MIXQUANT: ADAPTIVE QUANTIZATION FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-precision quantization improves post-training quantization by assigning higher precision to sensitive layers, but existing methods typically optimize bit allocation for a fixed memory budget. On edge devices, the memory available to an LLM can vary at runtime and is unknown during offline calibration. Adaptive quantization addresses this setting by calibrating once and deriving an allocation for the available budget at deployment. However, accurately calibrating before the deployment-time allocation is known is challenging because a layer's distortion depends on the bitwidths of its upstream layers. We show that this dependence substantially changes layer scores and the resulting allocations. We propose MIXQUANT, a quantizer-agnostic adaptive quantization framework that marginalizes over information unknown during offline calibration while exploiting the fixed quantizer and allocator. MIXQUANT estimates each layer's expected distortion over quantized upstream configurations, regularizes low-bit assignments that introduce disproportionate distortion, and calibrates quantizer parameters on plans produced by the deployment-time allocator. At deployment, a greedy solver produces an allocation for any available budget in $O(L|\mathcal{B}| \log L)$ time. Across three LLMs, two base quantizers, multiple memory budgets, and diverse downstream tasks, MIXQUANT consistently improves perplexity and average task accuracy. On Llama-3.2-3B at $1.25\,\mathrm{GB}$, it improves average accuracy over the strongest baseline by 7.7 and 8.4 percentage points with AWQ and GPTQ, respectively. Under tight memory constraints, competing methods incur $2.5\times$–$8.75\times$ the degradation from FP16 of MIXQUANT across the evaluated task categories.

1 INTRODUCTION

Large Language Models (LLMs) now power everyday tools from conversational assistants to code completion (Jiang et al., 2026) and document understanding. There is growing demand to run them directly on user devices such as phones, laptops, vehicles, and embedded assistants. These devices, however, operate under tight memory budgets that a full-precision LLM far exceeds (Zheng et al., 2025). Mixed-precision quantization (MPQ) is a widely adopted approach to shrink a model's memory footprint by storing different layers at different bit-widths, assigning higher precision only where it is needed, so a model that would not otherwise fit can be deployed on constrained hardware (Dong et al., 2020; Yao et al., 2020).

Deploying an LLM well on the edge poses two challenges. First, the memory available to a model is not fixed: it shifts at runtime as concurrent workloads and other applications compete for it, and is unknown in advance (Yu et al., 2026; Chen et al., 2026; Xiao et al., 2026). Second, the memory budget alone does not determine performance, how the bits are distributed across layers matters just as much (Harma et al., 2026). In Table 1 we show that even for near identical memory budget bit allocations, quality does not track the unweighted average bit-width. Both Llama-3B (1.25 GB) and Llama-7B (3.0 GB) plans at 4.0 and 4.1 average bits reach lower perplexity than plans at 4.4 and 4.3. MPQ addresses the second challenge. However, they solve allocation for a single known budget (Wang et al., 2019; Koryakovskiy et al., 2023), not ideal for practical deployment. Adaptive Quantization (AQ) calibrates the model once and produces metrics from which a bit allocation for the available budget is derived at deployment (Jin et al., 2020; Yu et al., 2021; Zhao et al., 2026), making it a natural approach for adapting to dynamic memory budgets.

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.

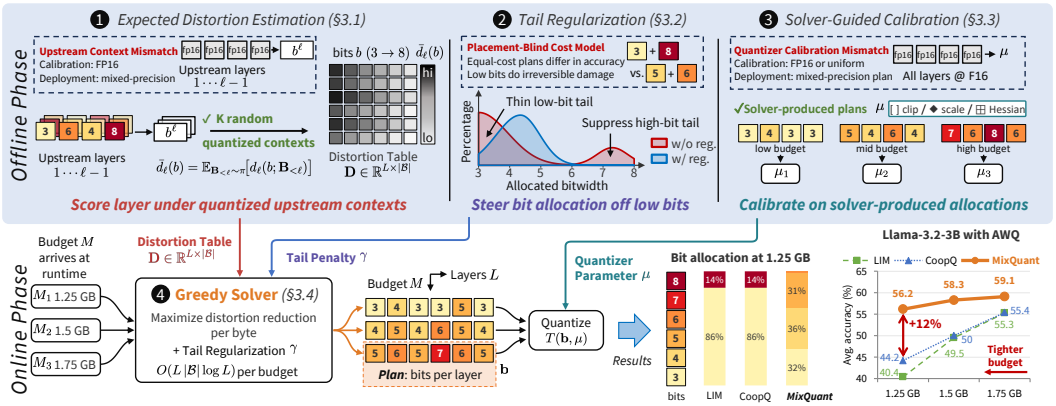


Figure 1: Overview of MIXQUANT. ❶ builds a budget-agnostic distortion table $\mathbf{D} \in \mathbb{R}^{L\times|\mathcal{B}|}$ by scoring each (linear layer, bit-width) pair under K random quantized upstream contexts (Section 3.1). ❷ adds a tail penalty γ that steers spare budget away from the lowest bit-widths (Section 3.2). ❸ estimates the quantization technique parameters μ on activations induced by solver generated bit allocations, rather than on FP16 activations (Section 3.3). ❹ In the online phase, each budget M_i is served by a single greedy solve over $\mathbf{D}$, μ, and γ.

However, applying AQ to mixed-precision LLMs presents a key challenge: calibration occurs before the deployment-time bit allocation is known. Accurate allocation requires per-layer scores and quantizer hyperparameters that reflect the fully quantized model deployed at runtime. Yet the space of possible mixed-precision configurations is combinatorially large, making it impractical to calibrate every configuration in advance. Existing methods tackle this challenge with assumptions and restrictions that can limit their accuracy or applicability at deployment. First, a layer's *distortion score*, which measures quantization-induced change using metrics such as quantization error or KL divergence, depends on the bit-widths of its upstream layers. Yet common methods such as LIM (Dumitru et al., 2024) measure this score with the upstream layers held at FP16.[1] Second, while CoopQ (Zhao et al., 2026) accounts for the quantization of upstream layers, it does not leverage the implicit bit allocation preference of its ILP solver to further tune the quantizer's hyperparameters. Third, methods that support multiple budgets, such as Any-Precision LLM (Park et al., 2024), are tied to a single quantization scheme, specifically the non-uniform codebooks of SqueezeLLM (Kim et al., 2024), and do not transfer to widely used PTQ techniques like AWQ (Lin et al., 2024) and GPTQ (Frantar et al., 2022). No existing method addresses all three limitations. We introduce MIXQUANT to close this gap.

The key idea behind MIXQUANT is to **marginalize over what is unknown during offline calibration while exploiting what is known about the deployment pipeline**. The deployment-time memory budget, and hence the resulting bit allocation, is unknown during calibration, whereas the solver, which maps a memory budget to a bit allocation, and base quantizer are fixed in advance (Figure 1). ❶ marginalizes over upstream bit allocations when estimating each layer's distortion, preventing the scores from being tied to any particular budget-specific configuration. ❷ builds on the observation that low-bit assignments can introduce disproportionately large distortion that propagates through subsequent layers, and penalizes such assignments to encourage available memory to raise layers from the lowest bitwidths before further increasing the precision of layers already at higher bitwidths. Finally, ❸ exploits the fixed deployment structure by using the solver to generate representative plans and calibrating the quantizer's hyperparameters under the activations induced by those plans.

