Less Random, More Incoherent: Greedy Hadamard Alignment for LLM Quantization

Yun Ye, Zhaoqi Wu, Xueshu Wang, Cong Zhang,
Zhonghua Sun, Yongbo Zhu, Beryl Xu
Intel China
yun.ye@intel.com

Abstract

Hadamard preprocessing is widely used in large language model (LLM) quantization to reduce activation coherence and suppress peaks. Existing methods typically rely on random sign flips before the Hadamard transform, which do not use information from the target data distribution. We propose Greedy Hadamard Alignment (GHA), a simple training-free method to build a signed-permutation preconditioner from calibration data. GHA uses a pooled representative activation, a greedy search over flat targets, and an efficient alignment step that is analytically optimal once the target is fixed. We also study when one shared GHA preconditioner can transfer across LLM activations. GHA works best when activations are directionally aligned and stable across calibration sources. Overall, GHA improves post-quantization model quality over randomized Hadamard baselines, with the strongest gains in KV-cache quantization, more moderate gains for hidden states, and little compute overhead during inference.

1 Introduction

As large language models (LLMs) grow, their memory and computation costs become harder to manage. Quantization is a common way to reduce this burden while keeping the accuracy loss acceptable.

One main difficulty in LLM quantization is that activations can be highly concentrated in a few coordinates, creating a small number of large outliers (Bondarenko et al., 2021; Sun et al., 2024). This concentration can be viewed as high activation coherence, which becomes more pronounced at larger model scales (Dettmers et al., 2022) and makes low-bit quantization harder by enlarging the value range seen by quantizers.

Among existing solutions, Hadamard-based incoherence preprocessing has become one of the most practical choices for LLM quantization (Gong et al., 2025; Liu et al., 2025a). It reduces activation coherence by spreading directional concentration across dimensions, and it can be implemented efficiently by the Fast Walsh-Hadamard Transform (FWHT) (Fino and Algazi, 1976) without general multiplications. In practice, many methods apply a random sign matrix before the Hadamard transform to avoid unfavorable alignment with the fixed Hadamard basis (Ailon and Chazelle, 2006; Halko et al., 2011). This Randomized Hadamard Transform (RHT) is simple and often effective, but its random signs are selected independently of the target data. Therefore, standard RHT does not directly optimize the post-transform activation range for the target data.

This raises the main question of this work: can we replace random sign flips with better preconditioning before the Hadamard transform, so as to promote incoherence more effectively? Based on this idea, we propose Greedy Hadamard Alignment (GHA), a simple training-free method that constructs a signed-permutation preconditioner DP from a pooled representative activation computed from calibration data so as to reduce the post-transform peak $\|HDPx\|_\infty$, where H is a Hadamard matrix, D is a diagonal sign matrix, P is a permutation matrix, and x denotes an activation vector. GHA requires no model training or gradient-based optimization and only uses a small calibration set. It uses a surrogate objective as an effective optimization target for reducing the post-transform peak. A key observation is that, once a flat target is fixed, the difficult signed-permutation step admits an analytical solution. By the rearrangement inequality (Hardy et al., 1952), the optimal permutation is obtained by sorting and matching the magnitudes of the input and the target. GHA then only needs to search greedily over the flat target itself. We also implement $HDPx$ with a fused kernel, where the extra inference overhead of applying DP over a plain Hadamard transform is nearly negligible in our benchmarks. Figure 1

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 1526–1540
November 6-10, 2026 ©2026 Association for Computational Linguistics

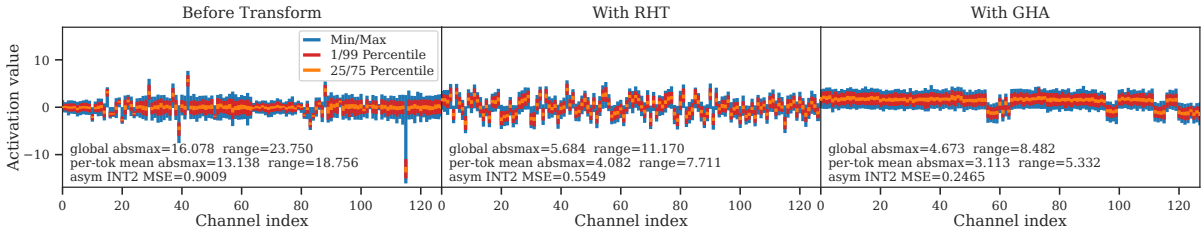


Figure 1: Illustrative pre-RoPE Key distribution from Llama-2-7B (layer 5, head 0) before transform, with RHT, and with GHA (calibrated using channel-wise medians).

gives an illustrative pre-RoPE Key example, where GHA produces a visibly flatter distribution than both the original tensor and a representative RHT draw.

The contributions of this paper are three-fold. First, we propose GHA, a training-free method that combines a greedy flat-target search with an analytically optimal alignment step for any fixed target. Second, we evaluate GHA across KV-cache and hidden-state quantization and show that it outperforms randomized Hadamard baselines overall, especially for KV-cache quantization. Third, we show that, with a fused implementation, GHA introduces almost no extra inference overhead.

2 Related Work

Work on reducing activation coherence in LLM quantization mainly falls into three groups. These are mixed-precision methods (Dettmers et al., 2022), scaling-based equivalent transformations (Xiao et al., 2023; Wei et al., 2023; Shao et al., 2024), and orthogonal-transform methods. Our work belongs to the third group and focuses on preprocessing before the Hadamard transform.

Randomized projections and orthogonal mixing have a longer history in randomized preconditioning (Avron et al., 2010; Ailon and Chazelle, 2006). In LLM quantization, QUIP (Chee et al., 2023) studies random orthogonal pre- and post-processing, QuIP# (Tseng et al., 2024) uses the Randomized Hadamard Transform (RHT), and QuaRot (Ashkboos et al., 2024) extends Hadamard-based rotation to end-to-end W4A4KV4 quantization. DFRot (Xiang and Zhang, 2025) studies why randomized Hadamard preprocessing can outperform randomized dense orthogonal rotations in low-bit settings.

Some later works add more structure. RotateKV (Su et al., 2025) adds calibrated channel reordering for KV tensors. GSR (Choi et al., 2025) uses grouped Walsh blocks. SpinQuant (Liu et al., 2025b), FlatQuant (Sun et al., 2025), ButterflyQuant (Xu et al., 2026), DuQuant (Lin et al., 2024), and PeRQ (Sanjeet et al., 2026) replace purely random rotations with learned, heuristic, or calibrated transformations.

Our work is closest to methods that keep the Hadamard structure while improving its surrounding transformations. GHA is training-free. It uses calibration data but requires no model training or gradient-based optimization. It constructs a signed-permutation preconditioner from a Hadamard-based surrogate objective, with an analytical alignment step and a lightweight greedy target search. To our knowledge, this is the first work to study signed-permutation preconditioning for Hadamard-based LLM quantization.

3 Greedy Hadamard Alignment

3.1 Problem Formulation

Let H be a normalized Hadamard matrix of size $d \times d$, satisfying

$$H^\top H = I, \quad H_{ij} \in \left\{ \pm \frac{1}{\sqrt{d}} \right\}.$$

Given an input vector x with dimension d, let P be a permutation matrix and let D be a diagonal sign matrix with entries in $\{\pm 1\}$. The objective is to minimize the peak value of the transformed output:

$$\min_{P,D} \|HDPx\|_\infty. \tag{1}$$

Direct optimization of this discrete combinatorial problem is difficult because the ℓ_∞ objective is non-smooth and many local changes do not affect the current maximum. Instead of solving this problem exactly, GHA uses a surrogate view to extract useful structure and turn it into an efficient alignment procedure.

