COSTBIT: CANDIDATE-COST-INFORMED BIT ALLOCATION FOR POST-TRAINING QUANTIZATION OF LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training quantization (PTQ) is a practical route to reducing the memory and bandwidth requirements of large language models (LLMs). However, uniform bit-width quantization becomes increasingly inefficient in the aggressive sub-3-bit regime due to heterogeneous quantization sensitivity across weight blocks. Although existing mixed-precision schemes alleviate this issue, their allocation rules fail to account for the quantization error variation of each weight block across candidate bit-widths, resulting in suboptimal accuracy–compression trade-offs. To address this limitation, we introduce **CostBit**, a candidate-**Cost**-informed block-wise **Bit** allocation method for PTQ of LLMs. Specifically, CostBit evaluates the Hessian-weighted cost associated with each candidate bit-width for every weight block and leverages a Lagrangian allocator to closely match the target bit budget, supporting non-integer values. Extensive experiments on OPT, LLaMA-2, and LLaMA-3 spanning 125M to 70B parameters demonstrate that CostBit yields improved performance at comparable target bit budgets, with significant gains in the sub-3-bit regime. Further, at an average bit-width of approximately 2.5 bits, it reduces WikiText-2 perplexity from 7.69 to 6.43 and improves the average zero-shot accuracy from 60.04% to 63.79% compared with OWQ on LLaMA-2-7B. CostBit also enables fully GPU-resident inference of LLaMA-2-70B on a single RTX 4090 (24 GB) at approximately 2.5 bits, achieving 75.70% average zero-shot accuracy, only 1.37% below the FP16 baseline. Our code is available at https://anonymous.4open.science/r/CostBit.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable performance across language understanding and generation tasks (Touvron et al., 2023; Grattafiori et al., 2024). However, deploying LLMs remains expensive since their billions of parameters incur substantial memory capacity and bandwidth requirements (Pope et al., 2023; Sheng et al., 2023). Post-training quantization (PTQ) is able to quantize model weights, activations, or both to low-bit numerical formats without requiring full retraining, thereby becoming a practical approach to efficient LLM inference (Yao et al., 2022; Xiao et al., 2023). Note that we only focus on weight-only PTQ for LLMs in this study.

Recent weight-only PTQ methods (Frantar et al., 2023; Chee et al., 2023; Li et al., 2025) make substantial progress at a fixed 3-bit or 4-bit quantization. However, allocating the same bit-width to all weights becomes increasingly inefficient under aggressive sub-3-bit regime since LLM weights exhibit highly non-uniform sensitivity to quantization perturbations (Lin et al., 2024; Kim et al., 2024). In particular, insensitive weights may receive unnecessarily high bit-widths, while sensitive weights may be over-compressed. Consequently, performing effective bit allocation across weight blocks with heterogeneous sensitivity becomes a critical challenge for sub-3-bit quantization.

Mixed-precision quantization provides a natural way to address this issue by assigning different bits to different weights. For instance, OWQ (Lee et al., 2024) retains a small set of weak columns in FP16 while keeping the remaining weights in a uniform low-bit format. SliM-LLM (Huang et al., 2025) performs group-wise mixed-precision quantization by ranking groups according to salience and using output KL divergence to select the number of paired $(N-1)$-bit and

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.

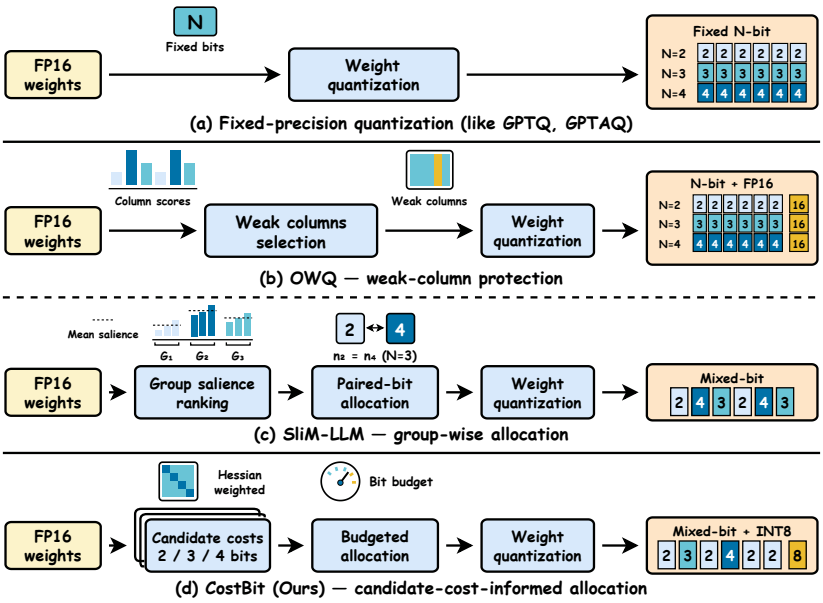
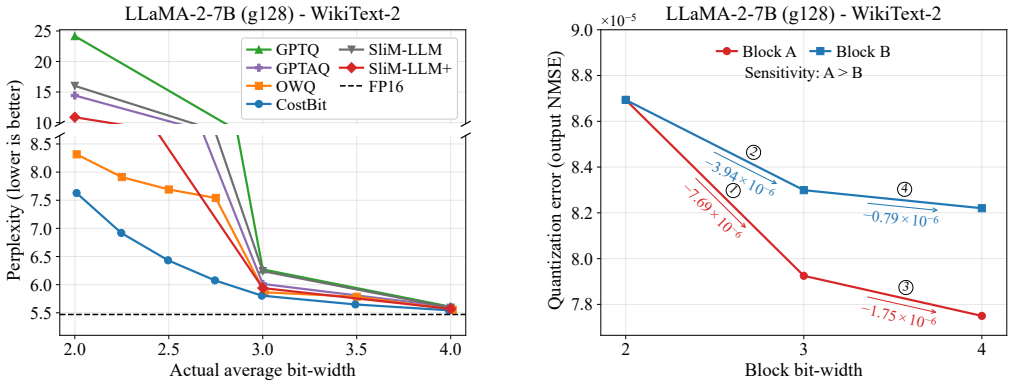


Figure 1: Comparison of bit-allocation paradigms.

$(N+1)$-bit groups around a target of N bits. However, they still rely on importance ranking to reflect block sensitivity and implement bit allocation to each block accordingly, thereby failing to reveal which candidate bit-width is most appropriate for each block. As a result, these methods still exhibit less favorable accuracy–compression trade-offs in sub-3-bit regime. Figure 1 provides a comparison between existing allocation rules and CostBit. Figure 2(a) compares perplexity across bit-widths, highlighting CostBit's advantage in the sub-3-bit regime.

This limitation is especially pronounced in the sub-3-bit regime where the target bit budget (i.e., average bit-width of per-layer weights) is severely constrained and each bit must be allocated to provide the greatest reduction in quantization error. Therefore, effective bit allocation should consider not only the sensitivity of each weight block but also the quantization error associated with each candidate bit-width. As shown in Figure 2(b), for candidate bit-widths of 2, 3, and 4 bits, increasing Block A from 2 to 3 bits produces the greatest NMSE reduction of 7.69×10^{-6}, whereas further increasing it further to 4 bits only reduces NMSE by 1.75×10^{-6}. By contrast, increasing the less sensitive Block B from 2 to 3 bits achieves a larger reduction of 3.94×10^{-6} than increasing Block A from 3 to 4 bits. *Thus, if only one additional bit is available for allocation, assigning it to Block B rather than further increasing Block A yields a larger reduction in quantization error, despite $A > B$ in sensitivity*. These results illustrate that a single importance ranking fail to determine the appropriate bit-width for each block since different blocks exhibit distinct quantization error responses to different candidate bit-widths. This observation leads to the core question of this work: **How should candidate bit-widths be judiciously allocated across weight blocks to minimize quantization error subject to a prescribed target bit budget?**

To answer this question, we propose **CostBit**, a candidate-**Cost**-informed blockwise **Bit** allocation method for PTQ of LLMs. Before blockwise bit allocation, CostBit identifies a small set of extremely sensitive columns according to Hessian-weighted probe quantization error and preserves them in configurable high precision, defaulting to INT8. For the remaining columns, it estimates the candidate cost of every block–bit pair by weighting quantization errors with the diagonal of the calibration-derived Hessian. Based on these costs, a Lagrangian allocator selects block bit-widths under a target bit budget, while explicitly accounting for the storage cost of extremely sensitive columns. Furthermore, binary search over the bit penalty and local adjustment closely match the target bit budget, including non-integer values. Finally, CostBit performs sequential weight reconstruction with second-order error compensation according to the selected blockwise bit-width plan.