We propose MIXQUANT, a quantizer-agnostic adaptive quantization framework that wraps any base quantizer, such as AWQ or GPTQ. MIXQUANT comprises three offline stages, ❶ expected-distortion estimation, ❷ tail regularization, and ❸ solver-guided calibration (Figure 1). At deployment, a single greedy solve ❹ produces an allocation for the available budget in $O(L|\mathcal{B}|\log L)$ time while achieving accuracy comparable to an ILP solver (Section 4). We evaluate MIXQUANT

[1] LIM considers baselines using both FP16 and BF16; for clarity, we refer to FP16 throughout.

Table 1: Bit allocations for a fixed memory budget per model corresponding to runs in Tables 2, 3

Model (budget)	Metric	Sampled plans (avg. bit-width →)					
		P1	P2	P3	P4	P5	P6
Llama-3B ($\sim 1.25\,\mathrm{GB}$)	Avg. bit-width	3.7	3.9	4.0	4.0	4.3	4.4
	WikiText PPL (↓)	17.1	16.6	**12.4**	12.5	12.9	13.3
	Avg. task acc. (↑)	40.4	44.2	47.8	**48.5**	46.9	46.0
Llama-7B ($\sim 3.0\,\mathrm{GB}$)	Avg. bit-width	3.9	4.0	4.1	4.1	4.3	4.3
	WikiText PPL (↓)	12.2	11.9	**10.0**	10.1	10.4	10.3
	Avg. task acc. (↑)	43.5	**45.0**	44.3	44.6	42.9	42.7

across three LLMs, two base quantizers, multiple memory budgets, and tasks spanning language completion, scientific reasoning, commonsense reasoning, and factual knowledge. MIXQUANT consistently improves perplexity and average task accuracy, with its largest gains at tight memory budgets. On Llama-3.2-3B at $1.25\,\mathrm{GB}$, MIXQUANT improves average task accuracy over the strongest baseline by 7.7 and 8.4 percentage points with AWQ and GPTQ, respectively, while reducing perplexity from 12.4 to 10.7 with AWQ and from 13.3 to 11.2 with GPTQ. At tight memory budgets, MIXQUANT also preserves quality consistently across task domains, with competing methods incurring $2.5\times$–$8.75\times$ the degradation from FP16 of MIXQUANT across all four evaluated task categories.

Contributions. This work makes three contributions. **(1)** We identify a mismatch between offline calibration and deployment-time mixed-precision configurations, showing that upstream bit allocations affect layer distortion scores and the resulting allocations. **(2)** We introduce MIXQUANT, a quantizer-agnostic framework that combines expected-distortion estimation, tail regularization, and solver-guided calibration with an efficient $O(L|\mathcal{B}|\log L)$ online solver. **(3)** We evaluate MIXQUANT across three LLMs, two base quantizers, multiple memory budgets, and diverse downstream tasks, and provide empirical insights into how bit-allocation patterns and tail regularization affect model quality.

2 RELATED WORK

Post-Training Quantization. *Post-training quantization* (PTQ) compresses pretrained LLMs without retraining. AWQ (Lin et al., 2024) protects salient weight channels identified by activation magnitude, and GPTQ (Frantar et al., 2022) uses approximate second-order information to compensate quantization error column by column; MIXQUANT wraps both as base quantizers. Other methods suppress outliers via scaling (Xiao et al., 2023) and rotations (Ashkboos et al., 2024; Liu et al., 2025), or absorb residual error into a low-rank term (Zhang et al., 2024). *Mixed-precision* methods allocate bits non-uniformly: HAWQ-v2 (Dong et al., 2020) ranks layers by Hessian trace and AMQ (Lee et al., 2025) searches per-layer precisions under a budget. SpQR (Dettmers et al., 2023) applies mixed precision within each layer, keeping a small fraction of sensitive outlier weights in 16-bit while quantizing the rest to 3–4 bits. These methods assume the memory budget is known a priori, so every change in the runtime budget requires re-optimization.

Adaptive Quantization. *Adaptive quantization* (AQ) removes this dependence: a single offline calibration yields per-layer scores from which an allocation for *any* budget is recovered by one cheap solve at deployment. Early AQ methods, developed for CNNs and small transformers, achieve this through quantization-aware retraining: AdaBits (Jin et al., 2020) jointly trains a single model to run at any of a fixed set of bit-widths, BitMixer (Bulat & Tzimiropoulos, 2021) trains a meta-network that selects arbitrary mixed-precision configurations at runtime, and RobustQuant (Shkolnik et al., 2020) regularizes weight kurtosis so layer distributions stay resilient across precisions. All recover accuracy through gradient updates, which is infeasible at LLM scale; we therefore focus on retraining-free AQ for LLMs. LIM (Dumitru et al., 2024) scores each transformer block by the cosine similarity between its input and output hidden states, and proposes a greedy algorithm for bit allocation. CoopQ (Zhao et al., 2026) attributes a contribution to each layer via Shapley values

and allocates bits with an ILP. Any-Precision LLM (Park et al., 2024) stores one model that serves multiple bit-widths, built on the non-uniform codebooks of SqueezeLLM (Kim et al., 2024).

3 METHOD

We now formalize MIXQUANT's allocation problem and objective, then describe the three offline stages and the greedy solver summarized in Figure 1.

Formulation. A pretrained LLM contains L linear layers indexed by $\ell \in \{1, \dots, L\}$, where layer ℓ has P_ℓ parameters. We quantize the model using a PTQ technique T with calibration parameters μ, such as smoothing factors and clipping ranges for AWQ or Hessian information for GPTQ. bit-widths are drawn from an ordered set $\mathcal{B}$, and a plan $\mathbf{b} = (b_1, \dots, b_L) \in \mathcal{B}^L$ where b_ℓ is the bit-width at layer ℓ. $\mathbf{b}_{<\ell}$ is the bit allocation of layer 1 to $l-1$. $y_\ell^{(b,\mathbf{b}_{<\ell})}$ is defined as the output of layer ℓ quantized to bit-width b while the upstream layers are quantized to $\mathbf{b}_{<\ell}$. Given a memory budget M, deployment seeks the plan and technique parameters that minimize the loss $\mathcal{L}(T(\mathbf{b}, \mu))$ of the quantized model subject to $\mathrm{mem}(\mathbf{b}) \le M$ where $\mathrm{mem}(\mathbf{b}) = \frac{1}{8}\sum_{\ell=1}^{L} P_\ell b_\ell$.