3.2 Flat-Target Surrogate Objective

If we relax the search space to arbitrary orthogonal transforms, then we have

$$\min_{Q^\top Q=I} \|Qx\|_\infty = \frac{\|x\|_2}{\sqrt{d}}.$$

The minimum is attained when all coordinates of Qx have equal magnitude. It suggests the following flat target set:

$$\mathcal{F}_\rho = \left\{ \pm \frac{\rho}{\sqrt{d}} \right\}^d, \quad \rho = \|x\|_2.$$

Let $y = HDPx$. Then, for any $z \in \mathcal{F}_\rho$,

$$\begin{aligned} \|y - z\|_2 &\geq \|y - z\|_\infty \\ &\geq \|y\|_\infty - \|z\|_\infty \\ &= \|y\|_\infty - \frac{\rho}{\sqrt{d}}. \end{aligned}$$

Therefore, the ℓ_2 distance from y to the flat target set $\mathcal{F}_\rho$ gives an upper bound on $\|y\|_\infty$:

$$\|y\|_\infty \leq \frac{\rho}{\sqrt{d}} + \min_{z \in \mathcal{F}_\rho} \|y - z\|_2.$$

This motivates the following distance-based surrogate objective:

$$\min_{P,D,z \in \mathcal{F}_\rho} \|HDPx - z\|_2^2.$$

Here z is a flat target for the optimization over P and D. Although the bound may not be tight, we find that minimizing the surrogate effectively reduces the actual post-transform peak (see Appendix A). Using the orthogonality of H, we can rewrite it as

$$\min_{P,D,z \in \mathcal{F}_\rho} \|DPx - H^\top z\|_2^2.$$

In this pullback form, $H^\top z$ is the corresponding target in the pre-Hadamard domain. Expanding the square gives

$$\begin{aligned} \|DPx - H^\top z\|_2^2 &= \|DPx\|_2^2 + \|H^\top z\|_2^2 \\ &\quad - 2\langle DPx, H^\top z \rangle \\ &= 2\rho^2 - 2\langle DPx, H^\top z \rangle. \end{aligned}$$

Therefore, we can equivalently work with the following inner-product form of the surrogate objective:

$$\max_{P,D,z \in \mathcal{F}_\rho} \langle DPx, H^\top z \rangle.$$

3.3 Signed Permutation Alignment

For fixed z, let $u = H^\top z$. Then the diagonal sign matrix can be optimized coordinate-wise for any permutation P, so the problem reduces to

$$\max_P \sum_{i=1}^d |(Px)_i| \, |u_i|.$$

Since P only permutes the coordinates of x, sorting $|Px|$ gives the same sequence as sorting $|x|$. Once $z \in \mathcal{F}_\rho$ is fixed, the remaining step is a rearrangement problem. By the rearrangement inequality, the optimum matches the entries of $|x|$ and $|u|$ in the same order. We therefore sort both vectors by magnitude, use the resulting match to build $\hat{P}$, and then choose the signs so that $\hat{D}\hat{P}x$ agrees coordinate-wise with u.

This reduces the fixed-target surrogate alignment problem to a deterministic sorting problem once z is fixed. It does not solve Eq. (1) globally. The procedure is summarized in Algorithm 1.

Algorithm 1 Signed Permutation Alignment

$u \leftarrow H^\top z$
$\mathrm{idx}_x \leftarrow \mathrm{argsort}_\downarrow(|x|)$
$\mathrm{idx}_u \leftarrow \mathrm{argsort}_\downarrow(|u|)$
$\hat{P} \leftarrow 0 \in \{0,1\}^{d \times d}$
for $t \leftarrow 1$ **to** d **do**
    $i_t \leftarrow \mathrm{idx}_x[t], \quad c_t \leftarrow \mathrm{idx}_u[t]$
    $\hat{P}_{c_t, i_t} \leftarrow 1$
end for
$v \leftarrow \hat{P}x$, $\hat{D} \leftarrow I_d$
for $i \leftarrow 1$ **to** d **do**
    $\hat{D}_{i,i} \leftarrow -1$ if $u_i v_i < 0$ else 1
end for
return $(\hat{P}, \hat{D})$

3.4 Greedy Search for the Flat Target

We choose the flat target z by greedily minimizing the pullback surrogate

$$\left\| |x|_\downarrow - |H^\top z|_\downarrow \right\|_2^2, \tag{2}$$

where $|\cdot|_\downarrow$ denotes descending sorting by absolute value. Let $\rho = \|x\|_2$ and $\alpha = \rho/\sqrt{d}$. Starting from $z^{(0)} = 0$, at step t we consider every still-unassigned coordinate c and both sign choices $s \in \{\pm 1\}$, which induces the candidate update

$$\begin{aligned} z' &= z^{(t)} + s\alpha e_c, \\ u' &= H^\top z'. \end{aligned}$$

Here $e_c \in \mathbb{R}^d$ denotes the c-th standard basis vector. At each step, we pick the candidate with the smallest surrogate value, fix that coordinate, and continue until all d coordinates of z are assigned. We then run the fixed-target alignment core on the final z. The full GHA procedure is summarized in Algorithm 2.

Algorithm 2 Greedy Hadamard Alignment

$\rho \leftarrow \|x\|_2, \alpha \leftarrow \rho/\sqrt{d}$
$z \leftarrow 0 \in \mathbb{R}^d, u \leftarrow 0 \in \mathbb{R}^d, \mathcal{C} \leftarrow \{1, 2, \ldots, d\}$
for $t \leftarrow 1$ **to** d **do**
  $S_t(c, s) := \left\| |x|_\downarrow - |u + s\alpha H^\top_{c,:}|_\downarrow \right\|_2^2$
  $(c_t, s_t) \leftarrow \underset{c \in \mathcal{C},\ s \in \{\pm 1\}}{\arg\min}\ S_t(c, s)$
  $z_{c_t} \leftarrow s_t \alpha$
  $u \leftarrow u + s_t \alpha H^\top_{c_t,:}$
  $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c_t\}$
end for
return SIGNEDPERMALIGN(x, z)

4 Extending GHA to LLM Activations

The formulation above develops GHA for one vector x. In LLM quantization, one shared preconditioner must be reused across many token activations. This turns the problem into calibration. We therefore ask two questions. Can one representative vector summarize activations from one source? How well does that representative transfer across sources?

4.1 Calibration Representativeness

Given a calibration set

$$\mathcal{X}_{\text{cal}} = \{x_j\}_{j=1}^N, \quad x_j \in \mathbb{R}^d,$$

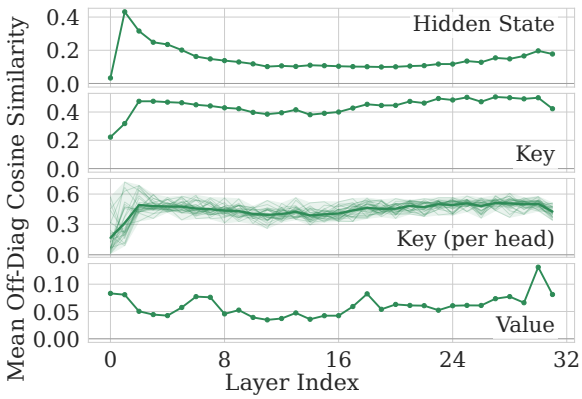


Figure 2: Layer-wise mean pairwise cosine similarity of token activations in Llama-2-7B.

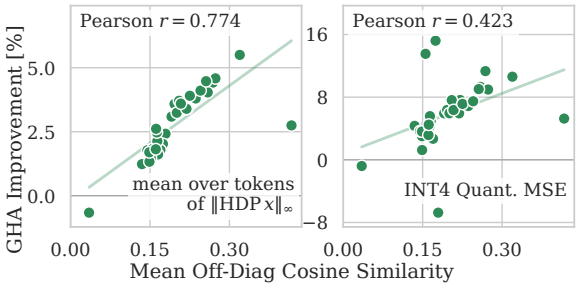


Figure 3: Layer-wise median-based GHA improvement over the mean RHT baseline vs. mean pairwise cosine similarity on the input hidden states of Llama-2-7B.

the practical task is to construct one pair $(\hat{D}, \hat{P})$ that keeps post-transform peaks small across the set. Joint optimization over all inputs is often too expensive, so we approximate the set with one representative vector x_{rep}, run GHA once on x_{rep}, and reuse the resulting $(\hat{P}, \hat{D})$ during inference. This approximation is most plausible when token activations still share some directional structure.

We test this by sampling real LLM activations and computing mean pairwise cosine similarity. Fig. 2 shows the layer-wise values for Key, Value, and hidden states in Llama-2-7b (Touvron et al., 2023). The consistently positive values indicate shared directional structure and support the use of one representative calibration vector.

