POST-TRAINING ADAPTATION OF LLMS TO STOCHASTIC WEIGHT PERTURBATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Analog in-memory computing (AIMC) performs matrix–vector multiplications inside memory arrays that store weights, avoiding costly weight movement during LLM inference. The price it pays is stochastic noise on every weight that is fundamentally different from the deterministic rounding error in digital quantization. Hardware-aware training can mitigate the resulting accuracy loss, but requires a full training pipeline with escalating costs as model sizes grow. Post-training quantization (PTQ) requires only a calibration set, but fails to address stochastic weight noise. We propose a post-training robustification (PTR) method that adapts weights using a noise-aware objective: expected layer-output distortion, decomposed into weight-error and noise-variance terms. Weight-Range Adaptation for Robustness to Perturbations (WARP) combines per-channel input scaling and weight clamping with exactly-solved thresholds for each weight group. WARP+ adds GPTQ-style error redistribution on top. We evaluate 0.6B to 32B models of the Qwen3 family in an AIMC simulator with a hardware-derived programming-noise model. WARP+ improves accuracy across all model sizes and completes in $<$3 hours on 4 A100 GPUs for the 32B model. On Qwen3-14B, WARP+ recovers 40% of the accuracy loss, more than 2× the recovery achieved by PTQ, and reaches within 3.5% points of noise-free accuracy. On Qwen3-32B, the remaining gap is $< 1.5\%$ points. Using WARP+ for PTQ by applying rounding error as weight noise yields a Qwen3-4B model with 3-bit weights at $<$9.2% accuracy loss, less than half of OmniQuant (19.3%) and GPTQ (22.4%).

1 INTRODUCTION

The dominant computational cost of large language models (LLMs) has shifted from training to inference (McDonald et al., 2022). This cost is hard to reduce because data movement, rather than arithmetic, dominates energy use on modern accelerators. Fetching an operand from memory costs far more than computing on it (Jouppi et al., 2021), and every forward pass must stream the model's weights to the compute units. Low-precision formats reduce this traffic, but they do not remove its cause, the separation of memory and computation known as the von Neumann bottleneck.

Analog in-memory computing (AIMC) addresses this bottleneck by performing matrix-vector multiplications directly inside the weight-storing memory array (Burr et al., 2017; Sebastian et al., 2020), alleviating weight movement. Array capacity has so far restricted AIMC to models smaller than current LLMs (Le Gallo et al., 2023). With 3D-stacked non-volatile memory, however, models with hundreds of billions of parameters become a plausible target (Hu et al., 2022; Büchel et al., 2025b).

The price of computing in the analog domain is accuracy. The dominant source of error is programming noise (Le Gallo et al., 2023): a device cannot be set to its target conductance exactly, so every stored weight is perturbed by a random amount that is fixed once programmed but unknown in advance. This noise is stochastic and thus fundamentally different from the deterministic rounding error of digital quantization. In addition, the converters at the boundary of the array quantize the inputs and outputs of every matrix-vector multiplication (MVM). Together, these effects substantially degrade the accuracy of off-the-shelf LLMs deployed on AIMC hardware.

Adaptation to hardware non-idealities has been pursued along two lines. The first fully retrains the model with the non-idealities simulated during training, which recovers much of the lost accuracy under analog noise but requires a full training pipeline (Büchel et al., 2025a), a cost that grows with

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

model size. The second adapts the model post-training using a small calibration set, and has proven effective for the deterministic rounding error of digital accelerators (Frantar et al., 2023; Xiao et al., 2024; Liu et al., 2025; Shao et al., 2024). For stochastic weight noise, however, no such post-training adaptation exists. The only post-training method for analog hardware we are aware of Hou et al. (2025) rescales activations to reduce the converters' quantization error, shifting the error burden onto the weights rather than reducing the noise on them. This work addresses this gap (Table 1).

Table 1: Hardware-aware training (HWA), post-training quantization (PTQ), and post-training robustification (PTR, this work). HWA token count from Büchel et al. (2025a).

	HWA	**PTQ**	**PTR (ours)**
Data (tokens)	training, $\sim$20B	calibration, $\sim$1M	calibration, $\sim$1M
Weight updates	full retraining	layer-wise	layer-wise
Compute	multi-GPU cluster, days	few GPUs, hours	few GPUs, hours
Error model	deterministic or stochastic	deterministic	stochastic
Target hardware	digital & analog	digital	analog & digital

This work introduces post-training robustification (PTR): Adapting a pretrained model to stochastic weight perturbations using only a calibration set. Rather than attempting to predict or correct an unknown realization of programming noise, PTR reshapes the pretrained weights so that future perturbations have a smaller effect. Our approach minimizes the expected layerwise output distortion under hardware noise, decomposing it into a weight-error term and a noise-variance term. The optimization therefore balances the distortion introduced by modifying the weights against the reduction in their sensitivity to stochastic noise. We derive two complementary mechanisms for reducing noise-inducing weight ranges. *Input scaling* reduces large weight magnitudes by transferring scale from the weights to the corresponding activations while preserving the layer output exactly. *Weight clamping* then clips the remaining outliers that determine the programming range, with the clipping thresholds selected by a hyperparameter-free solver. Together they form *WARP* (Weight-Range Adaptation for Robustness to Perturbations). *WARP+* extends WARP with a GPTQ-style update that compensates for clipping error by adjusting the remaining unprocessed weights. WARP+ operates layer by layer without retraining and completes in under three hours on 4 A100 GPUs for a 32B model.

Our experiments on the Qwen3 family lead to four findings. First, digital PTQ methods transfer poorly to analog noise. Second, robustness to analog noise improves with model size: Under identical hardware configurations, the accuracy drop of off-the-shelf Qwen3 models shrinks from 22.3% at 1.7B to 4.1% at 32B. Third, WARP effectively recovers accuracy across all model sizes and brings Qwen3-32B to within 1.5% of its noise-free accuracy. Fourth, using WARP with rounding error in place of the programming noise yields a competitive method for low-bit weight quantization. In summary, our contributions are:

- A noise-aware layer-wise objective for deploying LLMs on AIMC hardware, with an exact decomposition of the expected output distortion into weight error and noise variance.
- WARP, a post-training method derived from this objective that combines input scaling with a hyperparameter-free solver for weight clamping thresholds, and its extension to WARP+ with GPTQ-style error redistribution.
- A study of robustness to AIMC noise across model scale (0.6B–32B) and an evaluation of digital post-training quantization methods under analog noise.
- An extension of the framework to low-bit weight quantization, where the rounding error takes the place of the programming noise and the quantization ranges are obtained in closed form.

2 Background and Related Work

Analog in-memory computing. AIMC performs MVMs directly in the memory array storing the weights. Each non-volatile memory (NVM) device encodes a weight as a conductance (Figure 1e), while the input vector is applied as voltages. Ohm's law produces the weight–input products, and

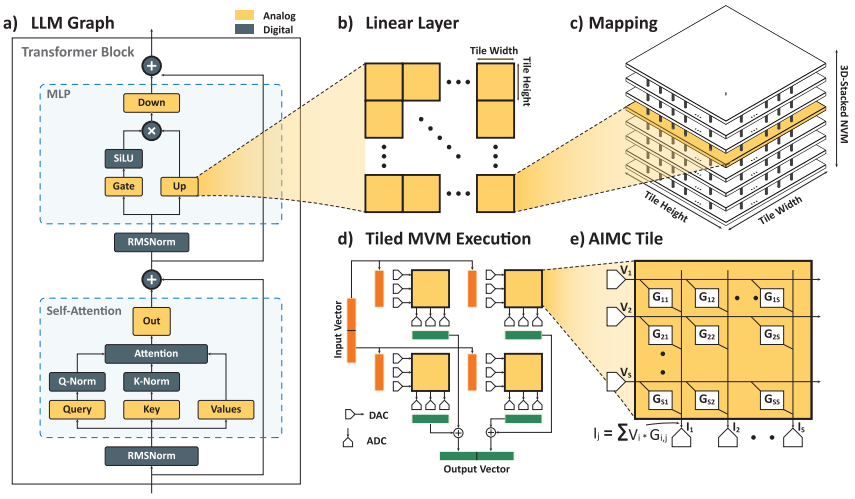


Figure 1: Mapping an LLM onto AIMC hardware. (a) The linear layers of the transformer block (yellow) run in analog; all other operations stay digital (gray). (b) Each weight matrix is partitioned into tiles. (c) The tiles are mapped onto the layers of a 3D-stacked non-volatile memory array. (d) An MVM runs tile by tile: DACs convert the input into voltages, each tile computes a partial sum in analog, ADCs convert it back to digital, and the partial sums are accumulated digitally. (e) Within a tile, each device is programmed to a conductance $G_{i,j}$ encoding one weight $W_{i,j}$, and the current on output line j, $I_j = \sum_i V_i\, G_{i,j}$, is the dot product of the input with the j-th weight column.

Kirchhoff's current law sums them along the output lines to yield the dot products. Since all devices operate in parallel, the entire MVM is completed in a single physical step.

We consider every linear layer of the transformer block executed in analog, while the remaining operations stay in digital (Figure 1a). A crossbar has a fixed size of $S \times S$ devices ($S = 512$ in this work), so a weight matrix $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ is partitioned into tiles of S rows and S columns (Figure 1b,c). Each tile computes the partial dot products over its S input dimensions, and the partial sums of different tiles are added digitally (Figure 1d). Because device conductances are bounded by a maximum value $G_{\max}$, the weights of a tile are normalized before programming. This is done per output channel: for row r and the column group $\mathcal{G}$ of a tile, let $s_{r,\mathcal{G}} = \max_{j \in \mathcal{G}} |\widehat{W}_{r,j}|$ be the absolute maximum of that row segment. The normalized weight $\widetilde{W}_{r,j} = \widehat{W}_{r,j} / s_{r,\mathcal{G}} \in [-1, 1]$ is programmed as a conductance, with ± 1 corresponding to $\pm G_{\max}$, and the readout of row r is multiplied by $s_{r,\mathcal{G}}$ to recover the original scale. In what follows, we express all device-level quantities in these normalized units. Throughout, $\widehat{\boldsymbol{W}}$ denotes the target weights that a method sends to the hardware. The values actually stored are $\widetilde{\boldsymbol{W}}$ plus the programming noise introduced below. The target weights equal the pretrained $\boldsymbol{W}$ unless a post-training method has changed them.

Error sources. The crossbar operates in the analog domain while the rest of the computation is digital, so the inputs pass through a digital-to-analog converter (DAC) and the outputs through an analog-to-digital converter (ADC) (Figure 1d). This pipeline has three main sources of error: the DAC quantizes the inputs, programming the weights is imprecise, and the ADC quantizes the outputs. The DAC produces a finite set of voltage levels of fixed bit resolution, with a dynamic range computed per token and per tile (Dai et al., 2021). The ADC converts each column current into a digital value with a fixed number of bits and a fixed saturation level, beyond which currents are clipped. We model it as a uniform quantizer with a single static range shared by all tiles and layers, a step digital accelerators do not have (Zhang et al., 2024b). AIMC hardware exhibits further non-idealities, including read noise, conductance drift, and IR drop along the crossbar lines (Sebastian et al., 2020). Simulating them is prohibitive at the model sizes we study, and we restrict this work to the three sources above, which dominate accuracy degradation (Le Gallo et al., 2023).

Programming noise. A device cannot be set to its target conductance exactly: the value it reaches differs from the target by a random amount that is fixed once programmed but unknown beforehand.

In normalized units the stored value is $\widetilde{W}_{r,j} + \widetilde{E}_{r,j}$, and after the readout is multiplied by $s_{r,\mathcal{G}}$ the partial sum of row r becomes

$$s_{r,\mathcal{G}} \sum_{j \in \mathcal{G}} \left(\widetilde{W}_{r,j} + \widetilde{E}_{r,j}\right) x_j \;=\; \sum_{j \in \mathcal{G}} \left(\widehat{W}_{r,j} + s_{r,\mathcal{G}}\, \widetilde{E}_{r,j}\right) x_j. \tag{1}$$

The effective noise on the weight is thus $E_{r,j} = s_{r,\mathcal{G}}\, \widetilde{E}_{r,j}$, with standard deviation $\phi_{r,j} = s_{r,\mathcal{G}}\, \sigma\big(\widetilde{W}_{r,j}\big)$, where $\sigma(\cdot)$ is the device noise standard deviation in normalized units. The noise on every weight in a row segment is therefore proportional to the segment's largest magnitude: a single outlier raises the noise on all S weights that share its scale. The entries of $\widetilde{\mathbf{E}}$ are zero-mean, independent of each other, and independent of the input. We use two models for σ. The first is *additive Gaussian* noise with a constant $\sigma = \alpha$, which gives $\phi_{r,j} = \alpha\, s_{r,\mathcal{G}}$. The second is a *state-dependent* model in which σ depends on the programmed conductance, taken from measurements of a phase-change memory (PCM) based AIMC chip Le Gallo et al. (2023). Appendix B shows both models and details PCM equations.

Robustness to Analog Noise. Analog Foundation Models Büchel et al. (2025a) address the same deployment setting through hardware-aware retraining, with the non-idealities simulated during training. Retraining recovers much of the lost accuracy at a cost of 230 hours on 96 V100 GPUs for a 3.8B-parameter model, a cost that grows with model size. RAOQ Zhang et al. (2024b) also trains, using quantization-aware training with low-rank adaptation to make models robust to static output quantization, but neglects weight noise. NORA Hou et al. (2025) is training-free: it rescales activations per layer to shift the error burden from the converters onto the weights, and thus targets a different error source than we do. ROMER Zhou et al. (2026) calibrates the routers of mixture-of-experts models and is complementary to per-layer weight adaptation.

Post-Training Quantization. Post-training quantization (PTQ) adapts a pretrained model to low-precision digital inference without retraining. GPTQ Frantar et al. (2023) quantizes each layer to minimize its output error on a calibration set, compensating the rounding error of each weight through the weights not yet quantized, a mechanism we build on in Section 3.4. Other methods reshape the weight and activation distributions before quantization: SmoothQuant Xiao et al. (2024) migrates activation outliers into the weights through per-channel scaling, SpinQuant Liu et al. (2025) applies rotations, MagR Zhang et al. (2024a) reduces weight magnitudes through a proximal update, and OmniQuant Shao et al. (2024) learns clipping thresholds and equivalent scaling factors per block by gradient descent. All of these methods minimize the deterministic reconstruction error of the quantized layer. None model the stochastic perturbation of the weights, and their behavior under analog noise has not been systematically evaluated. Conversely, since quantization error is a range-dependent perturbation of the weights, our objective transfers to quantization, where the quantization ranges are obtained in closed form (Section 3.5).

