

HARP: Hadamard-Preconditioned Adaptive Rotation Processor for Extreme LLM Quantization

Artur Zagitov^{1,2,3} Gleb Molodtsov^{1,2} Aleksandr Beznosikov^{1,2,3}
¹MIRAI ²BRAIn Lab ³Innopolis University

Abstract

Post-training quantization (PTQ) is essential for deploying LLMs under memory and bandwidth constraints. However, extreme low-bit quantization remains highly sensitive to activation outliers and anisotropic weight curvature. Existing incoherence-based PTQ methods mitigate this issue with fixed randomized Hadamard transforms (RHTs), which improve quantization robustness but cannot adapt the rotated basis to the layer, calibration distribution, or quantizer. We introduce HARP (**H**adamard-**p**reconditioned **A**daptive **R**otation **P**rocessor), a learnable structured two-sided orthogonal processor that replaces fixed Hadamard mixing while preserving exact full-precision equivalence. HARP represents each rotation as a product of sparse butterfly-like block-orthogonal stages, supports non-power-of-two dimensions through Mixed-Radix schedules, and initializes to the RHT processor up to a fixed permutation. Fitted only on calibration data, HARP adapts the quantization basis to each layer and backend. Across 2–4-bit settings on Llama models from 1B to 70B, HARP consistently improves perplexity and yields its clearest zero-shot gains at 2 bits; a 2-bit Qwen3-8B experiment shows the same transfer beyond the Llama family. HARP also preserves deployment efficiency: on Llama 2 7B at 2 bits, it reaches 128 tok/s, retaining 90% of RHT throughput (142 tok/s) and running approximately 2.1× faster than FP16 (61 tok/s).

1 Introduction

Large language models (LLMs) have become central to modern NLP, yet their scale makes deployment increasingly constrained by memory bandwidth. During inference, repeated weight and activation transfers through the memory hierarchy dominate latency and serving cost. Quantization targets this bottleneck by compressing weights and activations to fewer bits.

Post-training quantization (PTQ) is especially attractive because it operates on a fixed pretrained model, uses only a small calibration set, and avoids the expense of full training (Frantar et al., 2023; Xiao et al., 2023; Lin et al., 2024; Shao et al., 2024). Successful PTQ pipelines do more than choose a rounding rule. They are defined by several coupled design choices, including calibration, representation preprocessing, the reconstruction objective, and the code family used for compression.

In the extreme low-bit regime, outlier handling becomes central. At such bitwidths, small reconstruction errors are amplified across many layers, and heavy-tailed statistics make the quantization problem substantially harder. As a result, the coordinates in which quantization is performed can fundamentally change the difficulty of the problem.

Incoherence processing exploits this degree of freedom by applying structured orthogonal changes of basis before quantization. The full-precision layer remains unchanged, but weight mass and curvature-sensitive directions are spread across coordinates, making them less aligned with a few outlier axes (Chee et al., 2023). In practice, randomized Hadamard transforms (RHTs) have become the default preprocessing choice because they are exactly orthogonal, fast, and admit simple $O(d \log d)$ kernels (Tseng et al., 2024a,b).

Although the standard Hadamard/RHT processor provides a strong generic mixing basis, it does not adapt to the layer, the calibration distribution, or the block structure of the downstream quantizer. Meanwhile, recent evidence shows that the choice of rotation is not incidental: end-to-end rotation methods reveal substantial variation across rotations and motivate learning them from data (Ashkboos et al., 2024b; Liu et al., 2025), learnable butterfly rotations adapt structured orthogonal mixing to calibration data (Xu et al., 2025), and closed-form data-aware transforms can improve over fixed Hadamard mixing under specific quan-

tizer assumptions (Chen et al., 2026). These alternatives, however, often target different operating points, relax the drop-in structure of Hadamard-based PTQ, or introduce rotations that are too costly to use. This leaves a gap: the preprocessing module should be learnable from calibration data, remain an exact orthogonal change of basis, and preserve the Hadamard-like efficiency that makes incoherence processing practical.

We present HARP (Hadamard-preconditioned Adaptive Rotation Processor), a learnable two-sided orthogonal processor for extreme low-bit PTQ. HARP satisfies three requirements simultaneously: (i) adaptivity, by fitting rotations to each layer and quantizer backend; (ii) faithfulness, by remaining an exact change of basis that preserves the full-precision model; and (iii) efficiency, by using staged butterfly-like block-orthogonal factors (Dao et al., 2019) instead of dense rotations. Crucially, HARP is drop-in compatible with existing Hadamard-based PTQ pipelines: at initialization, it recovers the RHT processor up to a fixed permutation convention, after which calibration learns a structured orthogonal refinement around this strong baseline.

Contributions.

Our main contributions are:

- We introduce HARP, a learnable two-sided orthogonal incoherence processor for PTQ that is fit using only calibration data and compatible with both scalar and vector-quantized backends.
- We design HARP as a drop-in replacement for fixed RHT/Hadamard preprocessing: at initialization, it recovers the randomized Hadamard processor up to a fixed permutation, and calibration learns a structured orthogonal refinement.
- We demonstrate consistent quality gains over fixed RHT across 2–4-bit settings on Llama models from 1B to 70B, transfer the same method to Qwen3-8B without backend retuning, and retain most of the RHT throughput advantage through hardware-aware kernels. Our code is publicly available¹.

We next formalize the setting (Section 2), introduce the HARP processor (Section 3), and evaluate it (Section 4). A detailed literature review is provided in Appendix L.

2 Setup

Layerwise PTQ objective. Modern PTQ pipelines typically cast quantization as a layerwise reconstruction problem.

We consider PTQ of a linear layer with $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and calibration inputs $x \in \mathbb{R}^{d_{\text{in}}}$. As in Hessian-aware PTQ, we use the empirical second moment as a curvature proxy:

$$H := \mathbb{E}[xx^\top] \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}. \quad (1)$$

Given that, PTQ seeks an approximation $\widehat{W}$ constrained to a code family $\mathcal{Q}$ that minimizes a proxy for the output error. Scalar quantization rounds weights independently, which is simple and kernel-friendly but can be inefficient at very low bitwidths. Vector quantization (VQ) instead quantizes small blocks jointly using a structured code family. By shaping quantization noise in a higher-dimensional space, VQ can significantly reduce distortion at a fixed bitrate, but it introduces more complex encoding/decoding and raises hardware considerations for fast inference. A common choice is a second-moment or Hessian-weighted quadratic objective, which captures how errors along different input directions affect the layer output.

$$\mathcal{L}(W, \widehat{W}) := \text{Tr}\left((W - \widehat{W})H(W - \widehat{W})^\top\right). \quad (2)$$

This perspective yields scalable algorithms because it allows optimization independently per layer (and often per block within a layer), using tractable curvature approximations derived from calibration activations (Frantar et al., 2023; Hassibi and Stork, 1992; Hassibi et al., 1993; Wu et al., 2024; Frantar and Alistarh, 2022).

Incoherence processing for PTQ. Extreme low-bit PTQ is often dominated by anisotropy and outliers: a small subset of coordinates can carry disproportionate mass in W and in the curvature proxy H . This harms blockwise quantizers because their scales and code assignments must cover rare directions, reducing effective resolution for the majority of weights.

Incoherence processing addresses this by applying an orthogonal change of basis so that energy is less concentrated in a few coordinates. For a linear layer, we choose two orthogonal processors $U \in O(d_{\text{out}})$ and $V \in O(d_{\text{in}})$, rotate the weight and curvature proxy as

$$\widetilde{W} := U^\top W V, \quad \widetilde{H} := V^\top H V.$$

The backend then quantizes the rotated weight, producing $\widehat{\widetilde{W}}$, and the quantized layer is mapped

¹<https://github.com/brain-lab-research/HARP>

back to the original basis as $\widehat{W} := U\widehat{W}V^\top$. Because U and V are orthogonal, cyclicity of trace gives the invariant reconstruction objective

$$\mathcal{L}(W, \widehat{W}) = \text{Tr}\left((\widetilde{W} - \widehat{W}) \widetilde{H} (\widetilde{W} - \widehat{W})^\top\right). \quad (3)$$

Thus, the full-precision layer and the target reconstruction objective are unchanged; only the basis exposed to the finite-code quantizer changes.

This is particularly important because low-bit quantizers are generally not invariant to rotations. Scalar quantizers are axis-aligned, and blockwise VQ backends operate on fixed contiguous groups. If large weights or curvature-sensitive directions align with a few coordinates, scales and code assignments must cover rare outliers, reducing resolution for the majority of values. Randomized Hadamard transforms (RHTs) reduce this concentration generically. HARP keeps the same orthogonal reparameterization principle, but learns a structured rotation that adapts to the layer and downstream blockwise quantizer.

Following QuIP, one can measure generic concentration using weight incoherence

$$\mu_W(A) := \sqrt{mn} \frac{\|A\|_\infty}{\|A\|_F}, \quad A \in \mathbb{R}^{m \times n}, \quad (4)$$

and Hessian incoherence

$$\mu_H(H) := \sqrt{n} \|Q\|_\infty, \quad H = Q\Lambda Q^\top. \quad (5)$$

Lower values indicate that weights or curvature-sensitive directions are less concentrated in a few coordinate axes. Classical incoherence scores are useful but not sufficient for modern blockwise quantizers. A rotation can reduce outliers while still misaligning curvature with the quantizer’s blocks, or conversely increase a generic Hessian incoherence score while producing lower backend-specific reconstruction error. For this reason, we evaluate both classical metrics and quantizer-aligned diagnostics in Appendix F: pre-quantization weight incoherence, quantized-weight incoherence, off-block Hessian energy, diagonal Hessian-weighted distortion, and classical Hessian incoherence. The diagnostics show that HARP does not simply optimize every generic incoherence score. Rather, it learns a basis that is more favorable for the deployed blockwise quantizer.

Fixed RHTs are a popular choice because they are orthogonal and fast. However, they ignore per-layer statistics and quantizer structure. HARP

keeps Hadamard-like efficiency and exact orthogonality while learning a structured refinement from calibration data.

3 HARP processors: structured orthogonal rotations

A dense learned rotation would be too expensive to store or apply at LLM hidden dimensions. HARP therefore parameterizes each processor as a product of sparse stride stages, following the same high-level structure as fast Hadamard and butterfly transforms. Fix a transformed dimension d (either d_{in} or d_{out}) and choose a Mixed-Radix schedule

$$\mathbf{b} = (b_0, \dots, b_{m-1}), \quad b_t \geq 2, \quad \prod_{t=0}^{m-1} b_t = d. \quad (6)$$

The schedule determines which coordinates are mixed at each stage. In all experiments we use a greedy schedule with preferred radix 8 (unless otherwise noted). Appendix K.2 gives the construction and Appendix G.2 ablates the radix choice.

At stage t , define the stride $s_t := \prod_{j < t} b_j$ and the number of blocks $D_t := d/b_t$. Conceptually, a perfect-shuffle permutation P_t groups the b_t coordinates mixed by stage t into contiguous blocks. The stage operator is

$$\begin{aligned} S_t(\Theta_t) &:= P_t^\top \mathcal{B}_t(\Theta_t) P_t, \\ \mathcal{B}_t(\Theta_t) &:= \text{BlkDiag}(B_{t,1}, \dots, B_{t,D_t}), \end{aligned} \quad (7)$$

where each $B_{t,c} = B_{t,c}(\theta_{t,c}) \in \mathbb{R}^{b_t \times b_t}$ is orthogonal. The one-pass HARP transform is

$$T(\Theta) := S_{m-1}(\Theta_{m-1}) \cdots S_0(\Theta_0). \quad (8)$$

Since each stage is orthogonal, $T(\Theta)$ is orthogonal. Multiple passes can be used by composing independent copies of the same schedule, although we use one pass by default.

