_j.

Headline. CLAWS at d=0.5 recovers dense Avg5 while improving over matched-recipe CATS+LoRA by **6.8pp** MMLU (49.6 vs. 42.8), while preserving the CATS-compatible row-sparse execution pattern.

4.2. Routing-Signal Hierarchy

Each row of Table 2 (excluding dense and CLAWS rows) is the same architecture with a different routing score, so gaps between rows are pure routing-quality gaps. We make three observations.

(i) Saliency-calibrated row routing improves CATS at the same deployment cost. The CLAWS signal alone (without LoRA) lifts MMLU from CATS's 29.2 to 35.4 (+6.2pp) at identical inference cost. The improvement comes entirely from c_j, a static buffer derived from deployment-distribution calibration data, with no training and no extra runtime work beyond one elementwise multiply. Appendix B measures the underlying mask movement with the CATS score as the base. At $d = 0.5$ the multiplier changes roughly 10% of the selected set per token.

(ii) CLAWS is competitive with LaRoSA without training and leads it on accuracy once trained. Among the no-training methods, LaRoSA posts the best Avg5 (48.60) while the CLAWS signal leads on MMLU (35.4 vs 34.0) and PPL (2.17 vs 2.65), so neither is strictly stronger. Under the matched LoRA recipe the accuracy comparison favors CLAWS, ahead of LaRoSA by 3.0pp MMLU and on Avg5 as well, though LaRoSA reaches a lower PPL (1.54 vs 1.70, Table 2). The saliency calibration behind CLAWS's gain is specific to the native gate score and does not transfer to LaRoSA's rotated routing. Section 5 explains this through the geometry of the two base scores.

(iii) With matched LoRA, CLAWS recovers dense accuracy on the row-sparse pattern. The lower block of Table 2 applies the same rank-128 LoRA recipe to all methods, so routing quality is the only variable. CLAWS reaches 49.6 MMLU / 53.36 Avg5, against 44.4 / 50.72 for LoRA-tuned Dense, 42.8 / 52.60 for CATS, and 46.6 / 50.00 for LaRoSA. It recovers dense Avg5 at d=0.5 and improves on matched CATS by 6.8pp MMLU, all on the same row-sparse execution path as CATS.

Table 3. Down-proj GEMV (CLAWS, PowerInfer, GGML) and full-MLP column-sparse (LaRoSA) speedup vs density on ARM (cold-cache 35-layer cycling, GGML Q4_0 / Q8_0), with H100 reference. The LaRoSA row measures a NEON int8 outer-product kernel applied column-sparsely to W_g and W_u at the same quantization (Path 4 of Appendix E).

Backend	d=0.3	d=0.5	d=0.6
GGML dense (baseline)	1.00×	1.00×	1.00×
PowerInfer scatter (down-proj)	1.33×	1.81×	2.17×
Cactus KMI4_fast (down-proj, CLAWS)	**2.20**×	**3.15**×	**3.88**×
LaRoSA Path 4 NEON int8 (W_g + W_u col-sparse)	0.69×	0.71×	0.66×
H100 cuBLAS (decode)	—	0.62×	—

4.3. ARM Deployment

We measure down-proj GEMV latency at Gemma-4-E2B-IT dimensions ($D_{\text{FFN}} = 6144$, $D_{\text{model}} = 1536$, batch=1) on Apple M-series ARM, GGML Q4_0 weights with Q8_0 activations, in a **cold-cache 35-layer cycling regime**. We visit 35 distinct weight matrices (177 MB total, exceeding typical L3) sequentially so each layer's weights are fetched from DRAM per pass. This matches realistic LLM decode memory pressure. We report per-layer median μs over 50 timed cycles. As a GPU reference we measure the same operation on H100 BF16 (Llama-3-8B MLP geometry, n_decode=200).