(a) Perplexity comparison (b) Quantization error across block bit-widths

Figure 2: (a) Perplexity across bit-widths for CostBit and existing methods. (b) Quantization error of the first-layer query projection, measured by the normalized mean squared error (NMSE) between original and reconstructed quantized outputs. Starting from an allocation targeting 2.5 bits, each curve varies one 128-column block over 2, 3, and 4 bits while fixing other bit assignments. ①-④ reveal the priority of bit-width upgrades of CostBit, where ① corresponds to the highest priority.

Note that CostBit supports non-integer target bit-widths, enabling model precision to better match hardware memory constraints in practical deployment. For instance, when deploying LLaMA-2-70B on a single RTX 4090 (24 GB), fixed bit-width quantization methods such as GPTQ and GPTAQ face a capacity–accuracy dilemma. The main reason is that storing LLaMA-2-70B weights at 3 bits already exceeds the GPU memory, whereas 2-bit quantization incurs substantial accuracy loss. Although OWQ attains an actual average bit-width of 2.5 bits by retaining more weak columns in FP16, its released CUDA kernels lack support for the required 2-bit base precision. In contrast, CostBit enables fully GPU-resident inference at approximately 2.5 bits, with generation examples provided in Appendix C. It achieves 75.70% average accuracy across six zero-shot tasks, outperforming OWQ by +2.47% at the same bit budget and trailing FP16 baseline by only 1.37% (Table 2).

Our contributions are as follows: 1) We introduce a Hessian-weighted candidate cost relying on the quantization errors of each block–bit pair. By comparing these costs across candidate bit-widths, it enables blockwise bit allocation that reflects how each block responds to different bit-widths, rather than relying on importance ranking; 2) We develop a training-free blockwise Lagrangian allocator that accounts for protected columns and closely matches the per-layer target bit budget, including non-integer values. Moreover, it maintains a deployment-friendly layout for efficient inference; 3) We provide experiments across the OPT, LLaMA-2, and LLaMA-3 families, spanning sizes from 125M to 70B, demonstrating that CostBit consistently achieves a better accuracy–compression trade-off than existing PTQ methods. The perplexity improvements are most evident in the sub-3-bit regime (Table 1), with consistent gains in six-task zero-shot accuracy (Table 2). Experiments on real hardware further show that CostBit achieves decoding speedups and substantial reductions in both weight and runtime memory (Table 4), and supports fully GPU-resident inference of LLaMA-2-70B on a single RTX 4090 (24 GB) at approximately 2.5 bits (Appendix C).

2 Related Work

Post-training quantization for LLMs. PTQ has become a practical approach to reducing LLM inference costs. LLM.int8() (Dettmers et al., 2022) enables 8-bit matrix multiplication for large transformer inference by handling outlier features separately. SmoothQuant (Xiao et al., 2023) addresses activation outliers by migrating quantization difficulty from activations to weights through an equivalent smoothing transformation, enabling accurate W8A8 quantization. For aggressive low-bit settings, AWQ (Lin et al., 2024) protects a small fraction of salient weights through activation-aware scaling, improving low-bit weight-only quantization. OmniQuant (Shao et al., 2024) introduces learnable weight clipping and equivalent transformation to handle weight extremes and activation outliers. AffineQuant (Ma et al., 2024) extends the optimization space of PTQ through equivalent

affine transformations to reduce quantization error. However, they largely assume a fixed bit-width without explicitly addressing how to allocate a limited bit budget across different parts of the model.

Hessian-based weight-only quantization. Second-order information has long been utilized in model compression (LeCun et al., 1989; Hassibi & Stork, 1992), employing Hessian-derived criteria to quantify how parameter perturbations affect model loss. For LLMs, GPTQ (Frantar et al., 2023) uses second-order information for sequential weight quantization and error compensation, enabling accurate low-bit quantization. Several methods extend second-order information in different directions, including Hessian-incoherence processing in QuIP (Chee et al., 2023) and QuIP# (Tseng et al., 2024), sensitivity-aware outlier isolation in SpQR (Dettmers et al., 2024), and asymmetric calibration against full-precision layer outputs in GPTAQ (Li et al., 2025). Consequently, Hessian information is primarily used for error estimation and compensation after the bit-width has been fixed, rather than for deciding how bit-widths should be allocated across weight blocks.

Mixed-precision quantization. Mixed-precision quantization allocates different bit-widths to different layers, blocks or channels. Earlier mixed-precision methods for convolutional and vision models use Hessian-based sensitivity to determine bit-width assignments. HAWQ (Dong et al., 2019) and HAWQ-V2 (Dong et al., 2020) rank layers using Hessian-derived sensitivity, while HAWQ-V3 (Yao et al., 2021) further formulates hardware-aware mixed-precision selection as an integer linear program. For LLMs, mixed-precision quantization has also been explored to improve the accuracy–compression trade-off under aggressive bit budgets. OWQ (Lee et al., 2024) protects weak columns using higher precision, while SliM-LLM (Huang et al., 2025) proposes salience-driven group-wise mixed-precision quantization. Nevertheless, their allocation decisions remain guided by importance rankings. Such rankings capture the relative sensitivity of weight blocks yet cannot identify the most suitable candidate bit-width for each individual block. Hence, they still struggle to achieve favorable accuracy-compression trade-offs in sub-3-bit regime.

3 PRELIMINARIES

Layer-wise quantization. Let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote the full-precision weights of a linear layer and $X \in \mathbb{R}^{d_{\text{in}} \times N}$ denote the input activations for N calibration tokens. Weight-only PTQ seeks a low-precision approximation of W that minimizes the discrepancy between the original and quantized layer outputs. Denoting the dequantized weight matrix by $\widehat{W}$, we use the layer reconstruction objective (Nagel et al., 2020; Hubara et al., 2021; Frantar et al., 2023):

$$E(\widehat{W}) = \left\| WX - \widehat{W}X \right\|_F^2 = \left\| (W - \widehat{W})X \right\|_F^2 . \tag{1}$$

Reconstruction Hessian. Let w_r and $\widehat{w}_r$ denote the r-th rows of W and $\widehat{W}$, respectively. The layer-wise quantization error (Lee et al., 2024) can then be expressed as a sum of row-wise errors:

$$E(\widehat{W}) = \sum_{r=1}^{d_{\text{out}}} \|(w_r - \widehat{w}_r)X\|_2^2 . \tag{2}$$

Each row-wise objective is quadratic in $\widehat{w}_r$ and has the same Hessian matrix:

$$H = 2XX^\top \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}, \tag{3}$$

where the coordinates of H correspond to input channels, or equivalently, columns of W.

Quantization blocks. Let $\{G_k\}_{k=1}^K$ be a partition of the input-channel indices $\{1, \ldots, d_{\text{in}}\}$. For each group G_k, we define the weight block $W_{:,G_k}$, which contains all output rows and input columns indexed by G_k. For each output row, the Hessian restricted to these coordinates is the principal submatrix H_{G_k,G_k}, whose derivation is shown in Appendix D. CostBit uses these column groups as the units of bit-width allocation and the associated Hessian submatrices to estimate block sensitivity.