3 Noise-Aware Post-Training Adaptation

3.1 Layer-Wise Optimization

Following state-of-the-art PTQ methods Frantar et al. (2023); Lin et al. (2026), we adapt the network for analog deployment one layer at a time, choosing each layer's deployed weights to preserve the layer's output on the calibration distribution. In contrast to standard PTQ, weights are perturbed by programming noise, which we account for in the objective function. For a linear layer, we minimize the expected output distortion

$$\mathcal{J}(\widehat{\boldsymbol{W}}) \;=\; \mathbb{E}_{\mathbf{x},\,\mathbf{E}} \left\| \boldsymbol{W}\mathbf{x} - (\widehat{\boldsymbol{W}} + \mathbf{E})\,\mathbf{x} \right\|^2, \tag{2}$$

where $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ is the pretrained weight matrix, $\mathbf{x} \in \mathbb{R}^d$ is the layer input drawn from the calibration distribution, and $\widehat{\boldsymbol{W}} \in \mathbb{R}^{n \times d}$ is the deployed weight matrix produced by the method under consideration. The random matrix $\mathbf{E} \in \mathbb{R}^{n \times d}$ models the programming noise: its entries are zero-mean and mutually independent, independent of $\mathbf{x}$, with standard deviations $\phi_{r,j}$ determined by the deployed weights through the noise model of Section 2. Because the noise is zero-mean and independent of the input, the cross term between the weight error $\boldsymbol{\Delta} := \boldsymbol{W} - \widehat{\boldsymbol{W}}$ and the noise

vanishes, and evaluating the expectations over the input second moment $\boldsymbol{H} := \mathbb{E}[\mathbf{x}\mathbf{x}^\top]$, estimated as $\boldsymbol{H} \approx \frac{1}{m}\boldsymbol{X}^\top\boldsymbol{X}$ from a calibration set $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ of m sampled layer inputs, gives (Appendix A.1)

$$\mathcal{J}(\widehat{\boldsymbol{W}}) \;=\; \underbrace{\sum_{r=1}^{n} \boldsymbol{\Delta}_{r,:}\,\boldsymbol{H}\,\boldsymbol{\Delta}_{r,:}^\top}_{\text{weight error}} \;+\; \underbrace{\sum_{r=1}^{n}\sum_{j=1}^{d} H_{j,j}\;\phi_{r,j}^2}_{\text{noise variance}}, \tag{3}$$

where the noise term involves only the diagonal of $\boldsymbol{H}$ because the noise entries are mutually independent. Both steps are exact consequences of the noise model. Since the noise magnitudes $\phi_{r,j}$ depend on the deployed weights, the noise variance can be reduced by modifying $\widehat{\boldsymbol{W}}$, which can in turn introduce weight error. Minimizing equation 3 balances the two.

3.2 Weight Clamping

Programming noise is set by the largest weight of each output channel of a tile: a single outlier raises the noise on all S weights that share its range. Clamping removes such outliers, reducing the range-dependent part of the noise on every weight of the channel at the price of a clipping error on the few weights that are clipped. The decomposition equation 3 lets us balance the two, and we choose the threshold of every output channel of every tile as the minimizer of this trade-off.

Objective. Consider one output channel of a tile with weights $\boldsymbol{w} \in \mathbb{R}^S$, and let $h_j := H_{j,j}$ be the corresponding diagonal entries of the input second moment $\boldsymbol{H}$. Clamping at a threshold c replaces each weight by $\widehat{w}_j(c) = \operatorname{sign}(w_j)\,\min\{|w_j|,\, c\}$, so that every weight with $|w_j| > c$ is clipped and the programmed range becomes c. The noise on weight j is then $\phi(\widehat{w}_j(c);\, c)$, which equals αc for the additive model and $c\,\sigma_{\mathrm{PCM}}\big(|\widehat{w}_j(c)|/c\big)$ for the PCM model. Substituting the clamped weights into equation 3, the noise term separates into one term per output channel of each tile, and we approximate the weight-error term by the diagonal of $\boldsymbol{H}$ (Appendix A.2). Each threshold is then the solution of an independent problem in the single variable c,

$$c^\star \;=\; \arg\min_{c>0}\; J(c), \qquad J(c) \;=\; \underbrace{\sum_{j:\,|w_j|>c} h_j\,\big(|w_j| - c\big)^2}_{\text{clipping error}} \;+\; \underbrace{\sum_{j=1}^{S} h_j\,\phi\big(\widehat{w}_j(c);\, c\big)^2}_{\text{noise variance}}. \tag{4}$$

Solution. The difficulty in equation 4 is that the set of clipped weights changes with c. We handle this by splitting the search range, sorting the magnitudes $|w_{(1)}| \geq \cdots \geq |w_{(S)}|$ and setting $|w_{(S+1)}| := 0$. As c decreases from $|w_{(1)}|$, the weights become clipped one at a time, so the sorted magnitudes divide the search range into S intervals $\mathcal{I}_p = [\,|w_{(p+1)}|,\, |w_{(p)}|\,]$, and on $\mathcal{I}_p$ exactly the p largest weights are clipped. Within one interval the set of clipped weights is fixed, so the restriction J_p of J to $\mathcal{I}_p$ is a smooth function of the threshold, and we minimize it on the interval. Because the intervals cover the whole search range and each candidate is optimal on its own interval, the best of the S candidates is the global minimizer of equation 4. The procedure runs in parallel over all output channels of a tile, and only the minimization within an interval depends on the noise model.

Additive noise. For $\phi(w; c) = \alpha c$ every weight carries the same noise variance $\alpha^2 c^2$, and on interval p the objective is the convex quadratic

$$J_p(c) \;=\; (A_p + \lambda)\,c^2 \;-\; 2B_p\,c \;+\; D_p, \qquad \lambda \;:=\; \alpha^2 \sum_{j=1}^{S} h_j, \tag{5}$$

with $A_p = \sum_{j\leq p} h_{(j)}$, $B_p = \sum_{j\leq p} h_{(j)}|w_{(j)}|$ and $D_p = \sum_{j\leq p} h_{(j)}|w_{(j)}|^2$. Its minimizer on the interval is

$$c_p \;=\; \operatorname{proj}_{\mathcal{I}_p}\!\left(\frac{B_p}{A_p + \lambda}\right), \tag{6}$$

where the projection takes the nearer endpoint if the stationary point lies outside $\mathcal{I}_p$, and $c^\star$ is the c_p with the smallest $J_p(c_p)$. A larger α lowers the threshold and clips more weights. Without noise nothing is clipped. The additive case is therefore solved exactly and in closed form: the global minimizer is the best of S closed-form candidates, at the cost of one sort and $O(S)$ arithmetic per output channel of a tile.

PCM noise. Under the PCM model the noise of an unclipped weight also depends on its own programmed value, so J_p is a polynomial in c and $1/c$ rather than a quadratic, and its stationary point has no closed form. The intervals, the coefficients from one sort and cumulative sums, and the best-of-S-candidates rule are unchanged. Only the minimization within an interval becomes numerical, by a safeguarded Newton iteration from a few seed points, which reaches the same thresholds as a dense grid at a small fixed number of evaluations per interval (Appendix C.1.1).

3.3 Input Scaling

Clamping reduces the noise-variance term of equation 3 at the expense of increased weight error. Input scaling reduces it without adding weight error, by redistributing magnitudes between weights and inputs while leaving the layer output unchanged. For a positive vector $\boldsymbol{s} \in \mathbb{R}^d_{>0}$ with entries s_j, the scaled layer has weights and input

$$\boldsymbol{W}' \;=\; \boldsymbol{W}\,\mathrm{diag}(\boldsymbol{s})^{-1}, \qquad \mathbf{x}' \;=\; \mathrm{diag}(\boldsymbol{s})\,\mathbf{x}, \tag{7}$$

so that $\boldsymbol{W}'\mathbf{x}' = \boldsymbol{W}\mathbf{x}$ for every input. As in SmoothQuant Xiao et al. (2024), we absorb the scale into the preceding normalization gain or linear layer offline, adding no computation or tile mapping changes. Since the noise is defined in normalized units, the relative noise and the output quantization of the preceding layer are unchanged (Appendix C.2). The scaled layer computes the same function as the original, so we apply the objective equation 3 to it, with $\boldsymbol{W}'$ and $\mathbf{x}'$ in place of $\boldsymbol{W}$ and $\mathbf{x}$ and, accordingly, the second moment of the scaled input $\mathrm{diag}(\boldsymbol{s})\,\boldsymbol{H}\,\mathrm{diag}(\boldsymbol{s})$ in place of $\boldsymbol{H}$. For input scaling, deploying $\widehat{\boldsymbol{W}} = \boldsymbol{W}'$ introduces no weight error and the objective reduces to the noise-variance term of the scaled layer. A scale acts on this term in two ways: dividing input channel j by s_j lowers the ranges, and hence the noise, of the output channels in which channel j holds the largest weight, while the same factor amplifies the channel's contribution through $s_j^2 H_{j,j}$. The scales balance the two and are free to move in either direction:

$$\boldsymbol{s}^\star \;=\; \arg\min_{\boldsymbol{s}>0} \sum_{r=1}^{n}\sum_{j=1}^{d} s_j^2\, H_{j,j}\; \phi_{r,j}\big(\boldsymbol{W}\,\mathrm{diag}(\boldsymbol{s})^{-1}\big)^2, \tag{8}$$

minimized by gradient descent with $H_{j,j}$ estimated once per layer from the calibration set. In our experiments, we additionally weight each output channel by its importance to the language-modeling loss of the full model. We replace $H_{j,j}$ in equation 8 with $u_r\,H_{j,j}$, where $u_r = \mathbb{E}[g_r^2]$ and g_r is the gradient of that loss with respect to output channel r.

3.4 WARP and WARP+

Input scaling and weight clamping act on the same quantity, the range of each output channel of a tile, along different axes. Scaling rescales an input channel across all rows and moves magnitude between weights and activations with no weight error, while clamping reduces the range within a row at the price of a small clipping error. We call this combination WARP (Weight-Range Adaptation for Robustness to Perturbations): The network is first transformed by input scaling, the input second moment is recomputed on the rescaled inputs, and the clamping thresholds are solved on the transformed weights. After scaling, fewer weights need clipping and the clipping error is smaller.

WARP leaves the clipping error uncorrected. WARP+ compensates it with the GPTQ update Frantar et al. (2023), applied to the clipping error instead of the rounding error. The thresholds are fixed as in WARP, which fixes every range and hence the noise term of equation 3, and the clipping error of each weight is redistributed over the not-yet-processed input channels of the same output channel. A weight pushed beyond its threshold by an update is clipped when its turn comes, so the ranges and noise term stay the same as WARP and only the weight-error term decreases.

3.5 Quantization Instantiation

The same framework covers weight quantization. A b-bit symmetric quantizer with range c places $L = 2^{b-1} - 1$ grid points on each side of zero, rounds each weight with $|w_j| \le c$ to the nearest grid point, $\frac{c}{L}\big\lfloor \frac{L}{c}\,w_j \big\rceil$, and clips each weight with $|w_j| > c$ to $\pm c$. The range trades a clipping error on the largest weights against a rounding error on all others, exactly as clamping trades it against

the programming noise. We consider two models of the rounding error. The *uniform* model treats it as a stochastic perturbation of known variance, uniform over one grid step and independent across weights. Its standard deviation is proportional to the range, $\phi = \alpha_b\, c$ with $\alpha_b = 1/(\sqrt{12}\, L)$, same as the additive noise model in Section 2, so the solver in Section 3.2 applies. The *exact* model computes the deterministic rounding error of each weight, which replaces the noise variance in equation 4:

$$J(c) \;=\; \underbrace{\sum_{j:\,|w_j|>c} h_j \left(|w_j| - c\right)^2}_{\text{clipping error}} \;+\; \underbrace{\sum_{j:\,|w_j|\le c} h_j \left(w_j - \tfrac{c}{L}\left\lfloor \tfrac{L}{c}\, w_j \right\rceil\right)^2}_{\text{rounding error}}. \tag{9}$$

A weight changes grid point only at finitely many values of c, so between consecutive breakpoints the rounded values are fixed and equation 9 is again a convex quadratic in c with a closed-form minimizer. The solver is the same as for clamping, except that each weight contributes up to L breakpoints instead of one, so the sweep covers $O(LS)$ intervals rather than S (Appendix C.1.2). Both models are instantiations of the same objective. In the exact model, our solver coincides with PiSO Amboage et al. (2026), a concurrent and independent work that arrives at the same piecewise closed-form sweep over rounding breakpoints for round-to-nearest quantization. Input scaling and error redistribution in WARP+ carry over with the rounding error instead of the programming noise.

4 Experiments

4.1 Experimental Setup

We evaluate off-the-shelf instruct checkpoints of the Qwen3 family Yang et al. (2025) (0.6B to 32B) using recommended sampling settings. We fix the generation seed and vary only the noise seed, isolating hardware-noise variance. The hardware simulation follows Section 2: tile size $S = 512$, 8-bit dynamic per-token, per-tile input quantization, 8-bit static output quantization of tile's partial sum, and programming noise from either the PCM model of Le Gallo et al. (2023) or the additive Gaussian model with α from 1% to 10%. All transformer-block linear layers are mapped to analog tiles (Figure 1a). The lm-head remains digital (Appendix F.1). Every noisy experiment reports the mean and standard deviation over 10 noisy instances. We evaluate on GSM8K Cobbe et al. (2021) with strict answer extraction, ARC-C Clark et al. (2018), GPQA-diamond Rein et al. (2023), HumanEval+ and MBPP+ Chen et al. (2021); Austin et al. (2021); Liu et al. (2023). Prompting and harness details are in Appendix D.1. Calibration uses 512 sequences of 2048 tokens from UltraChat Ding et al. (2023) and WARP+ runs layer by layer on a single A100 GPU, under 30 minutes for Qwen3-4B, and under 3 hours for Qwen3-32B—where the gradient pass of the Fisher weighting uses 4 GPUs (Appendix D.1). We denote a hardware setting by the triple Aa–Ww–Oo: a-bit input quantization, the state of the weights (W_{noisy} for analog weights with programming noise, Wb for b-bit quantized weights), and o-bit output quantization. The AIMC pipeline is A8–W_{noisy}–O8. *Noise-free* denotes evaluation with BF16 activations and weights. *Off-the-shelf* indicates the model weights have not been modified by any post-training method.

4.2 Comparison with Existing Post-Training Methods

Table 2 compares WARP against existing PTQ methods on Qwen3-14B under the AIMC pipeline. We run the PTQ methods of Section 2 unchanged and program their transformed weights onto analog tiles without quantization, exposing them only to PCM programming noise. GPTQ, which requires a quantizer, is programmed with its 4-bit weights and thus incurs quantization error too. For MagR we report MagR-tiled, a variant whose penalty acts per output channel of each tile and performs better than the original under analog noise. Baseline configurations are in Appendix D.1 and D.2.3.

Under the AIMC pipeline, the off-the-shelf model loses 5.8% on average. SpinQuant, MagR-tiled and GPTQ lose even more and SmoothQuant does not improve significantly. This shows that reducing weight ranges is not sufficient by itself: MagR-tiled shrinks the slices whose maxima set the noise, but the single penalty strength clips too much where the noise is small and too little where it is large. WARP sets each group's threshold from its own noise. OmniQuant helps, recovering 1.0%.