Full MLP block (gate + up + down). On Cactus, the full row-sparse MLP is $\times 1.24$ over dense at d=0.5 (down-only $\times 1.14$) and $\times 2.11$ over GGML dense. Sparsity gives a comparable $\times 1.31$ on GGML (PowerInfer). We report both because the Cactus dense kernel is already about $\times 1.7$ faster than GGML's. This is an isolated MLP speedup. Since we measured the MLP at roughly half of per-token decode, the sparsity implies a full-model decode gain near $\times 1.1$. **Deployment interpretation.** Row-sparse access aligns with quantized row-major K-blocked layouts because retained rows of W_d are contiguous block sequences. Column-sparse access over W_g and W_u, as induced by LaRoSA-style input sparsity, touches isolated K positions inside quantization blocks, so bytes read remain close to dense unless weights are repacked into a nonstandard layout. This explains why the optimized LaRoSA-style Q4_0/Q8_0 kernel is slower than dense on ARM while CLAWS reaches $\times 1.24$ on the full MLP block. On H100 BF16, the same scatter strategy is $\times 0.62$ at d=0.5 because dense tensor-core kernels dominate at these matmul sizes. Detailed kernel results are in Appendix E.

4.4. Cross-Model Transfer

We replicate the recipe on Llama-3-8B-base (Dubey et al., 2024) with C4 calibration. Table 4 reports the full set of baselines benchmarked under the same protocol on the base model.

We also successfully replicate the results on Llama-3-8B-base. The CLAWS signal lifts MMLU from CATS's 57.6 to 59.6 (+2.0pp), reproducing on a second model family the saliency-calibrated native-row benefit observed on Gemma-4-IT. As Section 3.3 explains, LoRA adds little once the calibration data already matches deployment, thus on this base model fine-tuning leaves the run unchanged (58.6 vs 59.6 no-FT). Additional Llama-family comparison details are in Appendix D.

Table 4. Cross-model transfer to Llama-3-8B-base at d=0.5, $n = 500$ per task, 5-shot MMLU. Avg5 over the same five tasks. C4 calibration (matched to the base-model deployment distribution).

Method	PPL	MMLU	Avg5	Δ MMLU
Dense Llama-3-8B (base)	9.61	62.4	72.40	0
CATS (no FT)	12.48	57.6	67.52	−4.8
GRIFFIN-prefill (no FT)	53.94	27.4	49.20	−35.0
CLAWS signal only (no FT)	11.87	**59.6**	68.00	**−2.8**
CLAWS (ours, +LoRA)	11.25	58.6	70.04	−3.8

5. Why Saliency Moves Native Routing but Not the Rotated Score

The same static multiplier that lifts CLAWS over CATS barely moves LaRoSA's rotated score, and the geometry of the two base scores explains why.

LaRoSA answers a different question from CLAWS. Instead of selecting native FFN rows, it rotates the layer input, $z = Q^\top x$, fuses the rotation into W_g and W_u, and routes by magnitude in the rotated input basis. This produces a strong baseline, but it changes the sparsity pattern to column sparsity over the gate and up projections.

As a diagnostic, we augment LaRoSA's rotated-input magnitude score with the saliency calibration vector constants:

$$r_k^{\text{rot}}(x) = |z_k(x)| \cdot c_k^{\text{rot}}, \quad z = Q^\top x. \quad (4)$$

Each variant of c_k^{rot} enters multiplicatively, as c_j does in Eq. (3). The absolute-gradient variant sets $c_k^{\text{rot}} = \mathbb{E}_{\mathcal{D}_{\text{calib}}}\left[|\partial L/\partial z_k|\right]$, the RMS variant replaces the expectation with a root mean square, and continuous-mask and local-reconstruction variants are described in Appendix B. None of these modifications meaningfully changes algorithm behavior. As Table 5 shows, $|z_k|$ has much wider top-tail spread than its saliency modifier, so multiplication rarely moves coordinates across the top-K boundary. In contrast, CLAWS applies its modifier to the native gate score after GELU has compressed much of the gate distribution, giving the same kind of static multiplier substantially larger rank leverage (Figure 3). A fuller account, including the leverage ratio λ that predicts when a static multiplier can reorder a magnitude score, is in Appendix G.