Proxy objective. Minimizing this loss exactly would require searching over all $|\mathcal{B}|^L$ plans, evaluating each by quantizing the model and running a forward pass, which is infeasible. Following prior mixed-precision work (Yao et al., 2020), we replace $\mathcal{L}$ with a sum of per-linear layer distortion scores $\bar{d}_\ell(b)$ and cast allocation as a multiple-choice knapsack problem (MCKP):

$$\min_x \sum_{\ell=1}^{L}\sum_{b\in\mathcal{B}} \bar{d}_\ell(b)\, x_{\ell,b} \quad \text{s.t.} \quad \sum_{b\in\mathcal{B}} x_{\ell,b} = 1 \;\; \forall \ell, \qquad \tfrac{1}{8}\sum_{\ell=1}^{L}\sum_{b\in\mathcal{B}} P_\ell\, b\, x_{\ell,b} \le M, \tag{1}$$

where $x_{\ell,b} \in \{0,1\}$ assigns bit-width b to module ℓ. Unlike the MPQ setting, where M is fixed and Equation 1 is solved once offline, here M is a runtime constraint, so Equation 1 must be solved at runtime whenever a new memory budget arrives. We estimate μ with a heuristic that exploits the trends of the plans resulting from solving Equation 1 (Section 3.3).

3.1 STAGE 1: EXPECTED DISTORTION ESTIMATION

A layer's distortion score depends on the bit allocation of the upstream layers, so fixing their precision to FP16, as in prior work LIM, biases the score toward a configuration that does not reflect deployment, where upstream layers are also quantized. We call the bit allocation of the upstream layers *context*. Thus, measuring a layer's score in an unrepresentative context can bias the allocation returned by the ILP solver. When the memory budget is unknown until deployment, the scores should avoid bias toward any particular budget-specific context and be able to support allocation across a range of deployment budgets.

MIXQUANT reduces dependence on any single context by averaging each layer's distortion over randomly sampled contexts, yielding an estimate of its expected distortion across deployment configurations. We sample upstream bit-widths independently and uniformly over $\mathcal{B}$, inducing the prior $\pi(\mathbf{b}_{<\ell}) = \prod_{m=1}^{\ell-1} \mathrm{Unif}_{\mathcal{B}}(b_m)$. Under this prior, we define the *decoupled expected distortion* $\bar{d}_\ell(b)$ as the expected distortion of layer ℓ at bit-width b. We estimate this expectation using K Monte Carlo samples:

$$\bar{d}_\ell(b) = \mathbb{E}_{\mathbf{B}_{<\ell}\sim\pi}\big[d_\ell(b; \mathbf{B}_{<\ell})\big], \qquad \hat{d}_\ell(b) = \frac{1}{K}\sum_{k=1}^{K} d_\ell\big(b; \mathbf{b}_{<\ell}^{(k)}\big), \quad \mathbf{b}_{<\ell}^{(k)} \sim \pi. \tag{2}$$

where $d_\ell(b; \mathbf{b}_{<\ell}) = \mathbb{E}_x \|y_\ell^{(b,\mathbf{b}_{<\ell})} - y_\ell^{\mathrm{fp16}}\|^2 / \mathbb{E}_x \|y_\ell^{\mathrm{fp16}}\|^2$, $\mathbf{B}_{<\ell}$ denotes a random upstream configuration distributed according to π and x is a calibration sample.

For each sampled context $\mathbf{b}_{<\ell}^{(k)}$, we calibrate the parameters $\mu^{\mathbf{b}_{<\ell}^{(k)}}$ of technique T under that context. Thus, every entry in $\mathbf{D}$ is measured with quantized upstream layers rather than an FP16 context. These context-specific parameters are used only to estimate $d_\ell\big(b; \mathbf{b}_{<\ell}^{(k)}\big)$ and are not deployed. The deployment parameters μ are estimated separately in Stage 3 (Section 3.3).

In effect, MIXQUANT marginalizes over the upstream context under π, yielding a score $\bar{d}_\ell(b)$ that depends only on (ℓ, b). With finite K, the estimator $\hat{d}_\ell(b)$ approximates this expectation and converges to $\bar{d}_\ell(b)$ as $K \to \infty$.

3.2 STAGE 2: TAIL REGULARIZATION

Low-bit assignments can introduce disproportionately large distortion that cannot be fully mitigated by assigning higher precision to downstream layers. We therefore reserve the lowest bit-widths for cases where they are necessary to satisfy the memory budget, reducing the propagation of quantization error through subsequent layers.

We encode this preference with a penalty that favors upgrading layers at the lowest bit-width. Let b_0 be the lowest bit-width in $\mathcal{B}$, and let b^+ denote the successor of b in the ordered set $\mathcal{B}$. Increasing a layer's bit-width from b to b^+ reduces distortion at the cost of additional memory. We quantify the efficiency of this increase as the distortion reduction per unit of additional memory,

$$e_\ell(b) \;=\; \frac{\bar{d}_\ell(b) - \bar{d}_\ell(b^+)}{P_\ell\,(b^+ - b)}. \tag{3}$$

We then discount this efficiency according to how far b is above the minimum bit-width,

$$\tilde{e}_\ell(b) \;=\; \frac{e_\ell(b)}{1 + \gamma\,(b - b_0)}, \qquad \gamma \geq 0. \tag{4}$$

The term $b - b_0$ measures the distance from the minimum bit-width. Consequently, increases at higher bit-widths receive progressively larger discount than increases from lower bit-widths, encouraging the solver to allocate additional precision to layers at low bit-widths before allocating precision to layers already at higher bit-width. The parameter γ controls the strength of this preference.

3.3 STAGE 3: SOLVER GUIDED CALIBRATION

While the deployment-time memory budget is unknown offline, the quantization technique T and solver are known and can be incorporated into offline calibration. Stage 1 reduces dependence on prior knowledge of the memory budget while incorporating T into its distortion estimates (Section 3.1). The solver, however, remains unaccounted for. Its objective and implementation can induce systematic preferences over bit allocations, making some configurations more relevant to deployment than others. Stage 3 therefore uses the solver's allocation behavior to guide the estimation of the deployment parameters μ.

We therefore partition the feasible memory range $[M_{min}, M_{max}]$ into I intervals and estimate a separate set of quantization parameter μ_i for each interval. Within interval i, we sample J memory budgets and solve for each using the same greedy algorithm used online (Section 3.4), yielding plans $\mathbf{b}^{i,1}, \cdots, \mathbf{b}^{i,J}$. For each plan, we estimate $\mu^{(i,j)}$ from the calibration dataset using quantization technique T under $\mathbf{b}^{(i,j)}$, and average these parameters within the interval: $\mu_i = \frac{1}{J}\sum_{j=1}^{J} \mu^{(i,j)}$.

Because our greedy solver is deterministic, a sampled budget $M^{(i,j)}$ yields the same plan offline as it would online. Estimating $\mu^{(i,j)}$ for each such plan and averaging the resulting parameters across the interval yields hyperparameters representative of the solver's allocations within interval i.

3.4 GREEDY SOLVER.

Given a feasible budget M, the solver starts from the uniform minimum-bit-width plan $\mathbf{b} = (b_0, \ldots, b_0)$ and maintains one candidate transition per layer, from its current bit-width b_ℓ to its successor b_ℓ^+, scored by the penalized efficiency $\tilde{e}_\ell(b_\ell)$ in Equation 4. At each iteration, it applies the highest-scoring transition that fits within the remaining budget. After increasing layer ℓ to b_ℓ^+, the solver adds the next available transition for that layer to the candidate set. The procedure terminates when no candidate transition fits within the remaining budget.