Although Eq. (7) is written with a permutation, no explicit gather is used. The input is reshaped to expose the stride- s_t groups, transposed so that each length- b_t group is contiguous, multiplied by the corresponding block kernel, and reshaped back. This preserves the staged execution pattern that makes Hadamard preprocessing efficient. Appendix K.1 gives an index-view example of the stride layout.

HARP uses independent processors for the input and output sides, giving the rotations V and U used in Eq. (3). As in RHT preprocessing, we also include diagonal Rademacher sign flips. In our

QuIP# integration, these signs are already part of the pipeline. HARP reuses them and replaces only the fixed Hadamard mixing component.

3.1 Hadamard-preconditioned block kernels and initialization

HARP uses block kernels of the form

$$B_{t,c}(\theta_{t,c}) := Q_{t,c}(\theta_{t,c}) G_{b_t}, \quad (9)$$

where $Q_{t,c}(\theta_{t,c}) \in SO(b_t)$ is learnable and G_{b_t} is a fixed orthogonal base mixer. For power-of-two radices, G_{b_t} is the normalized Sylvester Hadamard matrix H_{b_t} . For non-power-of-two radices, we use a deterministic orthogonal fallback computed by QR factorization of a fixed Gaussian matrix. Thus $G_{b_t}^\top G_{b_t} = I$, so every block remains orthogonal.

Hadamard preconditioning serves two purposes. First, it anchors HARP to a strong baseline: at $\Theta = 0$, all learnable blocks satisfy $Q_{t,c}(0) = I$, so $B_{t,c}(0) = G_{b_t}$. For power-of-two schedules, the staged product therefore recovers Hadamard-family preprocessing up to a fixed permutation convention, so HARP is a drop-in upgrade for Hadamard-based incoherence processing pipelines. Second, it makes fitting easier in practice: rather than learning mixing from scratch, HARP learns a structured residual around an already useful incoherence processor, which is helpful when the quantizer objective is highly non-smooth at extreme bitwidths. Appendix G.3 ablates the base mixer choice.

3.2 Exact initialization: $\Theta = 0$ recovers RHT

HARP initializes all learnable parameters to zero, so $Q_{t,c}(0) = I_{b_t}$ and $B_{t,c}(0) = G_{b_t}$ for every stage t and block c . With Hadamard base mixers, the stride stages therefore recover Hadamard-family preprocessing exactly, up to the fixed permutation induced by the reshape convention. Under the same Kronecker fallback used by QuIP# for non-power-of-two dimensions, the initialization also matches QuIP#’s convention. Thus, before calibration, HARP is numerically equivalent to the existing RHT processor up to implementation-level floating-point differences.

Assumption 1 (Exact RHT equivalence conditions). For each transformed dimension, the following conditions hold:

- (i) For a transformed dimension d , either: (a) $d = 2^L$ with a power-of-two stride schedule, or (b) $d = K \cdot 2^L$ with the Kronecker fallback $T_d(\Theta)$ from Eq. (12).

- (ii) For every power-of-two stage radix b_t , the base mixer is the normalized Sylvester Hadamard $G_{b_t} = H_{b_t}$.
- (iii) Initialization is exact: $Q_{t,c}(0) = I_{b_t}$, hence $B_{t,c}(0) = G_{b_t}$.
- (iv) Random sign flips use the same Rademacher signs and placement as the corresponding RHT processor: $S_U \in \{\pm 1\}^{d_{\text{out}}}$ and $S_V \in \{\pm 1\}^{d_{\text{in}}}$ are applied as diagonal matrices.

Theorem 2. *Under Assumption 1, HARP at $\Theta = 0$ implements QuIP#’s randomized Hadamard incoherence processor exactly up to the same fixed permutation convention. With the inherited random signs, the full two-sided processor is the corresponding RHT. Proof in Appendix A.*

3.3 Orthogonal parameterization

For $b_t = 2$, we use a Givens rotation:

$$Q(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}. \quad (10)$$

For $b_t > 2$, we use the Cayley map. If $A(\theta) \in \mathbb{R}^{b_t \times b_t}$ is skew-symmetric, then

$$Q(\theta) := (I + A(\theta))^{-1}(I - A(\theta)) \quad (11)$$

is orthogonal whenever $I + A(\theta)$ is invertible. In implementation, $A(\theta)$ is obtained by antisymmetrizing an unconstrained tensor.

3.4 Mixed-Radix dimensions and Kronecker fallback

The Mixed-Radix schedule in Eq. (6) supports non-power-of-two dimensions directly. For example, 5120 is handled by the schedule (8, 8, 8, 5, 2), using Hadamard mixers for the power-of-two stages and the QR fallback for the radix-5 stage. This avoids padding while preserving exact orthogonality and staged execution.

We also support an optional Kronecker fallback that aligns the initialization more closely with QuIP#’s Hadamard convention and reduces the number of learnable parameters. If $d = K \cdot n_2$ with $n_2 = 2^L$, and a Hadamard-like table $\tilde{H}_K \in \{\pm 1\}^{K \times K}$ satisfies $\tilde{H}_K \tilde{H}_K^\top = KI$, we define

$$T_d(\Theta) := \left(\frac{1}{\sqrt{K}} \tilde{H}_K \right) \otimes T_{n_2}(\Theta). \quad (12)$$

Operationally, this reshapes a vector into a $K \times n_2$ tensor, applies HARP along the power-of-two axis, and mixes the K rows with the fixed Hadamard-like table. Mixed-Radix HARP is more general and expressive. The Kronecker fallback is useful when closer compatibility or lower overhead is desired.

Algorithm 1 HARP transform $y = x T(\Theta)$

Require: Input $x \in \mathbb{R}^{B \times d}$, stage radices $(b_0, \dots, b_{m-1})$ with $\prod_t b_t = d$
Require: Fixed base mixers $G_{b_t} \in \mathbb{R}^{b_t \times b_t}$ and learnable parameters $\Theta = \{\theta_{t,c}\}$

- 1: $y \leftarrow x$
- 2: Precompute strides $s_t := \prod_{r < t} b_r$ for $t = 0, \dots, m-1$
- 3: **for** $t = 0$ to $m-1$ **do**
- 4: $s \leftarrow s_t, \quad g \leftarrow d/(b_t s)$
- 5: Reshape y to $Y \in \mathbb{R}^{B \times g \times b_t \times s}$
- 6: Transpose to $\tilde{Y} \in \mathbb{R}^{B \times g \times s \times b_t}$
- 7: Construct block kernels $K_t \in \mathbb{R}^{(d/b_t) \times b_t \times b_t}$; $K_t[c] \leftarrow Q(\theta_{t,c}) G_{b_t}$
- 8: View K_t as $\mathbb{R}^{g \times s \times b_t \times b_t}$
- 9: Apply blockwise multiplication along the last dimension
- 10: Invert transpose and reshape back to $y \in \mathbb{R}^{B \times d}$
- 11: **end for**
- 12: **return** y

3.5 Implementation details

3.5.1 Memory and compute

A dense learned orthogonal matrix in $\mathbb{R}^{d \times d}$ requires $\Theta(d^2)$ storage and application cost, which is infeasible at LLM hidden dimensions. HARP instead uses m sparse block stages. For a stage with radix b_t , there are $D_t = d/b_t$ blocks, each with $b_t(b_t - 1)/2$ degrees of freedom, so the parameter count is

$$D_t \frac{b_t(b_t - 1)}{2} = d \frac{b_t - 1}{2}. \quad (13)$$

With bounded radices, this is $O(dm)$ parameters per pass. The application cost is $O(D_t b_t^2) = O(db_t)$ per stage and $O(\sum_t db_t)$ overall, i.e., Hadamard-like $O(d \log d)$ for constant radices. This gives learnable exact orthogonal mixing at structured-transform cost.

3.5.2 Parameter quantization for deployment

After fitting, we optionally store HARP parameters in int8 form. For each block, we quantize either the Givens angle or the upper-triangular entries of the Cayley matrix using a per-block scale, and reconstruct the corresponding orthogonal block at runtime. This reduces processor storage with little effect on perplexity in our experiments. Appendix K.4 gives the exact packing rule.

3.5.3 Fitting HARP processors for PTQ

HARP is fit per layer using calibration statistics and a fixed quantizer backend. Let

$$\begin{aligned} \tilde{W}(\Theta) &= U(\Theta_U)^\top W V(\Theta_V), \\ \tilde{H}(\Theta) &= V(\Theta_V)^\top H V(\Theta_V). \end{aligned}$$

Directly optimizing the full proxy loss in Eq. (3) inside the QuIP# solver would be expensive: it

would require repeated multiplication by $\tilde{H}(\Theta)$ and repeated blockwise second-order quantization steps. We therefore optimize a lightweight surrogate that is cheap enough to evaluate during calibration while remaining aligned with the deployed codebook quantizer.

QuIP# ultimately refines blockwise code assignments using an $LDL^\top$ factorization of the curvature proxy. During HARP fitting, we do not nest this LDLQ procedure inside the optimizer. Instead, for the current rotated weight we compute a direct blockwise codebook target and treat it as stopped-gradient. This gives a practical training signal: a useful rotation should make $\tilde{W}$ easier for the downstream code family to represent, especially in curvature-important directions.

Let $\Delta(\Theta) := \tilde{W}(\Theta) - Q(\tilde{W}(\Theta))$, $w_j(\Theta) := |\tilde{H}_{jj}(\Theta)|$, and let $\bar{w} = w/\text{mean}(w)$. Our diagonal Hessian-weighted reconstruction proxy is

$$\mathcal{L}_{\text{diag}}(\Theta) := \frac{1}{d_{\text{out}} d_{\text{in}}} \sum_{i=1}^{d_{\text{out}}} \sum_{j=1}^{d_{\text{in}}} \Delta_{ij}(\Theta)^2 \bar{w}_j. \quad (14)$$

We stop gradients through $Q(\tilde{W})$, so gradients update the rotation parameters through $\tilde{W}(\Theta)$ rather than through discrete code-assignment changes. This is more stable than straight-through gradients through sharp codebook search.

To better align the rotated curvature proxy with blockwise quantization, we also penalize off-block Hessian energy:

$$\mathcal{R}_{\text{bd}}(\Theta_V) := \frac{1}{d_{\text{in}}^2} \sum_{\substack{p,q \\ p \neq q}} \|\tilde{H}_{pq}\|_F^2, \quad (15)$$

where $\tilde{H}_{pq}$ denotes a block under the quantizer's contiguous partition. This term gives V a direct signal to make curvature more local in the same block structure used by the backend. The total fitting loss is

$$\mathcal{L}_{\text{fit}}(\Theta) := \mathcal{L}_{\text{diag}}(\Theta) + \lambda_{\text{bd}} \mathcal{R}_{\text{bd}}(\Theta_V). \quad (16)$$

We optimize Θ_U, Θ_V with Adam from the exact RHT initialization. Appendix K.3 gives the full layerwise fitting procedure, including the target-refresh variant used to reduce calibration time.

4 Experiments

Scope of comparison. Our primary experimental comparison isolates a single component: the incoherence processor inside a fixed QuIP#-style backend. We therefore compare HARP directly

against RHT under the same backend, and separately report context-matched system-level comparisons to published AWQ, GPTQ, OmniQuant, ButterflyQuant, and the available weight-only SpinQuant result where evaluation protocols are compatible. We also include a QTIP experiment to test whether the same learned processor can be inserted into a distinct RHT-based PTQ backend.

Models. The main scaling study uses Llama 3.2 (1B, 3B) and Llama 2 (7B, 13B, 70B). Subsection 4.7 reports a separate 2-bit transfer experiment on Qwen3-8B, using the same E8P/LDLQ backend and HARP optimization settings.