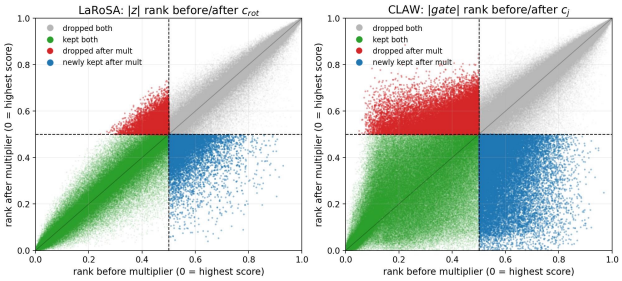

Figure 3. Rank movement before and after applying the static saliency multiplier. Ranks are normalized so 0 is the highest score, and the dashed lines mark the $d = 0.5$ top-K boundary. In the LaRoSA rotated-input basis (left), most points remain close to the diagonal after multiplying by c_{rot}, so top-K membership changes modestly. In the CLAWS native gate basis (right), the multiplier moves many more coordinates across the top-K boundary, producing larger selected-set changes.

Table 5. Distributional spread of base routing scores and saliency modifiers. In the LaRoSA rotated-input basis, the base magnitude $|Q^\top x|$ is already broader than its modifier c_{rot}, so multiplication has limited rank leverage. In the CLAWS native gate basis, GELU compresses $|W_g x|$ into a narrower $|\phi(W_g x)|$ distribution, making the static modifier c_j comparably influential and more likely to change top-K membership.

Quantity	std	p95/p50
$\lvert z\rvert = \lvert Q^\top x\rvert$	0.3507	5.6140
c_{rot}	8.997e-04	2.9601
$\lvert z\rvert \cdot c_{\text{rot}}$	6.395e-04	6.4518
$\lvert W_g x\rvert$	0.6495	5.0794
$\lvert \phi(W_g x)\rvert$	0.1831	2.8340
c_j	0.3062	2.9527
$\lvert \phi(W_g x)\rvert \cdot c_j$	0.2078	4.3465

6. Related Work

Activation-sparsity routers. CATS (Lee et al., 2024), GRIFFIN (Dong et al., 2024), and LaRoSA (Liu et al., 2025b), compared in Table 1, are the closest prior routers. Relative to CATS, which uses gate magnitude alone, CLAWS folds the gate $\times$ up interaction, the down-projection column norm, and calibration-time loss sensitivity into a single scalar c_j at the same inference cost. Because forced-log-likelihood benchmarks score prompts through prefill, we evaluate GRIFFIN-prefill, a variant that also applies GRIFFIN's mask during prefill scoring (Appendix C). We use LaRoSA's rotated-input score as a diagnostic for where static saliency multipliers have leverage, and CLAWS instead applies that signal under the native row-sparse constraint required for quantized deployment.

Wanda and TEAL. Wanda (Sun, Xie, and Kolter, 2024) scores weights by weight magnitude times input activation norm. TEAL (Liu et al., 2025a) keeps activations above calibrated magnitude thresholds. CLAWS instead selects native gated-MLP neurons per token using a loss-gradient saliency scalar that includes the gate $\times$ up term and W_d column norm, while preserving the CATS row-sparse down-projection execution pattern.

Other activation-sparsity systems. Deja Vu (Liu et al., 2023) predicts contextual sparsity with an auxiliary network. PowerInfer and PowerInfer-2 (Song et al., 2024; Xue et al., 2024) use neuron-pattern prediction and CPU/GPU hybrid execution. These systems do not study saliency-calibrated routing under row-sparse quantized CPU/ARM deployment.