The solver performs at most $L(|\mathcal{B}| - 1)$ upgrades, each selecting the best of at most L candidates. Maintaining these candidates in a priority queue requires $O(\log L)$ time per upgrade, giving a total

allocation cost of $O(L|\mathcal{B}|\log L)$. In Table 5, we compare MIXQUANT's greedy solver against the ILP variant (*Stage 1 + Stage 2 + ILP*) and show that it achieves comparable accuracy with lower asymptotic runtime complexity.

4 EVALUATION

Setup. We evaluate MIXQUANT on Llama-3.2-3B, Llama-2-7B, and Mistral-7B-v0.1 using quantizer AWQ (Lin et al., 2024) and GPTQ (Frantar et al., 2022). Plans assign each linear layer a bit-width in $\mathcal{B} = \{3, \ldots, 8\}$[2]. We use group-wise symmetric weight quantization (group size 128) and per-token 8-bit absmax activation quantization, and report linear-weight memory excluding scale overhead. We calibrate distortion scores and quantizer parameters on 128 random length-2048 windows from the WikiText-2 (Merity et al., 2016) training split. Unless stated otherwise, MIXQUANT uses $K = 10$ context samples per (layer, bit-width) pair, $I = 3$ budget intervals with $J = 5$ plans each, and $\gamma = 10$. We report word-level perplexity on the WikiText-2 validation split and accuracy on PIQA (Bisk et al., 2019), ARC-Challenge (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2019), and MMLU (Hendrycks et al., 2021) (humanities, social sciences, STEM, and other) using the EleutherAI lm-evaluation-harness (Gao et al., 2024). Experiments use four NVIDIA RTX 6000 Ada GPUs (48 GB each), Ubuntu 22.04 with NVIDIA driver 550.54.15 and CUDA 12.1, Python 3.10.12, PyTorch 2.4.1, Transformers 5.3.0, Datasets 4.0.0, and lm-evaluation-harness 0.4.11. Each experiment was run 5 times, we report the mean.

Baselines. We compare against two groups of baselines under AWQ and GPTQ, two widely used PTQ techniques with distinct quantization mechanisms. (1) **Adaptive methods**. LIM (Dumitru et al., 2024) scores each transformer block by input-output cosine similarity but does not directly support linear-layer granularity. CoopQ (Zhao et al., 2026) estimates module contributions via Shapley values under quantized context and solves an ILP; we report its original block-level variant, as our linear-layer adaptation performed substantially worse. We do not include Any-Precision LLM (Park et al., 2024) because it is designed specifically around SqueezeLLM's non-uniform quantization scheme and does not directly support AWQ or GPTQ. (2) **FP16-prior metrics**. To broaden the comparison beyond existing adaptive methods, we construct four baselines that score quantized layers relative to the FP16 model. *Quant. Err.* measures activation error, *KL Div* measures output-distribution divergence, *HAWQ-v2* (Dong et al., 2020) combines Hessian-based importance with quantization error, and *Fisher* uses a Fisher approximation of the Hessian as an importance metric. These cover the main families of prior scoring metrics: error, divergence, importance, and their combination (formulation in Appendix A.3). We pair each with the same ILP solver for Equation 1.

Quality across memory budgets. Across all three models, both quantizers, and the memory budgets, MIXQUANT consistently achieves the highest or near-highest downstream accuracy and lowest perplexity, with its largest gains at tight budgets.

On Llama-3.2-3B (Table 2), MIXQUANT achieves the highest average accuracy under both AWQ and GPTQ at every budget. At 1.25 GB, it exceeds the next-best baseline by 7.7 and 8.4 percentage points under AWQ and GPTQ, respectively. Similarly, on Llama2-7B (Table 3), MIXQUANT achieves the highest average accuracy under both AWQ and GPTQ at every budget. At 3.0 GB, it exceeds the next-best baseline by 3.8 and 2.7 percentage points under AWQ and GPTQ, respectively. Results for Mistral-7B (Table 6) follow the same trend and are reported in Appendix A.1.

Perplexity results are similarly consistent (Table 4). MIXQUANT achieves the lowest perplexity for every model, budget, and quantizer. At 1.25 GB on Llama-3.2-3B, it reduces perplexity from 12.4 to 10.7 under AWQ and from 13.3 to 11.2 under GPTQ relative to the next-best baseline.

To visualize the trends observed across models and quantizers, Figure 2 shows Llama-3.2-3B with AWQ as a representative case. MIXQUANT's reduction in perplexity and increase in accuracy relative to the baselines are largest at the tightest budget (Figures 2a,b), the regime most relevant when memory available to LLMs on edge devices is constrained. Relative to FP16, MIXQUANT also preserves quality consistently across task domains. At 1.25 GB, the baselines incur $2.5\times$–$8.75\times$ the degradation of MIXQUANT across all four categories (Figure 2c).

[2]We exclude 2-bit quantization because AWQ and GPTQ suffer severe accuracy degradation at this precision.