Input scaling alone reaches 75.2% and clamping alone 74.0%. Combined, WARP reaches 76.1%. The two are complementary: Scaling redistributes magnitude into the activations, and clamping

Table 2: Post-training methods on Qwen3-14B under the AIMC pipeline (A8–W_{noisy}–O8), unless indicated otherwise in the left-most column. Mean ± one standard deviation over 10 noise instances.

	Benchmark accuracy (%)					
Method	**GSM8K**	**ARC-C**	**GPQA-D**	**HumanEval+**	**MBPP+**	**Avg.**
noise-free off-the-shelf (W16)	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.8	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
off-the-shelf	71.4 ± 8.1	91.8 ± 0.6	51.9 ± 5.7	82.9 ± 1.6	71.4 ± 1.1	73.9 ± 1.6
GPTQ (A8–$W4_{noisy}$–O8)	62.5 ± 13.5	91.0 ± 0.4	48.0 ± 4.2	81.4 ± 2.4	70.3 ± 1.1	70.6 ± 2.4
SmoothQuant	72.2 ± 6.1	91.8 ± 0.5	52.0 ± 3.7	83.3 ± 1.6	71.1 ± 1.0	74.1 ± 1.1
SpinQuant	60.8 ± 11.1	88.4 ± 0.8	34.6 ± 5.3	71.4 ± 2.9	61.0 ± 9.4	63.2 ± 3.1
MagR-tiled	67.9 ± 7.8	**92.0** ± 0.4	52.5 ± 3.4	83.2 ± 2.1	71.0 ± 1.3	73.3 ± 1.5
OmniQuant	73.4 ± 5.3	91.8 ± 0.3	54.7 ± 2.9	83.2 ± 1.9	71.6 ± 0.9	74.9 ± 0.5
Input scaling	73.2 ± 6.1	91.7 ± 0.4	54.8 ± 3.9	84.3 ± 1.3	71.7 ± 1.0	75.2 ± 1.7
Clamping	70.1 ± 8.8	91.9 ± 0.4	52.8 ± 2.3	83.6 ± 1.7	71.8 ± 1.5	74.0 ± 1.9
Clamping + redistribution	73.3 ± 7.6	91.8 ± 0.6	52.3 ± 3.0	83.5 ± 1.6	71.5 ± 1.6	74.5 ± 1.8
WARP	77.4 ± 4.7	91.9 ± 0.3	**54.9** ± 2.5	**84.8** ± 1.7	71.4 ± 1.2	76.1 ± 1.3
WARP+	**79.0** ± 4.5	91.8 ± 0.4	53.8 ± 2.0	84.2 ± 1.8	**72.0** ± 1.9	**76.2** ± 1.1

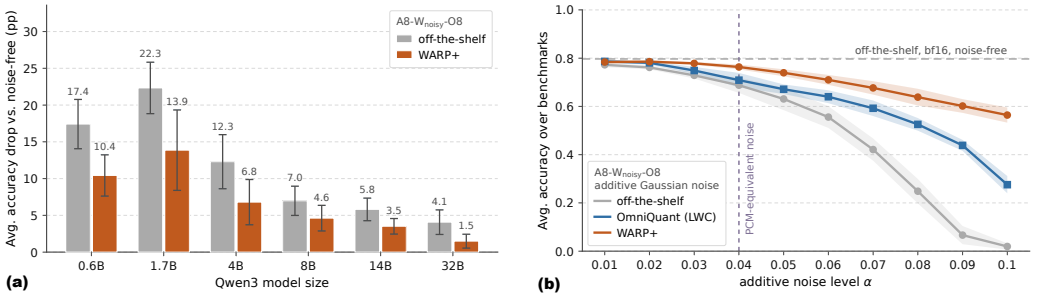


Figure 2: Robustness under the AIMC pipeline. (a) Accuracy drop of the Qwen3 family under *PCM* noise, averaged over the five benchmarks. (b) Average accuracy of Qwen3-14B under *additive Gaussian* noise of level α. WARP+ is adapted for each α, OmniQuant is calibrated once for 4 bits. The dashed vertical line marks the additive noise level ($\alpha = 4\%$) whose expected output distortion matches that of the PCM noise. Per-benchmark accuracies in Appendix E.1 and E.2.

the reduced ranges costs little clipping error. WARP+ reaches 76.2% and recovers 40% of the 5.8% degradation relative to only 17% recovery for OmniQuant, the best baseline. Since error redistribution never hurts, we use WARP+ in the remaining experiments.

4.3 Robustness Improves with Model Size and WARP+

Figure 2a shows accuracy drop of Qwen3 family under the AIMC pipeline, relative to the noise-free models. Off-the-shelf models degrade substantially, but the degradation shrinks with size. The 0.6B model deviates (17.4 pp) because its noise-free accuracy is close to chance on several benchmarks. WARP+ reduces the drop further at every size, and reaches within 1.5 pp of its noise-free accuracy at 32B. It also removes almost all of the loss due to the DAC and ADC (Appendix F.2). Robustness from model size and model adaptations combines, and at the largest scales, analog inference converges to the accuracy of digital inference.

4.4 WARP+ Robustness across Noise Magnitudes

To evaluate WARP+ at higher noise levels than PCM, we sweep the additive Gaussian model from $\alpha = 1\%$ to 10% on Qwen3-14B, adapting WARP+ at each level (Figure 2b). The off-the-shelf model and OmniQuant collapse beyond 7% and 9%, respectively. WARP+ degrades gradually across the entire sweep, and retains most of its accuracy on ARC-C, HumanEval+ and MBPP+ (Appendix E.2). The method generalizes well and remains effective over a wide range of noise levels.

Table 3: Weight-only quantization of Qwen3-4B at 4/3-bit (BF16 activations, group size 128). Entries are mean ± one standard deviation over 5 evaluation seeds. OmniQuant uses 4-bit Learnable Weight Clipping.WARP+ variants differ in rounding-error models used for thresholds (Section 3.5).

Method	GSM8K	ARC-C	GPQA-D	HumanEval+	MBPP+	Avg.
noise-free off-the-shelf (W16)	75.1 ± 1.8	87.7 ± 0.0	48.7 ± 1.8	76.4 ± 1.3	59.8 ± 1.0	69.5 ± 0.6
W4, group size 128						
GPTQ	74.5 ± 1.2	85.4 ± 0.2	45.5 ± 2.4	72.5 ± 1.5	56.7 ± 0.8	66.9 ± 0.5
OmniQuant	63.1 ± 2.8	85.5 ± 0.0	$\mathbf{46.1} \pm 1.2$	73.8 ± 1.3	59.9 ± 0.9	65.7 ± 0.4
WARP+ (uniform)	$\mathbf{81.6} \pm 1.1$	$\mathbf{86.2} \pm 0.1$	45.1 ± 2.1	$\mathbf{78.2} \pm 1.7$	$\mathbf{60.8} \pm 0.9$	$\mathbf{70.4} \pm 0.6$
WARP+ (exact)	71.2 ± 1.8	85.0 ± 0.1	44.1 ± 3.5	75.7 ± 1.0	58.6 ± 1.7	66.9 ± 0.6
W3, group size 128						
GPTQ	26.5 ± 2.0	76.0 ± 0.2	29.0 ± 1.7	55.4 ± 2.6	48.5 ± 2.4	47.1 ± 0.7
OmniQuant	19.4 ± 0.8	$\mathbf{82.4} \pm 0.1$	$\mathbf{33.5} \pm 1.0$	57.9 ± 0.8	$\mathbf{52.8} \pm 0.7$	49.2 ± 0.3
WARP+ (uniform)	54.3 ± 1.3	78.1 ± 0.0	24.5 ± 3.9	58.9 ± 2.9	45.2 ± 0.8	52.2 ± 1.2
WARP+ (exact)	$\mathbf{70.7} \pm 1.7$	82.0 ± 0.0	30.7 ± 2.5	$\mathbf{66.2} \pm 0.7$	52.1 ± 1.8	$\mathbf{60.3} \pm 0.7$

4.5 Transfer to Low-Bit Weight Quantization

We evaluate the quantization instantiation in Section 3.5 on Qwen3-4B at 4 and 3 bits with group size 128 against GPTQ and OmniQuant at the same settings (Table 3). The WARP+ variants differ only in the rounding-error model used to optimize the input scales and clamping thresholds.

At 4-bit the uniform variant is the best method on every benchmark except GPQA-D. The quantization is fine relative to the weight distribution, so the uniform model of the rounding error is accurate. At 3-bit the picture reverses: the uniform variant falls to 52.2%, while the exact variant reaches 60.3%. Controlling the weight range therefore matters beyond analog hardware: choosing the scales and thresholds under the objective, with thresholds solved exactly, is competitive with established quantization methods. The transfer requires only the perturbation model to be plugged into the objective and, for the exact variant, the corresponding breakpoints.

5 Discussion and Limitations

Standard PTQ targets a different error than what is present in AIMC hardware. Quantization error is deterministic and known once the quantized weights are fixed, allowing the realized error to be directly compensated. Programming noise, by contrast, is stochastic and unknown at adaptation time; only its statistics are available. Robustness therefore cannot rely on correcting a particular realization of the error, but must instead reduce the model's expected sensitivity to future perturbations. This distinction explains the poor transfer of conventional digital PTQ methods to analog hardware (Table 2). A noise-aware objective is critical: it balances weight clipping error against noise variance induced by the deployed weights, and applies to any noise model, including state-dependent ones. The benefit shows in the accuracy variation as well as the mean, since adaptation reduces the variation across noise instances (Section 4.2). This also serves to make accuracy of deployed models more predictable across repeated programmings. Replacing programming noise with rounding error in the same objective also yields a competitive low-bit quantization method.

Our study limitations include the fact that all results are simulated and only include three dominant error sources. We deliberately omit IR drop, conductance drift and read noise, which would add degradation on top of what we report. Robustness versus model size is also hard to measure cleanly, because task difficulty and model size interact: a fixed benchmark is easier for a larger model, so its smaller drop may reflect the easier task rather than greater robustness. GSM8K and GPQA-D remain demanding at every size and show the same trend, but a definitive test would use benchmarks whose difficulty is matched to each model size. The method also has limits: at $\alpha = 10\%$ WARP+ still loses 23 pp on average and falls to 26% on GSM8K and 36% on GPQA-D (Appendix E.2).Finally, we evaluate a single model family, Qwen3, and dense models only.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, to draft and format text, tables and figure captions, to find related work, for code validation and generation, and to check mathematical derivations. AI assistance was used for checking and helping to write the derivations in the appendix, in particular the safeguarded Newton solver for the PCM interval minimization and the breakpoint structure of the exact quantization instantiation. We have not used generative AI tools for the conception of the key ideas of this work, exact weight clamping and input scaling, for the derivation of the objective and its decomposition, or for the closed-form solution of the additive case. We have reviewed all AI-assisted work: LLM-generated code was verified and tested by the authors, all derivations checked with AI assistance were re-verified by hand, related-work descriptions were confirmed against the cited papers, and all AI-drafted text was revised and approved by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All models and data used in this work are public: the Qwen3 checkpoints, the UltraChat calibration data, and the lm-eval-harness and EvalPlus frameworks. The hardware simulation is fully specified: the tile mapping and normalization in Section 2, the PCM noise model with its polynomial coefficients and $G_{\max}$ in Appendix B, and the DAC and ADC settings in Appendix B.2. The methods are given in full: the objective and its decomposition in Section 3.1 with the derivation in Appendix A.1, the clamping solver in Section 3.2 with the PCM case in Appendix C.1.1 and the quantization case in Appendix C.1.2, input scaling in Section 3.3 with its optimization settings in Appendix C.2, and the redistribution step of WARP+ in Appendix C.1.3. The experimental protocol, including generation settings, prompting, layer mapping, noise seeds and compute, is described in Section 4.1 and Appendix D.1, and the configuration of every baseline in Appendix D.2. All results in the main text are reported as mean and standard deviation over 10 noise instances, and the per-benchmark and per-model numbers behind every figure and table are given in Appendix E.1, E.2. We will release the code for WARP and WARP+ upon publication.

REFERENCES

Juan Amboage, Pablo Monteagudo-Lago, Ian Colbert, Giuseppe Franco, and Nicholas Fraser. Optimal post-training quantization scales and where to find them, 2026. URL https://arxiv.org/abs/2606.10890.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Julian Büchel, Iason Chalas, Giovanni Acampa, An Chen, Omobayode Fagbohungbe, Sidney Tsai, Kaoutar El Maghraoui, Manuel Le Gallo, Abbas Rahimi, and Abu Sebastian. Analog foundation models, 2025a. URL https://arxiv.org/abs/2505.09663.

Julian Büchel, Athanasios Vasilopoulos, William Andrew Simon, Irem Boybat, HsinYu Tsai, Geoffrey W. Burr, Hernan Castro, Bill Filipiak, Manuel Le Gallo, Abbas Rahimi, Vijay Narayanan, and Abu Sebastian. Efficient scaling of large language models with mixture of experts and 3d analog in-memory computing. *Nature Computational Science*, Jan 2025b. ISSN 2662-8457. doi: 10.1038/s43588-024-00753-x. URL https://doi.org/10.1038/s43588-024-00753-x.

Geoffrey W. Burr, Robert M. Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, Lucas L. Sanches, Irem Boybat, Manuel Le Gallo, Kibong Moon, Jiyoo Woo, Hyunsang Hwang, and Yusuf Leblebici and. Neuromorphic computing using non-volatile memory. *Advances in Physics: X*, 2(1):89–124, 2017. doi: 10.1080/23746149.2016.1259585. URL https://doi.org/10.1080/23746149.2016.1259585.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,

Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge, 2018. URL https://arxiv.org/abs/1803.05457.

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. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *Proceedings of Machine Learning and Systems*, 3:873–884, 2021.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023. URL https://arxiv.org/abs/2305.14233.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Yayue Hou, Hsinyu Tsai, Kaoutar El Maghraoui, Tayfun Gokmen, Geoffrey W. Burr, and Liu Liu. Nora: Noise-optimized rescaling of llms on analog compute-in-memory accelerators. In *2025 Design, Automation Test in Europe Conference (DATE)*, pp. 1–7, 2025. doi: 10.23919/DATE64628.2025.10993217.

Han-Wen Hu, Wei-Chen Wang, Chung-Kuang Chen, Yung-Chun Lee, Bo-Rong Lin, Huai-Mu Wang, Yen-Po Lin, Yu-Chao Lin, Chih-Chang Hsieh, Chia-Ming Hu, Yi-Ting Lai, Han-Sung Chen, Yuan-Hao Chang, Hsiang-Pang Li, Tei-Wei Kuo, Keh-Chung Wang, Meng-Fan Chang, Chun-Hsiung Hung, and Chin-Yuan Lu. A 512gb in-memory-computing 3d-nand flash supporting similar-vector-matching operations on edge-ai devices. In *International Solid-State Circuits Conference (ISSCC)*, volume 65, pp. 138–140, 2022. doi: 10.1109/ISSCC42614.2022.9731775.

Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. Ten lessons from three generations shaped google's tpuv4i. In *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ISCA '21, pp. 1–14. IEEE Press, 2021. ISBN 9781450390866. doi: 10.1109/ISCA52012.2021.00010. URL https://doi.org/10.1109/ISCA52012.2021.00010.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Manuel Le Gallo, Riduan Khaddam-Aljameh, Milos Stanisavljevic, and et al. A 64-Core Mixed-Signal In-Memory Compute Chip Based on Phase-Change Memory for Deep Neural Network Inference. *Nature Electronics*, 6(9):680–693, 2023. doi: 10.1038/s41928-023-01010-1. URL https://doi.org/10.1038/s41928-023-01010-1.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration, 2026. URL https://arxiv.org/abs/2306.00978.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM quantization with learned rotations, 2025. URL https://arxiv.org/abs/2405.16406.

Joseph McDonald, Baolin Li, Nathan Frey, Devesh Tiwari, Vijay Gadepally, and Siddharth Samsi. Great power, great responsibility: Recommendations for reducing energy for training language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1962–1970. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-naacl.151. URL http://dx.doi.org/10.18653/v1/2022.findings-naacl.151.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof QA Benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature Nanotechnology*, 15(7):529–544, Jul 2020. doi: 10.1038/s41565-020-0655-z.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models, 2024. URL https://arxiv.org/abs/2308.13137.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, 2024. URL https://arxiv.org/abs/2211.10438.

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, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, 2025. URL https://arxiv.org/abs/2505.09388.

Aozhong Zhang, Naigang Wang, Yanxia Deng, Xin Li, Zi Yang, and Penghang Yin. MagR: Weight Magnitude Reduction for Enhancing Post-Training Quantization, 2024a. URL https://arxiv.org/abs/2406.00800.

Bonan Zhang, Chia-Yu Chen, and Naveen Verma. Reshape and adapt for output quantization (RAOQ): Quantization-aware training for in-memory computing systems. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=fM9xTkpAdu.

Wenyong Zhou, Yuannuo Feng, Yizhe Chen, Taiqiang Wu, Wendong Xu, Wenbo Qi, Zhengwu Liu, Wang Kang, and Ngai Wong. ROMER: Expert Replacement and Router Calibration for Robust MoE LLMs on Analog Compute-in-Memory Systems, 2026. URL https://arxiv.org/abs/2605.11800.

A Appendix

A.1 Derivation of the Error Decomposition

Throughout, $\widehat{\boldsymbol{W}}$ is deterministic, and the entries of $\mathbf{E}$ are zero-mean, mutually independent, and independent of $\mathbf{x}$, with standard deviations $\phi_{r,j}$.

Derivation of equation 3. Since $\boldsymbol{W}\mathbf{x} - (\widehat{\boldsymbol{W}} + \mathbf{E})\mathbf{x} = \boldsymbol{\Delta}\mathbf{x} - \mathbf{E}\mathbf{x}$, expanding the square in equation 2 gives

$$\mathcal{J}(\widehat{\boldsymbol{W}}) \;=\; \mathbb{E}\,\big\|\boldsymbol{\Delta}\mathbf{x}\big\|^2 \;-\; 2\,\mathbb{E}\big[(\boldsymbol{\Delta}\mathbf{x})^\top \mathbf{E}\,\mathbf{x}\big] \;+\; \mathbb{E}\,\big\|\mathbf{E}\mathbf{x}\big\|^2. \tag{10}$$

We show that the cross term is zero. By the tower property, we can first take the expectation over the noise with the input held fixed:

$$\mathbb{E}_{\mathbf{x},\,\mathbf{E}}\big[(\boldsymbol{\Delta}\mathbf{x})^\top \mathbf{E}\,\mathbf{x}\big] \;=\; \mathbb{E}_{\mathbf{x}}\Big[\,\mathbb{E}_{\mathbf{E}}\big[(\boldsymbol{\Delta}\mathbf{x})^\top \mathbf{E}\,\mathbf{x}\,\big|\,\mathbf{x}\big]\Big] \;=\; \mathbb{E}_{\mathbf{x}}\Big[(\boldsymbol{\Delta}\mathbf{x})^\top\,\mathbb{E}\big[\mathbf{E}\,\big|\,\mathbf{x}\big]\,\mathbf{x}\Big]. \tag{11}$$

In the second equality, the vectors $\boldsymbol{\Delta}\mathbf{x}$ and $\mathbf{x}$ are fixed by the conditioning and the expression is linear in $\mathbf{E}$, so the conditional expectation acts on $\mathbf{E}$ alone. Independence of $\mathbf{E}$ and $\mathbf{x}$ gives $\mathbb{E}[\mathbf{E} \mid \mathbf{x}] = \mathbb{E}[\mathbf{E}]$, which is zero because the noise is zero-mean. The cross term therefore vanishes, and equation 10 reduces to

$$\mathcal{J}(\widehat{\boldsymbol{W}}) \;=\; \mathbb{E}_{\mathbf{x}}\big\|\boldsymbol{\Delta}\mathbf{x}\big\|^2 \;+\; \mathbb{E}_{\mathbf{x},\,\mathbf{E}}\big\|\mathbf{E}\mathbf{x}\big\|^2. \tag{12}$$

It remains to evaluate the two expectations. Each squared norm in equation 12 is a sum over the n output rows. For the weight-error term, row r contributes

$$\mathbb{E}\Big[\big(\boldsymbol{\Delta}_{r,:}\,\mathbf{x}\big)^2\Big] \;=\; \boldsymbol{\Delta}_{r,:}\,\mathbb{E}\big[\mathbf{x}\mathbf{x}^\top\big]\;\boldsymbol{\Delta}_{r,:}^\top \;=\; \boldsymbol{\Delta}_{r,:}\,\boldsymbol{H}\,\boldsymbol{\Delta}_{r,:}^\top, \tag{13}$$

which is the first sum in equation 3. For the noise term, expanding the square of row r and using the independence of $\mathbf{E}$ and $\mathbf{x}$ to factor each summand,

$$\mathbb{E}\Big[\big(\mathbf{E}_{r,:}\,\mathbf{x}\big)^2\Big] \;=\; \sum_{j=1}^{d}\sum_{k=1}^{d}\mathbb{E}\big[\mathbf{E}_{r,j}\,\mathbf{E}_{r,k}\big]\;\mathbb{E}\big[\mathrm{x}_j\,\mathrm{x}_k\big]. \tag{14}$$

Because the noise entries are zero-mean and mutually independent,

$$\mathbb{E}\big[\mathbf{E}_{r,j}\,\mathbf{E}_{r,k}\big] \;=\; \begin{cases}\phi_{r,j}^2 & j = k,\\ 0 & j \neq k,\end{cases} \tag{15}$$

so only the diagonal terms of equation 14 survive, and with $\mathbb{E}[\mathrm{x}_j^2] = H_{j,j}$,

$$\mathbb{E}\Big[\big(\mathbf{E}_{r,:}\,\mathbf{x}\big)^2\Big] \;=\; \sum_{j=1}^{d}\phi_{r,j}^2\,H_{j,j}. \tag{16}$$

Summing both contributions over the rows yields equation 3.

A.2 Diagonal Approximation of the Weight-Error Term

The weight-error term of equation 3 for one output channel is the quadratic form $\boldsymbol{\Delta}_{r,:}\,\boldsymbol{H}\,\boldsymbol{\Delta}_{r,:}^\top$, which couples the errors of different input channels through the off-diagonal entries of $\boldsymbol{H}$. For clamping, the error of channel j is $\delta_j(c) = \operatorname{sign}(w_j)\,(|w_j| - c)_+$, and the exact term is $\sum_{j,k}\delta_j(c)\,H_{j,k}\,\delta_k(c)$. Section 3.2 keeps only the diagonal, $\sum_j h_j\,\delta_j(c)^2$ with $h_j = H_{j,j}$, which drops the cross terms between clipped weights. This has two consequences. First, the objective of each output channel

of a tile depends only on its own weights and on the diagonal of $\boldsymbol{H}$, so all thresholds of a layer are solved independently and in parallel. Second, the per-interval coefficients of the solver are simple sums over the clipped weights, which gives the closed form of equation 6 and the $O(S)$ cost. The noise term needs no such approximation: it involves only the diagonal of $\boldsymbol{H}$ exactly, because the noise entries are independent (Appendix A.1). The same diagonal approximation is made by GPTQ when deciding how to round each weight, and by the uniform and exact quantization instantiations of Section 3.5, whose rounding errors are likewise coupled through $\boldsymbol{H}$ only in the dropped cross terms. The cross terms are recovered to the extent that the GPTQ update of WARP+ uses the full $\boldsymbol{H}^{-1}$ when redistributing the clipping error.

B Noise Models

B.1 Programming noise

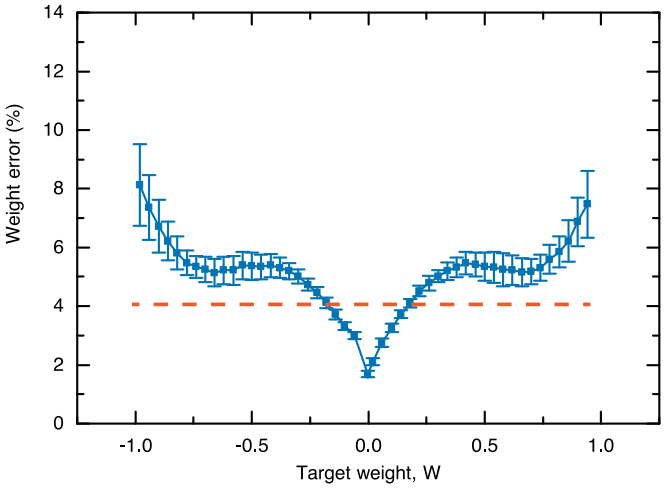


Figure 3: Programming noise of the two models as a function of the normalized target weight $\widetilde{W}$. Solid blue: state-dependent PCM model measured on the IBM HERMES project chip (Le Gallo et al., 2023); error bars are one standard deviation across its 64 cores. Dashed orange: additive Gaussian model with constant $\sigma = 4\%$, independent of the programmed value. The PCM noise is lowest near zero and highest at $|\widetilde{W}| = 1$, the value to which clamped weights are mapped.

State-dependent PCM model. The state-dependent model describes the programming noise of phase-change-memory devices as measured on the IBM HERMES Project Chip (Le Gallo et al., 2023). For each target value, the measurement records how far the conductance reached after programming deviates from the target, normalized by the maximum conductance. Figure 3 plots the standard deviation of this deviation against the normalized target weight $\widetilde{W}$: it is smallest for weights near zero and grows toward the ends of the range. The spread across the 64 cores of the chip is shown as error bars. In this chip each weight is stored in a unit cell of four PCM devices, two per polarity (Le Gallo et al., 2023); we assume that a weight is encoded by two devices, which lowers its programming noise by about $\sqrt{2}$ compared with a single device.

The measured curve is fitted by a cubic polynomial in the target conductance G, $\sigma_G(G) = 1.23 \times 10^{-5}\, G^3 - 3.06 \times 10^{-3}\, G^2 + 2.45 \times 10^{-1}\, G + 2.11$, with G and σ_G in μS and $G_{\max} = 180\,\mu$S. Since a normalized weight $\widetilde{W} \in [-1, 1]$ is programmed as the conductance $G = G_{\max}\,|\widetilde{W}|$, and the noise in normalized units is $\sigma_G / G_{\max}$, the same model expressed directly in normalized weights reads

$$\sigma_{\text{PCM}}\big(\widetilde{W}\big) \;=\; 0.3995\,\big|\widetilde{W}\big|^3 \;-\; 0.5499\,\big|\widetilde{W}\big|^2 \;+\; 0.2453\,\big|\widetilde{W}\big| \;+\; 0.01172, \tag{17}$$

which is smallest near $\widetilde{W} = 0$ and largest at $|\widetilde{W}| = 1$. This is the noise we simulate; the noise on the weight is then $\phi_{r,j} = s_{r,\mathcal{G}}\,\sigma_{\text{PCM}}(\widetilde{W}_{r,j})$, as in Section 2.

Additive Gaussian model. The additive model is the state-independent counterpart of the PCM model: every weight receives noise of the same standard deviation $\sigma = \alpha$, regardless of its programmed value, so that $\widetilde{E}_{r,j} \sim \mathcal{N}(0, \alpha^2)$ and $\phi_{r,j} = \alpha\, s_{r,\mathcal{G}}$. This noise model is not tied to a particular device technology, so results under this model indicate how our findings transfer to AIMC hardware other than PCM. In this figure we show $\alpha = 4\%$.

B.2 Input and Output Quantization Settings

The DAC quantizes the input of each tile to 8 bits with a dynamic range, computed per token and per tile from the absolute maximum of the input values entering that tile. The ADC quantizes the partial sum of each tile to 8 bits with a static range: the normalized MVM output is clipped to $[-12, 12]$ and uniformly quantized within it, with the same range for every output channel of every tile in every layer, and this quantization is applied to each tile's partial sum before the partial sums of the tiles are accumulated digitally. The ADC range is a hardware specification and not a tunable parameter: it is fixed once for the whole evaluation and is neither adjusted per model nor per benchmark. The value of 12 is chosen to cover the typical distribution of normalized partial sums: with weights and inputs normalized to $[-1, 1]$ and roughly Gaussian, the partial sum over $S = 512$ inputs has a standard deviation of about 2.5, so the range corresponds to roughly five standard deviations.