4 METHOD

We propose CostBit, a candidate-cost-informed blockwise bit allocation method for PTQ of LLMs. CostBit can accept a user-specified target bit budget, which may take non-integer values. Figure 3 illustrates the overall pipeline.

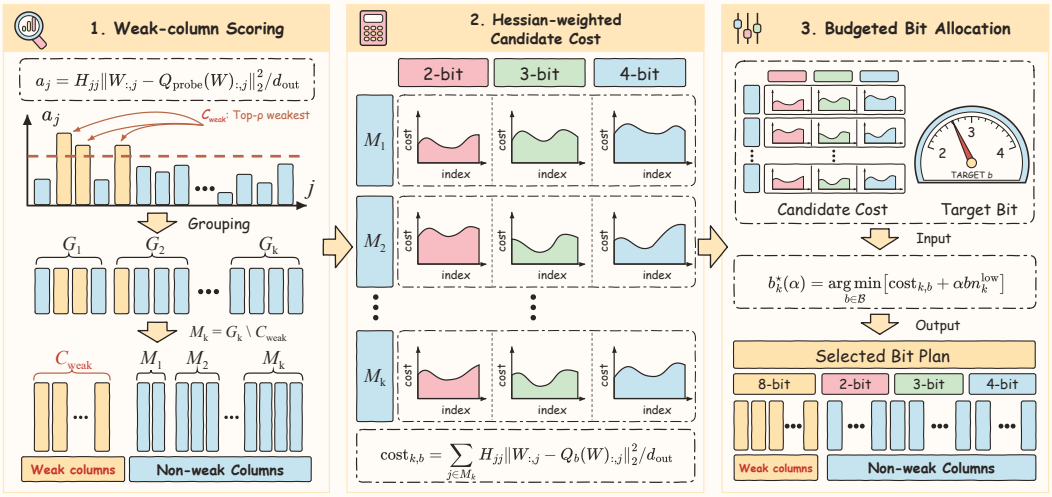


Figure 3: The overall pipeline of CostBit.

4.1 Weak-Column Scoring

Before computing candidate costs, CostBit protects extremely sensitive columns from low-bit quantization. For each input-channel column j, we use a probe quantizer Q_{probe} to compute its quantization error and Hessian-weighted saliency:

$$e_j = \frac{1}{d_{\text{out}}} \|W_{:,j} - Q_{\text{probe}}(W)_{:,j}\|_2^2, \qquad a_j = H_{jj} e_j. \tag{4}$$

where a_j is the Hessian-weighted column sensitivity score. CostBit preserves the top ρ fraction of columns according to a_j as weak columns, denoted by $\mathcal{C}_{\text{weak}}$. We store weak columns at $b_{\text{high}} = 8$ by default with $\rho = 0.005$. The weak-column cost is included when computing the actual average bit-width. Further, we partition the input-channel axis into consecutive groups $\{G_k\}_{k=1}^K$. We then estimate each candidate bit-width's Hessian-weighted cost and allocate shared bit-widths to non-weak columns in each group under budget constraints, as shown in the subsequent subsections.

4.2 Hessian-Weighted Candidate Cost

For each input-channel group G_k and each candidate bit-width $b \in \mathcal{B} = \{2, 3, 4\}$, CostBit estimates the cost of assigning bit-width b to the non-weak columns in this group. Let $M_k = G_k \setminus \mathcal{C}_{\text{weak}}$ denote the set of non-weak columns in group G_k. For each column $j \in M_k$, we compute the column-wise quantization error under bit-width b as:

$$\epsilon_{j,b} = \frac{1}{d_{\text{out}}} \|W_{:,j} - Q_b(W)_{:,j}\|_2^2, \tag{5}$$

where $Q_b(\cdot)$ denotes the temporary b-bit quantizer.

Further, CostBit weights this quantization error by the corresponding Hessian diagonal entry. The candidate cost of assigning bit-width b to M_k is defined as:

$$\text{cost}_{k,b} = \sum_{j \in M_k} H_{jj} \epsilon_{j,b}, \tag{6}$$

where H_{jj} is the sensitivity of the layer-wise quantization error to perturbations of weight column j. Hence, quantization errors on columns with larger Hessian diagonal values receive larger weights in the candidate cost. Eq. 6 can be viewed as a diagonal approximation to the Hessian-weighted quantization error over M_k under bit-width b. In ablation study in Table 3, we compare the candidate cost in Eq. 6 with alternative costs based on the mean Hessian trace and Hessian spectral norm.

Note that the summed cost is adopted rather than the average cost over M_k so that the candidate cost captures each group's total quantization error and scales naturally with non-weak column counts.

This is consistent with the budgeted allocation, where the corresponding bit cost also scales with $|M_k|$. Although Eq. 6 uses column-level Hessian diagonal entries, CostBit does not assign bit-widths independently to individual columns. Instead, it aggregates column-wise costs into group-level estimates and assigns a shared bit-width to all non-weak columns per group. It preserves fine-grained Hessian information for block-sensitivity estimation while maintaining quantized-layout regularity. Compared with per-column mixed-precision quantization, bit-width sharing eases weight packing, dequantization, and CUDA kernel development at inference time.

4.3 Budgeted Bit Allocation

Given the candidate costs $\{\mathrm{cost}_{k,b}\}$ and a target bit budget $\bar{b}$, CostBit selects one bit-width for each block. Specifically, we introduce a Lagrange multiplier α to balance the candidate cost and bit cost. For a fixed α, each group independently selects its bit-width as:

$$b_k^\star(\alpha) = \arg\min_{b\in\mathcal{B}} \left[\mathrm{cost}_{k,b} + \alpha\, b\, n_k^{\mathrm{low}}\right], \tag{7}$$

where $n_k^{\mathrm{low}} = |M_k|$ is the number of non-weak columns in group G_k. The first term measures the cost of assigning candidate bit-width b to block M_k, while the second term penalizes bit cost. A larger α imposes a stronger penalty on higher bit-widths, thereby favoring lower-bit assignments.

In the current implementation, the budgeted bit allocation is performed independently for each quantized linear layer. For a single layer, the actual average bit-width is:

$$\bar{b}_{\mathrm{layer}} = \frac{\sum_{k=1}^{K} b_k n_k^{\mathrm{low}} + b_{\mathrm{high}} \sum_{k=1}^{K} n_k^{\mathrm{high}}}{d_{\mathrm{in}}}, \tag{8}$$

where b_k is the bit-width assigned to the non-weak columns in group G_k, b_{high} is the weak-column bit-width, and n_k^{high} is the number of weak columns in that group. The denominator is d_{in} because the column groups form a partition of the input-channel dimension: $\sum_k (n_k^{\mathrm{low}} + n_k^{\mathrm{high}}) = d_{\mathrm{in}}$.

For reporting the final average bit-width of the whole quantized model, CostBit uses a parameter-weighted average over all quantized linear layers:

$$\bar{b}_{\mathrm{model}} = \frac{\sum_{\ell\in\mathcal{L}} \sum_{k=1}^{K_\ell} d_{\mathrm{out}}^{(\ell)} \left(b_k^{(\ell)} n_{k,\ell}^{\mathrm{low}} + b_{\mathrm{high}} n_{k,\ell}^{\mathrm{high}}\right)}{\sum_{\ell\in\mathcal{L}} d_{\mathrm{out}}^{(\ell)} d_{\mathrm{in}}^{(\ell)}}, \tag{9}$$

where $\mathcal{L}$ denotes the set of quantized linear layers, the superscript (ℓ) indicates the layer index, and K_ℓ is the number of groups in layer ℓ. The terms $n_{k,\ell}^{\mathrm{low}}$ and $n_{k,\ell}^{\mathrm{high}}$ denote the numbers of non-weak and weak columns in group k of layer ℓ, respectively, and b_{high} is the bit-width of weak columns.