7. Conclusion and Limitations

The contribution of neuron j to a gated-MLP output depends on the gate $\times$ up product, the down-projection column, and the downstream loss sensitivity of that term. CLAWS uses saliency calibration to fold these factors into dynamic top-K routing. A simple leverage ratio predicts when a static modifier can reorder a magnitude-based router: the multiplier that lifts CLAWS barely moves LaRoSA's rotated score because its score is already much broader than the modifier. CLAWS recovers dense Avg5 on Gemma-4-E2B-IT at d=0.5 and outperforms matched-recipe CATS+LoRA by 6.8pp MMLU while preserving row-sparse execution. A custom NEON scatter-gather kernel turns the row-sparse masks into a $\times 1.24$ full-MLP-block speedup on Apple M-series ARM.

Limitations. (i) Two model families are tested (Gemma-4-E2B-IT and Llama-3-8B-base), so broader scaling is unverified. (ii) The ARM benchmark isolates down-projection GEMV at Gemma-4 dimensions. Full end-to-end on-device decode latency including attention is left for future work. (iii) Evaluation focuses on standard zero/few-shot benchmarks. Longer-form reasoning evaluation (e.g., GSM8K chain-of-thought, multi-turn generation quality) is not reported in the main body and is partial in the appendix. (iv) Eq. (2) bundles gate magnitude, up magnitude, $\|W_d[:, j]\|_2$, and activation-gradient magnitude. Factor-by-factor ablations are left to future work.

References

Dong, H., Chen, B., and Chi, Y. Prompt-prompted adaptive structured pruning for efficient llm generation. In *Conference on Language Modeling*, 2024. arXiv:2404.01365.

Dubey, A. et al. The llama 3 herd of models, 2024. arXiv:2407.21783.

Google. Gemma technical reports and model cards. https://ai.google.dev/gemma, 2025.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowl-

edge in a neural network. In *NeurIPS Deep Learning Workshop*, 2015. arXiv:1503.02531.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. arXiv:2106.09685.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. arXiv:1611.01144.

Lee, D., Lee, J.-Y., Zhang, G., Tiwari, M., and Mirhoseini, A. Cats: Contextually-aware thresholding for sparsity in large language models. In *Conference on Language Modeling*, 2024. arXiv:2404.08763.

Liu, J., Ponnusamy, P., Cai, T., Guo, H., Kim, Y., and Athiwaratkun, B. Training-free activation sparsity in large language models (teal). In *International Conference on Learning Representations*, 2025a. arXiv:2408.14690.

Liu, K., Xu, B., et al. Larosa: Enhancing llm efficiency via layerwise rotated sparse activation. In *International Conference on Machine Learning*, 2025b. arXiv:2507.01299.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., and Chen, B. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, 2023. arXiv:2310.17157.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. arXiv:1611.00712.

Shazeer, N. Glu variants improve transformer, 2020. arXiv:2002.05202.

Song, Y., Mi, Z., Xie, H., and Chen, H. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *ACM Symposium on Operating Systems Principles*, 2024. arXiv:2312.12456.

Xue, Z., Song, Y., Mi, Z., Chen, L., Xia, Y., and Chen, H. Powerinfer-2: Fast large language model inference on a smartphone, 2024. arXiv:2406.06282.

A. Full Density Sweep

Five densities $\times$ four no-training methods on Gemma-4-E2B-IT, $n = 500$ per task, 5-shot MMLU, single H100 calibration run per entry. **Bold** Avg5 is the within-density winner.

CATS degrades sharply at low density. At $d = 0.3$ its PPL rises to 397.79, while the CLAWS signal remains at PPL 4.05. LaRoSA posts the best no-training Avg5 from d=0.4 upward, while the CLAWS signal has the lowest PPL of any no-training method at every density and the best MMLU through d=0.5 (Table 6).