Mem (GB)	Method	AWQ								GPTQ							
		Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg	Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg
1.25	Quant. Err.	34.3	71.3	62.1	33.8	34.7	43.0	43.1	46.0	32.8	70.2	61.6	31.7	30.5	36.0	35.4	42.6
	KL Div	34.8	71.7	62.8	33.5	36.0	43.2	46.6	46.9	33.6	70.0	61.1	34.0	31.5	37.6	37.2	43.6
	HAWQ-v2	34.4	73.3	64.2	35.1	37.0	44.6	46.0	47.8	35.8	72.6	63.9	35.1	34.7	40.7	39.5	46.0
	Fisher	35.6	73.1	63.9	35.2	37.7	46.5	47.6	48.5	33.9	71.3	63.5	32.8	31.4	35.3	37.2	43.6
	CoopQ	34.5	73.1	61.6	32.1	32.4	37.0	38.6	44.2	32.3	69.3	58.9	30.7	30.9	35.8	33.5	41.6
	LIM	35.4	72.3	59.9	25.8	28.4	30.0	31.0	40.4	32.1	67.9	58.5	29.5	29.5	31.2	30.0	39.8
	MIXQUANT	**43.4**	**75.9**	**69.0**	**42.3**	**44.9**	**59.2**	**58.7**	**56.2**	**42.0**	**76.3**	**66.9**	**40.5**	**43.9**	**55.4**	**56.5**	**54.4**
1.5	Quant. Err.	39.2	73.9	65.5	36.2	40.0	48.8	51.8	50.8	39.2	74.6	64.5	37.6	40.1	47.5	49.9	50.5
	KL Div	39.4	74.0	64.6	37.1	39.9	48.6	51.1	50.7	36.8	73.2	64.9	35.0	36.3	46.0	45.7	48.3
	HAWQ-v2	39.2	75.0	63.5	38.4	41.5	52.4	53.4	51.9	39.5	74.3	64.2	36.0	37.1	44.8	47.5	49.1
	Fisher	39.1	74.1	64.6	35.9	40.3	49.5	51.6	50.7	36.4	73.6	63.5	36.9	36.3	43.6	45.0	47.9
	CoopQ	36.6	74.1	64.7	37.4	39.1	48.1	50.1	50.0	35.6	72.1	62.9	35.7	37.5	46.1	48.6	48.4
	LIM	37.4	73.0	64.8	37.3	37.4	47.4	49.3	49.5	33.6	72.1	62.2	35.9	37.4	44.5	43.9	47.1
	MIXQUANT	**45.9**	**76.9**	**68.7**	**44.8**	**48.6**	**61.1**	**62.0**	**58.3**	**44.9**	**76.7**	**68.9**	**45.1**	**48.0**	**62.0**	**61.6**	**58.2**
1.75	Quant. Err.	43.6	76.6	66.6	42.6	47.6	60.0	59.9	56.7	43.5	76.4	66.9	42.2	47.1	59.0	56.9	56.0
	KL Div	43.1	76.6	67.9	40.6	45.7	59.3	59.3	56.1	42.4	75.4	65.6	40.7	42.7	54.4	52.5	53.4
	HAWQ-v2	44.1	**76.8**	67.3	43.0	47.0	59.5	59.3	56.7	44.0	76.4	67.1	40.8	46.1	55.8	56.3	55.2
	Fisher	41.9	76.2	68.0	39.5	45.2	58.3	57.2	55.2	40.4	75.3	67.1	39.3	41.9	53.0	50.9	52.6
	CoopQ	41.3	75.4	68.3	42.4	44.6	59.0	57.2	55.5	38.8	74.3	64.9	43.0	45.2	56.2	55.4	54.0
	LIM	39.6	74.6	67.4	42.9	45.5	57.8	58.9	55.2	39.6	74.2	65.7	44.1	46.5	60.1	59.2	55.6
	MIXQUANT	**46.2**	76.7	**69.1**	**45.9**	**49.3**	**63.6**	**62.6**	**59.1**	**46.3**	**77.0**	**69.1**	**45.5**	**49.1**	**62.7**	**62.5**	**58.9**

Table 2: Downstream mixed-precision quantization results on Llama-3.2-3B. Accuracies (%). **Bold** = best, underline = second best per column within each memory budget.

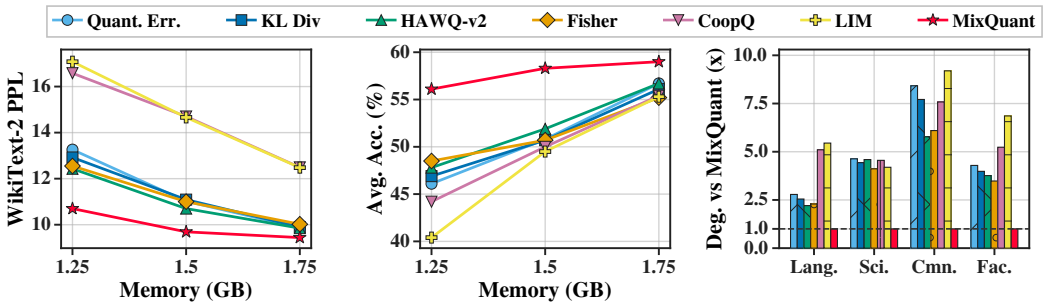
Mem (GB)	Method	AWQ								GPTQ							
		Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg	Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg
3.0	Quant. Err.	42.8	75.6	66.0	24.7	27.7	31.2	32.0	42.9	41.7	76.3	65.1	29.5	29.7	36.9	35.5	45.0
	KL Div	42.3	76.2	66.5	24.9	28.1	29.8	31.4	42.7	41.4	76.0	65.7	29.3	29.4	35.1	34.8	44.5
	HAWQ-v2	43.3	75.9	67.4	26.4	30.1	32.9	33.9	44.3	42.5	76.7	65.8	29.8	31.2	37.8	35.8	45.7
	Fisher	43.0	76.0	67.1	27.6	30.3	33.5	34.8	44.6	42.3	76.7	65.2	30.5	29.8	37.6	36.0	45.4
	CoopQ	41.2	75.9	67.2	29.8	28.8	35.1	37.0	45.0	39.8	76.2	66.9	27.5	31.2	30.9	32.5	43.6
	LIM	40.9	75.9	66.1	29.2	28.8	30.9	32.9	43.5	40.9	75.5	66.5	24.6	30.4	28.5	31.1	42.5
	MIXQUANT	**45.2**	**77.3**	**69.1**	**31.3**	**35.9**	**40.9**	**41.8**	**48.8**	**44.9**	**77.5**	**69.1**	**31.8**	**35.7**	**40.2**	**39.7**	**48.4**
3.5	Quant. Err.	43.2	76.6	69.1	30.8	35.2	40.6	42.0	48.2	42.6	77.3	65.9	32.3	32.9	40.8	39.1	47.3
	KL Div	42.6	76.9	67.7	30.4	35.0	39.9	41.8	47.8	42.9	76.6	66.0	33.2	31.2	39.7	36.6	46.6
	HAWQ-v2	43.9	76.6	67.8	29.8	35.7	40.4	42.4	48.1	43.0	77.5	66.2	33.1	33.7	41.1	39.6	47.7
	Fisher	44.5	76.3	**69.4**	28.6	34.1	38.8	40.0	47.4	41.9	76.4	67.9	33.0	31.9	40.2	38.4	47.1
	CoopQ	42.4	77.3	67.6	28.2	29.7	30.8	35.1	44.4	42.2	77.3	**68.7**	30.0	35.3	36.1	39.7	47.0
	LIM	43.1	76.6	66.5	31.5	31.8	36.6	38.1	46.3	42.0	76.6	68.5	28.0	33.8	34.7	36.0	45.7
	MIXQUANT	**45.8**	**77.7**	69.1	**35.6**	**39.0**	**47.8**	**47.6**	**51.8**	**46.1**	**78.0**	**68.7**	**33.7**	**37.9**	**45.7**	**45.2**	**50.8**
4.0	Quant. Err.	45.5	77.5	68.4	33.7	38.6	45.1	45.5	50.6	44.5	77.5	66.9	**35.4**	38.2	**47.4**	46.5	50.9
	KL Div	**46.6**	77.3	68.4	34.2	37.4	44.8	44.6	50.5	**46.1**	77.5	67.1	34.5	38.3	45.9	45.8	50.7
	HAWQ-v2	45.2	77.2	68.5	33.1	38.7	45.6	45.7	50.6	45.4	77.7	67.3	35.0	38.0	46.5	45.4	50.8
	Fisher	45.1	77.3	68.7	32.4	37.5	44.6	44.2	50.0	42.7	77.4	68.3	33.9	33.9	43.1	41.2	48.6
	CoopQ	45.1	77.0	69.0	28.7	32.0	35.5	38.0	46.5	43.7	77.5	68.2	31.3	36.6	41.7	42.4	48.8
	LIM	43.4	77.5	68.4	34.3	36.6	44.4	45.6	50.0	44.2	77.1	**68.8**	30.2	36.3	39.7	42.0	48.3
	MIXQUANT	46.2	**77.8**	**69.1**	**35.1**	**39.7**	**47.8**	**48.8**	**52.1**	45.8	**78.2**	**68.8**	34.9	**40.2**	47.1	**47.3**	**51.8**

Table 3: Downstream mixed-precision quantization results on Llama-2-7B. Accuracies (%); **Bold** = best, underline = second best per column within each memory budget.