According to Eq. 8, the default configuration with 0.5% INT8 weak columns has a minimum average bit-width of approximately 2.03 bits when all non-weak columns use 2 bits. To reach the extreme-compression point at 2.01 bits, we bypass blockwise allocation at this endpoint, adopt a 2-bit base, and retain only enough INT8 weak columns to consume an additional budget of 0.01 bits per weight.

Binary Search. CostBit finds an allocation close to the target $\bar{b}$ by binary search over α. For a given α, Eq. 7 determines the bit-width of each group independently. The resulting actual average bit-width is then computed based on Eq. 8. If the actual average bit-width is larger than the target bit budget, α is increased to penalize higher bit-widths more strongly. if it is smaller than the target, α is decreased to allow more high-bit assignments. The assignment with the actual average bit-width is closest to $\bar{b}$ is retained.

Local Adjustment. Since the selection of bit-width is discrete, the binary search may not match the target exactly. Hence, CostBit applies a small local adjustment step after the search. If the actual average bit-width is above the target, CostBit greedily decreases selected blocks to the next lower candidate bit-width, prioritizing changes with the smallest loss increase per saved bit. If it is below the target, CostBit greedily increases selected blocks to the next higher candidate bit-width, prioritizing changes with the largest loss reduction per additional bit. The adjustment stops once no adjacent single-block change can further reduce the target mismatch.

Second-Order Reconstruction. After bit allocation, CostBit conducts sequential second-order reconstruction with assigned block bit-widths and inverse-Hessian error compensation, retaining weak columns in INT8 format. The full procedure is outlined in the Appendix A.

Table 1: WikiText-2 perplexity on the LLaMA and OPT families. Lower is better.

		LLaMA					OPT							
Method	Bits	2-7B	2-13B	2-70B	3-8B	3-70B	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B
FP16	16.0	5.47	4.88	3.31	5.75	2.90	27.65	22.00	14.62	12.47	10.86	10.13	9.56	9.34
GPTQ	2.0	24.16	12.17	6.72	926.64	25.01	283.18	435.15	53.77	31.70	16.36	16.85	11.97	713.93
GPTAQ	2.0	14.43	8.84	6.55	89.65	20.92	134.30	79.10	42.46	24.92	15.14	15.51	11.78	17.88
OWQ	2.01	8.32	7.23	4.94	15.11	8.00	91.82	60.14	21.26	16.53	13.05	11.91	10.59	10.12
CostBit	2.01	**7.63**	**6.53**	**4.85**	**12.04**	**7.51**	**52.59**	**34.71**	**20.53**	**15.97**	**12.87**	**11.58**	**10.52**	10.26
OWQ	2.25	7.91	6.82	4.79	12.80	7.33	45.50	56.63	19.81	16.11	13.03	11.60	10.56	10.08
CostBit	2.25	**6.92**	**6.02**	**4.50**	**10.10**	**6.82**	**42.29**	**30.41**	**18.16**	**14.91**	**12.27**	**11.11**	**10.24**	**9.93**
OWQ	2.5	7.69	6.69	4.76	12.14	7.00	41.98	49.33	19.57	16.01	12.84	11.55	10.54	10.03
CostBit	2.5	**6.43**	**5.64**	**4.17**	**8.77**	**6.06**	**34.34**	**27.78**	**16.88**	**14.00**	**11.75**	**10.75**	**9.97**	**9.68**
OWQ	2.75	7.54	6.54	4.68	11.31	6.93	41.74	63.16	19.13	15.88	12.80	11.47	10.42	10.05
CostBit	2.75	**6.08**	**5.36**	**3.88**	**7.84**	**5.18**	**32.88**	**25.58**	**16.00**	**13.40**	**11.37**	**10.47**	**9.78**	**9.53**
GPTQ	3.0	6.27	5.30	3.76	19.58	4.88	34.09	26.30	15.91	13.50	11.36	10.44	9.72	10.78
GPTAQ	3.0	6.01	5.23	3.75	9.06	4.79	32.39	24.67	16.02	13.35	11.30	10.37	9.76	10.37
OWQ	3.01	5.86	5.16	3.62	7.24	4.25	35.20	24.96	15.30	12.95	11.15	10.33	9.64	9.42
CostBit	3.0	**5.80**	**5.14**	3.62	**7.16**	4.27	**30.56**	**23.80**	15.36	**12.94**	**11.12**	**10.28**	9.68	9.42
OWQ	3.5	5.79	5.13	3.59	6.99	4.08	30.15	24.43	15.26	12.86	11.10	10.30	9.66	9.42
CostBit	3.5	**5.65**	**5.02**	**3.49**	**6.72**	**3.65**	**29.28**	**22.90**	**14.97**	**12.74**	**11.00**	**10.23**	**9.62**	**9.38**
GPTQ	4.0	5.61	4.96	3.40	6.96	3.25	29.50	23.21	14.87	12.76	10.97	10.18	9.59	9.51
GPTAQ	4.0	5.60	4.95	3.39	6.53	3.21	28.83	22.61	14.85	12.67	10.95	10.19	9.59	9.42
OWQ	4.01	5.56	4.94	3.39	6.42	3.18	29.48	22.83	14.83	12.59	10.92	10.15	9.58	9.34
CostBit	4.0	**5.54**	4.94	**3.38**	**6.38**	**3.16**	**28.69**	**22.31**	**14.75**	**12.57**	**10.91**	10.16	9.58	9.35

5 EXPERIMENTS

5.1 SETTINGS

Quantization. We evaluate CostBit at target bit budget ranging from 2.01 to 4 bits. Candidate bit-widths are selected from $\mathcal{B} = \{2, 3, 4\}$ and assigned to input-channel blocks of size 128. For the 2.01-bit endpoint, we use the extreme-compression fallback described in Section 4.3. For other target bit budgets, we preserve 0.5% of columns as INT8 weak columns. Throughout the paper, the Bits column reports the nominal target for due to blockwise allocation. Unless otherwise specified, calibration uses 128 randomly sampled WikiText-2 sequences (Merity et al., 2017) of length 2048. For each block, CostBit constructs an asymmetric quantizer for each candidate bit-width using a 10-step one-dimensional MSE range search and estimates the corresponding candidate cost for bit-width allocation. Once the allocation is fixed, CostBit performs the final low-bit quantization using a 40-step asymmetric two-dimensional clipping search, followed by GPTAQ reconstruction with $\alpha_{\text{GPTAQ}} = 0.25$. We use true-sequential quantization in all experiments, with activation ordering disabled for the LLaMA models. For reference, at the 2.5-bit target, CostBit quantizes LLaMA-2-70B in approximately 3–4 hours on a single NVIDIA H20 (96 GB).

Models. We evaluate OPT (Zhang et al., 2022) from 125M to 66B, LLaMA-2 (Touvron et al., 2023) from 7B to 70B, and LLaMA-3 (Grattafiori et al., 2024) from 8B to 70B.

Evaluation. All models are calibrated on WikiText-2. For evaluation, we report perplexity on WikiText-2 (Merity et al., 2017) and C4 (Raffel et al., 2020), together with average zero-shot accuracy on PIQA (Bisk et al., 2020), ARC-easy and ARC-challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), and LAMBADA (Paperno et al., 2016). Zero-shot evaluation uses the Language Model Evaluation Harness (Gao et al., 2023), with identical settings across all evaluated methods and models.

Baselines. We compare CostBit with GPTQ (Frantar et al., 2023), GPTAQ (Li et al., 2025) , and OWQ (Lee et al., 2024). The introductory comparison additionally includes the published SliM-LLM and SliM-LLM+ results (Huang et al., 2025). The reported bit-widths for GPTQ and GPTAQ correspond to their integer bit-widths, whereas OWQ bit-widths include the 0.01-bit overhead introduced by retaining weak columns at high precision. We use the FP16 perplexities reported by SliM-LLM (Huang et al., 2025) as the FP16 baseline reference for LLaMA-2 and LLaMA-3.