Table 6. Full density sweep on Gemma-4-E2B-IT across no-training methods.

d	Method	MMLU	ARC-E	ARC-C	BoolQ	HSwag	Avg5	PPL
1.0	Dense	46.2	—	—	—	—	51.44	1.27
0.3	CLAWS signal	31.0	49.0	26.6	55.8	46.6	41.80	4.05
0.3	CATS	25.0	35.0	23.2	47.0	35.0	33.04	397.79
0.3	LaRoSA	26.8	40.8	26.8	70.2	42.2	41.36	9.23
0.3	GRIFFIN-pf	24.6	48.6	26.6	56.8	44.4	40.20	12.55
0.4	CLAWS signal	31.4	49.0	29.2	59.8	52.8	44.44	2.80
0.4	CATS	26.4	37.2	27.8	52.2	47.0	38.12	8.92
0.4	LaRoSA	29.0	47.6	31.0	76.2	44.0	**45.56**	3.87
0.4	GRIFFIN-pf	27.8	49.2	31.4	64.4	50.0	44.56	5.84
0.5	CLAWS signal	35.4	47.4	30.8	61.6	53.4	45.72	2.17
0.5	CATS	29.2	43.2	28.6	62.8	51.4	43.04	3.56
0.5	LaRoSA	34.0	47.8	32.8	78.8	49.6	**48.60**	2.65
0.5	GRIFFIN-pf	25.4	47.2	32.2	72.8	53.2	46.16	3.94
0.6	CLAWS signal	40.0	48.6	33.0	68.2	52.0	48.36	1.77
0.6	CATS	38.4	47.2	32.4	70.6	51.6	48.04	2.44
0.6	LaRoSA	40.8	49.4	35.2	79.0	50.8	**51.04**	2.21
0.6	GRIFFIN-pf	30.6	49.2	34.2	75.6	52.8	48.48	3.06
0.7	CLAWS signal	42.0	48.0	32.5	75.0	49.0	49.30	1.52
0.7	CATS	44.5	46.5	31.5	76.5	47.5	49.30	2.09
0.7	LaRoSA	42.4	48.6	35.6	77.0	52.2	**51.16**	2.05
0.7	GRIFFIN-pf	33.0	47.0	34.0	82.0	49.0	49.00	2.57

B. Rotated-Input Saliency Diagnostic Results

We tested static saliency modifications of the LaRoSA score at $d = 0.5$ by multiplying the rotated-input magnitude $|z_k| = |(Q^\top x)_k|$ by per-coordinate constants estimated from calibration gradients. The variants included RMS gradient constants, absolute-gradient constants, continuous-mask scoring, and local-reconstruction-inspired constants. They produced only small changes around the LaRoSA baseline rather than a new reliable method. The mechanistic diagnostics in Appendix G explain why the multiplier has limited leverage in this basis.

The most compact summary is the induced rank movement. We compare the normalized rank before and after multiplication by the static saliency modifier, with ranks scaled to $[0, 1]$ within each layer and token.

Table 7. Rank movement induced by static saliency modifiers in rotated-input and native gate-space routing.

Score family	Base score	Modifier	rank std(Δ)	rank p95($\lvert\Delta\rvert$)	mean selected-set change
LaRoSA rotated-input	$\lvert z\rvert = \lvert Q^\top x\rvert$	c_{rot}	0.052	0.114	4.95%
CLAWS gate-space	$\lvert\phi(W_g x)\rvert$	c_j	0.124	0.275	10.01%

The modifier distributions are similarly spread, but the base scores are not. In the rotated-input basis, $|z|$ has a wider top-tail than its modifier, so multiplication mostly preserves the original magnitude ranking. In gate space, GELU compresses much of the score distribution, making the same style of multiplier more likely to move coordinates across the top-K boundary. The gate-space row also serves as a direct comparison between the CATS and CLAWS routers,

since its base score $|\phi(W_g x)|$ is exactly the CATS routing score and its modifier is the CLAWS calibration constant.

C. GRIFFIN-Prefill Correction

The upstream GRIFFIN implementation (Dong et al., 2024) selects the sequence-level top-K mask from a single prefill token's $|\text{gate} \cdot \text{up}|$ score, then applies that mask to all *decode* steps. Forced-log-likelihood evaluation protocols for MMLU, ARC, BoolQ, and HellaSwag score the entire prompt + answer choice through prefill (no autoregressive decode), so the sequence-level mask is never applied during benchmarking under the upstream implementation. The reported "sparse" GRIFFIN benchmark numbers therefore equal the dense numbers.