No-finetuning comparison. Our main tables intentionally compare quantization *without* QuIP#’s weight fine-tuning stages. This isolates the effect of replacing the fixed RHT mixer with HARP: the codebooks, solver, signs, calibration statistics, and evaluation harness are kept fixed. This choice should not be read as an incompatibility with fine-tuning. Appendix I shows that HARP can be combined with QuIP#’s fine-tuning-during-quantization stage, and that HARP with only this stage already improves over the RHT setting.

Evaluation protocol. We report perplexity (PPL, ↓) on WikiText2 and C4. Llama 3.2 results use context length 8192, while the main Llama 2 comparison uses context length 4096, matching the corresponding QuIP# calibration statistics. For comparison with published AWQ/GPTQ/OmniQuant/ButterflyQuant and available SpinQuant results, we additionally evaluate Llama 2 at context length 2048 in Table 6. Zero-shot accuracy is evaluated with `lm_eval` using `acc` on ARC-Challenge, ARC-Easy, PIQA, and WinoGrande. Evaluation is deterministic for a fixed checkpoint; Appendix C reports benchmark standard errors for the principal 2-bit results.

Methods compared. *FP16* is the unquantized baseline. *RHT* uses fixed randomized Hadamard preprocessing as in QuIP#. Unless explicitly attached to another backend (e.g., QTIP+HARP), *HARP* denotes the same QuIP# backend with only its fixed Hadamard mixer replaced by learned HARP processors; random signs, codebooks, solver, and evaluation remain unchanged. The primary variant is Mixed-Radix HARP. The optional Kronecker compatibility variant is reported separately in Appendix G.1.

Effective bitrate accounting. We report effective bits-per-parameter (BPP) assuming the underlying quantized weights contribute exactly the nominal bitrate and adding storage overhead from HARP processor parameters and metadata. RHT is therefore reported at exactly the nominal bitrate. We report both floating-point HARP parameters and an int8 parameter-storage variant.

Figure 1 visualizes the 2-bit quality–size trade-off. Across both model families, HARP shifts the RHT curve downwards at nearly the same stored model size when int8 parameter storage is used. The improvement is greatest where the RHT model is farthest from FP16, while the gap naturally narrows for larger models whose 2-bit RHT baseline is already closer to full precision. This suggests HARP is most effective when fixed incoherence processing leaves substantial quantization error.

4.1 Language modeling perplexity

Table 1 reports the controlled comparison between fixed RHT and Mixed-Radix HARP. Replacing only the mixer lowers perplexity for every evaluated model and bitwidth while preserving the codebooks, random signs, Hessians, and QuIP# solver. The largest gains occur at 2 bits, where scalar and fixed-RHT quantizers leave the most reconstruction error. Figure 1 shows that these gains persist at nearly matched stored size after int8 parameter packing.

At 3 and 4 bits, RHT is already close to FP16, so the remaining absolute improvement is necessarily smaller. We therefore view these settings as evidence that HARP remains useful as quantization becomes less constrained, while the main operating point is extreme 2-bit PTQ. Appendix B provides context-matched comparisons with published scalar and rotation baselines.

Kronecker compatibility and parameter storage. The optional Kronecker construction follows QuIP#’s non-power-of-two convention with lower processor overhead, but is less expressive than the primary Mixed-Radix processor. Its available 1B–7B results are reported separately in Appendix G.1. Int8 storage changes perplexity only marginally and reduces the HARP premium to 0.02–0.08 BPP in the 2-bit experiments.

4.2 Quantizer-aligned structural analysis

To examine why the learned basis improves quantization, we dequantize the E8P indices for all

Table 1: Perplexity (PPL ↓) for QuIP# with fixed RHT and Mixed-Radix HARP. Llama 3.2 uses context length 8192 and Llama 2 uses 4096. BPP includes HARP processor storage.

BITS	METHOD	LLAMA 3.2 1B			LLAMA 3.2 3B			LLAMA 2 7B			LLAMA 2 13B			LLAMA 2 70B		
		BPP	W2	C4	BPP	W2	C4	BPP	W2	C4	BPP	W2	C4	BPP	W2	C4
16	FP16	16.00	11.57	13.19	16.00	9.58	10.61	16.00	5.12	6.63	16.00	4.57	6.05	16.00	3.12	4.97
2	QuIP# (RHT)	2.00	26.27	25.24	2.00	16.59	15.88	2.00	8.22	10.86	2.00	6.05	8.06	2.00	4.16	6.01
2	HARP	2.14	22.30	22.57	2.10	15.02	14.77	2.11	7.23	9.49	2.05	5.71	7.63	2.04	4.01	5.81
2	+ INT8 PARAMS	2.08	22.32	22.57	2.05	15.03	14.76	2.06	7.25	9.51	2.03	5.73	7.64	2.02	4.01	5.82
3	QuIP# (RHT)	3.00	14.02	15.32	3.00	11.04	11.72	3.00	5.61	7.35	3.00	4.89	6.49	3.00	3.40	5.20
3	HARP	3.14	13.46	14.56	3.10	10.60	11.50	3.11	5.51	7.19	3.05	4.81	6.38	3.04	3.35	5.15
3	+ INT8 PARAMS	3.08	13.48	14.59	3.05	10.61	11.52	3.06	5.52	7.21	3.03	4.82	6.40	3.02	3.35	5.16
4	QuIP# (RHT)	4.00	12.37	13.95	4.00	10.13	11.04	4.00	5.27	6.85	4.00	4.70	6.20	4.00	3.22	5.05
4	HARP	4.14	12.17	13.66	4.10	9.88	10.80	4.11	5.22	6.78	4.05	4.64	6.14	4.04	3.19	5.02
4	+ INT8 PARAMS	4.08	12.18	13.69	4.05	9.90	10.83	4.06	5.24	6.81	4.03	4.66	6.15	4.02	3.20	5.02

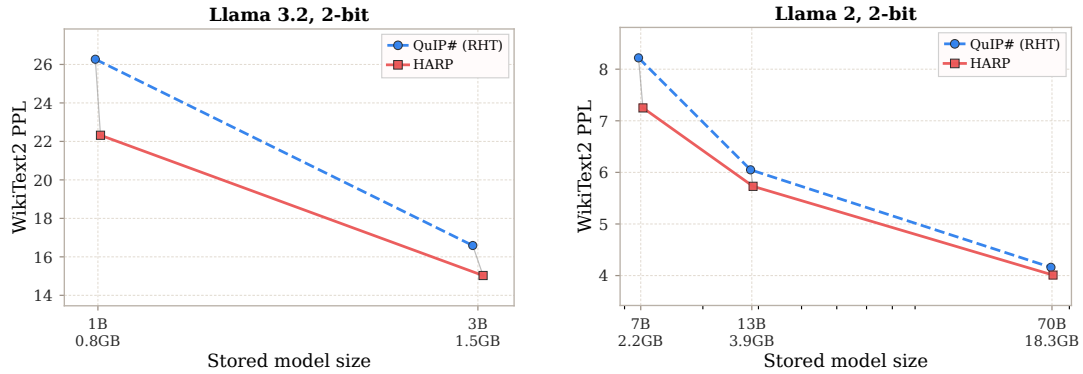


Figure 1: WikiText2 quality-size scaling at 2 bits. HARP uses int8 parameter storage. The plots are split by model family to keep the quality gaps visible across scales.

128 linear matrices of 2-bit Llama 2 7B and evaluate $E = \hat{W} - W$ against each calibration Hessian. Table 3 reports the QuIP# proxy $\text{tr}(EHE^T)/\text{tr}(WHW^T)$, relative Frobenius error, and gain-error component $c = \langle E, W \rangle / \|W\|_F^2$, where $c < 0$ indicates systematic shrinkage.

HARP reduces mean proxy error by 9.2%, relative Frobenius error by 11.0%, and systematic shrinkage by 21.3%. The improvement is not explained solely by scalar flattening. E8P quantizes contiguous 8-dimensional blocks with a bounded codeword radius. Blocks outside this representable shell incur unavoidable radial distortion. Figure 2 shows that HARP moves source-block norms toward the shell and lowers distortion across the norm range. For the layer-0 output projection, mean block distortion falls from 6.90 to 2.50, while the fraction of blocks above the maximum codeword norm falls from 39.0% to 11.0%. Extended diagnostics are reported in Appendix F. The regularizer ablation in Appendix G.4 shows that mild off-block regularization improves quality, while

most of the gain remains without this term.

4.3 Zero-shot evaluation

Table 2 reports zero-shot accuracy on Llama 2. At 2 bits, HARP improves every listed task over fixed RHT, with the largest gains on ARC. These results show that the perplexity improvements are accompanied by stronger downstream multiple-choice performance. At 3–4 bits, several task-level differences are small relative to benchmark standard error, consistent with the narrower perplexity gaps in this regime. We therefore emphasize the consistent 2-bit improvements rather than claiming uniform task-level gains at every bitrate. Appendix C reports standard errors for the principal 2-bit results.

4.4 Inference latency and storage

A central motivation for HARP is to improve incoherence processing without giving up the speed advantages of a quantized backend. We measure single-token latency using the QuIP# decoding-

Table 2: Zero-shot accuracy (acc, $\uparrow$) under QuIP# (RHT) vs. HARP on Llama 2 using 1m_eval. HARP uses Mixed-Radix. Context length follows the evaluation harness defaults.

BITS	METHOD	LLAMA 2 7B					LLAMA 2 13B					LLAMA 2 70B				
		BPP	ARC	ARC	PIQA	WINO	BPP	ARC	ARC	PIQA	WINO	BPP	ARC	ARC	PIQA	WINO
16	FP16	16.00	40.0	69.3	78.5	67.3	16.00	45.6	73.3	78.7	69.6	16.00	51.1	77.7	81.1	77.0
2	QuIP# (RHT)	2.00	29.7	56.7	70.8	62.1	2.00	33.8	65.1	74.4	64.3	2.00	47.4	76.9	79.5	75.0
2	HARP	2.11	33.0	63.7	72.4	62.9	2.05	36.4	67.3	75.8	67.2	2.04	48.5	77.8	79.9	75.5
3	QuIP# (RHT)	3.00	37.7	69.3	76.2	66.7	3.00	41.4	71.6	77.5	68.0	3.00	50.4	78.4	80.7	77.0
3	HARP	3.11	38.0	69.1	77.1	67.3	3.05	42.3	71.9	78.5	68.4	3.04	51.8	78.6	80.6	76.8
4	QuIP# (RHT)	4.00	40.7	70.6	77.2	66.7	4.00	45.6	74.5	78.6	69.4	4.00	51.5	78.1	80.6	78.1
4	HARP	4.11	40.7	69.2	78.3	68.7	4.05	45.0	74.8	78.9	69.4	4.04	51.9	78.2	81.2	78.4

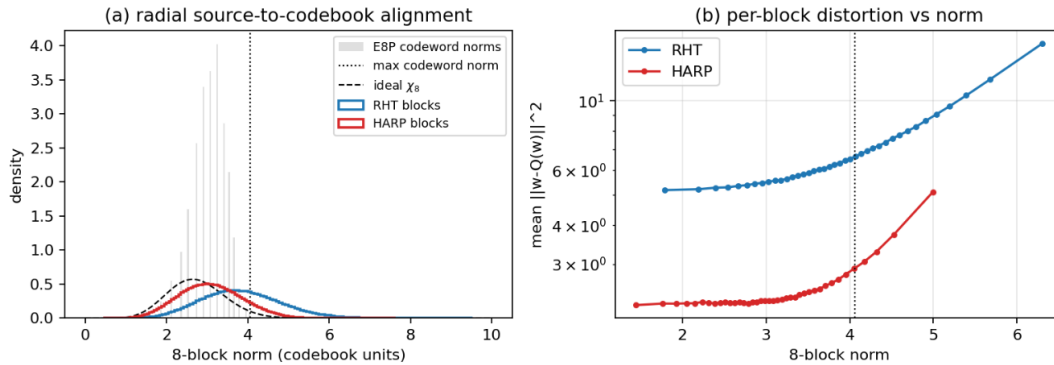


Figure 2: E8P codebook alignment for the layer-0 output projection of 2-bit Llama 2 7B under fixed RHT versus HARP. The dotted vertical line marks the maximum E8P codeword norm.