Table 2: Average zero-shot accuracy (%) on the LLaMA family over PIQA, ARC-easy, ARC-challenge, HellaSwag, WinoGrande, and LAMBADA. Higher is better.

Method	Bits	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B	LLaMA-3-8B	LLaMA-3-70B
FP16	16.0	69.82	72.59	77.07	73.22	79.94
GPTQ	2.0	34.28	40.07	57.66	29.50	36.49
GPTAQ	2.0	37.71	49.45	56.86	31.00	35.45
OWQ	2.01	55.98	58.60	72.23	44.99	66.70
CostBit	2.01	**56.02**	**60.95**	**72.57**	**47.23**	**68.92**
OWQ	2.25	59.53	62.00	73.32	54.92	72.34
CostBit	2.25	**61.63**	**65.05**	**74.60**	54.57	**72.36**
OWQ	2.5	60.04	63.04	73.23	56.93	72.33
CostBit	2.5	**63.79**	**67.29**	**75.70**	**63.54**	**74.76**
OWQ	2.75	59.92	63.78	73.40	57.33	73.46
CostBit	2.75	**66.24**	**69.69**	**76.34**	**65.87**	**77.16**
GPTQ	3.0	65.91	70.23	75.11	60.86	73.63
GPTAQ	3.0	66.61	70.48	75.84	65.24	75.46
OWQ	3.01	67.57	71.23	76.10	69.48	77.23
CostBit	3.0	**68.65**	71.19	**76.83**	**69.92**	**77.59**
OWQ	3.5	68.51	71.59	76.63	70.59	78.48
CostBit	3.5	**68.56**	**71.70**	**76.69**	**71.94**	**78.60**
GPTQ	4.0	69.37	72.21	77.02	72.16	77.69
GPTAQ	4.0	68.73	72.15	76.95	72.28	78.38
OWQ	4.01	69.44	72.06	76.93	72.93	79.02
CostBit	4.0	68.86	**72.52**	76.92	72.41	78.74

5.2 Perplexity Results

Table 1 reports WikiText-2 perplexity for the LLaMA and OPT families, with the corresponding C4 results in Appendices B.1 and B.2. As shown in the tables, CostBit generally outperforms prior weight-only post-training quantization methods across the OPT, LLaMA-2, and LLaMA-3 model families over bit-widths ranging from 2.01 to 4 bits. At approximately 2.5 bits, CostBit reduces perplexity by 12.4%–27.8% relative to OWQ across the five evaluated LLaMA models. The OPT results further show that CostBit is effective across model scales, delivering substantial perplexity reductions even on smaller models, ranging from OPT-125M to OPT-2.7B. The advantage is particularly pronounced in the sub-3-bit regime, where quantization error is more severe and bit-width allocation becomes more critical. This trend highlights the benefit of allocating bit-widths according to sensitivity. At higher bit-widths, the performance gap narrows as quantization error decreases and all methods approach the FP16 baseline.

5.3 Zero-Shot Results

Table 2 compares the average zero-shot accuracy of CostBit and baselines across five LLaMAs. At 2.5 bits, CostBit improves the six-task average accuracy over OWQ by +2.43% $\sim$ +6.61%, and its accuracy on LLaMA-2-70B and LLaMA-3-70B is only 1.37% and 5.18% below the corresponding FP16 baselines, respectively. At 3.5 bits, these gaps narrow to 0.38% and 1.34%. It shows that candidate-cost-informed bit allocation preserves downstream performance under aggressive compression and achieves near-FP16 zero-shot accuracy for LLaMA-2-70B and LLaMA-3-70B.

5.4 Ablation Studies

Unless otherwise specified, all ablation experiments use WikiText-2 calibration, a sequence length of 2048, a group size of 128, GPTAQ reconstruction with $\alpha_{\mathrm{GPTAQ}} = 0.25$, and 0.5% INT8 weak columns. In each experiment, only the component under study is changed. The weak-column-ratio and scale-dependent activation-ordering studies are reported in Appendices B.3 and B.4.

Table 3 reports the core component ablations on LLaMA-2-7B and LLaMA-3-8B. **(1) Allocation rule:** sensitivity-based Allocation ranks blocks using Hessian-weighted scores and allocates bit-

Table 3: Ablations at 2.5 bits on LLaMA-2-7B/LLaMA-3-8B. We report WikiText-2 and C4 perplexity. Lower is better. For each component, the variant adopted in CostBit is listed first.

Component	Variant	Actual bits	WikiText-2	C4
Allocation	**Candidate Cost**	2.4961/2.4976	**6.43/8.77**	**9.78/16.80**
	Sensitivity	2.4939/2.4970	6.76/9.55	10.60/19.84
	Random	2.4989/2.4991	6.65/9.21	10.29/18.55
Block cost	**Hessian-weighted error**	2.4961/2.4976	**6.43**/8.77	**9.78**/16.80
	Mean Hessian trace	2.4923/2.4947	6.46/**8.74**	9.82/**16.68**
	Hessian spectral norm	2.4962/2.4984	6.51/8.84	9.90/17.23
Reconstruction	**GPTAQ**	2.4961/2.4976	**6.43/8.77**	**9.78/16.80**
	GPTQ	2.4961/2.4977	6.55/9.28	9.94/18.16
Weak format	**INT8**	2.4961/2.4976	**6.43/8.77**	**9.78/16.80**
	FP16	2.5021/2.5033	6.51/8.89	9.93/17.48

widths according to the ranking. Relative to Salience, Cost-based Allocation reduces WikiText-2/C4 perplexity by 0.33/0.82 on LLaMA-2-7B and 0.78/3.04 on LLaMA-3-8B; relative to the mean of five Random allocation seeds, the reductions are 0.22/0.51 and 0.44/1.75. These results highlight the benefit of selecting block bit-widths according to candidate costs. **(2) Block cost:** Hessian-weighted error performs best on LLaMA-2-7B, while mean Hessian trace is marginally better on LLaMA-3-8B and Hessian spectral norm is consistently worse; **(3) Reconstruction backend:** GPTAQ consistently outperforms GPTQ, reducing WikiText-2/C4 perplexity by 0.12/0.16 and 0.51/1.36 on the two models; **(4) Weak-column format:** With the weak-column ratio fixed at 0.5%, INT8 outperforms FP16 on both models at comparable actual average bit-widths, since FP16 preserves weak columns more accurately but leaves less budget for low-bit blocks.

5.5 Deployment Efficiency

We implement the CostBit CUDA kernel to support mixed 2/3/4-bit weights and INT8 weak columns. Table 4 reports decoding efficiency on a NVIDIA H20 (96GB) with CUDA graphs, a one-token prompt, 128 generated tokens, and preallocated KV storage. We also deploy LLaMA-2-70B Base at 2.4987 bits on a RTX 4090 (24GB), with generation examples provided in Appendix C. Table 4 shows that CostBit reaches 182.76 token/s on LLaMA-2-7B at 2.4961 bits, 38.5% above FP16, while reducing weight memory by 80.1% relative to FP16 (a compression ratio of 5.02×). This result shows that CostBit can retain its memory advantage without sacrificing decoding throughput.

Table 4: Deployment with our CUDA kernel. WikiText-2 perplexity (PPL), weight memory (WM), runtime memory (RM), and decoding speed on an NVIDIA H20 (96GB) are reported.