We construct the representative vector by applying a statistic independently to each channel,

$$(x_{\text{rep}})_i = \phi\left(\{x_{j,i}\}_{j=1}^N\right), \quad \phi \in \{\text{mean}, \text{median}, \text{extreme}\},$$

where the extreme statistic emphasizes the largest-magnitude channel responses while retaining their signs. We evaluate all three choices in Section 5. This approach can effectively reduce the post-transform range and quantization error. Appendix D provides an example on Llama-2-7B hidden states.

Figure 3 further relates this behavior to directional structure. For Llama-2-7B hidden states, GHA improvement over the mean RHT baseline grows with mean pairwise cosine similarity. The linear fit also suggests that as cosine similarity approaches zero, GHA tends to match RHT rather than underperform it. We therefore use cosine similarity as a practical indicator of representative-vector reliability.

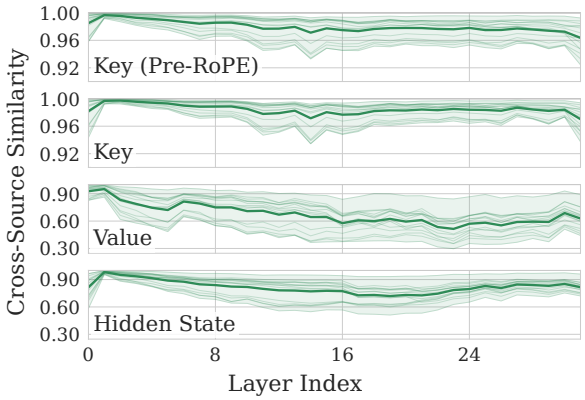


Figure 4: Cross-source similarity of the channel-wise median in Llama-2-7B across six Pile sub-categories. Each panel shows the 15 off-diagonal pairwise cosines, their min/max envelope, and their mean.

4.2 Calibration Transfer Across Sources

The analysis above assumes a fixed calibration set. In practice, however, the preconditioner $(\hat{D}, \hat{P})$ is built once and reused. The key question is whether the representative vector x_{rep} stays similar across data sources. If it does, calibration can transfer. If it does not, GHA becomes much more sensitive to the calibration corpus.

We test this on six categories from the validation split of the uncopyrighted variant of The Pile (Gao et al., 2020): ArXiv, GitHub, FreeLaw, DM Mathematics, Pile-CC, and Wikipedia. For each category, we sample 128 documents of length at least 2048. For each layer in Llama-2-7B, we take the channel-wise median across the 128 documents to obtain one 4096-dimensional x_{rep} for each category. We then compute cosine similarity over all 15 cross-category pairs.

Figure 4 shows two clear patterns. Key activations, before or after RoPE, remain highly stable across sources. Their pairwise cosines stay above 0.92 and vary little. In contrast, Value and hidden-state activations are much more sensitive.

4.3 Practical Considerations

4.3.1 Windowed Greedy Search

For high-dimensional Hadamard transforms in LLMs, the full greedy search in Algorithm 2 is expensive because each step evaluates all remaining coordinates and both sign choices. In practice, we use a windowed variant that only considers the first m available coordinates at each step. The surrogate in Eq. (2) is still evaluated on the full candidate vector. This only reduces the number of candidates examined per step. When m is at least the number of available coordinates, the method becomes the full search.

In our implementation, m follows an anchor-based schedule. Given an anchor a, we use $m = a$ when $d \leq a$, and otherwise set $m = a^2/2^{\lceil \log_2 d \rceil}$. We use $a = 1024$ in all experiments.

4.3.2 Fused Kernel Implementation

A naive implementation such as `had(x[:,p]*d)` in PyTorch applies $\hat{D}$ and $\hat{P}$ as separate preprocessing steps. This can materialize intermediate tensors and make the operation memory-bandwidth bound. We instead fuse both into the Hadamard kernel. $\hat{P}$ is absorbed into the first load, and $\hat{D}$ is applied just before the first butterfly stage. Input data therefore moves from input to output in one read/write pass.

We provide this fused interface in two backends. The CUDA backend extends Dao (2023) with an indexed gather and an in-register sign flip. The CPU backend mirrors the same design on AVX-512. The overhead over a plain Hadamard transform is small, and the fused version is much faster than a dense `matmul` baseline.

5 Experiments

5.1 KV Cache Quantization

The cross-source transferability observed in Section 4 allows us to evaluate KV-cache quantization on RULER-4096 without calibrating on RULER data. We evaluate LLaMA-3.1 8B Instruct (Grattafiori et al., 2024) and Qwen3 4B (Yang et al., 2025). For each model, we run prefill on the full prompt, quantize and dequantize the KV cache once, and then continue generation from the updated cache. We calibrate GHA on 128 WikiText-2 training sequences of length 4096.

5.1.1 2-Bit RULER-4096 Evaluation

We evaluate per-layer (PL) and per-head (PH) transformations applied either before or after RoPE. For GHA, we compare mean, median, and extreme representative vectors. The mean is streamed over all tokens. For median and extreme, we first compute a within-sequence channel-wise median or signed maximum-magnitude value, respectively, and then take the median across sequences. This avoids storing all token activations.

GSR (Choi et al., 2025) uses 128-dimensional Walsh blocks aligned with attention heads, so we report it under PH. MassDiff instead reblocks channels globally before per-head transforms, so we

Model	Method	Post-PL	Post-PH	Pre-PL	Pre-PH
LLaMA-3.1 8B	FP16		95.71		
	Direct Quant.	87.44	87.44	63.86	63.86
	RHT	94.25±0.28	92.99±0.39	83.12±1.39	79.23±0.79
	GHA (mean)	**94.39±0.22**	**93.51±0.22**	**84.61±0.93**	**79.63±2.04**
	GHA (median)	**94.26±0.13**	**93.30±0.52**	**84.77±0.49**	**80.00±1.61**
	GHA (extreme)	94.07±0.23	**93.07±0.38**	**84.38±0.80**	**80.67±1.21**
	DFRot	94.06±0.33	91.77±0.75	**86.13±1.37**	77.96±1.68
	ButterflyQuant	93.61±0.26	**93.01±0.54**	81.71±1.60	72.25±2.46
	GSR	–	**93.23**	–	**81.13**
	MassDiff	94.14±0.24	–	**85.04±0.72**	–
	MassDiff+GHA	**94.27±0.23**	–	**85.27±0.70**	–
Qwen3 4B	FP16		93.36		
	Direct Quant.	84.27	84.27	57.39	57.39
	RHT	91.53±0.45	90.36±0.40	78.61±1.90	74.25±0.69
	GHA (mean)	**91.89±0.30**	**91.02±0.18**	76.07±3.30	**78.29±1.13**
	GHA (median)	**91.83±0.24**	**91.14±0.38**	76.53±1.66	**77.79±1.52**
	GHA (extreme)	**91.95±0.34**	**91.01±0.30**	**79.99±3.23**	**77.11±1.30**
	DFRot	**91.60±0.17**	89.49±0.47	**81.73±1.96**	70.77±1.95
	ButterflyQuant	**91.71±0.29**	**90.72±0.38**	74.64±3.75	67.52±3.68
	GSR	–	**90.78**	–	**74.69**
	MassDiff	91.51±0.36	–	**81.04±1.82**	–
	MassDiff+GHA	**91.60±0.32**	–	**79.97±2.24**	–

Table 1: Overall RULER-4096 (Hsieh et al., 2024) accuracy under 2-bit asymmetric KV-cache quantization (Keys: per-channel; Values: per-token). Results with stochastic preconditioning report mean ± standard deviation over 10 calibration seeds, while deterministic GSR is reported as a single value. Direct Quant. uses no preconditioning. Results above RHT are bolded; the best result in each column is also underlined.

report MassDiff and MassDiff+GHA under PL, although the latter refits GHA per head.

Table 1 shows that the best GHA representative improves over RHT in all eight model-setting combinations, although no single aggregation is uniformly best. The different trends for LLaMA and Qwen3 suggest that the most effective aggregation strategy varies by model. GHA is strongest among the standalone methods in both post-RoPE PH settings, while DFRot gives the best standalone pre-RoPE PL results. Adding a per-head GHA refit to MassDiff improves three of the four MassDiff results, with Qwen3 pre-RoPE as the exception. A major practical advantage of GHA is its training-free, low-cost calibration. It calibrates substantially faster than DFRot and ButterflyQuant, as shown by the end-to-end timings in Appendix G.