GRIFFIN-prefill is a minimal modification that monkey-patches `GemmaMLP.forward` to apply the top-K mask during prefill scoring as well. This makes GRIFFIN genuinely sparse during eval under the same forced-log-likelihood protocol. Its results (Table 2 and Appendix A) confirm that GRIFFIN's mask is the weakest of the four no-training routing signals on Gemma-4-E2B-IT, with near-chance MMLU at d=0.3 and d=0.5.

D. Llama-3-8B-Base Full Results

Table 8. Full Llama-3-8B-base transfer results at d=0.5.

Method	PPL	MMLU	ARC-C	ARC-E	BoolQ	HSwag	Avg5
Dense Llama-3-8B (base)	9.61	62.4	54.2	81.8	85.4	78.2	72.40
CATS no-FT	12.48	57.6	46.8	77.2	79.8	76.2	67.52
GRIFFIN-prefill no-FT	53.94	27.4	33.8	67.0	60.2	57.6	49.20
CLAWS signal (no FT)	11.87	59.6	45.0	78.4	79.8	77.2	68.00
CLAWS (+LoRA, C4 calib)	11.25	58.6	48.8	82.0	83.8	77.0	70.04
CLAWS (+LoRA-KL, C4 calib)	11.68	58.2	44.4	79.2	80.4	75.6	67.56
CATS (+LoRA-KL, C4 calib)	12.92	55.8	45.2	77.6	77.6	76.2	66.48

The CATS+LoRA-KL row is the direct counterfactual to the IT-model recipe. On a base model with C4 calibration, LoRA-KL training on top of CATS *regresses* vs CATS no-FT (55.8 vs 57.6 MMLU). The same regression appears on CLAWS+LoRA-KL (58.2 vs 59.6 no-FT). This pattern is consistent with Section 3.3. When $\mathcal{D}_{\text{calib}}$ already matches deployment, no-FT is near-optimal and LoRA on the same distribution adds nothing.

E. ARM Kernel Details

Benchmark setup. Apple M-series ARM, GGML Q4_0 weights with Q8_0 activations. We use a cold-cache 35-layer cycling regime where we visit 35 distinct Q4_0 weight matrices (5.1 MB each, 177 MB total, exceeding typical L3) in sequence so each layer's weights are fetched from DRAM per pass. 35 layers $\times$ 5 warmup + 50 timed passes (1750 GEMVs total). Single down-projection GEMV at Gemma-4-E2B-IT dimensions ($D_{\text{FFN}} = 6144$, $D_{\text{model}} = 1536$, batch=1). Per-layer median μs.

Table 9. Down-projection GEMV latency on Apple M-series ARM under the cold-cache 35-layer cycling benchmark.

sp%	K_{live}	GGML dense (μs)	PowerInfer scatter	Cactus KMI4_fast
30%	4288/6144	206.2	155.0 ($\times$1.33)	93.6 ($\times$2.20)
50%	3072/6144	208.5	115.5 ($\times$1.81)	66.2 ($\times$3.15)
60%	2464/6144	206.0	95.0 ($\times$2.17)	53.0 ($\times$3.88)

LaRoSA-paradigm column-sparse measurements. We measured the LaRoSA column-sparse W_g/W_u matmul at the same Q4_0/Q8_0 quantization and same harness as Table 3, using two implementations to separate kernel-engineering quality from the structural layout question (source `scripts/bench_larosa_layout_arm_int4.c`). Path 3 dequants each live column to FP32 then runs a dense FP32 GEMV (the naive "translate to FP32" approach). Path 4 unpacks Q4_0 nibbles directly with NEON `vandq_s8` / `vshrq_n_s8`, multiplies by the int8 activation broadcast via `vmull_s8`, widens through int16 $\rightarrow$ int32 $\rightarrow$ fp32, and applies the combined block scale via `vmlaq_f32`. Path 4 has no per-element dequant, and it does all multiply-accumulate work in int8/int32. The Path 4 kernel quality is below Cactus KMI4_fast by a factor we estimate at $\sim \times 1.2$ (less optimized layout and interleaving), and the residual $\sim \times 1.5$ of the gap to Cactus is structural (LaRoSA pays sparse-access cost on two larger matmuls, while CLAWS pays it on one smaller matmul).