C Supplementary Methods

C.1 Weight Clipping Details

C.1.1 Minimization of the PCM Clamping Objective

Fix one output channel of a tile and drop its index: $\boldsymbol{w} \in \mathbb{R}^S$ are its weights and $h_j \geq 0$ the corresponding diagonal entries of $\boldsymbol{H}$. Under the PCM model a deployed weight $\widehat{w}_j = \operatorname{sign}(w_j) \min\{|w_j|, c\}$ has normalized value $\widehat{w}_j/c \in [-1, 1]$, since the range of the channel is c, and is programmed at the conductance $G_j = |\widehat{w}_j|\, G_{\max}/c$. The programming noise on a conductance G has standard deviation given by the fitted cubic $\Pr(G) = a_3 G^3 + a_2 G^2 + a_1 G + a_0$ (Section 2). In weight units the noise standard deviation of weight j is

$$\phi_j(c) \;=\; \frac{c}{G_{\max}}\, \Pr\!\left(\frac{|\widehat{w}_j|\, G_{\max}}{c}\right), \tag{18}$$

and the clamping objective is equation 4 with this ϕ_j in place of αc:

$$J(c) \;=\; \sum_{j:\,|w_j|>c} h_j \left(|w_j| - c\right)^2 \;+\; \sum_{j=1}^{S} h_j\, \phi_j(c)^2, \tag{19}$$

What is unchanged. A clipped weight is deployed at $\pm c$ and programmed at $G_{\max}$, so its noise is $\phi_j = \sigma c$ with $\sigma := \Pr(G_{\max})/G_{\max}$: it has the range-proportional form of the additive model, with σ in the role of α. An unclipped weight is deployed at w_j, and substituting $G_j = |w_j| G_{\max}/c$ into equation 18 gives,

$$\phi_j(c) \;=\; a_3 G_{\max}^2\, \frac{|w_j|^3}{c^2} \;+\; a_2 G_{\max}\, \frac{|w_j|^2}{c} \;+\; a_1\, |w_j| \;+\; \frac{a_0}{G_{\max}}\, c, \qquad |w_j| \leq c. \tag{20}$$

Each power of G in the noise curve turns into one power of $1/c$, because a smaller range programs the same weight at a higher conductance, and the conversion to weight units contributes one power of c. On any range of c over which the clipped set is fixed, both equation 20 and σc are smooth in c, so the objective changes form only when a weight becomes clipped. The interval structure of Section 3.2 therefore applies unchanged: the breakpoints are the S sorted magnitudes $|w_{(1)}| \geq \cdots \geq |w_{(S)}|$, on the interval $\mathcal{I}_p = (|w_{(p+1)}|, |w_{(p)}|]$ exactly the p largest weights are clipped, and J is continuous at every breakpoint. At $c = |w_{(p)}|$ the weight $w_{(p)}$ sits exactly at the range and is programmed at $G_{\max}$, so both expressions give the same noise: its clipping error is zero, and equation 20 evaluated at $|w_j| = c$ reduces to σc, the clipped value. The minimizer over $(0, \|\boldsymbol{w}\|_\infty]$ is then the best of the S per-interval minimizers. In the additive case each J_p is a convex quadratic, and its per-interval minimizer is the projected stationary point of equation 6. Under the

PCM model J_p can have several local minima within an interval, so its per-interval minimizer is found by a one-dimensional search over the interval, described below, and the result is exact up to the accuracy of that search. As in the additive case we restrict the search to $c \le \|\boldsymbol{w}\|_\infty$. Under the PCM model this is a design choice rather than a consequence of the objective: the terms in c^{-1} and c^{-2} keep decreasing beyond $\|\boldsymbol{w}\|_\infty$, so a range larger than the largest weight, which programs every weight, including the largest, below $G_{\max}$, could lower the noise further at no clipping cost. We leave this extension to future work.

The function on each interval. Write A_p, B_p, D_p for the sums over the p clipped weights of equation 5 and define the corresponding moments of the $S-p$ unclipped weights,

$$M_{p,m} \;:=\; \sum_{j>p} h_{(j)}\,|w_{(j)}|^m, \qquad m = 0,\dots,6. \tag{21}$$

Expanding the squares in equation 19 on $\mathcal{I}_p$ gives

$$J_p(c) \;=\; A_p\,c^2 \;-\; 2B_p\,c \;+\; D_p \;+\; N_p(c), \qquad N_p(c) \;=\; \sigma^2 A_p\,c^2 \;+\; \sum_{m=0}^{6} \kappa_m\,M_{p,m}\,c^{\,2-m}. \tag{22}$$

The noise of an unclipped weight is the fitted cubic evaluated at its normalized value $|w_j|/c$ and converted back to weight units by the factor $c/G_{\max}$, which is how the four powers c^{-2}, c^{-1}, c^0, c^1 of equation 20 arise. Squaring it multiplies every pair of these terms, so the products collect by total degree m in $|w_j|/c$ into the seven powers c^{2-m}, with the constants

$$\begin{aligned}
&\kappa_0 = \frac{a_0^2}{G_{\max}^2}, && \kappa_1 = \frac{2a_1a_0}{G_{\max}}, && \kappa_2 = a_1^2 + 2a_2a_0, \quad \kappa_3 = 2G_{\max}\,(a_3a_0 + a_2a_1),\\
&\kappa_4 = G_{\max}^2\,(a_2^2 + 2a_3a_1), && \kappa_5 = 2G_{\max}^3\,a_3a_2, && \kappa_6 = G_{\max}^4\,a_3^2.
\end{aligned} \tag{23}$$

The first three terms of equation 22 are the clipping error, which is the same as in the additive case since clipping does not depend on the noise model. N_p is the noise variance of all S weights and generalizes the additive λc^2: the first term is the noise of the p clipped weights, the sum that of the $S-p$ unclipped ones, and it runs over the powers $c^2, c^1, \dots, c^{-4}$. With $a_1 = a_2 = a_3 = 0$ only $\kappa_0 = a_0^2/G_{\max}^2 = \sigma^2$ survives and $N_p = \sigma^2(A_p + M_{p,0})\,c^2 = \lambda c^2$, recovering equation 5. As in the additive case, all ten sums are obtained for every p at once from one sort and cumulative sums along the sorted order, here seven of them ($m = 0,\dots,6$), so setting up all S interval functions costs $O(S)$. The coefficients $\kappa_1,\dots,\kappa_5$ can have either sign, so J_p is in general not convex on its interval and can have several local minima.

Minimizing one interval. What remains is a standard one-dimensional minimization. On the closed interval $\overline{\mathcal{I}}_p = [\,|w_{(p+1)}|,\,|w_{(p)}|\,]$, J_p is a finite sum of powers c^k, $k = -4,\dots,2$, given explicitly by equation 22; the interval excludes $c = 0$ whenever negative powers are present, so J_p is twice continuously differentiable with bounded derivatives, and its value and derivatives are available exactly at negligible cost. Its stationarity condition $J_p'(c) = 0$, multiplied by c^5, is a polynomial equation of degree six with no closed-form solution, so we minimize numerically, in two ways.

Dense grid. Evaluate J_p at K equally spaced points of the interval and keep the best. This locates the minimizer to within the grid spacing at $O(K)$ evaluations per interval, and serves as the reference.

Safeguarded Newton. The intervals are the gaps between consecutive sorted magnitudes of a channel and are therefore narrow, so on each of them J_p is well approximated by its second-order Taylor expansion, a parabola, and a few seed points already lie close to the minimizer. This makes Newton's method the natural choice. Evaluate J_p at n_s equally spaced points of the interval and take the best as $c^{(0)}$. Then apply n_N Newton steps on J_p',

$$c^{(t+1)} \;=\; \mathrm{proj}_{\mathcal{I}_p}\!\left(c^{(t)} - \frac{J_p'(c^{(t)})}{J_p''(c^{(t)})}\right), \qquad J_p'(c) = \sum_k k\,\gamma_{p,k}\,c^{k-1}, \quad J_p''(c) = \sum_k k(k-1)\,\gamma_{p,k}\,c^{k-2}, \tag{24}$$

where $\gamma_{p,k}$ denotes the coefficient of c^k in equation 22, $k = -4,\dots,2$. A step is taken only where $J_p''(c^{(t)}) > 0$ and skipped otherwise, every iterate is projected back into the interval, and the final

iterate is kept only if it lowers J_p below its value at $c^{(0)}$. The candidate is therefore never worse than the best of the seed points, and where J_p is locally convex it converges to the nearby minimizer at Newton's quadratic rate. The accuracy of the search is set by the seed spacing, since a local minimum that no seed lies close to can be missed. Each evaluation of J_p, J_p' or J_p'' is a few multiply-adds on the seven coefficients, so the cost is $O\big(S\,(n_s + n_N)\big)$ per output channel, batched over all intervals, channels and tiles. For all WARP experiments we use $n_s = 8$ seed points and $n_N = 4$ Newton steps per interval.

C.1.2 Exact Minimization of the Quantization Objective

Fix one output channel of a tile and drop its index: $\boldsymbol{w} \in \mathbb{R}^S$ are its weights and $h_j \geq 0$ the corresponding diagonal entries of $\boldsymbol{H}$. For a range c the grid points are $\pm k\,c/L$, $k = 0, \ldots, L$. A weight with $|w_j| \leq c$ is rounded to the nearest grid point and a weight with $|w_j| > c$ is clipped to the top grid point $\pm c$. In both cases the weight is deployed at $\operatorname{sign}(w_j)\, q_j(c)\, c/L$, where

$$q_j(c) \;:=\; \min\Big\{L,\; \Big\lfloor \frac{L\,|w_j|}{c} \Big\rceil \Big\} \;\in\; \{0, 1, \ldots, L\} \tag{25}$$

is its *level*, the index of the grid point it is deployed at. We also say that the weight is in *bin* $q_j(c)$. A clipped weight has level L. Since the sign cancels inside every square, the objective equation 9 reads

$$J(c) \;=\; \underbrace{\sum_{j:\,|w_j|>c} h_j\,\big(|w_j| - c\big)^2}_{\text{clipping error}} \;+\; \underbrace{\sum_{j:\,|w_j|\leq c} h_j\,\Big(|w_j| - \tfrac{q_j(c)}{L}\,c\Big)^2}_{\text{rounding error}}. \tag{26}$$

The clipping term is the same as in the additive objective. The second term is where the two models differ. In the additive model every weight carries the same noise variance $\alpha^2 c^2$, so the noise term $\alpha^2 c^2 \sum_j h_j$ is one quadratic in c with a fixed coefficient. In the quantization model a clipped weight sits exactly on the grid point $\pm c$ and has no rounding error, and each unclipped weight has its own rounding error $(|w_j| - q_j(c)\,c/L)^2$, which is a quadratic in c only as long as its level $q_j(c)$ does not change. The objective is therefore again piecewise quadratic, but its pieces start and end at different values of c than in the additive case. The method is the same: find the breakpoints, solve each piece in closed form, keep the best candidate.

Step 1: breakpoints. Picture c starting very large and decreasing towards zero. We cut the c-axis into intervals on which J is one fixed quadratic. The values of c at which J changes form are the *breakpoints*.

A clipped weight has level L and is deployed at $\pm c = \pm\frac{L}{L}\,c$, so both terms of equation 26 have the same form: every weight, clipped or not, contributes $h_j(|w_j| - \frac{q_j}{L}\,c)^2$. As long as its level q_j is fixed, this is a convex quadratic in c with fixed coefficients. So as long as no weight changes level, J is one convex parabola, whose minimizer has a closed form, and the breakpoints are the values of c at which some weight changes level. A weight changes level because the bins shrink with c while the weight stays where it is, so at some point it finds itself in the next bin.

The magnitudes $|w_j|$ are not breakpoints. This differs from the additive case, where the breakpoints are the S magnitudes. There, weight j contributes $h_j\alpha^2c^2 + h_j \max(|w_j| - c,\, 0)^2$: the clipping term switches on at $c = |w_j|$ and changes the coefficients of the quadratic. In the quantization model nothing switches on at $c = |w_j|$, because becoming clipped does not change the level. The grid has a point exactly at $\pm c$, and for c slightly above $|w_j|$ this is the grid point nearest to the weight. Just before it is clipped the weight is rounded to $\pm c$, and just after it is clipped to $\pm c$. Both times it has level L and contributes $h_j(|w_j| - c)^2$. Crossing $c = |w_j|$ moves the weight from one sum of equation 26 to the other without changing J.

The level $q_j(c)$ makes J non-convex. For a fixed level, the term $h_j(|w_j| - \frac{q_j}{L}\,c)^2$ is a convex quadratic in c. But the level is itself a function of c: $q_j(c)$ is a step function that jumps by one at every breakpoint, and each jump replaces the quadratic by a different one. The resulting function of c is not convex: the error of a single weight is zero at every $c = L|w_j|/k$, $k = 1, \ldots, L$, where the weight lies exactly on grid point k, and positive in between. The additive model has no such dependence, since its noise term $\alpha^2 c^2 \sum_j h_j$ involves no level, and its J is convex. Here J can have many local minima, and every interval has to be examined (Step 3).

Breakpoints of one weight. The boundary between bin k and bin $k+1$ is the midpoint $(k+\frac{1}{2})\,c/L$ between their grid points. The weight does not move. The midpoints do, since they are proportional to c. As c decreases, every midpoint slides toward zero and passes over the weight, in the order $k = 0, 1, \ldots, L-1$. The weight moves from bin k to bin $k+1$ when the midpoint reaches it, $(k+\frac{1}{2})\,c/L = |w_j|$, that is, at

$$c_{j,k} \;=\; \frac{L\,|w_j|}{k+\frac{1}{2}}, \qquad k = 0, 1, \ldots, L-1. \tag{27}$$

Each nonzero weight has exactly L breakpoints, $c_{j,0} > c_{j,1} > \cdots > c_{j,L-1}$, and climbs through the bins $0, 1, \ldots, L$ in order without returning to a lower one. The last one, $c_{j,L-1} = L|w_j|/(L-\frac{1}{2})$, is larger than $|w_j|$: the weight reaches the top bin before it becomes clipped, as explained above. A zero weight stays in bin 0 and has no breakpoints. Computing equation 27 for every weight of a channel gives LS breakpoints in total, which together partition the search range into the intervals on which the rounded values are fixed.

Example. Take $b = 3$, so $L = 3$, and $|w_j| = 0.6$. The breakpoints are 3.6, 1.2 and 0.72: the weight is at level 0 for $c > 3.6$, level 1 on $(1.2,\ 3.6]$, level 2 on $(0.72,\ 1.2]$ and level 3 on $(0,\ 0.72]$. On the last interval it contributes $h_j(0.6-c)^2$ throughout, as a rounded weight above $c = 0.6$ and as a clipped weight below, so 0.6 is not a breakpoint. Its error is zero at $c = 1.8$, 0.9 and 0.6 and positive in between.

The grid of intervals. Collecting the breakpoints of all nonzero weights and sort them in descending order we get the grid of intervals,

$$t_{(1)} \;\geq\; t_{(2)} \;\geq\; \cdots \;\geq\; t_{(K)} \;>\; 0, \qquad K \leq L\,S, \tag{28}$$

with $t_{(K+1)} := 0$. The intervals $\mathcal{I}_i := (t_{(i+1)},\ t_{(i)}]$ partition $(0, t_{(1)}]$, where $t_{(1)} = 2L\,\|\boldsymbol{w}\|_\infty$. By construction no breakpoint lies inside an interval, so no level $q_j(c)$ changes for any weight in the tile. On $\mathcal{I}_i$ every $q_j(c)$ is a constant, and J is a single convex quadratic. Coinciding breakpoints produce empty intervals and are harmless.