Method	Actual bits	PPL↓	WM (MB)	RM (MB)	token/s	Compression
FP16	16.0	5.47	12916.52	13082.00	131.94	1.00×
CostBit	3.9905	5.54	3732.35	3964.00	170.47	3.46×
CostBit	3.4925	5.65	3346.35	3624.00	181.50	3.86×
CostBit	2.9951	5.80	2960.35	3176.00	181.74	4.36×
CostBit	2.4961	6.43	2574.35	2808.00	182.76	5.02×
CostBit	2.0093	7.63	2190.34	2406.00	176.62	5.90×

6 Conclusion

We propose CostBit, a candidate-cost-informed blockwise bit allocation method for PTQ of LLMs. It estimates Hessian-weighted quantization costs for candidate bit-widths and allocates block bit-widths to closely match a user-specified target bit budget, while preserving weak columns at higher precision. Experiments across the OPT and LLaMA families demonstrate improved accuracy–compression trade-offs in perplexity and zero-shot accuracy, particularly below 3 bits. With the CostBit CUDA kernel, LLaMA-2-70B quantized to approximately 2.5 bits runs entirely on a single RTX 4090 (24 GB), showing the practical deployment benefits of non-integer average bit-widths.

AI USE STATEMENT

In this work, we used generative AI tools to assist with editing and polishing the writing for clarity and readability. We did not use generative AI tools for any tasks listed under required disclosure. All AI-assisted edits were reviewed by the authors to ensure correctness and preserve the intended meaning. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7432–7439, 2020.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems*, pp. 4396–4429, 2023.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality, 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

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. *arXiv preprint arXiv:1803.05457*, 2018.

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

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *International Conference on Learning Representations*, pp. 5733–5761, 2024.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 293–302, 2019.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Advances in Neural Information Processing Systems*, pp. 18518–18529, 2020.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023. URL https://github.com/EleutherAI/lm-evaluation-harness.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1992.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. SliM-LLM: Salience-driven mixed-precision quantization for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 25672–25692, 2025.

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Accurate post training quantization with small calibration sets. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 4466–4475, 2021.

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 23901–23923, 2024.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1989.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. OWQ: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 13355–13364, 2024.

Yuhang Li, Ruokai Yin, Donghyun Lee, Shiting Xiao, and Priyadarshini Panda. GPTAQ: Efficient finetuning-free quantization for asymmetric calibration. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 36690–36706, 2025.

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

Yuexiao Ma, Huixia Li, Xiawu Zheng, Feng Ling, Xuefeng Xiao, Rui Wang, Shilei Wen, Fei Chao, and Rongrong Ji. AffineQuant: Affine transformation quantization for large language models. In *International Conference on Learning Representations*, pp. 50932–50951, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 7197–7206, 2020.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, pp. 606–624, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: an adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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. In *International Conference on Learning Representations*, pp. 45472–45496, 2024.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 31094–31116, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

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

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V3: Dyadic neural network quantization. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 11875–11886, 2021.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

The appendix is organized as follows: Appendix A summarizes the complete CostBit procedure. Appendix B provides additional experimental results. Appendix C presents generation examples. Appendix D derives the weight–Hessian block correspondence used by CostBit.

A Overall Algorithm

Algorithm 1 summarizes the layerwise procedure under the regular configuration. The 2.01-bit setting is described separately in Section 4.3. For each quantized linear layer, CostBit identifies weak columns, estimates a candidate-specific cost for every block–bit pair, solves the budgeted bit allocation, and reconstructs the weights using the selected bit-widths.

Algorithm 1 CostBit: Candidate-Cost-Informed Blockwise Bit Allocation

Input: weights $\{W^{(\ell)}\}$, calibration data $\mathcal{D}$, $\mathcal{B} = \{2, 3, 4\}$, group size g, weak ratio ρ, weak-column bit-width b_{high}, and target bit budget $\bar{b}$
for each quantized linear layer W in model order **do**
  Collect its input activations X and form $H = 2XX^\top$
  Probe-quantize W and compute $a_j = H_{jj}e_j$ using Eq. 4
  Set $\mathcal{C}_{\text{weak}}$ to the top ρ fraction under a_j
  Move weak columns to the end (outliers last)
  Partition the input columns into consecutive groups $\{G_k\}_{k=1}^{K}$ of size g
  for each group G_k and candidate bit-width $b \in \mathcal{B}$ **do**
    Set $M_k = G_k \setminus \mathcal{C}_{\text{weak}}$ and compute $\epsilon_{j,b}$ using Eq. 5
    Compute $\text{cost}_{k,b} = \sum_{j \in M_k} H_{jj}\epsilon_{j,b}$
  end for
  Binary-search α using Eqs. 7 and 8, retaining the assignment closest to $\bar{b}$
  Locally adjust block bit-widths to reduce the budget mismatch while accounting for weak columns stored at b_{high} bits
  Quantize columns sequentially at their assigned bit-widths with second-order error compensation
  Restore the quantized weight columns to their original order
  Update calibration activations for subsequent layers using the quantized weights
end for
Output: quantized model and blockwise bit assignments

B Additional Experimental Results

B.1 LLaMA C4 Perplexity

Table 5: C4 perplexity on the LLaMA family. Lower is better.

Method	Bits	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B	LLaMA-3-8B	LLaMA-3-70B
FP16	16.0	6.97	6.46	5.52	9.22	6.85
GPTQ	2.0	54.15	25.78	11.92	3574.83	253.71
GPTAQ	2.0	30.97	16.54	11.50	512.01	66.11
OWQ	2.01	14.10	12.70	8.19	47.62	16.91
CostBit	2.01	**12.98**	**11.39**	**8.05**	**34.15**	**14.30**
OWQ	2.25	12.84	11.28	8.02	31.31	14.55
CostBit	2.25	**10.93**	**9.55**	**7.36**	**22.30**	**12.12**
OWQ	2.5	12.30	11.13	7.91	26.59	13.84
CostBit	2.5	**9.78**	**8.59**	**6.80**	**16.80**	**10.70**
OWQ	2.75	12.11	10.75	7.83	23.73	13.58
CostBit	2.75	**8.87**	**7.87**	**6.36**	**14.18**	**9.55**
GPTQ	3.0	8.67	7.56	6.15	17.73	15.96
GPTAQ	3.0	8.52	7.48	6.14	14.18	9.23
OWQ	3.01	8.20	7.34	6.02	12.53	8.48
CostBit	3.0	**8.16**	**7.33**	6.03	**12.41**	8.48
OWQ	3.5	8.05	7.24	5.99	11.80	8.27
CostBit	3.5	**7.71**	**7.04**	**5.87**	**11.13**	**7.88**
GPTQ	4.0	7.51	6.87	5.78	10.64	7.62
GPTAQ	4.0	7.47	6.86	5.78	10.34	7.69
OWQ	4.01	7.44	6.85	5.77	10.15	7.47
CostBit	4.0	7.44	6.85	5.77	10.16	**7.44**

B.2 OPT C4 Perplexity

Table 6: C4 perplexity on the OPT family. Lower is better.

Method	Bits	OPT-125M	OPT-350M	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B	OPT-66B
FP16	16.0	26.56	22.59	16.07	14.34	12.71	12.06	11.44	10.99
GPTQ	2.0	565.07	607.60	158.31	122.91	28.80	21.10	15.55	1367.93
GPTAQ	2.0	231.40	107.56	109.09	67.07	26.26	20.15	15.20	27.69
OWQ	2.01	147.53	76.88	27.83	22.17	17.56	15.93	13.80	12.70
CostBit	2.01	**61.04**	**43.05**	**26.81**	**21.97**	**17.36**	**15.57**	13.87	13.34
OWQ	2.25	54.24	65.52	26.34	21.85	17.43	15.43	13.60	12.65
CostBit	2.25	**45.17**	**35.31**	**22.59**	**19.59**	**15.68**	**14.36**	**12.98**	**12.34**
OWQ	2.5	49.47	59.37	25.46	21.26	17.06	15.34	13.50	12.55
CostBit	2.5	**35.58**	**31.14**	**19.96**	**17.26**	**14.51**	**13.49**	**12.39**	**11.74**
OWQ	2.75	48.83	68.44	25.05	20.79	16.91	15.12	13.41	12.47
CostBit	2.75	**33.21**	**27.69**	**18.43**	**16.09**	**13.77**	**12.88**	**11.98**	**11.39**
GPTQ	3.0	32.90	27.61	18.02	15.84	13.55	12.56	11.78	12.29
GPTAQ	3.0	31.95	25.91	18.15	15.64	13.49	12.54	11.77	11.97
OWQ	3.01	33.37	26.67	17.34	15.26	13.21	12.46	11.68	11.16
CostBit	3.0	**30.18**	**25.23**	**17.23**	15.27	13.25	12.46	11.69	11.18
OWQ	3.5	29.47	25.29	17.11	15.09	13.17	12.40	11.65	11.14
CostBit	3.5	**28.46**	**23.86**	**16.73**	**14.84**	**12.98**	**12.26**	**11.58**	**11.08**
GPTQ	4.0	28.01	24.12	16.53	14.71	12.91	12.16	11.51	11.11
GPTAQ	4.0	27.70	23.39	16.47	14.65	12.87	12.15	11.51	11.06
OWQ	4.01	27.88	23.42	16.34	14.56	12.82	12.12	11.49	11.01
CostBit	4.0	**27.62**	**23.08**	**16.32**	**14.52**	12.82	12.13	11.49	11.02