Table 3: Aggregate reconstruction diagnostics over 128 quantized matrices of Llama 2 7B at 2 bits.

Processor	Proxy	Rel. Frobenius	c	Proxy wins
RHT	0.0413	0.1288	0.0573	—
HARP	0.0375	0.1146	0.0451	123/128

style microbenchmark: batch size 1, sequence length 1, one warmup pass, and 2000 timed repetitions, with CUDA graphs and SDPA attention enabled. CUDA graph capture avoids repeated small-kernel launch overhead. All timings in Table 4 use an NVIDIA RTX 5080 with 16GB VRAM.

HARP preserves most of the fixed-RHT throughput while improving quantized-model quality. On Llama 2 7B at 2 bits, HARP reaches 128 tok/s versus 142 tok/s for RHT, retaining 90.1% of RHT throughput, and is approximately $2.1\times$ faster than FP16 at 61 tok/s. The remaining overhead comes mainly from the additional stride-stage rotations. We use a fused Triton kernel for each stage, avoiding materialized permutations and reducing latency by approximately 20% relative to an unfused implementation. Calibration is a one-

Table 4: Single-token latency on RTX 5080. HARP uses Mixed-Radix and int8 parameter storage.

MODEL	METHOD	BITS	SEC/TOKEN $\downarrow$	TOK/S $\uparrow$
LLAMA 2 7B	FP16	16	0.0162	61
	QuIP# (RHT)	2	0.0070	142
	HARP	2	0.0078	128
	QuIP# (RHT)	4	0.0100	100
	HARP	4	0.0110	91
LLAMA 2 13B	FP16	16	OOM	OOM
	QuIP# (RHT)	2	0.0110	91
	HARP	2	0.0119	84
	QuIP# (RHT)	4	0.0152	66
	HARP	4	0.0167	60

time model-preparation cost and is analyzed separately in Appendix D; int8 parameter packing keeps the stored-model overhead small (Appendix E).

4.5 Backend portability: QTIP

We also integrate the same HARP module into QTIP, which uses trellis-coded quantization together with incoherence processing (Tseng et al., 2024b). The integration changes only the preprocessing, the QTIP backend remains fixed. Table 5 reports Llama 2 results at 2 and 3 bits.

Table 5: Perplexity (PPL $\downarrow$) for QTIP with fixed RHT and HARP. HARP uses Mixed-Radix.

BITS	METHOD	$\sim$ BPP	LLAMA 2 7B		LLAMA 2 13B	
			W2	C4	W2	C4
2	QTIP (RHT)	2.00	6.87	9.00	5.64	7.46
2	QTIP + HARP	2.10	6.62	8.79	5.51	7.34
2	+ INT8	2.05	6.63	8.91	5.52	7.36
3	QTIP (RHT)	3.00	5.41	7.03	4.75	6.31
3	QTIP + HARP	3.10	5.37	6.98	4.74	6.26
3	+ INT8	3.05	5.37	6.99	4.74	6.27

HARP improves QTIP at every evaluated 2- and 3-bit setting, providing evidence that the learned processor is not specific to the QuIP# code family. The int8 variant preserves nearly all gains. Because QTIP quantizes Q , K , and V separately, their processors are also fit separately. Sharing the input-side processor across these projections is a possible route to lower calibration cost.

4.6 Comparison with published baselines

Many published weight-only baselines report Llama 2 perplexity at context length 2048, whereas the controlled main comparison uses 4096. Table 6 therefore reports a separate context-matched system-level comparison. HARP gives the lowest perplexity in all four 2-bit cells. The ButterflyQuant row uses its published scalar W2A16 pipeline, and the other baselines likewise use different quantizers and metadata. This table compares complete systems, while the fixed-backend RHT/HARP experiment isolates the processor.

Table 6: Context-matched 2-bit Llama 2 perplexity at context length 2048. Effective BPP is shown where directly comparable.

Method	Backend	BPP 7B/13B	7B W2	7B C4	13B W2	13B C4
			W2	C4	W2	C4
ButterflyQuant	scalar W2A16	—	15.40	16.61	10.24	12.48
AWQ	scalar, g128	2.14	2.2×10^5	1.7×10^5	1.2×10^5	9.4×10^4
GPTQ	scalar, g128	2.14	36.77	33.70	28.14	20.97
OmniQuant	scalar, g128	2.14	11.06	15.02	8.26	11.05
QuIP# (RHT)	E8P/LDLQ	2.00/2.00	8.95	11.22	6.52	8.32
HARP	E8P/LDLQ	2.11/2.05	7.85	9.68	6.13	7.87

4.7 Transfer to Qwen3-8B

We additionally evaluate Qwen3-8B at 2 bits and context length 4096. The integration changes only model-specific module mapping and requires fresh second-moment statistics. The E8P/LDLQ backend, Mixed-Radix processor, fitting objective, and optimization hyperparameters remain unchanged. Table 7 shows that HARP improves perplexity on both corpora, extending the controlled RHT-to-HARP gain beyond the Llama family.

Table 7: Two-bit Qwen3-8B perplexity at context length 4096 (mean $\pm$ evaluation standard error).

Method	WikiText2 $\downarrow$	C4 $\downarrow$
FP16	8.99 ± 0.27	12.48 ± 0.58
QuIP# + RHT	11.38 ± 0.35	15.30 ± 0.74
QuIP# + HARP	10.80 ± 0.33	14.71 ± 0.72

4.8 Calibration scalability

Table 8 summarizes the 70B preparation cost. Each quantized module group receives 1200 updates. The refresh interval controls how frequently the stopped-gradient codebook target is recomputed.

Table 8: Calibration and quantization cost for 2-bit Llama 2 70B on H100. GPU-hours exclude Hessian generation.

Method	Refresh k	GPU-hours	W2 PPL $\downarrow$
RHT	—	~ 6	4.16
HARP	1	~ 80	4.01
HARP	2	~ 48	4.03
HARP	4	~ 31	4.07

Target caching reduces HARP preparation cost by up to 61% while retaining an improvement over fixed RHT. Peak HARP memory is approximately 42–50 GB, so each module group fits on one 80 GB H100. Independent Transformer blocks can also be scheduled across GPUs. Appendix D reports the complete cross-scale results and Hessian-generation accounting.

5 Conclusion

We introduced HARP, a learnable structured orthogonal incoherence processor for low-bit PTQ. HARP initializes exactly to fixed RHT and learns a layer- and backend-aware refinement from calibration data. Mixed-Radix butterfly-like stages support practical transformer dimensions at controlled storage and inference overhead. Across Llama 3.2 and Llama 2 at 2–4 bits, HARP consistently improves over fixed RHT under an otherwise unchanged QuIP# backend, with the largest gains at 2 bits. The same configuration also improves Qwen3-8B and transfers to QTIP. Structural diagnostics indicate that the learned basis improves alignment with the deployed E8P codebook and blockwise curvature rather than merely optimizing generic incoherence. Int8 parameter storage preserves nearly all gains, and fused kernels retain most of the RHT throughput advantage.

Limitations

HARP adds a one-time layerwise fitting stage beyond fixed RHT. Appendix D reports this cost through Llama 2 70B: depending on target-refresh frequency, the 70B fit requires approximately 31–80 GPU-hours and 42–50 GB peak VRAM on H100 hardware. Target caching, chunk-size control, and layer-level parallelism provide direct cost–memory trade-offs. Crucially, no end-to-end model fine-tuning is required.

The evaluated QuIP# and QTIP backends are weight-only systems, so the controlled experiments isolate weight-only incoherence processing. Extending HARP to weight–activation and KV-cache quantization requires graph-level placement of rotations and coordination with residual streams, rotary embeddings, cache formats, and fused attention kernels.

The current code families expose 2–4-bit operating points. Applying HARP to binary or sub-one-bit quantizers requires a different code family, scaling rule, and quantizer-aligned objective. Finally, the structural analysis identifies codebook alignment and blockwise curvature as important mechanisms, but a complete account of residual-error directions and cross-layer accumulation remains future work.

Acknowledgments

This work was supported by the Ministry of Economic Development of the Russian Federation (agreement No. 139-15-2025-013, dated June 20, 2025, IGK 000000C313925P4B0002).

References

- Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefer, and Dan Alistarh. 2024a. QUIK: Towards end-to-end 4-bit inference on generative large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3355–3371.
- Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefer, and James Hensman. 2024b. QuaRot: Outlier-free 4-bit inference in rotated LLMs. *Advances in Neural Information Processing Systems*, 37:100213–100240.
- Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2023. QuIP: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36:4396–4429.
- Jiale Chen, Vage Egiazarian, Roberto L Castro, Torsten Hoefer, and Dan Alistarh. 2026. WUSH: Near-optimal adaptive transforms for LLM quantization. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR.
- Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. 2019. Learning fast algorithms for linear transforms using butterfly factorizations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1517–1527. PMLR.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.
- Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefer, and Dan Alistarh. 2024. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *The Twelfth International Conference on Learning Representations*.
- Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme compression of large language models via additive quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12284–12303. PMLR.
- Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488.
- Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. 2023. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.
- Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems*, 5.
- Babak Hassibi, David Stork, and Gregory Wolff. 1993. Optimal brain surgeon: Extensions and performance comparisons. *Advances in Neural Information Processing Systems*, 6.

- Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. 2024. SqueezeLLM: Dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 23901–23923. PMLR.
- Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2.
- Deokjae Lee and Hyun Oh Song. 2025. Q-Palette: Fractional-bit quantizers toward optimal bit allocation for efficient LLM deployment. *Advances in Neural Information Processing Systems*, 38:11525–11558.
- Egor Lifar, Semyon Savkin, Or Ordentlich, and Yury Polyanskiy. 2026. WaterSIC: Information-theoretically (near) optimal linear layer quantization. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR.
- Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Weiming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.
- Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, and 1 others. 2024a. Parameter-efficient orthogonal finetuning via butterfly factorization. In *The Twelfth International Conference on Learning Representations*.
- Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. 2024b. VPTQ: Extreme low-bit vector post-training quantization for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8181–8196.
- Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024c. LLM-QAT: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 467–484.
- Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2025. SpinQuant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*.
- Vladimir Malinovskii, Denis Mazur, Ivan Ilin, Denis Kuznedelev, Konstantin Burlachenko, Kai Yi, Dan Alistarh, and Peter Richtarik. 2024. PV-Tuning: Beyond straight-through estimation for extreme LLM compression. *Advances in Neural Information Processing Systems*, 37:5074–5121.
- Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. 2023. Controlling text-to-image diffusion by orthogonal finetuning. *Advances in Neural Information Processing Systems*, 36:79320–79362.
- 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*.
- Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.
- Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024a. 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.
- Albert Tseng, Qingyao Sun, David Hou, and Christopher M De Sa. 2024b. QTIP: Quantization with trellises and incoherence processing. *Advances in Neural Information Processing Systems*, 37:59597–59620.
- Albert Tseng, Zhaofeng Sun, and Christopher De Sa. 2025. Model-preserving adaptive rounding. *arXiv preprint arXiv:2505.22988*.
- Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. GPTVQ: The blessing of dimensionality for LLM quantization. *arXiv preprint arXiv:2402.15319*.
- Diyuan Wu, Ionut-Vlad Modoranu, Mher Safaryan, Denis Kuznedelev, and Dan Alistarh. 2024. The iterative optimal brain surgeon: Faster sparse recovery by leveraging second-order information. *Advances in Neural Information Processing Systems*, 37:139621–139649.
- 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. 2025. ButterflyQuant: Ultra-low-bit LLM quantization through learnable orthogonal butterfly transforms. *arXiv preprint arXiv:2509.09679*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xing-gang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. RPTQ: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*.