Step 2: one quadratic on every interval. On $\mathcal{I}_i$ no level changes, so each $q_j(c)$ equals a constant, which we write $q_j^{(i)}$. Every weight contributes $h_j(|w_j| - \frac{q_j^{(i)}}{L}\,c)^2$. Expanding the square and summing over the weights gives

$$J_i(c) \;=\; \Lambda_i\,c^2 \;-\; 2\,M_i\,c \;+\; N, \qquad c \in \mathcal{I}_i, \tag{29}$$

with

$$\Lambda_i \;=\; \sum_j h_j \Big(\frac{q_j^{(i)}}{L}\Big)^2, \qquad M_i \;=\; \sum_j h_j\,\frac{q_j^{(i)}}{L}\,|w_j|, \qquad N \;=\; \sum_j h_j\,|w_j|^2. \tag{30}$$

The sums run over all weights. A clipped weight enters with $q_j^{(i)} = L$. Only Λ_i and M_i depend on the interval. N is the same constant everywhere, so it affects neither the minimizer on an interval nor the comparison between intervals.

Step 3: closed-form minimizer and global optimum. Since $h_j \geq 0$, $\Lambda_i \geq 0$ and J_i is a convex parabola. Its minimizer on the interval is the M_i/Λ_i, moved to the nearest end of the interval if it falls outside:

$$c_i \;=\; \mathrm{proj}_{\mathcal{I}_i}\!\left(\frac{M_i}{\Lambda_i}\right), \qquad c^\star \;=\; \operatorname*{arg\,min}_{c\,\in\,\{c_1,\ldots,c_K\}} J(c). \tag{31}$$

If $\Lambda_i = 0$, J_i is constant and we take $c_i = t_{(i)}$. For $c > t_{(1)}$ every weight is rounded to zero and $J \equiv N = J(t_{(1)})$, so the search can be restricted to $(0, t_{(1)}]$, which the intervals partition. Because J is continuous and each c_i minimizes J on the closure of its interval, the best of the K candidates is the global minimizer of equation 9. Since J is non-convex (Step 1), all K intervals have to be examined. Since $\|\boldsymbol{w}\|_\infty$ lies in one of the intervals, $J(c^\star) \leq J(\|\boldsymbol{w}\|_\infty)$: the result is never worse than absmax quantization.

Asymmetric integer grid. Our quantization backend deploys the integer grid $\{-2^{b-1}, \ldots, 2^{b-1}-1\} \cdot s$, which has one extra negative level. The solver handles it by giving negative weights the top level $L+1$ instead of L. The derivation is otherwise unchanged.

C.1.3 WARP+: GPTQ-STYLE UPDATE DETAILS

We use the GPTQ update as implemented replacing the rounding error of each weight by its clipping error for the analog results, and by the combined clipping and rounding error for the quantization results. The update redistributes the error of a weight over all remaining input channels of its output channel, across tile boundaries, so it changes weights whose thresholds were solved on their original values. The thresholds, however, are solved per output channel and tile, and the solver does not account for this later redistribution. We therefore compared the two ways of handling this that the original GPTQ implementation (Frantar et al., 2023) offers. In the first, the thresholds of all tiles are solved on the original weights, and the update then processes input channels in decreasing order of the diagonal entry of $\boldsymbol{H}$ (GPTQ's activation ordering), regardless of tile membership. In the second, the parameters of each group are solved when the update reaches it, on the current weights (GPTQ's non-static groups): the threshold of a tile is solved on the current weights, its error is redistributed, and the next tile is processed on the updated weights. Since the tile membership of a weight is fixed by the hardware, we apply the activation ordering at tile granularity in this case, processing tiles in decreasing order of the mean diagonal entry of $\boldsymbol{H}$ over their input channels. For the analog results the two orderings give the same accuracy, and we use the first, which is faster and lets the thresholds of all tiles be solved in parallel. For the quantization results the second ordering is better, and we use it there.

C.2 INPUT SCALING DETAILS

C.2.1 ABSORBING THE SCALES

The scale $\mathrm{diag}(\boldsymbol{s})$ applied to the input of a layer is absorbed offline into the operation that produces that input: the gain of the preceding normalization layer, or the output channels of the preceding linear layer. Layers that read the same input, such as the query, key and value projections, share one scale. Absorbing the scale into a preceding linear layer multiplies each of its output channels by a constant. Since the noise of Section 2 is defined in normalized units, the normalized weights of that output channel are unchanged, and its noise is multiplied by the same constant as its weights. The relative noise of the preceding layer is therefore unchanged, and so is its output quantization, since the ADC operates on the normalized partial sums, which the scaling leaves unchanged. The only programming noise that depends on $\boldsymbol{s}$ is that of the scaled layer itself.

C.2.2 OPTIMIZATION

The objective equation 8 is invariant to multiplying all scales by the same constant, since the weights shrink by exactly the factor by which the inputs grow. To remove this redundancy we normalize the scales to unit mean. We parametrize $\boldsymbol{s} = \exp(\boldsymbol{u})$ to keep the scales positive, initialize $\boldsymbol{u} = \mathbf{0}$, and minimize over $\boldsymbol{u}$ with Adam for 200 steps [learning rate], with $H_{j,j}$ and u_r estimated once per layer from the calibration set.

C.2.3 INPUT SCALING FOR QUANTIZATION

Input scaling carries over to the quantization instantiation of Section 3.5 with the rounding error in place of the programming noise; the transform and its absorption are unchanged, and only the objective differs. For the scaled weights $\boldsymbol{W}\,\mathrm{diag}(\boldsymbol{s})^{-1}$, each output channel of a tile is quantized with the range $c_{r,\mathcal{G}}(\boldsymbol{s}) = \max_{j\in\mathcal{G}} |W_{r,j}|/s_j$ that covers its largest entry. Under the uniform model the rounding error is a stochastic perturbation with $\phi = \alpha_b\, c_{r,\mathcal{G}}(\boldsymbol{s})$, so the scales are found from the objective equation 8 exactly as in the analog case. Under the exact model the scales minimize the actual rounding error,

$$\boldsymbol{s}^\star \;=\; \arg\min_{\boldsymbol{s}>0} \sum_{r=1}^{n}\sum_{j=1}^{d} s_j^2\, H_{j,j} \left(\frac{W_{r,j}}{s_j} - \frac{c_{r,\mathcal{G}(j)}(\boldsymbol{s})}{L} \left\lfloor \frac{L}{c_{r,\mathcal{G}(j)}(\boldsymbol{s})} \frac{W_{r,j}}{s_j} \right\rceil \right)^2, \tag{32}$$

where $\mathcal{G}(j)$ is the tile containing column j. As in the analog case, a scale lowers the range of every tile whose largest entry it divides, at the cost of amplifying the rounding error seen through its input channel. The objective is piecewise smooth in $\boldsymbol{s}$, with a kink wherever a weight changes grid point; we hold the grid points fixed within each gradient step (straight-through estimator) and otherwise optimize exactly as above. Clipping is left to the clamping step that follows.

D Experimental Details

D.1 Evaluation Details

Generation. All models are evaluated with the sampling settings recommended for the Qwen3 family: temperature 0.6, top-p 0.95, top-k 20, the chat template, and a maximum of 32768 new tokens. The generation seed is fixed across all runs; only the noise-injection seed varies. For noise-free rows, which have no noise seed, the reported standard deviation is over repeated runs with the same generation seed and reflects the run-to-run nondeterminism of batched inference in vLLM (Kwon et al., 2023), not sampling variance. The quantization results of Section 4.5 are the exception: quantization is deterministic, so there we vary the generation seed and report the standard deviation over 5 generation seeds, as stated in the caption of Table 3.

Benchmarks. GSM8K, ARC-C and GPQA-diamond are evaluated 5-shot with lm-eval-harness (Gao et al., 2024). For GSM8K we report strict answer extraction, which requires the answer in the expected format and therefore also penalizes formatting errors caused by noise. The five benchmarks were chosen so that their difficulty spans the capability range of models from 0.6B to 32B parameters. HumanEval+ and MBPP+ are evaluated with the EvalPlus framework (Liu et al., 2023), reporting pass@1 with a single sample.

Layer mapping. All linear layers of the transformer block are simulated on analog tiles: the query, key, value and output projections of attention and the gate, up and down projections of the feed-forward network. Normalization layers, the attention score computation, the embeddings and the lm-head remain digital. The lm-head is kept digital for comparability with PTQ methods, which typically leave it unmodified. Results with the lm-head mapped to analog are given in Appendix F.1.

Compute. WARP+ runs layer by layer on a single A100 GPU. For Qwen3-4B, input scaling takes at most 11 minutes and clamping with redistribution 18 minutes, 29 minutes in total. For Qwen3-32B the two take at most 59 minutes and 1 hour 56 minutes, under 3 hours in total. The 32B model uses four GPUs for the single backward pass that estimates the Fisher weights u_r, since it requires the gradient of the full model; all other steps are per layer and run on one GPU. For comparison, hardware-aware retraining of a 3.8B model is reported at 230 hours on 96 V100 GPUs (Büchel et al., 2025a).

D.2 Baselines

All baselines are run on Qwen3-14B with the same calibration set as WARP, 512 sequences of 2048 tokens from UltraChat (Ding et al., 2023), and evaluated under the same AIMC pipeline (A8–W_{noisy}–O8, PCM programming noise) with 10 noise instances, unless stated otherwise. Each method is run in its original formulation and its transformed weights are programmed without quantization. GPTQ is the exception and is programmed with its 4-bit weights (group size 512). For methods with a hyperparameter we sweep it and report the best configuration in Table 2. The tables below give the full sweeps, with the noise-free and off-the-shelf rows repeated for reference.

D.2.1 SmoothQuant

SmoothQuant Xiao et al. (2024) rescales each input channel j by $s_j = \max|x_j|^\alpha / \max|W_{:,j}|^{1-\alpha}$, dividing the activations and multiplying the weights by s_j, so that the smoothing strength α sets where the magnitude ends up: $\alpha = 1$ moves it entirely into the weights, $\alpha = 0$ equalizes the weight columns and moves it into the activations, and $\alpha = 0.5$ balances the two. Since programming noise on the weights is the dominant error source, smaller α should help, and Table 4 confirms the direction: the default $\alpha = 0.8$, reported in Table 2, loses 1.0 pp against the off-the-shelf model, while $\alpha = 0.4$ is the best setting. Even the best setting, however, only matches the off-the-shelf model (74.1 against 73.9), and $\alpha = 0$ is worse than $\alpha = 0.8$, since moving all magnitude into the activations also amplifies the input quantization error. The reason is that the scales are not chosen against the noise: a single α governs every channel of every layer, and the scale of a channel depends only on its own activation and weight maxima, not on the noise it induces on the output channels whose range it sets. WARP's input scaling optimizes the same per-channel scales directly under

the noise-variance term of the objective, which is what lets it gain 1.3 pp where SmoothQuant gains none.

Table 4: SmoothQuant on Qwen3-14B under the AIMC pipeline (A8–W_{noisy}–O8, PCM programming noise) as a function of the smoothing strength α. Weights stay in BF16. only the smoothing scales change between rows. Accuracy is reported in percent as mean $\pm$ one standard deviation.

Method	**GSM8K**	**ARC-C**	**GPQA-D**	**HumanEval+**	**MBPP+**	**Avg.**
off-the-shelf, noise-free	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.8	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
Noisy AIMC pipeline: A8–W_{noisy}–O8 with PCM programming noise						
off-the-shelf	71.4 ± 8.1	91.8 ± 0.6	51.9 ± 5.7	82.9 ± 1.6	71.4 ± 1.1	73.9 ± 1.6
SmoothQuant ($\alpha = 0.0$)	62.2 ± 12.2	90.7 ± 0.4	48.2 ± 3.9	80.6 ± 2.5	69.1 ± 1.2	70.2 ± 1.9
SmoothQuant ($\alpha = 0.2$)	71.1 ± 9.2	91.4 ± 0.5	52.2 ± 4.5	83.6 ± 1.0	70.8 ± 0.7	73.8 ± 2.0
SmoothQuant ($\alpha = 0.4$)	72.2 ± 6.1	91.8 ± 0.5	52.0 ± 3.7	83.3 ± 1.6	71.1 ± 1.0	74.1 ± 1.1
SmoothQuant ($\alpha = 0.5$)	72.2 ± 5.4	91.7 ± 0.5	51.5 ± 4.2	83.6 ± 1.9	70.4 ± 1.0	73.9 ± 1.3
SmoothQuant ($\alpha = 0.8$)	69.0 ± 5.8	91.7 ± 0.6	49.7 ± 3.2	83.4 ± 1.1	70.5 ± 1.4	72.9 ± 1.1

D.2.2 SpinQuant

We use the SpinQuant Liu et al. (2025) configuration without online Hadamard transforms, in which only the rotations R_1 and R_2 are applied and merged into the weights, so the architecture of the model is unchanged and only the programmed weights differ. SpinQuant is the one experiment in this work for which we changed the ADC range. The rotations spread magnitude evenly across channels, which raises the normalized partial sums of the tiles. With the range of 12 used everywhere else the outputs saturate and the accuracy collapses, so to give the method a fair evaluation Table 2 reports SpinQuant with the range set to 24. As noted in Section 2, the ADC range is a hardware specification and is not tuned per model, method or benchmark. The exception here is in SpinQuant's favour. To separate the two effects, we also evaluated SpinQuant without output quantization (A8–W_{noisy}–O16), where it reaches 68.6 ± 2.6 average accuracy, still below the 73.9 of the off-the-shelf model under the full pipeline (A8–W_{noisy}–O8). The rotated weights are therefore more sensitive to programming noise irrespective of the ADC: merging the rotations into the weights lowers the rounding error they were designed for, but makes the model more sensitive to stochastic noise.

D.2.3 MagR

MagR (Zhang et al., 2024a) is a quantization-agnostic preprocessing step that reduces weight magnitudes before rounding: for each linear layer it shrinks the largest entries of every output channel while keeping the layer's outputs on a calibration set close to those of the original weights, trading a small reconstruction error for a smaller weight range. Since the noise model scales the programming noise of every 512-wide weight tile by that tile's maximum magnitude, MagR is a natural baseline for noise-aware preprocessing. We use the Brevitas implementation with the paper's default $\alpha = 0.01$ for 4-bit and 200 proximal-gradient steps per layer, and calibrate on the same 512 UltraChat conversations of 2048 tokens used for all other methods, so that every preprocessing method sees identical data. MagR is applied to all linear layers of the decoder blocks, the language-model head and the embeddings are left untouched. No quantization is applied afterwards: the MagR-processed weights are evaluated in bfloat16 through the same A8–W_{noisy}–O8 pipeline as the off-the-shelf model.

Original MagR penalizes only the single largest entry of each output row, so on a layer whose rows span k tiles at most one tile in k sees the penalty directly. We therefore also evaluate a tiled variant, MagR-tiled, in which the ℓ_∞ penalty is applied independently to every 512-column tile of every row, i.e. to exactly the slices whose maxima set the noise scale. Everything else, covariance estimation, step size, the Hessian-weighted gradient step, the penalty normalization, and the number of steps,

Table 5: MagR and its tiled variant on Qwen3-14B under the AIMC pipeline (A8–W_{noisy}–O8, PCM programming noise). Accuracy is reported in percent as mean ± one standard deviation.