Ablation studies. We ablate MIXQUANT on Llama-3.2-3B at 1.25 GB (Table 5). Removing distortion-guided allocation causes large degradation. *Random Memory* randomly assigns bitwidths under the budget, while *Mem. budget* selects a plan solely to maximize budget utilization; neither uses distortion scores. Their AWQ average accuracies drop from 56.2% to 47.3% and 49.0%, respectively. *Greedy (rev.)*, which selects the least efficient transition at each step, falls further to 40.3%, showing that the ordering induced by the efficiency scores is informative. Removing Stage 2 reduces AWQ average accuracy by 1.6 points, while removing Stage 3 reduces it by 2.3 points; re-

Model	Mem (GB)	AWQ							GPTQ						
		Quant. Err.	KL Div	HAWQ-v2	Fisher	CoopQ	LIM	MIXQUANT	Quant. Err.	KL Div	HAWQ-v2	Fisher	CoopQ	LIM	MIXQUANT
Llama-2-7B	3.0	10.3	10.4	10.0	10.1	11.9	12.2	**9.5**	10.5	11.0	10.2	10.8	12.0	12.1	**9.7**
	3.5	9.4	9.6	9.4	9.5	11.3	11.5	**9.1**	9.6	10.1	9.5	10.0	11.2	11.4	**9.1**
	4.0	9.1	9.2	9.1	9.2	10.7	10.8	**8.9**	9.1	9.4	9.1	9.7	10.7	10.7	**9.0**
Llama-3.2-3B	1.25	13.3	12.9	12.4	12.5	16.6	17.1	**10.7**	15.7	15.4	13.3	15.1	18.2	19.0	**11.2**
	1.5	11.0	11.1	10.7	11.0	14.7	14.6	**9.7**	10.8	12.3	10.8	12.5	15.4	15.8	**10.0**
	1.75	9.9	9.9	9.8	10.0	12.5	12.5	**9.4**	9.9	11.0	9.9	11.2	13.0	13.3	**9.5**
Mistral-7B	3.0	9.4	9.3	9.6	9.7	10.5	10.6	**8.9**	9.5	9.4	9.5	9.9	10.9	10.9	**9.1**
	3.5	8.9	8.9	8.9	9.3	9.9	10.2	**8.4**	9.0	9.0	9.0	9.2	10.4	10.3	**8.4**
	4.0	8.7	8.8	8.6	8.8	9.5	9.8	**8.2**	8.8	8.8	8.7	9.0	10.0	10.0	**8.2**

Table 4: WikiText word perplexity (↓) under mixed-precision quantization across three models and three per-model memory budgets (GB). **Bold** = best, underline = second best per row within each quantizer.



(a) Perplexity (b) Accuracy (c) Normalized FP16 degradation

Figure 2: Results on Llama-3.2-3B with AWQ. **(a)** WikiText-2 perplexity versus memory budget (lower is better). **(b)** Average task accuracy versus memory budget (higher is better). **(c)** Per-category degradation from FP16, normalized by MIXQUANT's (MIXQUANT $= 1$) at 1.25 GB. (Lang. = Language Generation, Sci. = Scientific Reasoning, Cmn. = Commonsense Reasoning, Fac. = Factual Knowledge). MIXQUANT performs best across the evaluated memory budgets.

moving both costs 3.1 points. GPTQ exhibits similar trends. Finally, replacing our greedy solver and tail regularizer with the ILP (*Stage 1 + Stage 2 + ILP*) yields comparable overall quality, and the greedy solver achieves lower perplexity under both quantizers while outperforming the ILP in many accuracy metrics. This is notable given MIXQUANT's greedy solver has $O(L|\mathcal{B}|\log L)$ complexity and substantially lower allocation overhead.

Bit-allocation analysis. Figure 3 compares per-layer allocations across the three models. The FP16-prior baselines (Quant. Err., KL Div., HAWQ-v2, Fisher) concentrate high bit-widths at the end of the model, while LIM and CoopQ favor the extremes, assigning higher precision to the first and last layers, consistent with the allocation patterns observed by LIM (Dumitru et al., 2024). In contrast, MIXQUANT distributes precision across intermediate bitwidths and reduces the use of lower bit-widths. This reflects the tail regularization, which can favor a $3 \to 4$ transition over a less beneficial $7 \to 8$ transition, reducing severe quantization at individual layers. Notably, our FP16-prior baselines also use intermediate bitwidths more frequently than LIM and CoopQ, which may partly explain their stronger empirical performance.

Sensitivity to the tail penalty. We sweep the tail-penalty strength γ from 1 to 100 at tight budgets. The penalty improves or preserves perplexity across all settings, with larger gains at tighter budgets. On Llama-2-7B, for example, the perplexity reduction reaches 3.2% at 3.0 GB, compared with 0.11% at 3.5 GB. Gains largely saturate beyond $\gamma \approx 10$, indicating limited sensitivity to larger values. After a certain threshold there are no low bit to high bit transitions left that the solver can find valid solutions for. The full sensitivity sweep is reported in Appendix A.4.

Variant	AWQ									GPTQ								
	Wiki.	Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg	Wiki.	Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg
Random memory	13.8	36.3	73.3	61.3	34.7	36.3	44.9	44.3	47.3	17.9	29.4	67.1	58.2	28.8	26.6	30.3	33.7	39.2
Mem. budget	13.0	35.8	73.1	62.2	37.7	37.3	47.6	49.3	49.0	16.6	34.2	71.1	59.4	30.4	28.5	32.2	34.6	41.5
Greedy (rev.)	17.6	32.5	69.9	59.8	27.0	29.3	30.6	33.3	40.3	∞	28.6	64.9	57.6	25.8	26.3	25.1	27.1	36.5
w/o Stage 2	11.0	41.5	75.6	63.5	41.8	44.2	58.6	56.9	54.6	11.4	41.0	75.7	63.6	40.2	43.5	55.1	55.5	53.5
w/o Stage 3	11.7	40.3	72.5	66.1	41.9	43.5	56.7	56.1	53.9	11.4	41.2	75.5	63.5	40.5	43.3	55.5	56.0	53.6
w/o Stage 2 & Stage 3	11.9	39.5	75.4	61.5	41.8	41.2	56.6	55.9	53.1	12.1	40.0	74.7	61.6	39.2	41.5	52.1	53.5	51.8
Stage 1 + Stage 2 + ILP	11.0	41.8	75.7	68.4	40.7	**45.4**	58.2	**59.0**	55.6	11.36	41.8	75.6	66.3	**43.4**	**44.7**	**56.5**	55.6	**54.8**
MIXQUANT	**10.7**	**43.4**	**75.9**	**69.0**	**42.3**	44.9	**59.2**	58.7	**56.2**	**11.19**	**42.0**	**76.3**	**66.9**	40.5	43.9	55.4	**56.5**	54.4

Table 5: Component ablation on Llama-3.2-3B at the 1.25 GB budget. Accuracies (%); Wiki. = word perplexity (↓). Avg excludes Wiki. **Bold** = best and underline = second best per column.