Shihao Zhang, Haoyu Zhang, Ian Colbert, and Rayan Saab. 2026. Qronos: Correcting the past by shaping the future... in post-training quantization. In *The Fourteenth International Conference on Learning Representations*.

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

A Hadamard stride factorization and initialized equivalence

Goal. We prove that HARP at $\Theta = 0$ recovers Hadamard-family incoherence processing under the conventions used in the main text. We show: (i) the Walsh–Hadamard transform admits a stride-stage factorization, (ii) initialized HARP implements this factorization up to a fixed permutation, and (iii) the Kronecker fallback matches the corresponding QuIP# convention.

Conventions. We work with column vectors in this appendix for clarity. The row-vector convention used in the main text is the transpose of these statements. For $b = 2^k$, let the unnormalized Sylvester matrices be defined by

$$\tilde{H}_1 = [1], \quad \tilde{H}_{2n} = \begin{bmatrix} \tilde{H}_n & \tilde{H}_n \\ \tilde{H}_n & -\tilde{H}_n \end{bmatrix},$$

and define $H_b := b^{-1/2} \tilde{H}_b$. Then $H_b^\top H_b = I_b$.

A.1 Walsh–Hadamard as stride stages

Assume $d = b^m$ with $b = 2^k$. For stage $t \in \{0, \dots, m-1\}$, define

$$s_t := b^t, \quad g_t := \frac{d}{bs_t} = b^{m-t-1},$$

and the stride stage

$$S_t := I_{g_t} \otimes H_b \otimes I_{s_t}.$$

Lemma 3 (Stride implementation equals the Kronecker stage). *Let $x \in \mathbb{R}^d$. Reshape x into $X \in \mathbb{R}^{g_t \times b \times s_t}$ by*

$$X[\alpha, r, \beta] := x[\alpha(bs_t) + rs_t + \beta].$$

Apply H_b along the middle index r independently for each (α, β) , and flatten back using the same indexing convention. The resulting vector is $S_t x$.

Proof. For fixed (α, β) , the operation multiplies the length- b vector over index r by H_b and leaves all other indices unchanged. This is exactly the action of $I_{g_t} \otimes H_b \otimes I_{s_t}$ under the stated flattening order. $\square$

Define

$$\mathcal{H}_d^{\text{stride}} := S_{m-1} \cdots S_0.$$

Lemma 4 (Stride product is a Hadamard transform up to permutation). *There exists a fixed permutation matrix P depending only on the digit/reshape convention such that*

$$\mathcal{H}_d^{\text{stride}} = P^\top H_d P.$$

Proof. Write each index as base- b digits, $i = \sum_{\ell=0}^{m-1} i_\ell b^\ell$. Stage S_t mixes digit i_t and holds all other digits fixed. The product therefore applies H_b once per digit, i.e., it is $H_b^{\otimes m}$ up to a digit-axis permutation. That permutation is represented by P . Under the Sylvester convention, $H_d = H_b^{\otimes m}$. $\square$

A.2 Initialized HARP equals the stride Hadamard mixer

Theorem 5 (Exact equivalence at $\Theta = 0$, restated as Thm. 2). *Assume $d = b^m$ with $b = 2^k$, and HARP uses $G_b = H_b$ for every stage. At initialization, $Q_{t,c}(0) = I$ for all stages and blocks. Then*

$$T(0) = \mathcal{H}_d^{\text{stride}} = P^\top H_d P.$$

Proof. At initialization, every HARP block is $B_{t,c}(0) = Q_{t,c}(0)G_b = H_b$. Thus each HARP stage is exactly the stride stage S_t from Lemma 3. Taking the product over stages gives $T(0) = S_{m-1} \cdots S_0 = \mathcal{H}_d^{\text{stride}}$. The final equality follows from Lemma 4. $\square$

A.3 Kronecker fallback

Assume $d = K \cdot 2^L$ and $\tilde{H}_K \in \{\pm 1\}^{K \times K}$ satisfies $\tilde{H}_K \tilde{H}_K^\top = K I_K$. Let $H_K := \tilde{H}_K / \sqrt{K}$. The Kronecker fallback defines

$$T_d(\Theta) = H_K \otimes T_{2^L}(\Theta).$$

Corollary 6 (Exact Kronecker equivalence at $\Theta = 0$). If $T_{2L}(0) = P^\top H_{2L} P$, then

$$T_d(0) = (I_K \otimes P)^\top (H_K \otimes H_{2L}) (I_K \otimes P).$$

Thus the initialization matches the Kronecker–Hadamard preprocessing convention up to the same fixed permutation on the power-of-two axis.

Proof. Substitute $T_{2L}(0) = P^\top H_{2L} P$ and use the mixed-product property of Kronecker products. $\square$

B Comparison to published PTQ baselines

The main Llama 2 results use context length 4096, while most published AWQ, GPTQ, OmniQuant, ButterflyQuant, and available SpinQuant weight-only values use 2048. Table 9 therefore reports a separate context-matched comparison. HARP is strongest at 2 and 3 bits. At 4 bits, all competitive systems are close to FP16 and the differences are correspondingly small. ButterflyQuant uses scalar W2A16, while HARP uses E8P/LDLQ. SpinQuant’s row is its available weight-only ablation rather than its main weight–activation configuration. These rows are system-level references across different quantizers and calibration procedures. The isolated processor comparison remains fixed RHT versus HARP under the identical QuIP# backend.

C Evaluation uncertainty

For a fixed quantized checkpoint, perplexity and multiple-choice evaluation are deterministic, no stochastic decoding is used. The uncertainty below is the standard error over evaluation examples, not variance across independently re-quantized checkpoints. Calibration data, Rademacher signs, and random seeds are fixed in the controlled RHT/HARP comparison.

The perplexity improvement is consistent on both corpora and every scale. All listed 2-bit zero-shot means are non-decreasing, although several smaller task differences overlap benchmark standard error. A multi-seed re-quantization study would measure sensitivity to calibration/sign initialization rather than evaluation uncertainty and is separate from the fixed-backend comparison here. Tables 10 and 11 report these standard errors.

D Calibration-time cost

HARP fits the fused QKV , output, fused up/gate, and down-projection groups independently within each Transformer block. Each group receives 1200 Adam updates. The refresh interval k changes only how frequently the stopped-gradient codebook target $Q(\tilde{W})$ is recomputed: 1200, 600, and 300 evaluations per group for $k = 1, 2, 4$, respectively. Table 12 reports calibration and QuIP# quantization time across model scales.

Refreshing every two steps reduces HARP GPU-hours by approximately 36–41% relative to $k = 1$, with no systematic quality loss in these runs. Refreshing every four steps reduces cost by approximately 59–61% while retaining an improvement over RHT at every scale. At 70B, these settings correspond to approximately 3.3, 2.0, and 1.3 days on one H100 for $k = 1, 2, 4$. Independent blocks can be scheduled across GPUs.

Peak memory is controlled by `harp_chunk_size`: smaller chunks reduce VRAM at a modest runtime cost. Disabling $\mathcal{R}_{bd}$ entirely saves approximately 20–30% VRAM and changes 7B WikiText2 PPL from 7.23 to 7.31. For 70B, it is disabled only for the largest down projection, keeping peak usage below 50 GB. Figure 3 gives the finer 7B refresh sweep, including $k = 3, 5, 6$.

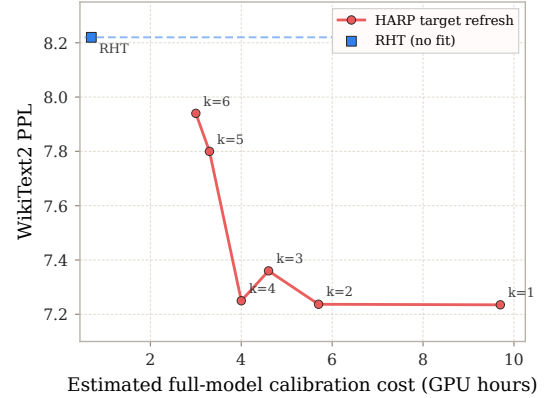


Figure 3: Calibration cost–quality trade-off for the detailed Llama 2 7B target-refresh sweep.

Hessian generation. Table 12 excludes generation of empirical second-moment statistics. For Llama 2 we reuse the released QuIP# statistics, whose original generation time was not reported. The same statistics are shared by RHT and HARP and reused across bitwidths and refresh settings. As a measured from-scratch reference, generating Qwen3-8B statistics from 6144 sequences at

Table 9: Context-matched weight-only PTQ comparison on Llama 2 at context length 2048. Effective BPP is reported for 7B/13B.

Bits	Method	Eff. BPP 7B/13B	7B W2	7B C4	13B W2	13B C4
2	ButterflyQuant (scalar W2A16)	—	15.40	16.61	10.24	12.48
2	AWQ (g128)	2.14/2.14	2.2×10^5	1.7×10^5	1.2×10^5	9.4×10^4
2	GPTQ (g128)	2.14/2.14	36.77	33.70	28.14	20.97
2	OmniQuant (g128)	2.14/2.14	11.06	15.02	8.26	11.05
2	QuIP# (RHT)	2.00/2.00	8.95	11.22	6.52	8.32
2	HARP	2.11/2.05	7.85	9.68	6.13	7.87
3	AWQ (g128)	3.15/3.15	6.24	7.84	5.32	6.94
3	GPTQ (g128)	3.15/3.15	6.29	7.89	5.42	7.00
3	OmniQuant (g128)	3.15/3.15	6.03	7.75	5.28	6.98
3	QuIP# (RHT)	3.00/3.00	6.00	7.60	5.23	6.85
3	HARP	3.11/3.05	5.89	7.46	5.15	6.74
4	AWQ (g128)	4.16/4.16	5.62	7.13	4.97	6.56
4	GPTQ (g128)	4.16/4.16	5.61	7.12	4.98	6.56
4	OmniQuant (g128)	4.16/4.16	5.58	7.12	4.95	6.56
4	SpinQuant (weight-only ablation)	—	5.60	—	5.00	—
4	QuIP# (RHT)	4.00/4.00	5.64	7.16	5.01	6.60
4	HARP	4.11/4.05	5.59	7.10	4.96	6.55

Table 10: Two-bit Llama 2 perplexity with evaluation standard errors.

Model	RHT W2	HARP W2	RHT C4	HARP C4
7B	8.22 ± 0.22	7.23 ± 0.19	10.86 ± 0.37	9.49 ± 0.35
13B	6.05 ± 0.16	5.71 ± 0.14	8.06 ± 0.27	7.63 ± 0.26
70B	4.16 ± 0.10	4.01 ± 0.10	6.01 ± 0.16	5.81 ± 0.14

context length 4096 took approximately 3.1 GPU-hours, or about 5–6 minutes per Transformer layer.

E Stored model size

Table 13 reports stored model size in GB for the fixed-RHT baseline, HARP with floating-point parameters, and HARP with int8 parameter storage. All sizes include the quantized model and processor metadata.

F Incoherence and quantizer-alignment diagnostics

The purpose of incoherence processing is to change the coordinates seen by the quantizer without changing the full-precision function. We complement the aggregate analysis in Table 3 with codebook-shell and curvature diagnostics over the same 128 Llama 2 7B matrices.

F.1 E8P shell alignment

E8P quantizes contiguous 8-dimensional source blocks with a bounded codeword radius. Table 14 reports representative matrices where HARP reduces both direct block distortion and the popula-

tion outside this radius.

Figure 4 shows that HARP generally lowers Hessian-weighted error even when changes in raw error energy are modest: it improves 123 of 128 matrices, with mean log proxy ratio -0.096 . This pattern is consistent with a learned basis that improves codebook fit and the interaction between residual error and layer curvature.