Full-MLP-block measurements (35-layer cold cycle, Gemma-4-E2B-IT dimensions, d=0.5):

Table 10. Full-MLP-block latency for dense, CLAWS row-sparse, and LaRoSA-style column-sparse kernels.

Configuration	μs / layer	μs / 35 layers	$\times$ dense	$\times$ CLAWS
GGML dense full MLP (gate + up + down)	618.5	21,648	1.000	0.475
PowerInfer (GGML sparse)	470.4	16,464	$\times$1.31	0.624
Cactus dense	365.3	12,786	$\times$1.69	0.804
CLAWS Cactus row-sparse (KMI4_fast)	293.6	10,276	$\times$2.11	1.000
LaRoSA Path 3 (FP32 dequant, naive)	1,560.2	54,607	$\times$0.396	$\times$0.188
LaRoSA Path 4 (NEON int8 optimized)	**869.0**	**30,415**	$\times$0.71	$\times$**0.338**

Both LaRoSA kernels are **slower than dense** at the same quantization. Even the optimized int8 outer-product kernel reaches only $\times 0.71$. The underlying issue is the column-sparse access pattern's mismatch with the row-major K-axis-blocked Q4_0 layout (Section 4.3). Kernel engineering closes the FP32-dequant$\rightarrow$int8 gap (Path 3 $\rightarrow$ Path 4 = $\times 1.80$) but cannot close the structural $\times 1.5$ gap to row-sparse on the smaller matmul.

Full 35-layer MLP stack (cold cache). Real scatter-gather for both up and down projections, gate dense:

End-to-end MLP-stack speedup at d=0.5 is $\times 1.24$, the deployment number for the full block (gate, up, and down combined).

Cactus KMI4_fast. K-major interleaved INT4 weight layout (4-row interleave, 64-byte alignment). Inner loop uses

Table 11. Full 35-layer MLP stack speedup with sparse down projection only and sparse up plus down projections.

sp%	dense (μs)	spDn (down only)	spUpDn (up + down, real scatter)
30%	12,872	12,059 (×1.07)	11,407 (×1.13)
50%	12,786	11,202 (×1.14)	10,276 (×1.24)
60%	12,850	10,005 (×1.28)	9,920 (×1.30)
70%	12,733	9,916 (×1.28)	9,038 (×1.41)

NEON `sdot` instructions with prefetch-optimized strides. A precomputed live-group bitmask provides group-level scatter, removing per-row branch overhead so all live groups are read in contiguous bursts.

PowerInfer scatter. PowerInfer-2's `libaz/az/cpu/aarch64/gemv.cpp` is a dense interleaved Q4_0×4 kernel identical to GGML's. The sparse path in `fused_sparse_ffn.cpp` calls `vec_dot_q4_0_q8_0` per live row. We replicate this strategy faithfully, it touches the same bytes as GGML dense (one row per live neuron, each row at non-contiguous Q4_0 blocks), so cache misses scale with K_{live}. Cactus's K-major layout avoids this by pre-packing live groups contiguously.

GPU H100 reference. Llama-3-8B MLP block decode bench at d=0.5, H100 BF16, n_decode=200, bs=1. Dense cuBLAS reaches 54.3 tps and index-gather Triton 33.7 tps (×0.62), with crossover at $d \approx 0.10$ (×1.009 over cuBLAS). Measured from a standalone kernel sweep with no model load.