5.1.2 Comparison with RotateKV

RotateKV applies channel reordering after the Hadamard transform and before quantization. Since it operates on the output side while GHA preconditions the input, a direct comparison is less informative than their combination. Inspired by RotateKV, we also test D_{out}, which applies calibrated sign flips without reordering, and $(DP)_{\text{out}}$, which further reorders channels globally across heads by descending output magnitude. Table 2 shows that GHA+RotateKV outperforms RotateKV in

Model	Method	Post-RoPE	Pre-RoPE
L-8B	RHT	83.29±0.66	15.53±1.71
	GHA (PH)	81.98±0.59	**37.66±2.60**
	GHA (PL)	**88.73±0.38**	**48.62±1.33**
	GHA+D_{out}	**89.44±0.34**	**69.44±0.59**
	GHA+$(DP)_{\text{out}}$	**92.54±0.23**	**74.48±1.14**
	RotateKV	**92.07±0.13**	**67.04±1.22**
	GHA+RotateKV	**92.43±0.29**	**72.59±1.39**
Q-4B	RHT	16.47±13.82	0.39±0.39
	GHA (PH)	**84.29±1.51**	**28.83±4.30**
	GHA (PL)	13.72±6.26	**17.07±13.78**
	GHA+D_{out}	**87.64±0.70**	**59.63±3.24**
	GHA+$(DP)_{\text{out}}$	**88.17±1.30**	**54.50±3.45**
	RotateKV	**85.67±0.38**	**50.39±1.20**
	GHA+RotateKV	**87.72±1.06**	**57.95±2.59**

Table 2: RULER-4096 accuracy under 2-bit asymmetric KV-cache quantization (Keys and Values: per-token; block size 128). L-8B and Q-4B denote LLaMA-3.1 8B and Qwen3 4B. All GHA variants use median calibration. Results are mean ± standard deviation over 10 calibration seeds. Results above RHT are bolded; the best in each column is also underlined.

Method	Post-PL	Post-PH	Pre-PL	Pre-PH
FP16	84.00			
RHT	73.03±1.57	72.84±0.87	60.25±1.81	58.71±2.56
GHA (m)	**73.94±1.17**	**73.37±1.09**	**62.32±1.75**	56.84±2.88
GHA (e)	72.11±2.67	71.87±2.16	**60.81±1.71**	**59.76±2.12**

Table 3: GSM8K accuracy under 2-bit asymmetric KV-cache quantization. GHA (m) and GHA (e) use median and extreme representatives, respectively. Results are mean ± std over 10 seeds. Results above RHT are bolded; the best in each column is also underlined.

all four settings, indicating that input-side GHA and post-Hadamard reordering are complementary. GHA+$(DP)_{\text{out}}$ performs best in three settings, but adding P_{out} hurts Qwen3 pre-RoPE.

5.1.3 Cross-Task Transfer to GSM8K

We directly reuse the preconditioners (D, P) calibrated in the RULER experiments and evaluate LLaMA-3.1 8B Instruct on GSM8K (Cobbe et al., 2021), without GSM8K-specific calibration. Table 3 shows that the better GHA representative outperforms RHT in all settings. Median is best in 3 settings, while extreme is best for pre-RoPE PH.

5.2 Hidden State Quantization

The representative-vector approximation is still useful for hidden states, but this setting is less favorable than KV-cache quantization. Hidden states have lower cosine similarity than Keys. They are also more sensitive to the calibration source. We therefore expect gains when calibration is well matched, but not in every model and task.

In this section, we use the same GPTQ (Frantar et al., 2023) calibration setting as QuaRot. We sample 128 sequences of length 2048 from the WikiText-2 training set. We then replace the hidden-state RHT in the QuaRot pipeline with the GHA-calibrated transform. All settings except the W16-prefixed controls use W4A4KV4 quantization. All other activation and weight quantization settings stay the same.

5.2.1 Language Modeling

For language modeling, we follow the QuaRot evaluation protocol across several models and quantization settings. On WikiText-2, we report perplexity and the KL divergence of logits. Perplexity measures errors on the correct next token. KL divergence measures distortion of the full output distribution. We evaluate RTN, GPTQ, and GPTQ-G. The suffix G means grouped quantization

Metric	Setting	Method	LLaMA-2 7B	LLaMA-2 13B	LLaMA-2 70B	LLaMA-3 8B	LLaMA-3 70B
PPL	RTN	QuaRot (RHT)	**8.235**±0.651	**6.122**±0.067	**4.123**±0.019	10.777±0.373	**45.409**±9.809
		QuaRot+GHA	8.295±0.386	6.160±0.052	4.141±0.023	**10.602**±0.396	47.565±7.237
	GPTQ	QuaRot (RHT)	6.116±0.015	5.381±0.011	3.793±0.009	8.198±0.052	**6.520**±0.126
		QuaRot+GHA	**6.100**±0.013	**5.367**±0.006	**3.781**±0.007	**8.135**±0.041	7.365±0.356
	GPTQ-G	QuaRot (RHT)	5.933±0.019	5.251±0.016	3.605±0.005	7.340±0.022	5.546±0.115
		QuaRot+GHA	**5.923**±0.013	**5.228**±0.008	**3.589**±0.003	**7.299**±0.021	**5.448**±0.121
	W16	QuaRot (RHT)	**5.905**±0.010	5.226±0.010	3.668±0.006	7.620±0.035	**5.463**±0.095
		QuaRot+GHA	5.908±0.011	**5.217**±0.004	**3.657**±0.005	**7.572**±0.017	5.906±0.193
	W16-G	QuaRot (RHT)	**5.769**±0.014	5.127±0.010	3.508±0.004	6.979±0.017	4.834±0.070
		QuaRot+GHA	5.770±0.012	**5.115**±0.007	**3.493**±0.004	**6.946**±0.012	**4.705**±0.090
KLD	RTN	QuaRot (RHT)	**0.4007**±0.0763	**0.2177**±0.0101	**0.2115**±0.0044	0.5570±0.0335	**2.7216**±0.2257
		QuaRot+GHA	0.4088±0.0476	0.2258±0.0077	0.2163±0.0039	**0.5413**±0.0354	2.7811±0.1561
	GPTQ	QuaRot (RHT)	0.1092±0.0011	0.0935±0.0016	0.1308±0.0008	0.2893±0.0057	**0.8220**±0.0186
		QuaRot+GHA	**0.1074**±0.0018	**0.0913**±0.0010	**0.1277**±0.0012	**0.2818**±0.0048	0.9420±0.0461
	GPTQ-G	QuaRot (RHT)	0.0798±0.0014	0.0705±0.0014	0.0807±0.0014	0.1811±0.0027	0.6644±0.0195
		QuaRot+GHA	**0.0776**±0.0013	**0.0666**±0.0009	**0.0761**±0.0004	**0.1746**±0.0022	**0.6473**±0.0229
	W16	QuaRot (RHT)	**0.0742**±0.0009	0.0647±0.0011	0.0970±0.0006	0.2159±0.0040	**0.6481**±0.0174
		QuaRot+GHA	0.0746±0.0015	**0.0636**±0.0004	**0.0944**±0.0007	**0.2098**±0.0019	0.7246±0.0309
	W16-G	QuaRot (RHT)	0.0517±0.0008	0.0470±0.0010	0.0535±0.0013	0.1292±0.0021	0.5285±0.0138
		QuaRot+GHA	**0.0515**±0.0006	**0.0441**±0.0006	**0.0497**±0.0005	**0.1241**±0.0012	**0.5024**±0.0192

Table 4: WikiText-2 language-modeling results for hidden-state quantization at sequence length 2048. PPL is perplexity and KLD is KL divergence of logits. Both methods report mean ± standard deviation across ten seeds. Lower is better, and better mean values within each model and setting are bolded.

with group size 128. Thus, GPTQ-G uses weight group size 128, and W16-G keeps weights in FP16 while using activation group size 128. We also include the no-weight controls W16 and W16-G, so we can isolate the effect of hidden-state quantization. For both the reproduced QuaRot baseline and QuaRot+GHA, we report mean ± standard deviation across ten seeds. Table 4 shows the results.