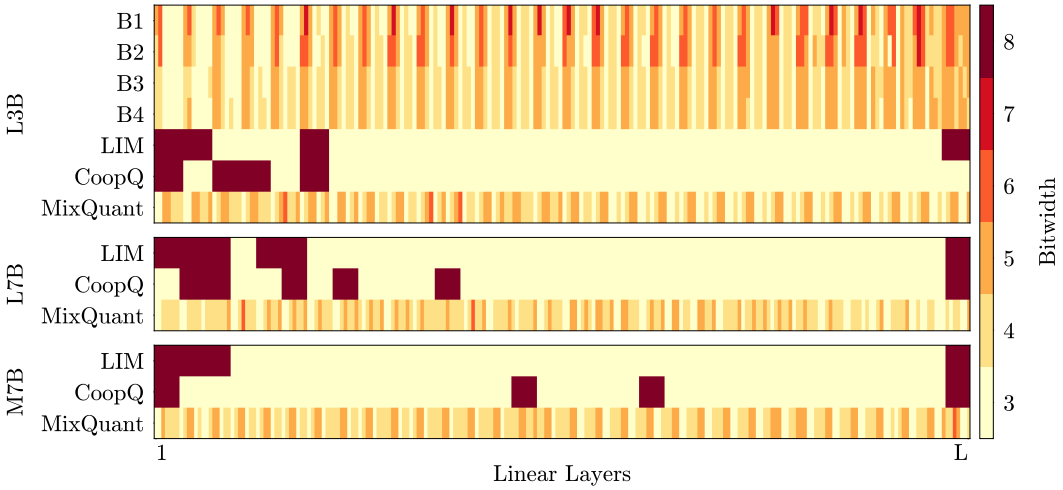


Figure 3: Per-linear-layer bit-width allocations by scoring method at a memory budget for Llama-3.2-3B (1.25 GB), Llama-2-7B (3.0 GB), and Mistral-7B (3.0 GB). The x-axis indexes layers from the first (1) to the last (L); each row is a method and shade encodes the bit-width. LIM and CoopQ concentrate mass at the extremes of $\mathcal{B}$, whereas the remaining methods distribute allocations across intermediate bit-widths. B1 = Quant. Err., B2 = KL Divergence, B3 = HAWQ-v2, B4 = Fisher

5 Conclusion and Discussion

We presented MIXQUANT, an adaptive mixed-precision quantization framework for deploying LLMs under dynamic memory budgets. MIXQUANT separates what is unknown during offline calibration from what is fixed in the deployment pipeline, marginalizing over upstream bit allocations while exploiting the known quantizer and the bit-width allocator. It further regularizes low-bit assignments to reduce severe quantization at individual layers. Across models, quantizers, memory budgets, and downstream tasks, MIXQUANT consistently preserves model quality, with the largest improvements under tight memory constraints. Our results show that accounting for both uncertainty in the deployment budget and the fixed structure of the quantization pipeline enables effective mixed-precision allocation without recalibration for each budget, making adaptive quantization practical for memory-constrained edge deployment.

Limitations and Future Work. MIXQUANT opens several directions for extension. First, solver-guided calibration currently uses solver-produced allocations to estimate the quantizer parameters μ. The same information could further specialize the distortion estimates by weighting upstream contexts according to their prevalence in solver-produced allocations. Second, the tail regularizer uses a simple fixed penalty to favor transitions from lower bitwidths. While effective across our evaluated settings, future work could explore adaptive penalties that account for model- or layer-specific allocation patterns. Third, MIXQUANT currently allocates the memory budget across model weights. Extending the allocation space to other memory components, such as the KV cache, could enable joint adaptation of multiple sources of LLM memory consumption.

AI USE STATEMENT

Large Language Models (LLMs) were used to assist with manuscript editing, including grammar, clarity, and readability, and to a limited extent with code debugging and routine code generation. All LLM-assisted text and code were reviewed, verified, and revised by the authors before inclusion. LLMs were not used to develop the research ideas, methodology, framework design, experimental design, or scientific contributions, nor were they used to analyze data or interpret experimental results. All LLM-assisted content was carefully reviewed, verified and revised by the authors. All experiments, technical decisions, and conclusions were conducted and determined by the authors. The authors take full responsibility for the content of the manuscript, including the correctness of its technical claims and experimental results.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics.[3] It does not involve human subjects, private data, or the collection of new datasets. We use publicly available pretrained models and benchmarks under their respective licenses. Our mixed-precision quantization method can reduce the memory and computational costs of language-model inference, potentially improving accessibility and resource efficiency. However, it does not mitigate biases, privacy risks, unsafe behavior, or potential misuse inherited from the underlying models, and quantization may affect performance unevenly across tasks. Practitioners should therefore conduct application-specific safety, fairness, and reliability evaluations before deployment.

REPRODUCIBILITY STATEMENT

We have taken several steps to support reproducibility. The formulation and individual stages of MIXQUANT are described in Section 3, and Section 4 documents the evaluated models, datasets, quantization methods, calibration data, memory budgets, hyperparameters, evaluation framework, hardware, and software versions. Complete stage-by-stage pseudocode for context-plan construction, distortion estimation, tail-regularized greedy allocation, and the offline and online phases is provided in Appendix A.2. The construction of the HAWQ-v2 and Fisher baselines is detailed in Appendix A.3.1, and their shared ILP allocator is described in Appendix A.3.3. Full experimental results across models, quantizers, memory budgets, and downstream tasks are reported in Section 4 and the corresponding appendix sections, together with component ablations and sensitivity analyses. All pretrained models and evaluation datasets used in this work are publicly available. Upon acceptance, we will release our implementation, experiment configurations, evaluation scripts, and documentation sufficient to reproduce the reported results.

[3] https://iclr.cc/public/CodeOfEthics

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/b5b939436789f76f08b9d0da5e81af7c-Abstract-Conference.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Adrian Bulat and Georgios Tzimiropoulos. Bit-mixer: Mixed-precision networks with runtime bit-width selection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5188–5197, 2021.

Zhixiong Chen, Bingjie Zhu, Jiangzhou Wang, Hyundong Shin, Arumugam Nallanathan, and Dusit (Tao) Niyato. Network edge inference for large language models: Principles, techniques, and opportunities. *ACM Computing Surveys*, 58(12):1–35, May 2026. ISSN 1557-7341. doi: 10.1145/3809166. URL http://dx.doi.org/10.1145/3809166.

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.

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, 2023. URL https://arxiv.org/abs/2306.03078.

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 Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d77c703536718b95308130ff2e5cf9ee-Abstract.html.