Figure 5 provides representative channel-level views. HARP lowers error energy over much of the curvature range, especially for the layer-0 output projection. These plots are diagnostic rather than a complete causal account. Residual directions and cross-layer error accumulation remain important topics for refining the objective.

F.2 Generic and block-aligned incoherence

We additionally measure pre- and post-quantization weight incoherence, off-block Hessian energy, diagonal Hessian-weighted distortion, and classical Hessian incoherence. Table 15 reports these diagnostics. Lower values are better.

HARP improves pre-quantization weight incoherence, quantized-weight incoherence, off-block Hessian energy, and diagonal Hessian-weighted distortion in almost all evaluated modules. It does not improve the classical Hessian incoherence score μ_H , for which RHT is better. This is consistent with the optimization target: HARP is not trained to minimize every generic incoherence measure, but to find an orthogonal basis favorable

Table 11: Two-bit Llama 2 zero-shot accuracy (percentage points, mean $\pm$ evaluation standard error).

Model	Method	ARC-C	ARC-E	PIQA	WinoGrande
7B	RHT	29.7 $\pm$ 1.34	56.7 $\pm$ 1.02	70.8 $\pm$ 1.06	62.1 $\pm$ 1.36
7B	HARP	33.0 $\pm$ 1.37	63.7 $\pm$ 0.99	72.4 $\pm$ 1.04	62.9 $\pm$ 1.37
13B	RHT	33.8 $\pm$ 1.38	65.1 $\pm$ 0.98	74.4 $\pm$ 1.02	64.3 $\pm$ 1.35
13B	HARP	36.4 $\pm$ 1.41	67.3 $\pm$ 0.96	75.8 $\pm$ 1.00	67.2 $\pm$ 1.32
70B	RHT	47.4 $\pm$ 1.46	76.9 $\pm$ 0.85	79.5 $\pm$ 0.94	75.0 $\pm$ 1.22
70B	HARP	48.5 $\pm$ 1.46	77.8 $\pm$ 0.87	79.9 $\pm$ 0.94	75.5 $\pm$ 1.21

Table 12: Calibration and quantization cost for 2-bit Llama 2 on H100 GPUs. Full-model GPU-hours exclude Hessian generation, model export, and evaluation.

Model	Method/ k	Target evals per group	Block time	Full-model GPU-hours	Peak HARP VRAM	W2 PPL $\downarrow$
7B	RHT	—	$\sim$ 80s	$\sim$ 0.7h	—	8.22
	HARP/1	1200	$\sim$ 1100s	$\sim$ 9.7h	12 GB	7.235
	HARP/2	600	$\sim$ 650s	$\sim$ 5.7h	12 GB	7.237
	HARP/4	300	$\sim$ 450s	$\sim$ 4.0h	12 GB	7.25
13B	RHT	—	$\sim$ 140s	$\sim$ 1.5h	—	6.05
	HARP/1	1200	$\sim$ 2000s	$\sim$ 22h	22 GB	5.71
	HARP/2	600	$\sim$ 1300s	$\sim$ 14h	22 GB	5.68
	HARP/4	300	$\sim$ 850s	$\sim$ 9h	22 GB	5.75
70B	RHT	—	$\sim$ 270s	$\sim$ 6h	—	4.16
	HARP/1	1200	$\sim$ 3650s	$\sim$ 80h	42–50 GB	4.01
	HARP/2	600	$\sim$ 2200s	$\sim$ 48h	42–50 GB	4.03
	HARP/4	300	$\sim$ 1400s	$\sim$ 31h	42–50 GB	4.07

Table 13: Stored model size in GB.

Bits	Model	RHT	HARP	HARP-INT8
2	LLAMA 3.2 1B	0.77	0.82	0.78
2	LLAMA 3.2 3B	1.50	1.60	1.52
2	LLAMA 2 7B	2.15	2.41	2.20
2	LLAMA 2 13B	3.84	4.12	3.89
2	LLAMA 2 70B	18.18	19.21	18.38
3	LLAMA 3.2 1B	0.90	0.95	0.91
3	LLAMA 3.2 3B	1.85	1.96	1.87
3	LLAMA 2 7B	2.96	3.22	3.01
3	LLAMA 2 13B	5.43	5.70	5.48
3	LLAMA 2 70B	26.75	27.78	26.95
4	LLAMA 3.2 1B	1.02	1.07	1.03
4	LLAMA 3.2 3B	2.21	2.31	2.23
4	LLAMA 2 7B	3.77	4.03	3.82
4	LLAMA 2 13B	7.01	7.29	7.07
4	LLAMA 2 70B	35.32	36.35	35.52

to the deployed blockwise quantizer. The combined evidence indicates that HARP better matches the E8P blocks and the layer’s second-order geometry, explaining why quantizer-aligned diagnostics improve even when classical Hessian incoherence does not.

Table 14: Selected E8P codebook-alignment diagnostics for 2-bit Llama 2 7B (RHT $\rightarrow$ HARP).

Matrix	Mean distortion	Above max norm
Layer-0 o	6.90 $\rightarrow$ 2.50	39.0% $\rightarrow$ 11.0%
Layer-1 up	1.14 $\rightarrow$ 0.86	3.75% $\rightarrow$ 0.29%
Layer-0 down	1.15 $\rightarrow$ 0.90	3.70% $\rightarrow$ 0.26%

Table 15: Incoherence and quantizer-alignment diagnostics on Llama 2 7B, 2-bit HARP. “HARP better” counts evaluated modules where HARP improves over RHT.

Metric	RHT	HARP	Δ	HARP BETTER
$\mu_W(\tilde{W}_{\text{pre}})$	5.7712	5.0972	-0.6741	118 / 128
$\mu_W(\tilde{W})$	3.0159	3.0049	-0.0110	125 / 128
OffBlk($\tilde{H}$)	0.9634	0.9237	-0.0397	103 / 128
$\mathcal{L}_{\text{diag}}(\tilde{W})$	2.538×10^{-4}	2.493×10^{-4}	-4.57×10^{-6}	126 / 128
$\mu_H(\tilde{H})$	6.5997	11.2700	+4.6703	13 / 96

G Additional experiments

G.1 Kronecker compatibility variant

The Kronecker variant follows QuIP#’s non-power-of-two Hadamard convention and reduces processor storage, while Mixed-Radix learns over the full dimension. Table 16 reports the models on which both variants were evaluated.

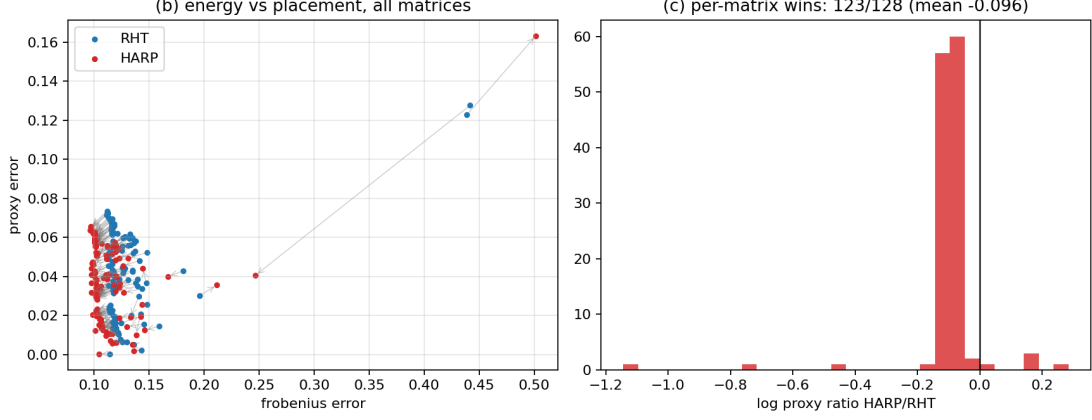


Figure 4: Relative Frobenius error and Hessian-weighted proxy error over all 128 matrices, with per-matrix HARP/RHT proxy ratios.

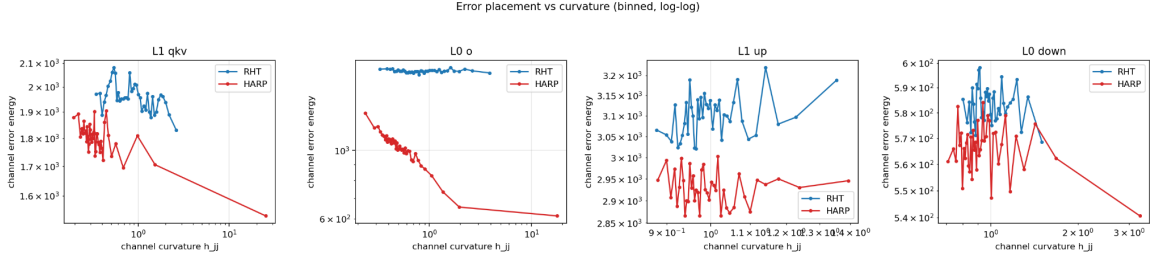


Figure 5: Binned channel-error energy versus diagonal curvature for four representative quantized matrices.

The Kronecker construction improves over RHT at lower overhead, while Mixed-Radix is consistently more accurate and is therefore the primary variant used in the complete cross-scale study.

G.2 Ablation: stage radix as a quality/overhead knob

The stage radix controls both expressivity and overhead. For fixed dimension d , larger radix reduces the number of stride stages (e.g., for $d = 4096$, the stage count is $\log_b d$), which can reduce reshape/-transpose and kernel-launch overhead in practice. However, larger blocks have more degrees of freedom and therefore increase parameter overhead, which is reflected in BPP. Table 17 shows this trade-off on Llama 2 7B.

G.3 Ablation: base mixer choice

Table 18 compares using an identity base mixer against the default Hadamard/QR base mixers. Hadamard preconditioning provides a stronger initialization and slightly improves the final perplexity.

G.4 Ablation: off-block regularization

The coefficient λ_{bd} controls how strongly the input-side rotation is encouraged to localize curvature within the quantizer’s contiguous blocks. Table 19 sweeps λ_{bd} on Llama 2 7B at 2 bits.

A mild penalty improves both datasets, while larger values increasingly constrain V and reduce quality. The modest difference between $\lambda_{bd} = 0$ and 0.1 shows that block localization is useful but is not the sole source of HARP’s gain.

H Additional scaling plots

Figure 6 shows WikiText2 quality–size scaling at 3 and 4 bits. The absolute gains are smaller than at 2 bits because the RHT baseline is already closer to FP16, but HARP continues to shift the curve downward at nearly unchanged storage when int8 parameter storage is used.

I Compatibility with QuIP# fine-tuning

The main experiments compare no-finetuning quantization to isolate the effect of the incoherence processor. QuIP# also reports results with additional fine-tuning stages. We distinguish two such stages.

Table 16: Two-bit Kronecker compatibility results and Mixed-Radix comparison.