Table 4 shows that the gains are setting dependent. GHA reduces perplexity in 17 of 25 model-setting pairs and KL divergence in 18 of 25. They are most consistent under grouped quantization. GPTQ-G improves both metrics for all five models, while W16-G improves KL divergence for all five and perplexity for four. In contrast, RTN improves only for LLaMA-3 8B. The clearest remaining exception is LLaMA-3 70B under RTN, GPTQ, and W16, where GHA degrades both metrics. This concentration of gains in the grouped settings, together with the weaker 70B results, is consistent with the lower hidden-state cosine similarity of the 70B models shown in Appendix C.

5.2.2 Zero-Shot Tasks

We also evaluate hidden-state quantization on zero-shot tasks. This lets us test whether the same preconditioners still help beyond WikiText-2 perplexity. We report results for LLaMA-2 7B, LLaMA-2 13B, and LLaMA-3 8B. We keep the same GHA calibration protocol instead of adding task-specific calibration data. This keeps the setup consistent with the rest of the paper and tests robustness under one practical recipe.

As in language modeling, we do not expect gains on every model and task. Table 5 reports PIQA, HellaSwag, ARC-e, ARC-c, WinoGrande, LAMBADA, and their average. GHA improves the average accuracy in 13 of the 15 model-setting combinations. It improves all five settings for LLaMA-2 13B and four of five for both LLaMA-2 7B and LLaMA-3 8B. The two exceptions are W16-G, where LLaMA-2 7B is effectively tied and LLaMA-

Model	Setting	Method	PIQA	HellaSwag	ARC-e	ARC-c	WinoGrande	LAMBADA	Avg
7B	RTN	QuaRot (RHT)	73.60±1.02	65.31±1.73	61.55±2.33	36.69±1.52	61.47±1.66	57.96±3.64	59.43±1.59
		QuaRot+GHA	**73.78**±0.93	**65.52**±1.24	**62.02**±2.68	**37.47**±1.50	**61.55**±2.12	**58.38**±4.75	**59.79**±1.68
	GPTQ	QuaRot (RHT)	74.04±8.44	68.18±13.57	64.73±13.67	40.62±4.10	64.30±4.94	64.02±22.05	62.65±11.11
		QuaRot+GHA	**76.62**±0.45	**72.47**±0.41	**69.44**±1.22	**42.53**±0.73	**65.40**±0.89	**71.32**±0.65	**66.30**±0.30
	GPTQ-G	QuaRot (RHT)	77.18±0.46	73.22±0.21	**71.34**±0.87	42.66±0.96	67.02±0.94	**72.04**±0.53	67.24±0.36
		QuaRot+GHA	**77.73**±0.51	**73.52**±0.22	71.16±0.74	**43.34**±0.95	**67.05**±0.67	71.56±0.64	**67.39**±0.33
	W16	QuaRot (RHT)	77.24±0.41	73.85±0.21	**70.91**±0.68	42.66±0.54	66.40±1.03	71.77±0.41	67.14±0.28
		QuaRot+GHA	**77.38**±0.52	**74.00**±0.25	70.53±0.69	**43.04**±0.62	**66.93**±0.93	**72.18**±0.49	**67.34**±0.38
	W16-G	QuaRot (RHT)	**77.79**±0.40	74.38±0.18	**72.48**±0.40	**44.47**±0.41	67.62±0.50	**72.60**±0.38	**68.23**±0.17
		QuaRot+GHA	77.76±0.27	**74.67**±0.18	72.19±0.44	44.17±0.76	**68.18**±1.14	72.37±0.42	68.22±0.33
13B	RTN	QuaRot (RHT)	76.90±0.63	72.91±1.02	71.08±1.47	**44.18**±0.98	67.50±1.10	**70.34**±1.13	67.15±0.49
		QuaRot+GHA	**77.42**±0.48	**73.67**±0.77	**71.22**±1.96	43.77±1.61	**67.97**±0.72	69.45±1.09	**67.25**±0.77
	GPTQ	QuaRot (RHT)	78.23±0.55	76.17±0.41	**73.56**±1.10	**46.49**±1.37	69.66±0.85	74.03±0.58	69.69±0.44
		QuaRot+GHA	**78.58**±0.33	**76.99**±0.29	73.08±1.30	46.01±0.62	**70.13**±0.53	**74.25**±0.58	**69.84**±0.29
	GPTQ-G	QuaRot (RHT)	78.93±0.67	76.93±0.27	**75.06**±0.79	**47.69**±0.97	70.68±0.87	75.18±0.32	70.74±0.36
		QuaRot+GHA	**79.39**±0.49	**77.50**±0.18	74.80±0.99	47.31±0.91	**71.35**±0.47	**75.40**±0.29	**70.96**±0.24
	W16	QuaRot (RHT)	79.07±0.50	77.37±0.20	**74.65**±0.46	**47.44**±0.83	70.39±0.80	**74.87**±0.51	70.63±0.30
		QuaRot+GHA	**79.30**±0.31	**78.01**±0.23	74.57±0.68	47.12±0.73	**70.92**±1.10	74.79±0.28	**70.79**±0.22
	W16-G	QuaRot (RHT)	**79.68**±0.24	77.97±0.22	**75.91**±0.61	**48.24**±0.78	70.70±0.78	75.37±0.47	71.31±0.23
		QuaRot+GHA	79.66±0.34	**78.49**±0.12	75.47±0.31	47.93±0.74	**71.81**±0.68	**75.44**±0.46	**71.47**±0.23
8B	RTN	QuaRot (RHT)	71.97±0.95	67.30±1.17	62.87±2.48	38.31±2.45	62.91±1.35	57.91±2.74	60.21±1.09
		QuaRot+GHA	**72.56**±1.04	**68.58**±0.97	**63.91**±1.71	**38.96**±1.33	**63.97**±1.27	**59.47**±1.96	**61.24**±0.79
	GPTQ	QuaRot (RHT)	**75.69**±0.77	73.09±0.68	69.89±1.66	**44.36**±1.46	66.74±0.89	67.42±0.71	66.20±0.64
		QuaRot+GHA	75.38±0.47	**73.28**±0.42	**69.90**±0.95	44.16±0.71	**67.30**±1.12	**68.00**±0.86	**66.34**±0.25
	GPTQ-G	QuaRot (RHT)	**78.04**±0.37	**75.73**±0.29	**74.89**±1.15	48.03±0.74	69.42±0.85	**71.27**±0.96	69.56±0.43
		QuaRot+GHA	77.68±0.43	75.55±0.38	74.33±1.45	**48.88**±0.89	**70.62**±0.98	70.93±0.75	**69.67**±0.46
	W16	QuaRot (RHT)	**76.98**±0.46	75.01±0.37	72.78±1.13	46.60±0.94	67.97±0.87	69.46±0.91	68.13±0.42
		QuaRot+GHA	76.65±0.54	**75.24**±0.18	**73.39**±0.91	**47.41**±0.83	**68.30**±0.98	**69.60**±0.66	**68.43**±0.28
	W16-G	QuaRot (RHT)	**78.72**±0.47	**77.03**±0.31	**76.30**±0.94	**50.83**±0.99	70.93±0.72	**72.41**±0.67	**71.04**±0.27
		QuaRot+GHA	78.28±0.51	**77.03**±0.29	75.22±1.38	50.53±1.20	**71.06**±0.92	72.12±0.51	70.71±0.44

Table 5: Zero-shot accuracy (%) for hidden-state quantization on PIQA, HellaSwag, ARC-e, ARC-c, WinoGrande, and LAMBADA. Both methods report mean ± standard deviation across ten seeds. Higher is better, and better mean values within each model and setting are bolded.

device	dtype	64	128	256	512	1024	2048	4096	8192
GPU	BF16	**1.14** (1.10)	**1.11** (1.10)	**1.14** (1.06)	**1.26** (1.11)	**1.51** (1.11)	**2.19** (1.11)	**4.46** (1.36)	**8.98** (2.32)
	FP16	**1.16** (1.12)	**1.33** (1.12)	**1.16** (1.08)	**1.33** (1.12)	**1.60** (1.13)	**2.20** (1.10)	**4.42** (1.31)	**8.68** (2.26)
	FP32	**1.12** (1.10)	**1.11** (1.08)	**1.28** (1.08)	**1.56** (1.09)	**2.52** (1.03)	**4.63** (1.16)	**9.26** (1.52)	**17.98** (2.27)
CPU	BF16	**2.45** (4.19)	**2.59** (4.44)	**2.86** (3.43)	**3.04** (1.85)	**2.42** (2.76)	**5.35** (4.07)	**8.05** (7.53)	**45.29** (8.17)
	FP16	**2.88** (4.79)	**3.26** (4.50)	**4.75** (3.52)	**6.44** (1.80)	**11.04** (2.42)	**24.40** (3.65)	**64.42** (5.71)	**122.45** (4.11)
	FP32	0.89 (3.76)	**1.10** (2.83)	**2.61** (1.92)	**5.54** (1.14)	**16.67** (1.20)	**36.79** (1.96)	**77.95** (2.96)	**231.01** (3.64)

Table 6: Speedup of our $HDPx$ implementation over a dense `matmul` baseline. The memory-copy ratio is shown in parentheses. Speedups above 1 are bolded.