B.3 WEAK-COLUMN RATIO

Table 7 evaluates the weak-column ratio at a target bit budget of 2.5 bits. A ratio of 0.25% achieves the lowest perplexity, but its actual average bit-width slightly exceeds the target. We use 0.5% by default, which provides comparable perplexity while more closely matching the target bit budget.

Table 7: Weak-column ratio ablation at 2.5 bits on LLaMA-2-7B and OPT-6.7B. We report WikiText-2 and C4 perplexity. Lower is better.

Model	Weak columns	Actual bits	WikiText-2	C4
LLaMA-2-7B	0%	2.5000	6.67	10.04
LLaMA-2-7B	0.25%	2.5102	**6.43**	**9.70**
LLaMA-2-7B	0.5%	2.4961	**6.43**	9.78
LLaMA-2-7B	0.75%	2.5065	6.47	9.84
LLaMA-2-7B	1%	2.4997	6.49	9.95
OPT-6.7B	0%	2.5000	11.84	14.78
OPT-6.7B	0.25%	2.5094	**11.71**	**14.47**
OPT-6.7B	0.5%	2.4983	11.75	14.51
OPT-6.7B	0.75%	2.5073	11.76	14.52
OPT-6.7B	1%	2.4971	11.82	14.62

B.4 ACTIVATION ORDERING ACROSS MODEL SCALE

We evaluate activation ordering at 2.5 bits across OPT, LLaMA-2-7B, and LLaMA-3-8B. Its effect is model-dependent: it helps the two smallest OPT models and LLaMA-3-8B, while changes on larger OPT models and LLaMA-2-7B are small or adverse.

Table 8: Activation-ordering ablation at 2.5 bits on the LLaMA and OPT families. We report WikiText-2 and C4 perplexity. Lower is better.

Model	Activation ordering	Actual bits	WikiText-2	C4
OPT-125M	off	2.5090	39.02	40.94
OPT-125M	on	2.5115	**34.34**	**35.58**
OPT-350M	off	2.5154	28.22	31.46
OPT-350M	on	2.5189	**27.78**	**31.14**
OPT-1.3B	off	2.5169	**16.88**	**19.96**
OPT-1.3B	on	2.5189	17.11	20.34
OPT-2.7B	off	2.4866	**14.00**	**17.26**
OPT-2.7B	on	2.4870	14.08	17.47
OPT-6.7B	off	2.4983	**11.75**	**14.51**
OPT-6.7B	on	2.4991	11.78	14.65
LLaMA-2-7B	off	2.4961	**6.43**	9.78
LLaMA-2-7B	on	2.4989	6.45	**9.73**
LLaMA-3-8B	off	2.4976	8.77	16.80
LLaMA-3-8B	on	2.4994	**8.46**	**15.41**

B.5 BLOCK-SIZE SENSITIVITY

We vary the input-channel block size on LLaMA-2-7B and LLaMA-3-8B at a target bit budget of 2.5 while keeping all other settings fixed. In the current implementation, the same grouping is used for blockwise bit allocation and group-wise scale estimation. As shown in Table 9, smaller blocks yield lower perplexity on both datasets at comparable actual average bit-widths. Reducing the block size from 128 to 32 lowers WikiText-2/C4 perplexity by 0.24/0.58 on LLaMA-2-7B and 0.79/2.31 on LLaMA-3-8B.

Table 9: Block-size ablation at 2.5 bits on LLaMA-2-7B/LLaMA-3-8B. We report WikiText-2 and C4 perplexity. Lower is better.

Block size	Actual bits	WikiText-2	C4
32	2.5010/2.5009	**6.19/7.98**	**9.20/14.49**
64	2.4993/2.5011	6.35/8.39	9.60/15.67
128	2.4961/2.4976	6.43/8.77	9.78/16.80

B.6 Calibration-Seed Robustness

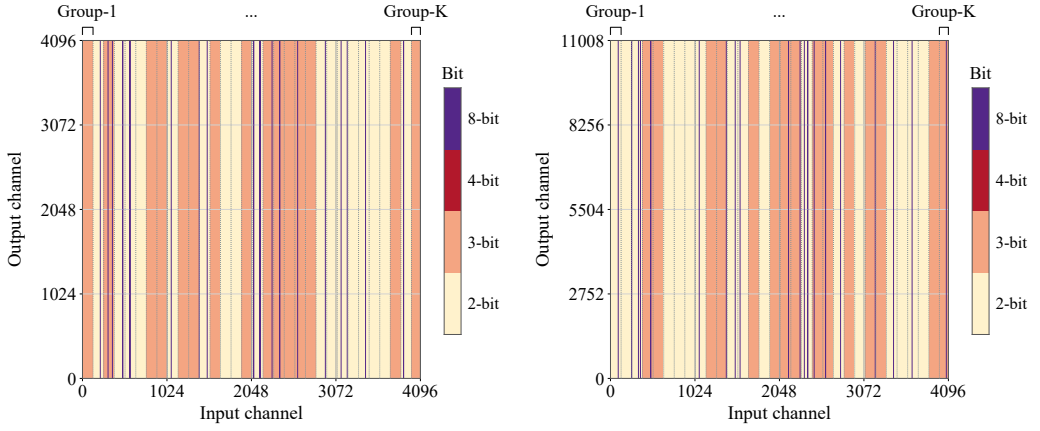
Table 10 evaluates CostBit on LLaMA-2-7B and LLaMA-3-8B with three calibration seeds while keeping the calibration set size and all quantization settings fixed. The small variation across seeds on both models indicates that the method is not dependent on a particular sampled calibration subset.

Table 10: Calibration-seed robustness at 2.5 bits on LLaMA-2-7B/LLaMA-3-8B. We report WikiText-2 and C4 perplexity. Lower is better.

Calibration seed	Actual bits	WikiText-2	C4
0	2.4962/2.4978	6.43/8.80	9.78/17.01
10	2.4960/2.4977	6.44/8.79	9.79/16.84
42	2.4961/2.4976	6.43/8.77	9.78/16.80

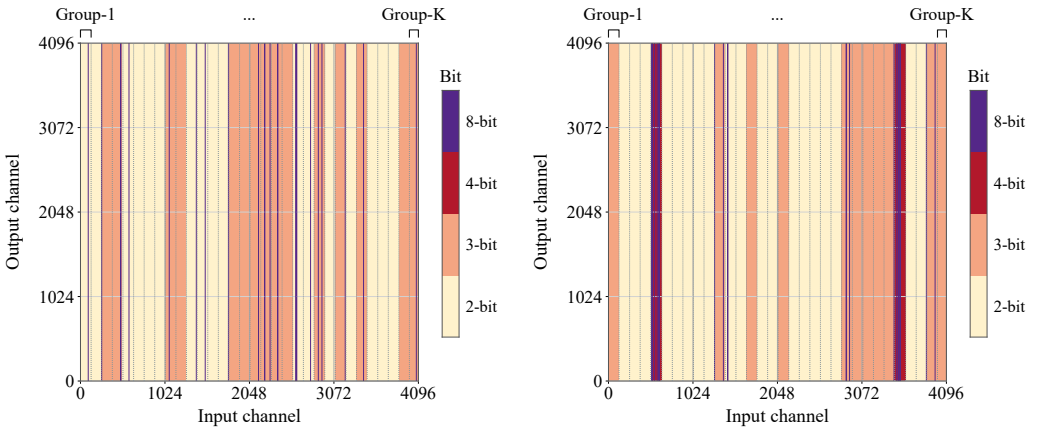
B.7 Bit Allocation Visualization

Figure 4 visualizes blockwise bit-width allocation for four representative LLaMA-2-7B projections. The vertical regions show how CostBit assigns different candidate bit-widths across input-channel blocks under the same target bit budget.