Method	Llama 3.2 1B			Llama 3.2 3B			Llama 2 7B		
	BPP	W2	C4	BPP	W2	C4	BPP	W2	C4
RHT	2.00	26.27	25.24	2.00	16.59	15.88	2.00	8.22	10.86
HARP (Kronecker)	2.13	23.36	22.88	2.06	15.97	15.29	2.02	7.61	9.98
+ int8 params	2.07	23.38	22.90	2.03	16.06	15.30	2.01	7.67	10.06
HARP (Mixed-Radix)	2.14	22.30	22.57	2.10	15.02	14.77	2.11	7.23	9.49

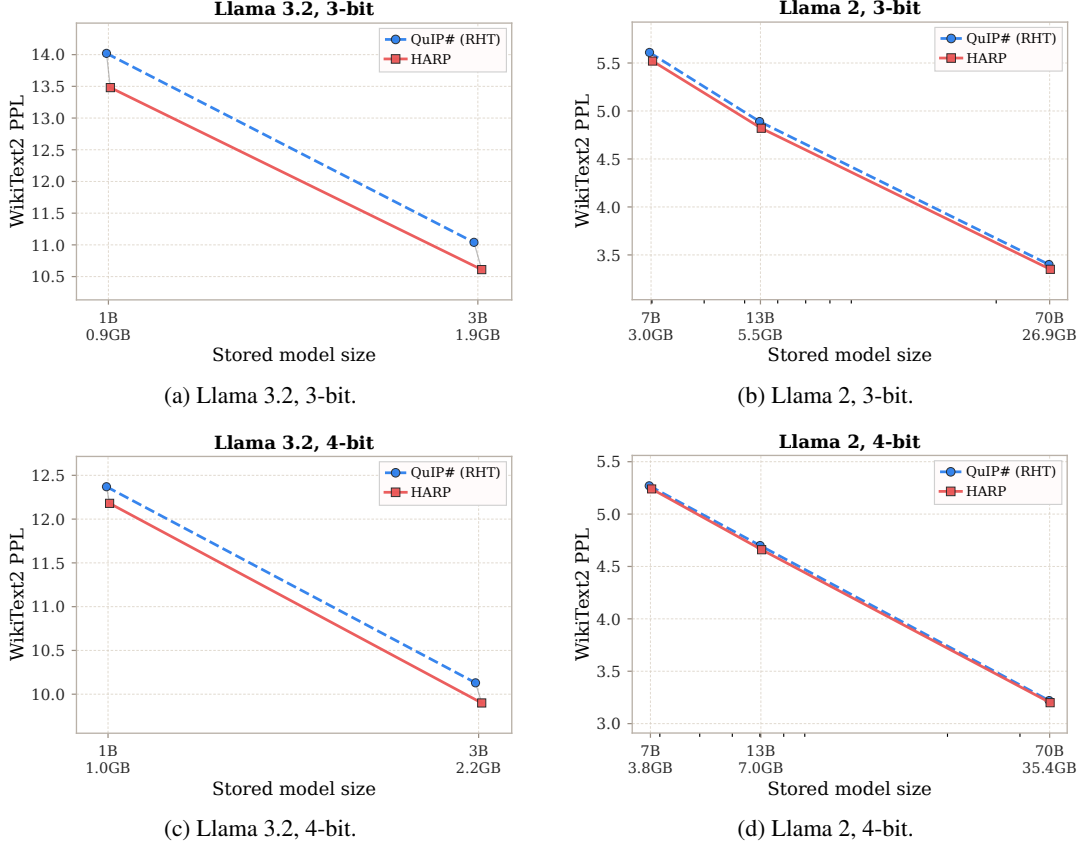


Figure 6: WikiText2 quality-size scaling at 3 and 4 bits. HARP uses int8 parameter storage. Llama 3.2 models use context length 8192, Llama 2 models use context length 4096.

Table 17: Effect of the preferred radix b on Llama 2 7B (2-bit PTQ)

b	STAGES (4096)	BPP	W2 PPL ↓	C4 PPL ↓
2	12	2.08	7.38	9.73
4	6	2.09	7.36	9.65
8	4	2.11	7.23	9.49
16	3	2.15	7.17	9.35

FT-quant denotes QuIP#’s fine-tuning-during-quantization stage. Modules within each Transformer block are quantized in a fixed order, starting with the joint QKV projection. After the current module or module group is quantized, optimization updates the channel scales for modules that have already been quantized and the full weights for modules that have not yet been quantized. Thus, FT-

Table 18: Base mixer choice for HARP blocks on Llama 2 7B, 2-bit PTQ.

MIXER	W2 PPL ↓	C4 PPL ↓
IDENTITY ($G_b = I$)	7.29	9.61
HADAMARD+QR	7.23	9.49

quant locally adapts the partially quantized block while quantization proceeds through the block.

E2E FT denotes the subsequent end-to-end fine-tuning stage after all modules have been quantized. At this stage, the procedure fine-tunes quantization scales together with the remaining unquantized parameters in the network, such as normalization parameters and the language-model head.

These stages improve final quality, but they introduce additional choices such as trainable param-

Table 19: Off-block regularizer ablation on Llama 2 7B at 2 bits.

λ_{bd}	W2 PPL ↓	C4 PPL ↓
0.0	7.31	9.56
0.1	7.23	9.49
0.2	7.33	9.64
0.3	7.32	9.61
0.5	7.42	9.71

eter sets, learning rates, schedules, and data budgets. For this reason, our main tables omit them and isolate the RHT-to-HARP processor change.

HARP is compatible with these stages because it only changes the orthogonal preprocessing used before quantization. As a preliminary check, we enable FT-quant on Llama 2 7B at context length 4096. Table 20 reports the result.

Table 20: Fine-tuning compatibility on Llama 2 7B, 2-bit, context length 4096. HARP uses Mixed-Radix.

METHOD	W2 PPL ↓	C4 PPL ↓
QUIP# + RHT + FT-QUANT ONLY	6.44	8.30
QUIP# + RHT + FT-QUANT + E2E FT	6.19	8.16
HARP + FT-QUANT ONLY	6.16	8.08

Even with only FT-quant, HARP improves over the corresponding RHT setting. It also slightly improves over the fully tuned RHT result that uses both FT-quant and E2E FT. We do not include full HARP + E2E FT experiments in the main table because the goal is to isolate the processor and avoid conflating it with fine-tuning schedule choices.

J Experimental Reproducibility Checklist

J.1 Hardware

Quantization and calibration experiments use NVIDIA H100 GPUs unless otherwise stated. Inference latency is measured on an NVIDIA GeForce RTX 5080 with 16GB VRAM. HARP performs 1200 Adam updates for each of four quantized module groups per Transformer block. Table 12 reports block times, full-model GPU-hours, and peak VRAM through Llama 2 70B. Hessian generation is a separate reusable preprocessing stage and is excluded from those totals.

J.2 Evaluation

Reported perplexity and multiple-choice values are deterministic for a fixed quantized checkpoint. Appendix C reports standard errors over evaluation examples for the principal 2-bit results. These

standard errors quantify benchmark sampling uncertainty, they do not measure variation across independently regenerated calibration sets, signs, or quantized checkpoints.

J.3 HARP fitting hyperparameters

Unless stated otherwise, we fit one pair of HARP processors (U, V) per quantized module group using $S = 1200$ Adam steps. We use stride ordering (Algorithm 1) with a single pass ($P = 1$) and the default Mixed-Radix schedule constructed by Algorithm 2 with preferred radix $b_{base} = 8$ and maximum radix $b_{max} = 8$.

QUIP# hyperparameters. We keep the QUIP# backend fixed throughout: codebook family and training, block sizes, quantization scales, and all QUIP# solver/rounding hyperparameters follow the settings in the original QUIP# paper (Tseng et al., 2024a). We do not retune QUIP#-specific hyperparameters for HARP, the only change is replacing the fixed Hadamard mixer with the learned HARP processor (and optionally enabling the Kronecker fallback where stated).

Calibration statistics. For the Llama experiments, we reuse the precomputed Hessian/second-moment statistics from the corresponding QUIP# pipeline. Following the QUIP# protocol, these statistics use 6144 sequences sampled from RedPajama 1T (Tseng et al., 2024a). For the Qwen3-8B transfer experiment, we generate statistics from 6144 sequences at context length 4096. This preprocessing takes approximately 3.1 GPU-hours in our setup. Hessian generation is a one-time stage whose outputs can be reused across quantization methods, bitwidths, hyperparameter sweeps, and target-refresh settings.

Optimization. We use learning rates $\eta_U = \eta_V = 3 \times 10^{-2}$ for the U and V parameters. We initialize rotation parameters at $\Theta = 0$, so that $Q_{t,c}(0) = I$ and HARP matches Hadamard-family preprocessing at initialization (Section 3.2).

Objective and regularization. We use the diagonal-weighted proxy objective from Eq. (14) and the block-diagonalization regularizer from Eq. (15) with $\lambda_{bd} = 0.1$ and block size $g = 8$. For Llama 2 70B down projection, we disable $\mathcal{R}_{bd}$ to keep peak calibration memory below 50 GB.

Base mixers. For power-of-two radices we use Hadamard base mixers, and for non-power-of-two

radices we use the QR-based orthogonal fallback (Section 3.1).

J.4 Code and artifact availability

The HARP implementation, QuIP#/QTIP integrations, deployment kernels, and reproduction scripts are available at <https://github.com/brain-lab-research/HARP>; quantized HARP checkpoints are available at <https://huggingface.co/collections/brain-lab/harp-quantized-models>.

J.5 Artifacts and licenses

We use publicly released Llama 2, Llama 3.2, and Qwen3 checkpoints under their original model licenses, and evaluate on WikiText2, C4, ARC-Challenge, ARC-Easy, PIQA, and WinoGrande through the standard evaluation harness. We release the software required to reproduce HARP experiments together with quantized checkpoints for research and reproducibility. Derived checkpoints remain subject to the licenses and access terms of their base models. We do not redistribute benchmark or calibration text. The released code is based on the public QuIP# and QTIP implementations and follows the upstream licenses stated in the repository README.

K Implementation details

This section reports implementation details omitted from the main text: the stride-layout example, the Mixed-Radix schedule construction, the layerwise fitting pseudocode, and parameter packing.

K.1 Stride-stage implementation

The permutation P_t in Eq. (7) is conceptual. A stage is implemented by reshaping, transposing, multiplying small blocks, and reshaping back.

Index-view example. For $d = 16$ and $b = 4$, write an index as two base-4 digits $i = i_0 + 4i_1$. The first stage mixes the low-order digit i_0 while holding i_1 fixed; these are contiguous groups. The second stage mixes the high-order digit i_1 while holding i_0 fixed; these are stride-4 groups. Thus each stage mixes one digit of the Mixed-Radix index representation, as in fast Walsh–Hadamard and Cooley–Tukey-style transforms, but implemented by reshape/transpose rather than by materializing a permutation.

K.2 Stage schedules and Mixed-Radix construction

Algorithm 2 gives the schedule construction used in all experiments. It peels off as many copies of the preferred radix b_{base} as divide d , factors the remainder with the largest admissible factor at most b_{max} , and keeps any leftover prime as a final stage. We use $b_{\text{base}} = b_{\text{max}} = 8$; the schedule is deterministic in d and needs no padding. For example, 4096 gives (8, 8, 8, 8) and 5120 gives (8, 8, 8, 5, 2), with the QR fallback of Section 3.1 used only for the radix-5 stage.

Algorithm 2 Greedy Mixed-Radix schedule (preferred radix first)

Require: Dimension $d \geq 2$, preferred radix b_{base} (default 8), maximum radix b_{max} (default 8)
Ensure: A list of stage radices $\mathbf{b} = (b_0, \dots, b_{m-1})$ such that $\prod_i b_i = d$

```

1:  $\mathbf{b} \leftarrow []$ ,  $r \leftarrow d$ 
2: while  $r \bmod b_{\text{base}} = 0$  do
3:   append  $b_{\text{base}}$  to  $\mathbf{b}$ 
4:    $r \leftarrow r / b_{\text{base}}$ 
5: end while
6: for  $f = \min(b_{\text{max}}, r)$  down to 2 do
7:   while  $r \bmod f = 0$  do
8:     append  $f$  to  $\mathbf{b}$ 
9:      $r \leftarrow r / f$ 
10:  end while
11: end for
12: if  $r \neq 1$  then
13:   append  $r$  to  $\mathbf{b}$ 
14: end if
15: return  $\mathbf{b}$ 
```

K.3 Layerwise fitting procedure

The full layerwise fitting procedure is given in Algorithm 3. It iterates over Adam steps, refreshing the quantized codebook target every k steps to reduce calibration cost.