3 8B favors QuaRot. The large LLaMA-2 7B GPTQ difference is partly driven by one QuaRot seed that collapses across all six tasks, so it indicates sensitivity to the random rotation rather than a typical per-seed gain. Together with the language-modeling results, these numbers support the view from Section 4: hidden-state gains are setting dependent, but are most consistent under grouped quantization, and GHA can avoid severe random-seed failures in some settings. Appendix E reports a separate W4A4KV4 evaluation implemented in the SpinQuant codebase.

5.3 CUDA & CPU Benchmarks

We benchmark the runtime of signed permutation preconditioning with the fused kernel from Section 4.3.2. We test both CUDA and CPU. We report speedup over a dense `matmul` baseline. We also show the ratio to memory copy in parentheses as a reference. We test bf16, fp16, and fp32. The GPU is an NVIDIA RTX 3090. The CPU is an Intel Xeon w7-3465X workstation processor. We test Sylvester Hadamard sizes from 64 to 8192. Table 6 shows the results.

6 Conclusion

We introduced Greedy Hadamard Alignment (GHA), a simple data-aware alternative to random sign flips before the Hadamard transform in LLM quantization. GHA builds a signed-permutation preconditioner from a pooled representative activation, combines a greedy flat-target search with an analytically optimal alignment step for any fixed target, and can be implemented with almost no extra inference overhead beyond a plain Hadamard transform. Unlike randomized Hadamard preprocessing, GHA uses calibration data to optimize the preconditioner for the target activations. Across tasks and models, this is enough to improve over randomized Hadamard baselines overall.

Our main empirical insight is that the value of a shared Hadamard preconditioner depends strongly on activation geometry. GHA works best when activations are directionally aligned and stable across calibration sources, which makes a shared representative vector more reliable. This explains why the gains are strongest in KV-cache quantization, especially for Keys, and more moderate and more calibration-dependent for hidden states. For hidden-state quantization, the gains are also more consistent on KL divergence than on perplexity, which suggests that GHA more reliably preserves the full output distribution than the single next-token probability. Cosine similarity and cross-source stability are therefore useful practical indicators of when shared preconditioning is likely to transfer.

Limitations

Our study has several limitations. The best choice of representative vector among the mean, median, and extreme statistics is model dependent, and we do not yet have a principled way to predict which representative will work best for a new model. The rearrangement inequality gives an analytical solution only for the alignment step once the flat target z is fixed. Searching for z remains computationally expensive, especially in high dimensions. Our greedy search is heuristic and does not guarantee a global optimum of the surrogate objective, while restricting the search to a window may miss better candidates. Better algorithms for this step are still lacking. Finally, our evaluation primarily covers LLaMA and Qwen models under 2-bit KV-cache and W4A4KV4 hidden-state quantization. Broader validation across more architectures, bitwidths, tasks, and deployment settings is still needed.

References

Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 557–563, New York, NY, USA. Association for Computing Machinery.

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

Haim Avron, Petar Maymounkov, and Sivan Toledo. 2010. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and overcoming the challenges of efficient transformer quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7947–7969, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems*, volume 36, pages 4396–4429. Curran Associates, Inc.

Euntae Choi, Sumin Song, Woosang Lim, and Sungjoo Yoo. 2025. Grouped sequency-arranged rotation: Optimizing rotation transformation for quantization for free. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 165–172, Vienna, Austria. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Tri Dao. 2023. Fast hadamard transform in cuda, with a pytorch interface. `https://github.com/Dao-AILab/fast-hadamard-transform`. Accessed: 2026-05-14.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, volume 35, pages 30318–30332. Curran Associates, Inc.

Fino and Algazi. 1976. Unified matrix treatment of the fast walsh-hadamard transform. *IEEE Transactions on Computers*, C-25(11):1142–1146.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The pile: An 800gb dataset of diverse text for language modeling. *Preprint*, arXiv:2101.00027.

Ruihao Gong, Yifu Ding, Zining Wang, Chengtao Lv, Xingyu Zheng, Jinyang Du, Yang Yong, Shiqiao Gu, Haotong Qin, Jinyang Guo, Dahua Lin, Michele Magno, and Xianglong Liu. 2025. A survey of low-bit large language models: Basics, systems, and algorithms. *Neural Networks*, 192:107856.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

N. Halko, P. G. Martinsson, and J. A. Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288.

Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. 1952. *Inequalities*. Cambridge University Press, Cambridge.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. 2024. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. In *Advances in Neural Information Processing Systems*, volume 37, pages 87766–87800. Curran Associates, Inc.

Kai Liu, Qian Zheng, Kaiwen Tao, Zhiteng Li, Haotong Qin, Wenbo Li, Yong Guo, Xianglong Liu, Linghe Kong, Guihai Chen, Yulun Zhang, and Xiaokang Yang. 2025a. Low-bit model quantization for deep neural networks: A survey. *Preprint*, arXiv:2505.05530.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2025b. Spinquant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*.

Sai Sanjeet, Ian Colbert, Pablo Monteagudo-Lago, Giuseppe Franco, Yaman Umuroglu, and Nicholas J. Fraser. 2026. Pushing the limits of block rotations in post-training quantization. In *Forty-third International Conference on Machine Learning*.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Zunhai Su, Hanyu Wei, Zhe Chen, Wang Shen, Linge Li, Huangqi Yu, and Kehong Yuan. 2025. Rotatekv: Accurate and robust 2-bit kv cache quantization for llms via outlier-aware adaptive rotations. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 6200–6208. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. 2024. Massive activations in large language models. In *First Conference on Language Modeling*.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, JiaxinHu, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, Wulong Liu, and Jun Yao. 2025. Flatquant: Flatness matters for LLM quantization. In *Forty-second International Conference on Machine Learning*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. QuIP#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 48630–48656. PMLR.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. 2023. Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1648–1665, Singapore. Association for Computational Linguistics.

Jingyang Xiang and Sai Qian Zhang. 2025. DFRot: Achieving outlier-free and massive activation-free for rotated LLMs with refined rotation. In *Second Conference on Language Modeling*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Bingxin Xu, Zhen Dong, Oussama Elachqar, and Yuzhang Shang. 2026. Butterflyquant: Ultra-low-bit llm quantization through learnable orthogonal butterfly transforms. *Preprint*, arXiv:2509.09679.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

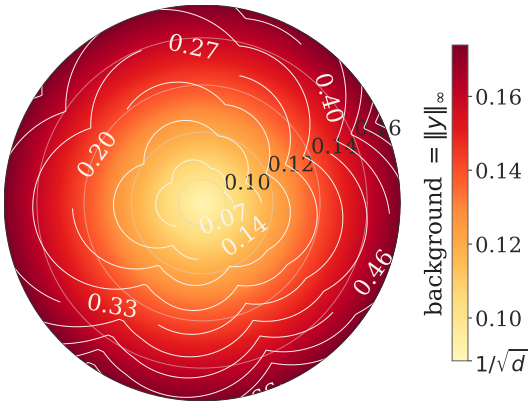


Figure 5: Unit-sphere visualization of the flat-target surrogate. Contours show $\|y - z\|_2$, while the polar radius and background color show $\|y\|_\infty$.