Method	GSM8K	ARC-C	GPQA-D	HumanEval+	MBPP+	Avg.
off-the-shelf, noise-free	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.8	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
Noisy AIMC pipeline: A8–W_{noisy}–O8 with PCM programming noise						
off-the-shelf	71.4 ± 8.1	91.8 ± 0.6	51.9 ± 5.7	82.9 ± 1.6	71.4 ± 1.1	73.9 ± 1.6
MagR ($\alpha = 0.01$)	66.0 ± 5.7	91.3 ± 0.4	42.9 ± 6.2	80.8 ± 1.4	69.4 ± 1.7	70.1 ± 1.6
MagR-tiled ($\alpha = 0.001$)	67.9 ± 7.8	92.0 ± 0.4	52.5 ± 3.4	83.2 ± 2.1	71.0 ± 1.3	73.2 ± 1.5
MagR-tiled ($\alpha = 0.0025$)	60.3 ± 8.1	91.7 ± 0.6	46.3 ± 4.7	82.4 ± 1.1	70.2 ± 2.0	69.9 ± 1.6

is identical to the original. Because every tile's maximum is now penalized, a given α removes considerably more range than in the original formulation. We needed to reduce the value of the hyperparameter α.

Table 5 reports accuracy under PCM programming noise. The best MagR configuration is the tiled variant with $\alpha = 0.001$, which is much better than the original formulation and matches the off-the-shelf model on four of the five benchmarks, but it remains 0.7 points below the off-the-shelf model on average, with GSM8K accounting for the difference. Original MagR and the stronger tiled setting ($\alpha = 0.0025$) both lose about 4 points on average.

D.2.4 OmniQuant

OmniQuantShao et al. (2024) has two learnable components: learnable weight clipping (LWC), which learns a clipping range per weight group, and learnable equivalent transformation (LET), which learns per-channel scales that move activation outliers into the weights, with a fixed rule. We evaluated LWC alone, calibrated for weight-only 4-bit quantization (W4A16) with 5 epochs, LWC with 20 epochs to test whether longer calibration helps, and LWC+LET calibrated for W4A8. In all cases the learned clipping ranges and scales are applied to the weights, which are then programmed without quantization. Table 6 reports the three configurations. LWC with 5 epochs is the best and is the one reported in Table 2. Longer calibration lowers the average by 1.1 pp, and adding LET does not help under programming noise.

Table 6: OmniQuant on Qwen3-14B under the AIMC pipeline (A8–W_{noisy}–O8, PCM programming noise). LWC denotes learnable weight clipping and LET learnable equivalent transformation. ep is the number of calibration epochs. Accuracy is reported in percent as mean ± one standard deviation.

Method	GSM8K	ARC-C	GPQA-D	HumanEval+	MBPP+	Avg.
off-the-shelf, noise-free	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.7	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
Noisy AIMC pipeline: A8–W_{noisy}–O8 with PCM programming noise						
off-the-shelf	71.4 ± 7.7	91.8 ± 0.6	51.9 ± 5.4	82.9 ± 1.6	71.4 ± 1.0	73.9 ± 1.5
OmniQuant LWC, W4A16, 5 ep	73.4 ± 5.0	91.8 ± 0.3	54.7 ± 2.7	83.2 ± 1.8	71.6 ± 0.9	74.9 ± 0.5
OmniQuant LWC, W4A16, 20 ep	67.2 ± 8.6	91.9 ± 0.4	54.9 ± 3.8	83.4 ± 1.9	71.8 ± 1.3	73.8 ± 1.9
OmniQuant LWC+LET, W4A8, 5 ep	72.4 ± 4.7	91.8 ± 0.4	55.0 ± 1.9	83.4 ± 0.7	70.5 ± 1.5	74.6 ± 1.2

E SUPPLEMENTARY RESULTS

E.1 PER-BENCHMARK RESULTS FOR THE SCALING STUDY

Table 7 reports the per-benchmark accuracies underlying Figure 2a. Two patterns are visible at this granularity. The degradation of the off-the-shelf models is far from uniform across tasks: GSM8K accounts for the largest share of the drop at every scale (e.g. $58.4 \rightarrow 7.5$ for Qwen3-1.7B), while ARC-C is barely affected for models of 4B and above. The variance over noise instances also changes with adaptation. For the larger models, adaptation reduces the spread: on GSM8K, Qwen3-32B goes from 83.6 ± 6.4 off-the-shelf to 92.7 ± 2.2 after adaptation, so the adapted model is not only more accurate but also more consistent across noise instances.

Table 7: For each Qwen3 model we report the clean baseline (A16–W16–O16), the noisy baseline (A8–W_{noisy}–O8, PCM programming noise), and WARP+ in the same noisy setting. Entries are mean $\pm$ one standard deviation over 10 noise instances, in percent. For the Avg. column, the five benchmark scores are first averaged within each noise instance; the reported spread is then the variation of this average across the 10 instances, the same quantity shown by the error bars in Figure 2a. Bold marks the better of the two noisy rows.

	Benchmark accuracy (%)					
Model / configuration	**GSM8K**	**ARC-C**	**GPQA-D**	**HumanEval+**	**MBPP+**	**Avg. (%)**
Qwen3-0.6B						
off-the-shelf, noise-free	21.9 ± 0.6	39.5 ± 0.4	23.7 ± 3.5	36.6 ± 0.0	45.4 ± 0.8	33.5 ± 0.6
off-the-shelf (A8–W_{noisy}–O8)	4.5 ± 5.8	28.6 ± 8.3	22.2 ± 5.2	11.3 ± 3.1	13.4 ± 2.7	16.0 ± 3.4
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{12.6 \pm 9.5}$	$\mathbf{35.6 \pm 10.0}$	$\mathbf{24.4 \pm 2.7}$	$\mathbf{18.8 \pm 2.5}$	$\mathbf{23.5 \pm 5.0}$	$\mathbf{23.0 \pm 2.8}$
Qwen3-1.7B						
off-the-shelf, noise-free	58.4 ± 0.4	71.9 ± 0.0	33.9 ± 1.9	65.2 ± 0.0	49.7 ± 0.0	56.0 ± 0.3
off-the-shelf (A8–W_{noisy}–O8)	7.5 ± 7.7	54.1 ± 8.5	24.5 ± 2.4	42.9 ± 3.4	38.5 ± 3.0	33.5 ± 3.5
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{25.1 \pm 21.7}$	$\mathbf{63.1 \pm 3.4}$	$\mathbf{26.5 \pm 3.7}$	$\mathbf{52.6 \pm 2.6}$	$\mathbf{42.5 \pm 2.3}$	$\mathbf{42.0 \pm 5.5}$
Qwen3-4B						
off-the-shelf, noise-free	75.0 ± 0.8	87.6 ± 0.2	48.6 ± 0.5	78.0 ± 0.0	61.0 ± 0.1	70.1 ± 0.1
off-the-shelf (A8–W_{noisy}–O8)	42.6 ± 18.5	82.8 ± 1.1	39.2 ± 2.3	69.0 ± 2.2	55.0 ± 2.1	57.7 ± 3.7
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{62.0 \pm 14.2}$	$\mathbf{83.9 \pm 0.9}$	$\mathbf{41.8 \pm 1.1}$	$\mathbf{71.8 \pm 1.7}$	$\mathbf{56.8 \pm 2.2}$	$\mathbf{63.2 \pm 3.1}$
Qwen3-8B						
off-the-shelf, noise-free	82.1 ± 0.3	90.6 ± 0.1	57.2 ± 1.3	79.3 ± 0.0	62.7 ± 0.0	74.4 ± 0.2
off-the-shelf (A8–W_{noisy}–O8)	68.8 ± 9.4	87.0 ± 0.9	47.7 ± 2.5	76.0 ± 2.6	57.5 ± 2.6	67.4 ± 2.0
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{71.6 \pm 9.3}$	$\mathbf{88.9 \pm 0.5}$	$\mathbf{49.8 \pm 1.5}$	$\mathbf{78.1 \pm 1.9}$	$\mathbf{60.5 \pm 2.9}$	$\mathbf{69.8 \pm 1.7}$
Qwen3-14B						
off-the-shelf, noise-free	85.0 ± 0.5	92.2 ± 0.1	60.2 ± 1.7	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
off-the-shelf (A8–W_{noisy}–O8)	71.4 ± 7.7	91.8 ± 0.6	51.9 ± 5.4	82.9 ± 1.6	71.4 ± 1.0	73.9 ± 1.5
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{79.0 \pm 4.2}$	$\mathbf{91.8 \pm 0.4}$	$\mathbf{53.8 \pm 1.9}$	$\mathbf{84.2 \pm 1.7}$	$\mathbf{72.0 \pm 1.8}$	$\mathbf{76.2 \pm 1.1}$
Qwen3-32B						
off-the-shelf, noise-free	94.7 ± 0.4	94.8 ± 0.1	60.5 ± 1.7	87.2 ± 0.4	71.4 ± 0.3	81.7 ± 0.3
off-the-shelf (A8–W_{noisy}–O8)	83.6 ± 6.4	94.2 ± 0.5	56.4 ± 2.2	82.4 ± 2.0	71.7 ± 1.3	77.7 ± 1.7
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{92.7 \pm 2.2}$	$\mathbf{94.4 \pm 0.3}$	$\mathbf{58.5 \pm 1.8}$	$\mathbf{83.6 \pm 1.8}$	$\mathbf{72.0 \pm 1.5}$	$\mathbf{80.2 \pm 0.9}$

E.2 PER-BENCHMARK RESULTS FOR THE NOISE-MAGNITUDE SWEEP

Table 8 reports the per-benchmark accuracies underlying Figure 2b, with every run evaluated through the 8-bit converter stack (8-bit DAC input, 8-bit ADC readout, lm_head in high precision). The collapse of the off-the-shelf model is again led by GSM8K, which falls below 10% already at $\alpha = 8\%$, while ARC-C degrades last. The adapted model preserves a usable accuracy on every task up to $\alpha = 10\%$, where the off-the-shelf model has been reduced to 1.9% on average. OmniQuant (LWC 4-bit) sits between the two throughout: its clipping buys real robustness over the off-the-shelf weights, 27.6% against 1.9% at $\alpha = 10\%$, but it still loses roughly half of the adapted model's average accuracy from $\alpha = 8\%$ onwards. At the smallest noise level ($\alpha = 1\%$) the three configurations are within noise of each other on every benchmark, which is expected since there is little degradation left to recover.

Table 8: Per-benchmark accuracies behind Figure 2b. Qwen3-14B under additive Gaussian programming noise (Section 2) at levels α from 1% to 10%, evaluated with the 8-bit converter stack active (A8-W_{noisy}-O8). At each level we report the off-the-shelf model, OmniQuant (LWC 4-bit), and the model adapted with WARP+ at that α. The first row gives the clean BF16 model without noise or converters, the ceiling all three series are read against, since the converter stack itself already costs accuracy. Entries are mean $\pm$ one standard deviation over 10 noise instances, in percent. The Avg. column is averaged within each noise instance, matching the bands in Figure 2b. Bold marks the best result at each noise level.

	Benchmark accuracy (%)					
Noise level / configuration	**GSM8K**	**ARC-C**	**GPQA-D**	**HumanEval+**	**MBPP+**	**Avg. (%)**
No noise						
off-the-shelf, BF16, noise-free	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.7	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
$\alpha = 0.01$						
off-the-shelf	82.4 ± 1.8	92.3 ± 0.2	53.9 ± 4.1	84.7 ± 2.0	72.9 ± 1.1	77.2 ± 1.0
OmniQuant (LWC)	$\mathbf{84.6 \pm 1.3}$	$\mathbf{92.5 \pm 0.3}$	$\mathbf{58.8 \pm 1.4}$	84.1 ± 1.2	$\mathbf{73.3 \pm 0.7}$	$\mathbf{78.7 \pm 0.5}$
ours: WARP+	84.2 ± 1.0	92.1 ± 0.3	57.8 ± 2.4	$\mathbf{84.7 \pm 1.0}$	73.2 ± 0.5	78.4 ± 0.7
$\alpha = 0.02$						
off-the-shelf	78.5 ± 3.8	92.3 ± 0.3	52.3 ± 2.9	84.7 ± 1.1	72.9 ± 1.1	76.2 ± 0.8
OmniQuant (LWC)	81.8 ± 2.6	$\mathbf{92.4 \pm 0.3}$	$\mathbf{58.6 \pm 1.7}$	84.5 ± 1.9	$\mathbf{73.1 \pm 1.0}$	78.1 ± 0.7
ours: WARP+	$\mathbf{83.9 \pm 1.9}$	92.1 ± 0.3	58.2 ± 2.0	$\mathbf{85.6 \pm 1.3}$	72.9 ± 0.7	$\mathbf{78.5 \pm 0.6}$
$\alpha = 0.03$						
off-the-shelf	68.2 ± 6.1	91.6 ± 0.5	51.1 ± 1.6	82.4 ± 1.8	71.1 ± 1.0	72.9 ± 1.3
OmniQuant (LWC)	73.5 ± 5.9	$\mathbf{92.1 \pm 0.3}$	53.5 ± 2.5	83.3 ± 1.7	72.0 ± 1.1	74.9 ± 1.5
ours: WARP+	$\mathbf{81.5 \pm 2.6}$	92.0 ± 0.2	$\mathbf{58.3 \pm 1.7}$	$\mathbf{85.0 \pm 1.6}$	$\mathbf{72.4 \pm 1.0}$	$\mathbf{77.8 \pm 0.6}$
$\alpha = 0.04$						
off-the-shelf	58.5 ± 14.5	90.6 ± 0.5	47.0 ± 3.3	79.6 ± 2.3	68.3 ± 1.0	68.8 ± 3.2
OmniQuant (LWC)	59.8 ± 12.4	$\mathbf{91.7 \pm 0.5}$	50.2 ± 3.8	82.0 ± 1.6	70.8 ± 1.0	70.9 ± 3.0
ours: WARP+	$\mathbf{77.3 \pm 3.8}$	91.7 ± 0.3	$\mathbf{56.2 \pm 2.3}$	$\mathbf{84.8 \pm 2.0}$	$\mathbf{71.8 \pm 1.3}$	$\mathbf{76.3 \pm 1.0}$
$\alpha = 0.05$						
off-the-shelf	47.8 ± 17.5	88.9 ± 0.6	39.7 ± 5.4	75.1 ± 1.9	64.0 ± 3.9	63.1 ± 4.6
OmniQuant (LWC)	50.1 ± 11.5	90.8 ± 0.5	46.8 ± 3.1	80.0 ± 2.0	68.0 ± 1.9	67.1 ± 2.0
ours: WARP+	$\mathbf{70.9 \pm 7.0}$	$\mathbf{91.2 \pm 0.4}$	$\mathbf{53.0 \pm 1.8}$	$\mathbf{83.7 \pm 2.1}$	$\mathbf{71.0 \pm 1.3}$	$\mathbf{74.0 \pm 1.5}$
$\alpha = 0.06$						
off-the-shelf	39.7 ± 11.0	86.1 ± 0.9	30.2 ± 3.5	67.6 ± 3.5	54.3 ± 12.3	55.6 ± 4.5
OmniQuant (LWC)	43.2 ± 14.7	89.7 ± 0.4	42.6 ± 3.8	77.8 ± 1.8	66.7 ± 3.1	64.0 ± 2.7
ours: WARP+	$\mathbf{63.4 \pm 9.9}$	$\mathbf{90.7 \pm 0.5}$	$\mathbf{50.1 \pm 2.3}$	$\mathbf{82.3 \pm 2.2}$	$\mathbf{68.6 \pm 1.7}$	$\mathbf{71.0 \pm 2.1}$
$\alpha = 0.07$						
off-the-shelf	21.1 ± 8.0	74.6 ± 7.0	24.2 ± 3.8	46.2 ± 5.0	44.5 ± 12.4	42.1 ± 4.6
OmniQuant (LWC)	33.9 ± 16.1	87.5 ± 1.0	38.8 ± 3.6	72.8 ± 2.2	63.3 ± 3.1	59.2 ± 3.2
ours: WARP+	$\mathbf{51.9 \pm 13.9}$	$\mathbf{89.6 \pm 0.8}$	$\mathbf{48.3 \pm 2.8}$	$\mathbf{81.2 \pm 2.0}$	$\mathbf{67.4 \pm 1.6}$	$\mathbf{67.7 \pm 2.8}$
$\alpha = 0.08$						
off-the-shelf	4.2 ± 3.0	48.7 ± 23.2	17.9 ± 3.8	17.9 ± 5.6	24.9 ± 8.9	24.9 ± 5.1
OmniQuant (LWC)	23.8 ± 17.8	84.0 ± 1.8	31.1 ± 3.3	64.9 ± 2.8	59.0 ± 1.6	52.6 ± 2.6
ours: WARP+	$\mathbf{40.2 \pm 15.8}$	$\mathbf{88.8 \pm 0.7}$	$\mathbf{45.6 \pm 3.4}$	$\mathbf{78.5 \pm 2.0}$	$\mathbf{66.1 \pm 2.0}$	$\mathbf{63.8 \pm 3.5}$
$\alpha = 0.09$						
off-the-shelf	0.1 ± 0.1	19.0 ± 16.3	8.7 ± 4.4	2.1 ± 1.5	3.4 ± 2.6	6.2 ± 3.7
OmniQuant (LWC)	14.6 ± 14.3	77.4 ± 3.8	26.8 ± 3.1	51.5 ± 2.6	49.0 ± 3.8	43.9 ± 2.3
ours: WARP+	$\mathbf{32.4 \pm 15.8}$	$\mathbf{87.4 \pm 0.9}$	$\mathbf{42.1 \pm 3.3}$	$\mathbf{74.9 \pm 2.3}$	$\mathbf{64.1 \pm 2.0}$	$\mathbf{60.2 \pm 2.8}$
$\alpha = 0.10$						
off-the-shelf	0.0 ± 0.1	4.4 ± 5.7	5.2 ± 2.7	0.0 ± 0.0	0.1 ± 0.1	1.9 ± 1.3
OmniQuant (LWC)	4.0 ± 5.2	52.6 ± 16.9	21.3 ± 4.1	24.4 ± 4.3	33.0 ± 3.4	27.6 ± 3.6
ours: WARP+	$\mathbf{25.6 \pm 16.8}$	$\mathbf{86.0 \pm 1.0}$	$\mathbf{35.8 \pm 2.9}$	$\mathbf{72.4 \pm 3.0}$	$\mathbf{62.2 \pm 2.4}$	$\mathbf{56.4 \pm 3.1}$