K.4 Int8 storage of HARP parameters

After fitting, we optionally quantize HARP parameters to 8-bit integers for storage efficiency. For $b_t > 2$ stages we store only the strict upper triangle of the skew-symmetric matrix $A(\theta)$, which has $b_t(b_t - 1)/2$ entries per block. For $b_t = 2$ stages we store the single Givens angle. For each block c , we compute a per-block scale

$$s_{t,c} = \max |a_{t,c}| / 127$$

and store

$$q_{t,c} = \text{round}(a_{t,c} / s_{t,c}).$$

At runtime we reconstruct $a_{t,c} \approx s_{t,c} q_{t,c}$ and build the corresponding orthogonal block using the Givens or Cayley parameterization.

Algorithm 3 Layerwise fitting of HARP for a linear layer

Require: Weight $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, second moment $H \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$
Require: Blockwise codebook quantizer $Q(\cdot)$; HARP initialization $\Theta_U = \Theta_V = 0$
Require: Adam step size η , steps S , block size g , regularizer λ_{bd} , target refresh interval k (default $k = 1$)
1: Initialize $U(\Theta_U) \in O(d_{\text{out}})$ and $V(\Theta_V) \in O(d_{\text{in}})$ as HARP processors
2: **for** $s = 1$ to S **do**
3: Compute rotated weights $\tilde{W} \leftarrow U^\top W V$
4: Compute rotated second moment $\tilde{H} \leftarrow V^\top H V$
5: $\tilde{w}_j \leftarrow |\tilde{H}_{jj}| / \text{mean}(|\tilde{H}_{jj}|)$ and $\text{stopgrad}(\tilde{w})$
6: **if** $s \bmod k = 1$ or $k = 1$ **then**
7: $\tilde{W} \leftarrow \text{stopgrad}(Q(\tilde{W}))$
8: **end if**
9: $\Delta \leftarrow \tilde{W} - \hat{\tilde{W}}$
10: $\mathcal{L}_{\text{diag}} \leftarrow \frac{1}{d_{\text{out}} d_{\text{in}}} \sum_{i,j} \Delta_{ij}^2 \tilde{w}_j$
11: Compute $\mathcal{R}_{\text{bd}}$ from Eq. (15)
12: $\mathcal{L}_{\text{fit}} \leftarrow \mathcal{L}_{\text{diag}} + \lambda_{\text{bd}} \mathcal{R}_{\text{bd}}$
13: Update Θ_U, Θ_V with Adam using $\nabla \mathcal{L}_{\text{fit}}$
14: **end for**
15: Optionally quantize HARP parameters to int8 as in Section 3.5.2
16: **return** Fitted processors U, V

L Detailed Literature Review

We organize related work into three parts: post-training quantization and its failure modes, transform-based reparameterizations, and structured orthogonal transforms for quantization.

L.1 Post-Training Quantization of LLMs

General compression and PTQ. LLM deployment under tight memory and bandwidth budgets has motivated a broad set of compression techniques, including one-shot pruning (Frantar and Alistarh, 2023; Sun et al., 2024), quantization-aware training (QAT) (Liu et al., 2024c), and post-training quantization (Zhu et al., 2024). We focus on PTQ because it operates on a fixed pretrained model with only a small calibration set, which keeps it practical at frontier scale where QAT or full retraining is costly.

Layerwise second-order reconstruction. The layerwise objective in Eq. (2) descends from classical second-order pruning (LeCun et al., 1989; Hassibi and Stork, 1992; Hassibi et al., 1993). OBC turns this analysis into an exact greedy solver for joint pruning and quantization (Frantar and Alistarh, 2022), GPTQ scales it to LLM-sized layers (Frantar et al., 2023), and later work accelerates the underlying sparse recovery (Wu et al., 2024). ZeroQuant pairs fine-grained group-wise quantization with lightweight layerwise distillation (Yao et al.,

2022). More recent rounding procedures correct errors inherited from previously quantized layers (Zhang et al., 2026) or target the divergence of the full model output directly (Tseng et al., 2025). All of these methods improve the rounding rule under a fixed basis. HARP is complementary: it changes the basis exposed to any such solver while leaving the rounding rule untouched.

Heavy-tailed statistics and outlier channels.

Extreme low-bit quantization is dominated by heavy-tailed weight distributions and a small number of high-magnitude outlier channels that inflate per-tensor scales and reduce effective precision (Liu et al., 2025; Ashkboos et al., 2024b). A direct remedy is to separate outliers and preserve them in higher precision: LLM.int8() (Dettmers et al., 2022) routes outlier features to FP16, while SpQR and SqueezeLLM decompose weights into a dense quantized core plus a sparse high-precision correction (Dettmers et al., 2024; Kim et al., 2024). QUIK extends the strategy to both weights and activations for end-to-end low-bit inference (Ashkboos et al., 2024a). Although effective, these methods introduce irregular memory access patterns and specialized kernel requirements that can negate the throughput advantages. This motivates approaches that maintain dense, uniform low-bit computation while reshaping layer statistics.

Calibration-based rescaling. A practical alternative to explicit outlier separation is to apply lightweight, calibration-guided channel-wise rescalings. AWQ derives per-channel scales from activation magnitudes to protect sensitive channels under weight-only quantization (Lin et al., 2024). SmoothQuant shifts part of the activation dynamic range into the weights via a matched diagonal rescaling, enabling standard integer kernels (Xiao et al., 2023). OmniQuant learns per-layer scale and clipping adjustments during calibration, though clipping is not a strict change of basis and can alter the function (Shao et al., 2024). RPTQ instead reorders and clusters activation channels so that similarly ranged channels share quantization parameters (Yuan et al., 2023). While practical, such rescaling and reordering cannot mix coordinates and therefore leave correlated outlier subspaces intact, a limitation that grows increasingly severe at extreme bitwidths. This motivates orthogonal reparameterizations that mix all coordinates while preserving the full-precision model exactly.

L.2 Transform-Based Reparameterizations

Incoherence processing. QuIP (Chee et al., 2023) introduces incoherence processing: it applies a structured random orthogonal change of basis so that weight mass and curvature-sensitive directions are spread across coordinates. QuIP# (Tseng et al., 2024a) replaces the general random orthogonal with the randomized Hadamard transform (RHT), which is exactly orthogonal and admits $O(d \log d)$ application. QTIP (Tseng et al., 2024b) retains the same preprocessing while advancing the quantizer backend to trellis-coded quantization. Beyond weight-only PTQ, QuaRot (Ashkboos et al., 2024b) inserts discrete orthogonal rotations into the full Transformer graph to suppress outliers in weights, activations, and the KV cache. A shared limitation of all these methods is the fixed transform.

Learned model-level rotations. SpinQuant (Liu et al., 2025) demonstrates that the choice of rotation substantially affects low-bit quality and learns Transformer-invariant rotations (including residual-stream and attention-head rotations) end-to-end against a quantized-network objective. The learned rotations are model-level objects wired into the Transformer graph, making SpinQuant an important baseline but not a drop-in replacement for the two-sided Hadamard preprocessor inside QuIP#-style pipelines. HARP, in contrast, operates at the level of a single linear layer and is fit within the PTQ calibration loop. It is designed as a reusable processor module for backends that already expose an incoherence-processing step.

Non-orthogonal data-aware transforms. WUSH (Chen et al., 2026) derives closed-form near-optimal blockwise transforms for joint weight-activation quantization under RTN AbsMax-scaled quantizers, combining a Hadamard backbone with a calibration-dependent second-moment factor. The resulting transform is generally non-orthogonal and targets standard RTN-style block quantizers at 4-bit weight/activation precision, a different operating point from HARP’s focus on extreme low-bit PTQ (2–4 bits per weight) with vector-quantized backends. More fundamentally, non-orthogonal transforms alter the full-precision computation and are not drop-in change-of-basis modules. HARP deliberately stays strictly orthogonal so the full-precision model is preserved exactly.

Rate allocation, information-theoretic views. WaterSIC (Lifar et al., 2026) analyzes linear-layer quantization through rate-distortion theory and derives a waterfilling allocation across input columns. Q-Palette (Lee and Song, 2025) connects near-optimal Gaussian-weight bit allocation with practical fractional-bit quantizers. Both are complementary to HARP: they modify bit allocation or the quantizer family, whereas HARP modifies the orthogonal basis for a fixed backend.

Structured code families. The effectiveness of extreme low-bit PTQ depends heavily on the quantizer code family. AQLM (Egiazarian et al., 2024) achieves extreme compression through additive multi-codebook quantization, and PV-Tuning further improves such extreme-compression pipelines by fine-tuning quantized representations beyond straight-through estimation (Malinovskii et al., 2024). GPTVQ and VPTQ demonstrate that Hessian-aware vector quantization can improve the size-accuracy trade-off by quantizing higher-dimensional weight blocks (Van Baalen et al., 2024; Liu et al., 2024b). QuIP# combines incoherence processing with lattice/codebook vector quantization, while QTIP replaces explicit codebooks with trellis coding to increase the effective quantization dimension (Tseng et al., 2024a,b). As these backends grow more powerful, the choice of coordinate system becomes more consequential: HARP learns this coordinate system, leaving the chosen backend unchanged.

L.3 Structured Orthogonal Transforms

Butterfly factorizations express a broad family of fast transforms as products of sparse orthogonal stages and make these stages learnable (Dao et al., 2019). Structured orthogonal parameterizations of this kind also power parameter-efficient fine-tuning, where orthogonal updates (Qiu et al., 2023) are made scalable through butterfly factorization (Liu et al., 2024a). The closest direction to HARP is the use of learnable structured orthogonal rotations as quantization preprocessors. ButterflyQuant (Xu et al., 2025) replaces fixed Hadamard rotations with learnable butterfly transforms parameterized by continuous Givens rotations, preserving orthogonality and $O(d \log d)$ cost while adapting the rotation to calibration data. HARP shares the principle that structured orthogonal transforms provide learnable mixing at deployable cost, but differs in target and compatibility

constraints.

Specifically, HARP is designed as a drop-in replacement for the two-sided RHT inside QuIP#-style PTQ pipelines, with four distinguishing properties. First, HARP initializes exactly to the corresponding RHT preprocessor (up to a fixed permutation convention), so calibration learns a structured refinement around an already-strong baseline rather than from scratch. Second, HARP uses Hadamard-preconditioned block kernels and supports non-power-of-two dimensions through Mixed-Radix schedules without padding. Third, HARP optimizes a layerwise objective directly coupled to the deployed blockwise quantizer, including a block-diagonalization regularizer that aligns the learned curvature with the backend’s block structure. Fourth, HARP is compatible with modern vector-quantized backends such as QuIP# and QTIP, enabling backend-agnostic deployment.

Table 6 includes ButterflyQuant’s published context-matched W2A16 perplexities as a system-level reference. HARP obtains lower perplexity in that comparison, but the scalar and E8P/LDLQ backends differ. The controlled evidence for the learned processor remains the RHT-to-HARP replacement within the same QuIP# pipeline.

Summary of positioning. Relative to fixed RHT (Tseng et al., 2024a,b), HARP preserves fast structured execution while adding per-layer, backend-aware adaptivity. Relative to model-level learned rotations (Liu et al., 2025), HARP uses a staged structured parameterization with controlled overhead and does not require rewiring the Transformer graph. Relative to non-orthogonal data-aware transforms (Chen et al., 2026), HARP stays strictly orthogonal so the full-precision model is preserved exactly. Rather than proposing a new standalone pipeline or quantizer, HARP treats the incoherence processor as a reusable module insertable into any Hadamard-based backbone.

Information About Use of AI Assistants

AI assistants were used to proofread the manuscript, support literature search, and organize the final repository structure. All AI-assisted outputs were reviewed by the authors.