Layer	GHA actual $\|y\|_\infty$	RHT mean $\|y\|_\infty$	GHA bound
0	0.00269	0.00779 ± 0.000606	0.0196
5	0.0653	0.164 ± 0.00961	0.454
10	0.105	0.305 ± 0.0219	0.651
15	0.170	0.512 ± 0.0416	1.21
20	0.336	1.07 ± 0.0784	1.09
25	0.685	2.13 ± 0.134	2.23
30	1.16	3.32 ± 0.212	4.81

Table 7: Empirical effect of the surrogate on selected LLaMA-2-7B hidden-state layers. The GHA bound is valid but can be loose, while the optimized actual peak remains substantially below the RHT mean.

A Surrogate Objective

The analytical inequality in Section 3.2 gives a valid upper bound on $\|y\|_\infty$, but the bound is not necessarily tight on real activations. Its main role in GHA is to provide a tractable surrogate objective for optimizing the flat target and the signed permutation. Figure 5 visualizes this surrogate on the unit sphere with a fixed flat target $z \in \mathcal{F}_1$. The contour lines are level sets of $\|y - z\|_2$, while the polar radius and background color encode $\|y\|_\infty$. The figure illustrates that moving toward the flat target decreases both the surrogate distance and the transformed vector's ℓ_∞ value.

We further evaluate whether the optimized surrogate is useful in practice. We sample hidden states from LLaMA-2-7B on WikiText-2 and compare GHA with RHT across layers. For GHA, we report the actual peak $\|y\|_\infty$ and the analytical bound $\rho/\sqrt{d} + \|y - z^*\|_2$. For RHT, we report the mean and standard deviation of the actual peak. Table 7 shows seven representative layers.

The bound is 3.2–7.3× the measured GHA peak in these layers, confirming that it should not be viewed as a tight predictor. However, the optimized GHA peak is 2.5–3.2× lower than the RHT mean and is below the RHT mean minus one standard deviation in every layer shown. Across all 32 layers, the optimized surrogate distance $\|y - z^*\|_2$ is strongly correlated with the actual peak $\|y\|_\infty$, as shown in Table 8.

Statistic	Value
Pearson r	0.920
Spearman ρ	0.935
Pearson r (log–log)	0.949

Table 8: Correlation between the optimized surrogate distance $\|y - z^*\|_2$ and the actual peak $\|y\|_\infty$ across all 32 LLaMA-2-7B hidden-state layers.

Together with the full-activation absmax and INT4 MSE results in Figure 7, these results show that the surrogate is an effective optimization target, without requiring it to be a tight bound or a global approximation to the original objective. The fixed-target alignment is exact for the surrogate, while the greedy target search remains heuristic.

B Efficient Computation of Off-Diagonal Cosine Similarity

One of the main reasons we choose off-diagonal cosine similarity as a metric to measure directional structure is that it can be computed exactly without forming the full pairwise cosine matrix. Given token activations $x_1, \ldots, x_T \in \mathbb{R}^D$, let

$$n_i = \frac{x_i}{\|x_i\|_2}, \qquad \cos_{\text{mean}} = \frac{\sum_{i \neq j} \langle n_i, n_j \rangle}{T(T-1)}.$$

If we define $s = \sum_{i=1}^{T} n_i$, then

$$\|s\|_2^2 = \sum_{i,j} \langle n_i, n_j \rangle = T + \sum_{i \neq j} \langle n_i, n_j \rangle,$$

because $\|n_i\|_2 = 1$ for every i. Therefore,

$$\cos_{\text{mean}} = \frac{\|s\|_2^2 - T}{T(T-1)}.$$

This identity turns the computation into a streaming reduction. We only need to normalize each activation, add it to s, and keep track of T. The final value then uses one D-dimensional dot product. The result is exact, uses $O(TD)$ total work and $O(D)$ memory, and avoids the $O(T^2)$ memory cost of explicitly forming the pairwise cosine matrix.

C Hidden-State Cosine Similarity Across Models

Figure 6 extends the hidden-state cosine-similarity analysis from LLaMA-2 7B to all five evaluated model scales. Two patterns are consistent across models. First, the mean pairwise cosine similarity remains positive throughout the network, so the representative-vector approximation is still meaningful beyond a single model. Second, the larger 70B models have visibly lower average similarity than the 7B/8B/13B models, especially in deeper layers, which is consistent with the weaker and less stable hidden-state gains observed in Table 4. Among the smaller models, LLaMA-2 7B and 13B are broadly similar, while LLaMA-3 8B shows a flatter profile with a clearer rise in the final layers.

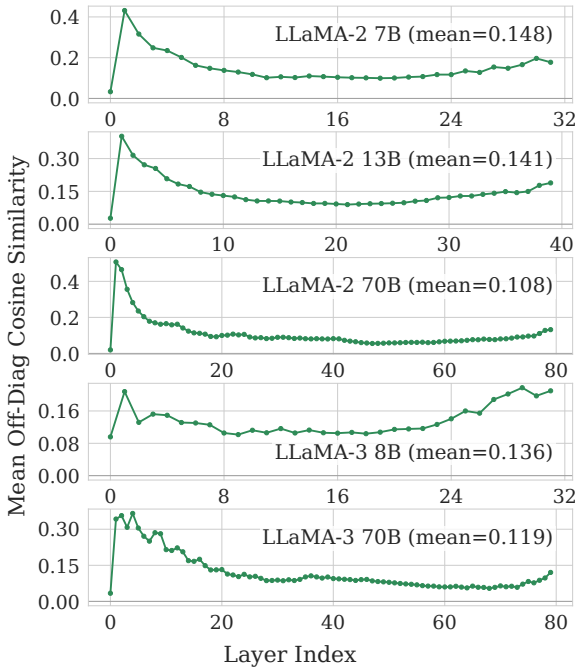


Figure 6: Layer-wise mean pairwise cosine similarity of hidden states across five evaluated models. Each panel reports the mean off-diagonal cosine similarity over all token pairs at each layer.

D Full-Activation Behavior Across Layers

As shown in Figure 7, GHA usually lies on the favorable side of the RHT trial distribution, but its separation on full token activations is smaller than on the calibration-derived representative. Optimizing one representative vector therefore captures useful structure without completely determining full-activation behavior.

The cosine-similarity annotations help explain why the size and stability of the gains vary by layer. Layers with larger hidden-state cosine similarity, such as the stronger-alignment early layer and the later layers where similarity rises again, tend to show clearer separation between GHA and the bulk of the RHT trials, especially on token-level absmax and often on quantization MSE as well. By contrast, layers with lower cosine similarity show much narrower gaps, which is consistent with the idea that a single representative vector becomes less reliable when token directions are less aligned.

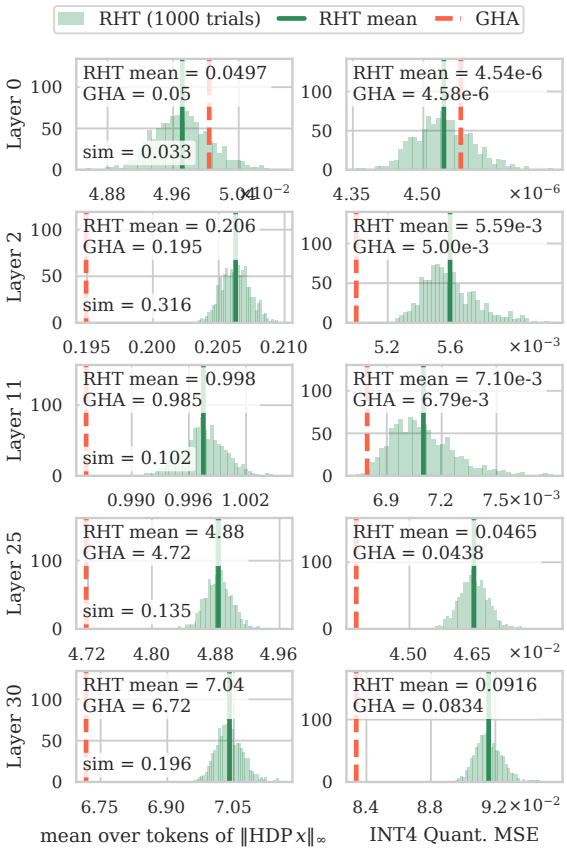


Figure 7: GHA versus 1000 RHT trials across selected LLaMA-2 7B hidden-state layers. Left: mean-over-tokens $\|HDPx\|_\infty$. Right: per-token INT4 symmetric quantization MSE. Each row also reports the layer-wise mean cosine similarity.