F Complementary Experiments

F.1 LM-head in AIMC

All results in the main text keep the lm-head digital, for comparability with PTQ methods, which typically leave it unmodified. Table 9 checks what this choice costs: on Qwen3-14B, mapping the lm-head to the AIMC pipeline as well changes the average accuracy by 0.1 pp (74.0 against 73.9), within the spread across noise instances. The lm-head is therefore not a sensitive component under programming noise at this model size, and the conclusions of the main text do not depend on keeping it digital.

Table 9: Effect of the AIMC pipeline on the LM head of Qwen3-14B (A8–PCM–O8: 8-bit input, PCM programming noise on the weights, 8-bit ADC readout). The two noisy rows differ only in whether the LM head is kept in high precision or is also mapped to the AIMC pipeline. Accuracy is reported in percent as mean $\pm$ one standard deviation over 10 noise instances. For the Avg. column, the five benchmark scores are first averaged within each instance.

Method	**GSM8K**	**ARC-C**	**GPQA-D**	**HumanEval+**	**MBPP+**	**Avg.**
off-the-shelf, noise-free	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.7	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
Noisy AIMC pipeline: A8–PCM–O8						
LM head in high precision	71.4 ± 7.7	91.8 ± 0.6	51.9 ± 5.4	82.9 ± 1.6	71.4 ± 1.0	73.9 ± 1.5
LM head on A8–PCM–O8	70.8 ± 7.7	91.7 ± 0.5	53.0 ± 5.0	83.6 ± 1.5	70.8 ± 1.2	74.0 ± 1.6

F.2 Noise Decomposition

Table 10 separates the three error sources of the AIMC pipeline by switching them on one at a time. Programming noise alone (A16–W_{noisy}–O16) is the dominant source for all but the largest model, and the two sources are not additive: the full pipeline loses about as much as the noise alone. At 32B the picture changes, with programming noise costing 2.1 pp and the converters 3.2 pp, most of it on GPQA-D (60.5 to 51.8). The explanation is the resolution of the ADC: the tile outputs of an off-the-shelf model occupy only a small part of the static range, so the 8-bit grid is coarse relative to them. WARP+ removes this loss almost entirely: evaluated with the converters but without programming noise (A8–W16–O8), it is within 0.5 pp of the noise-free accuracy at every size except 8B (1.1 pp). Since clamping lowers the range by which each tile row is normalized, the partial sums the ADC sees become larger and use more of its levels. So, tightening the weight ranges appears to help beyond the programming noise, by making better use of a fixed-resolution ADC.

Table 10: Error sources taken apart across the Qwen3 family. Configurations follow the notation of Section 4.1, with A16 and O16 denoting no input and no output quantization: A16–W_{noisy}–O16 isolates the programming noise, A8–W16–O8 isolates the converters, and A8–W_{noisy}–O8 is the full pipeline. W_{noisy} rows are mean $\pm$ one standard deviation over 10 noise instances. Avg. is averaged within each instance. Bold marks the better of the two A8–W_{noisy}–O8 rows.

Model / configuration	Benchmark accuracy (%)					Avg. (%)
	GSM8K	ARC-C	GPQA-D	HumanEval+	MBPP+	
Qwen3-0.6B						
noise-free off-the-shelf (W16)	21.9 ± 0.6	39.5 ± 0.4	23.7 ± 3.5	36.6 ± 0.0	45.4 ± 0.8	33.5 ± 0.6
off-the-shelf (A16–W_{noisy}–O16)	3.8 ± 4.6	30.9 ± 12.8	20.7 ± 3.2	15.7 ± 1.5	18.0 ± 3.7	17.8 ± 3.5
off-the-shelf (A8–W16–O8)	21.5 ± 0.5	35.1 ± 7.3	25.3 ± 2.5	30.1 ± 3.2	40.2 ± 1.2	30.4 ± 1.8
off-the-shelf (A8–W_{noisy}–O8)	4.5 ± 5.8	28.6 ± 8.3	22.2 ± 5.2	11.3 ± 3.1	13.4 ± 2.7	16.0 ± 3.4
ours: WARP+ (A8–W16–O8)	33.7 ± 0.3	44.0 ± 0.8	25.3 ± 0.0	31.7 ± 0.0	44.4 ± 0.0	35.8 ± 0.2
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{12.6 \pm 9.5}$	$\mathbf{35.6 \pm 10.0}$	$\mathbf{24.4 \pm 2.7}$	$\mathbf{18.8 \pm 2.5}$	$\mathbf{23.5 \pm 5.0}$	$\mathbf{23.0 \pm 2.8}$
Qwen3-1.7B						
noise-free off-the-shelf (W16)	58.4 ± 0.4	71.9 ± 0.0	33.9 ± 1.9	65.2 ± 0.0	49.7 ± 0.0	56.0 ± 0.3
off-the-shelf (A16–W_{noisy}–O16)	12.1 ± 13.9	53.3 ± 14.6	23.3 ± 3.1	45.9 ± 1.7	41.7 ± 3.6	35.3 ± 5.0
off-the-shelf (A8–W16–O8)	34.8 ± 1.2	68.2 ± 0.2	31.2 ± 2.8	61.6 ± 2.0	47.7 ± 1.1	48.7 ± 0.8
off-the-shelf (A8–W_{noisy}–O8)	7.5 ± 7.7	54.1 ± 8.5	24.5 ± 2.4	42.9 ± 3.4	38.5 ± 3.0	33.5 ± 3.5
ours: WARP+ (A8–W16–O8)	63.5 ± 0.5	71.7 ± 0.2	33.9 ± 1.2	60.4 ± 0.0	47.9 ± 0.0	55.5 ± 0.3
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{25.1 \pm 21.7}$	$\mathbf{63.1 \pm 3.4}$	$\mathbf{26.5 \pm 3.7}$	$\mathbf{52.6 \pm 2.6}$	$\mathbf{42.5 \pm 2.3}$	$\mathbf{42.0 \pm 5.5}$
Qwen3-4B						
noise-free off-the-shelf (W16)	74.9 ± 1.2	87.7 ± 0.1	49.2 ± 2.1	76.4 ± 1.5	59.6 ± 0.7	69.6 ± 0.6
off-the-shelf (A16–W_{noisy}–O16)	48.7 ± 18.2	83.6 ± 0.7	42.7 ± 1.6	70.3 ± 1.4	53.8 ± 2.5	59.8 ± 3.6
off-the-shelf (A8–W16–O8)	71.4 ± 1.3	87.7 ± 0.1	47.7 ± 1.5	75.2 ± 1.2	60.0 ± 1.0	68.4 ± 0.4
off-the-shelf (A8–W_{noisy}–O8)	42.6 ± 18.5	82.8 ± 1.1	39.2 ± 2.3	69.0 ± 2.2	55.0 ± 2.1	57.7 ± 3.7
ours: WARP+ (A8–W16–O8)	75.7 ± 1.0	86.8 ± 0.1	46.0 ± 1.1	76.1 ± 0.9	60.1 ± 0.1	68.9 ± 0.3
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{62.0 \pm 14.2}$	$\mathbf{83.9 \pm 0.9}$	$\mathbf{41.8 \pm 1.1}$	$\mathbf{71.8 \pm 1.7}$	$\mathbf{56.8 \pm 2.2}$	$\mathbf{63.2 \pm 3.1}$
Qwen3-8B						
noise-free off-the-shelf (W16)	82.1 ± 0.3	90.6 ± 0.1	57.2 ± 1.3	79.3 ± 0.0	62.7 ± 0.0	74.4 ± 0.2
off-the-shelf (A16–W_{noisy}–O16)	66.6 ± 10.2	86.3 ± 4.0	49.1 ± 2.7	76.8 ± 1.3	59.5 ± 2.9	67.7 ± 2.5
off-the-shelf (A8–W16–O8)	86.6 ± 0.7	90.5 ± 0.1	55.4 ± 2.1	79.3 ± 1.2	61.4 ± 0.9	74.6 ± 0.4
off-the-shelf (A8–W_{noisy}–O8)	68.8 ± 9.4	87.0 ± 0.9	47.7 ± 2.5	76.0 ± 2.6	57.5 ± 2.6	67.4 ± 2.0
ours: WARP+ (A8–W16–O8)	81.2 ± 0.7	90.9 ± 0.1	53.9 ± 1.1	78.0 ± 0.0	62.7 ± 0.0	73.3 ± 0.3
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{71.6 \pm 9.3}$	$\mathbf{88.9 \pm 0.5}$	$\mathbf{49.8 \pm 1.5}$	$\mathbf{78.1 \pm 1.9}$	$\mathbf{60.5 \pm 2.9}$	$\mathbf{69.8 \pm 1.7}$
Qwen3-14B						
noise-free off-the-shelf (W16)	85.1 ± 0.5	92.2 ± 0.1	60.2 ± 1.7	86.6 ± 0.4	74.3 ± 0.2	79.7 ± 0.5
off-the-shelf (A16–W_{noisy}–O16)	67.5 ± 12.6	91.9 ± 0.4	52.9 ± 2.2	83.1 ± 2.2	71.6 ± 1.1	73.4 ± 2.9
off-the-shelf (A8–W16–O8)	83.6 ± 1.4	92.5 ± 0.1	52.1 ± 6.2	86.5 ± 0.8	73.4 ± 0.6	77.6 ± 1.5
off-the-shelf (A8–W_{noisy}–O8)	71.4 ± 7.7	91.8 ± 0.6	51.9 ± 5.4	82.9 ± 1.6	71.4 ± 1.0	73.9 ± 1.5
ours: WARP+ (A8–W16–O8)	85.9 ± 0.4	91.6 ± 0.0	59.1 ± 2.0	87.8 ± 0.0	72.5 ± 0.0	79.4 ± 0.4
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{79.0 \pm 4.2}$	$\mathbf{91.8 \pm 0.4}$	$\mathbf{53.8 \pm 1.9}$	$\mathbf{84.2 \pm 1.7}$	$\mathbf{72.0 \pm 1.8}$	$\mathbf{76.2 \pm 1.1}$
Qwen3-32B						
noise-free off-the-shelf (W16)	94.7 ± 0.4	94.8 ± 0.1	60.5 ± 1.7	87.2 ± 0.4	71.4 ± 0.3	81.7 ± 0.3
off-the-shelf (A16–W_{noisy}–O16)	90.7 ± 3.4	94.4 ± 0.4	58.7 ± 1.3	82.9 ± 1.9	71.6 ± 0.9	79.6 ± 1.0
off-the-shelf (A8–W16–O8)	90.1 ± 0.8	94.5 ± 0.1	51.8 ± 2.1	83.9 ± 0.8	72.1 ± 0.8	78.5 ± 0.6
off-the-shelf (A8–W_{noisy}–O8)	83.6 ± 6.4	94.2 ± 0.5	56.4 ± 2.2	82.4 ± 2.0	71.7 ± 1.3	77.7 ± 1.7
ours: WARP+ (A8–W16–O8)	95.4 ± 0.3	94.7 ± 0.1	59.0 ± 1.8	87.2 ± 0.0	73.5 ± 0.0	82.0 ± 0.4
ours: WARP+ (A8–W_{noisy}–O8)	$\mathbf{92.7 \pm 2.2}$	$\mathbf{94.4 \pm 0.3}$	$\mathbf{58.5 \pm 1.8}$	$\mathbf{83.6 \pm 1.8}$	$\mathbf{72.0 \pm 1.5}$	$\mathbf{80.2 \pm 0.9}$