(a) layer-1-Q (b) layer-10-Up

Figure 4: Blockwise bit-width allocation on LLaMA-2-7B at a 2.5-bit target: (a) layer-1-Q; (b) layer-10-Up.



(c) layer-27-K (d) layer-31-Out

Figure 4: Continued. Blockwise bit-width allocation on LLaMA-2-7B at 2.5 bits: (c) layer-27-K; (d) layer-31-Out.

C Generation Examples

We present example responses to questions from the Vicuna Benchmark (Chiang et al., 2023), generated by CostBit-quantized LLaMA-2-70B Base (2.4987-bit) on a single RTX 4090 (24 GB). For these examples, the model remains fully GPU-resident, with a sampled peak process GPU memory usage of 23,644 MiB (23.09 GiB) measured by `nvidia-smi`.

LLaMA-2-70B Base

Prompt:	How can I increase my productivity while working from home?
CostBit (2.4987-bit):	`There are a few ways to increase your productivity while working from home. One way is to create a dedicated workspace that is free of distractions. Another way is to set specific goals and deadlines for yourself, and to hold yourself accountable to them. Finally, it is important to take breaks throughout the day to avoid burnout.`
Prompt:	What are some subtle clues that suggest someone is pretending to understand a topic or conversation when they are actually confused or uninformed?
CostBit (2.4987-bit):	`There are a few subtle clues that suggest someone is pretending to understand a topic or conversation when they are actually confused or uninformed. One clue is when the person is asking questions that seem irrelevant to the topic being discussed. Another clue is when the person is using words or phrases that seem out of place for the context of the conversation. Finally, when the person is making comments or statements that seem to contradict what has been said previously, it may be a sign that they are not understanding the conversation.`
Prompt:	As a space colonist on Mars, describe your daily life and the challenges you face living on another planet.
CostBit (2.4987-bit):	`On Mars, my days are filled with physical labor and the constant struggle to survive in an alien environment. I spend most of my time repairing equipment, farming, and maintaining our habitat. The lack of resources and the harsh climate make life difficult, but I am determined to overcome these challenges. Despite the difficulties, I find comfort in the beauty of Mars's landscape and the camaraderie of my fellow colonists.`

D Derivation of the Weight–Hessian Block Correspondence

We use the notation of Section 3. With W and X fixed, we regard layer-wise quantization error $E(\widehat{W})$ as a function of the real-valued entries of $\widehat{W}$. Let $\Delta w_r = w_r - \widehat{w}_r$ denote the quantization error in output row r.

Proposition (Quantization error induced by a column block). Let G_k be a group of input-channel indices, and suppose that $\widehat{W}_{:,j} = W_{:,j}$ for every $j \notin G_k$. Then the layer-wise quantization error can be written as

$$E(\widehat{W}) = \frac{1}{2} \sum_{r=1}^{d_{\text{out}}} \Delta w_{r,G_k} H_{G_k,G_k} \Delta w_{r,G_k}^{\top}, \tag{10}$$

where

$$H_{G_k,G_k} = 2X_{G_k,:}X_{G_k,:}^{\top} \in \mathbb{R}^{|G_k|\times|G_k|}. \tag{11}$$

For each output row r, H_{G_k,G_k} is the Hessian of $E(\widehat{W})$ with respect to $\widehat{w}_{r,G_k}$, with all remaining weight coordinates held fixed.

Proof. The layer-wise quantization error can be expanded over the output rows and calibration tokens as

$$\begin{aligned} E(\widehat{W}) &= \sum_{r=1}^{d_{\text{out}}} \|\Delta w_r X\|_2^2 \\ &= \sum_{r=1}^{d_{\text{out}}} \sum_{a=1}^{N} \left(\sum_{j=1}^{d_{\text{in}}} \Delta w_{r,j} X_{j,a} \right)^2 . \end{aligned} \tag{12}$$

Since $\Delta w_{r,j} = w_{r,j} - \widehat{w}_{r,j}$, for a fixed output row r,

$$\frac{\partial}{\partial \widehat{w}_{r,i}} \left(\sum_{j=1}^{d_{\text{in}}} \Delta w_{r,j} X_{j,a} \right) = -X_{i,a}. \tag{13}$$

Therefore,

$$\begin{aligned} \frac{\partial E(\widehat{W})}{\partial \widehat{w}_{r,i}} &= -2 \sum_{a=1}^{N} \left(\sum_{j=1}^{d_{\text{in}}} \Delta w_{r,j} X_{j,a} \right) X_{i,a} \\ &= -2\big(\Delta w_r X X^{\top}\big)_i. \end{aligned} \tag{14}$$

Differentiating once more gives

$$\frac{\partial^2 E(\widehat{W})}{\partial \widehat{w}_{r,i}\, \partial \widehat{w}_{s,j}} = \begin{cases} 2 \sum_{a=1}^{N} X_{i,a} X_{j,a}, & r = s, \\ 0, & r \neq s. \end{cases} \tag{15}$$

Thus, the Hessian associated with each output row is

$$H = 2XX^{\top}. \tag{16}$$

Restricting the input indices to G_k gives

$$H_{G_k,G_k} = 2X_{G_k,:}X_{G_k,:}^{\top}. \tag{17}$$

Moreover, since the mixed second derivatives between distinct output rows vanish, the Hessian with respect to the row-wise vectorization of $\widehat{W}_{:,G_k}$ is block diagonal, with d_{out} identical diagonal blocks, each equal to H_{G_k,G_k}.

It remains to express the layer-wise quantization error in terms of this Hessian block. Since $\widehat{W}_{:,j} = W_{:,j}$ for every $j \notin G_k$, we have

$$\Delta w_{r,j} = 0, \qquad j \notin G_k. \tag{18}$$

Hence, for each output row r,

$$\Delta w_r X = \sum_{j \in G_k} \Delta w_{r,j} X_{j,:} = \Delta w_{r,G_k} X_{G_k,:}. \tag{19}$$

Taking the squared Euclidean norm gives

$$\begin{aligned} \|\Delta w_r X\|_2^2 &= \Delta w_{r,G_k} X_{G_k,:} X_{G_k,:}^\top \Delta w_{r,G_k}^\top \\ &= \frac{1}{2} \Delta w_{r,G_k} H_{G_k,G_k} \Delta w_{r,G_k}^\top, \end{aligned} \tag{20}$$

where the second equality follows from $H_{G_k,G_k} = 2X_{G_k,:}X_{G_k,:}^\top$. Summing over $r = 1, \ldots, d_{\text{out}}$ yields Eq. 10. Since $E(\widehat{W})$ is quadratic in $\widehat{W}$, this identity is exact and requires neither a small-error approximation nor a full-rank assumption on X.

This correspondence clarifies the meaning of block size in CostBit. For example, when $|G_k| = 128$, the weight block $W_{:,G_k}$ has shape $d_{\text{out}} \times 128$, while the corresponding Hessian block H_{G_k,G_k} has shape 128×128. Thus, block size refers to the number of input-channel columns, rather than the dimensions of a square weight submatrix.