E Comparison with SpinQuant

This experiment is separate from the QuaRot-based evaluation in Section 5.2. We integrate GHA directly into the SpinQuant codebase and follow its W4A4KV4 pipeline on three smaller models.

SpinQuant achieves the best source-aligned perplexity for all three models, consistent with the greater expressiveness of its learned dense rotation. On zero-shot tasks, GHA has a small numerical

Model	Method	ARC-c	ARC-e	BoolQ	HellaS.	OBQA	PIQA	SIQA	WinoG.	Avg.
LLaMA-2 13B	RHT	46.45±1.19	74.52±0.72	**78.52±0.80**	76.35±0.28	43.32±0.63	78.92±0.60	45.56±0.50	69.39±0.81	64.13±0.34
	SpinQuant	**47.18**	**75.04**	77.28	**77.46**	43.40	**79.22**	45.91	68.19	64.21
	GHA	46.42	74.24	78.23	76.71	**44.20**	78.78	**46.67**	**71.03**	**64.54**
LLaMA-2 7B	RHT	41.60±0.94	69.83±1.20	**73.43±1.13**	72.70±0.33	40.32±0.96	76.81±0.60	43.60±0.43	65.60±0.78	60.49±0.27
	SpinQuant	**43.43**	**71.46**	73.36	72.90	38.80	76.06	44.93	63.22	60.52
	GHA	40.87	70.29	73.06	**73.20**	**41.40**	**76.99**	**44.98**	**66.93**	**60.96**
LLaMA-3 8B	RHT	45.60±1.06	71.36±1.66	74.65±2.76	74.12±0.65	41.36±0.88	76.79±0.34	44.91±0.75	68.07±1.11	62.11±0.44
	SpinQuant	**47.78**	74.96	75.81	**74.86**	41.80	77.09	44.63	68.51	63.18
	GHA	47.35	**75.04**	**76.64**	74.59	**42.40**	**77.37**	**45.19**	**69.93**	**63.56**

Table 9: Zero-shot accuracy (%) under W4A4KV4 quantization using the SpinQuant codebase. HellaS. and WinoG. denote HellaSwag and WinoGrande. Higher is better, and the best result for each model and task is bolded. RHT reports mean ± standard deviation over 10 seeds; SpinQuant and median-calibrated GHA are single runs.

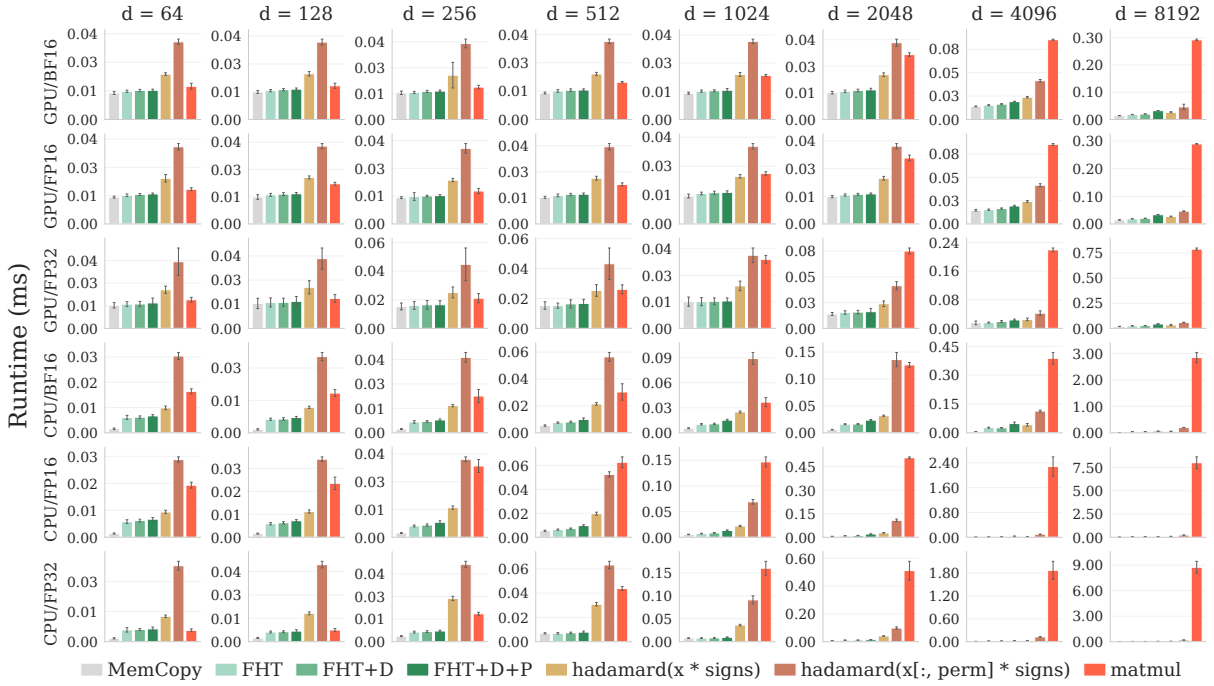


Figure 8: Detailed GPU and CPU benchmark runtimes for all benchmarked implementations. Each row corresponds to one device and data type, and each column corresponds to one Hadamard dimension. Bars show mean runtime in milliseconds, with error bars denoting the standard deviation across repeated runs. The methods are `MemCopy`, `FHT`, `FHT+D`, `FHT+D+P`, `hadamard(x * signs)`, `hadamard(x[:, perm] * signs)`, and `matmul`.

Model	RHT	SpinQuant	GHA
LLaMA-2 13B	5.38±0.01	**5.24**	5.35
LLaMA-2 7B	6.13±0.02	**5.94**	6.04
LLaMA-3 8B	7.86±0.05	**7.35**	7.75

Table 10: WikiText-2 perplexity under W4A4KV4 quantization using the SpinQuant codebase. Lower is better, and the best result for each model is bolded. RHT reports mean ± standard deviation over 10 seeds; SpinQuant and median-calibrated GHA are single runs.

advantage in average accuracy for all three models. However, SpinQuant and GHA are single-run results, so these differences do not establish statistical significance.

F Detailed CUDA and CPU Benchmark Breakdown

Figure 8 provides the full per-method runtime breakdown underlying the compact comparison in Table 6.

The appendix figure makes two qualitative patterns explicit. First, the fused `FHT+D+P` implementation consistently improves over the unfused permutation/sign baselines, especially on CPU and at larger transform sizes where materializing the preprocessing steps is increasingly expensive. Second, the gap to the dense `matmul` baseline widens quickly with dimension, which complements the summarized speedup ratios in Table 6.

Method	LLaMA-3.1 8B PH	LLaMA-3.1 8B PL	Qwen3 4B PH	Qwen3 4B PL
GHA	**341s** (1.00×)	**348s** (1.02×)	**366s** (1.00×)	**367s** (1.00×)
DFRot	1001s (2.94×)	5680s (16.66×)	527s (1.44×)	6350s (17.35×)
ButterflyQuant	4663s (13.67×)	1662s (4.87×)	5472s (14.95×)	1858s (5.08×)

Table 11: End-to-end calibration time on an Intel Xeon Platinum 8592+ and one NVIDIA RTX 5880. Parentheses report runtime relative to GHA-PH for each model. Lower is better.

G End-to-End Calibration Cost

GHA uses calibration data but is training-free and requires no gradient-based optimization. We compare its end-to-end calibration time with DFRot and ButterflyQuant on the RULER-4096 setup from Section 5. DFRot uses the official implementation with 100 iterations, while ButterflyQuant is reproduced with the paper's setting of 600 optimization steps.

Table 11 shows that GHA completes calibration in about six minutes for every setting. In contrast, DFRot and ButterflyQuant require 1.44–$17.35\times$ the GHA-PH time. This substantial reduction in end-to-end calibration cost is a major practical advantage of GHA, especially because our implementation uses unoptimized Python.