TEAL `splitk_sparse_gemv` reference (uncontended H100 BF16). We separately benchmarked TEAL's (Liu et al., 2025a) published kernel, which LaRoSA (Liu et al., 2025b) cites for its reported fp16 GPU speedups, under uncontended GPU conditions (n_warmup=20, n_iters=200). At Llama-3-style dimensions (D_m=4096, D_f=14336) the kernel reaches ×**1.092** over dense cuBLAS at d=0.3, falling to ×**0.989** at d=0.5 and ×**0.920** at d=0.7. At Gemma-4-E2B-IT dimensions (D_m=1536, D_f=6144) it is uniformly ×**0.66,** ×**0.65,** ×**0.63** across $d \in \{0.3, 0.5, 0.7\}$. These come from `logs/bench_teal_kernel_h100_clean.json`. We do not rely on this measurement, since LaRoSA's reported end-to-end speedups use a different model scale and harness, but it is consistent with Section 4.3's argument that the column-sparse mechanism is dimension- and harness-sensitive even on fp16, before any quantization-layout question arises. At the Gemma-4 dimensions we study, TEAL/LaRoSA-paradigm fp16 col-sparse is strictly slower than dense at every density.

F. Training Details

CLAWS on Gemma-4-E2B-IT uses LoRA rank 128 / alpha 128, applied to W_d only, with W_g, W_u, layer-norm parameters, and embeddings frozen. AdamW with $lr = 10^{-5}$, weight decay 0, $\beta = (0.9, 0.999)$. Linear LR warmup over the first 2.5% of steps, cosine decay thereafter. 2500 steps, batch size 8, max sequence length 1024.

The τ-anneal schedule starts at 1.0, decays linearly to 0.01 by step $0.6T = 1500$, and holds at 0.01 for the remaining 1000 steps. The mask collapses to hard top-K at inference.

Calibration data are 500 instruction-formatted sequences sampled equally from MMLU-aux dev and a chat instruction corpus, all formatted with the Gemma-4-IT chat template. Constants c_j are computed once per layer from gate / up activations and calibration gradients on this set with the model in eval mode.

For distillation, the teacher is the unsparsified Gemma-4-E2B-IT and the student is the sparsified + LoRA model, trained with loss $\mathrm{KL}(\text{student} \| \text{teacher}) + 0.2 \cdot \mathrm{CE}$ on teacher-forced ground-truth tokens.

CLAWS on Llama-3-8B-base uses the same hyperparameters but with 500 sequences of raw C4 for calibration, and a self-distillation teacher loaded from the same dense checkpoint.

G. Why Saliency Has Less Leverage in LaRoSA Space

This appendix records the diagnostic that explains why static saliency multipliers improve CLAWS but do not produce a comparable LaRoSA extension. Both methods use an input-dependent magnitude score multiplied by a static calibration constant, but the base-score distributions are different enough that the same multiplicative idea has different rank leverage.

Let $a_i(x)$ be the base score and c_i the static modifier. A simple measure of multiplicative leverage is

$$\lambda = \frac{\log_{10}(\mathrm{p95}(c)/\mathrm{p50}(c))}{\log_{10}(\mathrm{p95}(a)/\mathrm{p50}(a))}.$$

When λ is small, the modifier varies less than the score it multiplies, so rankings mostly remain magnitude-driven. When $\lambda \approx 1$, the modifier has comparable top-tail spread and can meaningfully change top-K membership.

The resulting leverage is $\lambda = 0.629$ for the LaRoSA rotated-input score and $\lambda = 1.039$ for the CLAWS gate-space score. The modifier distributions themselves are not the source of the difference, since c_{rot} and c_j have nearly identical p95/p50 spread. The difference is the base score. The rotated input magnitude $|z|$ has a wide top tail, so multiplying by c_{rot} makes comparatively few rank changes. GELU compresses the native gate score, especially because the raw Gemma gate is negative on 72.39% of measured activations, so the CLAWS modifier has comparable spread to the score

it modifies.

The rank diagnostic in Appendix B gives the operational consequence. CLAWS has 2.4x larger rank-delta standard deviation than the LaRoSA rotated-input multiplier (0.124 vs 0.052) and about twice the top-K selected-set change. This is why saliency calibration is useful for the CATS-compatible CLAWS router but is not, by itself, a strong static improvement to LaRoSA.