Razvan-Gabriel Dumitru, Vikas Yadav, Rishabh Maheshwary, Paul-Ioan Clotan, Sathwik Tejaswi Madhusudhan, and Mihai Surdeanu. Layer-wise quantization: A pragmatic and effective method for quantizing llms beyond integer bit-levels. *CoRR*, abs/2406.17415, 2024. doi: 10.48550/ARXIV.2406.17415. URL https://doi.org/10.48550/arXiv.2406.17415.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022. doi: 10.48550/ARXIV.2210.17323. URL https://doi.org/10.48550/arXiv.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. Eleutherai/lm-evaluation-harness: v0.4.3. Zenodo, July 2024. URL https://zenodo.org/records/12608602.

Simla Burcu Harma, Danila Mishin, Zhengyuan Su, Ayan Chakraborty, Elizaveta Kostenok, Dongho Ha, Babak Falsafi, Martin Jaggi, Yunho Oh, and Amir Yazdanbakhsh. Mxsens: Sensitivity-aware mixed-precision quantization for efficient llm inference, 2026. URL https://arxiv.org/abs/2607.17733.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, 35 (2):1–72, January 2026. ISSN 1557-7392. doi: 10.1145/3747588. URL http://dx.doi.org/10.1145/3747588.

Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2146–2156, 2020.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 23901–23923. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/kim24f.html.

Ivan Koryakovskiy, Alexandra Yakovleva, Valentin Buchnev, Temur Isaev, and Gleb Odinokikh. One-shot model for mixed-precision quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7939–7949, 2023.

Sangjun Lee, Seung-taek Woo, Jungyu Jin, Changhun Lee, and Eunhyeok Park. AMQ: enabling automl for mixed-precision weight-only quantization of large language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 35532–35550. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1799. URL https://doi.org/10.18653/v1/2025.emnlp-main.1799.

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 Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

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. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=ogO6DGE6FZ.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W. Lee. Any-precision LLM: low-cost deployment of multiple, different-sized llms. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39682–39701. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/park24e.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Moran Shkolnik, Brian Chmiel, Ron Banner, Gil Shomron, Yury Nahshan, Alexander M. Bronstein, and Uri C. Weiser. Robust quantization: One model to rule them all. *CoRR*, abs/2002.07686, 2020. URL https://arxiv.org/abs/2002.07686.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 2023. URL https://proceedings.mlr.press/v202/xiao23c.html.

Jie Xiao, Qianyi Huang, Xu Chen, and Chen Tian. Understanding large language models in your pockets: Performance study on cots mobile devices. *IEEE Transactions on Mobile Computing*, 25(8):13077–13094, 2026. doi: 10.1109/TMC.2026.3674827.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. HAWQV3: dyadic neural network quantization. *CoRR*, abs/2011.10680, 2020. URL https://arxiv.org/abs/2011.10680.

Haichao Yu, Haoxiang Li, Humphrey Shi, Thomas S. Huang, and Gang Hua. Any-precision deep neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 10763–10771. AAAI Press, 2021. doi: 10.1609/AAAI.V35I12.17286. URL https://doi.org/10.1609/aaai.v35i12.17286.

Hongseung Yu, Minsung Kim, Jongseok Park, and Kyunghan Lee. mzcache: On-device llm memory management under multitasking. *arXiv preprint arXiv:2609.01338*, 2026.

Cheng Zhang, Jianyi Cheng, George Anthony Constantinides, and Yiren Zhao. LQER: low-rank quantization error reconstruction for llms. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 58763–58779. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/zhang24j.html.

Junchen Zhao, Ali Derakhshan, Jayden Kana Hyman, Junhao Dong, Sangeetha Abdu Jyothi, and Ian G. Harris. Coopq: Cooperative game inspired layerwise mixed precision quantization for llms. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 7566–7578. Association for Computational Linguistics, 2026. URL https://aclanthology.org/2026.findings-acl.373/.

Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. A review on edge large language models: Design, execution, and applications, 2025. URL https://arxiv.org/abs/2410.11845.

A APPENDIX

A.1 QUALITY ACROSS MEMORY BUDGETS FOR MISTRAL-7B

Mem (GB)	Method	AWQ								GPTQ							
		Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg	Arc	PIQA	Wino	STEM	Hum.	Soc.	Othr	Avg
3.0	Quant. Err.	46.2	79.3	70.0	43.9	48.9	63.3	60.6	58.9	46.6	79.0	70.8	43.5	45.3	58.1	56.1	57.1
	KL Div	46.0	79.4	71.0	42.8	47.7	61.7	58.8	58.2	46.2	78.7	**71.7**	43.8	46.7	59.2	58.2	57.8
	HAWQ-v2	45.0	79.5	71.0	43.6	48.4	61.7	59.1	58.3	44.9	78.9	70.1	43.5	46.8	60.6	58.6	57.6
	Fisher	44.5	79.4	70.4	43.4	48.4	61.4	58.4	58.0	45.8	78.6	70.0	41.0	43.3	54.0	53.2	55.1
	CoopQ	42.9	78.0	68.8	40.6	44.4	56.8	55.0	55.2	43.9	78.0	70.1	37.3	39.6	49.9	48.5	52.5
	LIM	43.9	77.9	68.0	40.5	45.2	57.2	55.4	55.4	44.5	77.7	70.4	38.9	41.7	52.7	50.3	53.7
	MIXQUANT	**51.0**	**79.7**	**73.6**	**47.1**	**50.4**	**66.1**	**64.2**	**61.7**	**50.1**	**80.3**	71.6	**47.0**	**50.8**	**66.7**	**64.2**	**61.5**
3.5	Quant. Err.	49.1	79.8	72.5	45.2	49.8	63.5	61.6	60.2	50.6	80.0	72.0	46.1	48.6	63.6	61.4	60.3
	KL Div	49.1	79.8	72.0	45.8	50.2	63.2	61.3	60.2	49.7	79.4	72.5	46.2	49.1	63.9	62.1	60.4
	HAWQ-v2	49.7	**80.3**	72.5	45.9	50.1	64.1	63.3	60.8	49.7	80.0	73.4	46.7	49.2	63.6	62.1	60.7
	Fisher	48.3	80.2	72.9	45.6	50.5	63.5	62.0	60.4	48.9	79.9	70.6	46.4	48.4	62.7	59.9	59.5
	CoopQ	44.6	78.7	70.5	43.4	46.4	59.8	57.8	57.3	45.6	78.8	69.6	39.5	41.3	51.7	50.5	53.9
	LIM	46.8	79.2	71.3	44.5	48.5	61.1	60.0	58.8	46.1	78.7	71.0	42.2	44.9	58.9	57.5	57.0
	MIXQUANT	**52.7**	79.9	**73.6**	**49.2**	**52.7**	**69.1**	**66.1**	**63.3**	**52.1**	**80.9**	**73.5**	**49.3**	**53.0**	**69.1**	**66.5**	**63.5**
4.0	Quant. Err.	49.5	80.5	**73.2**	46.9	51.5	65.1	64.2	61.6	50.5	80.3	**74.0**	47.7	49.6	65.3	63.8	61.6
	KL Div	50.3	**80.7**	73.0	47.1	51.2	64.7	63.5	61.5	50.2	80.0	72.7	47.4	49.2	65.0	63.3	61.1
	HAWQ-v2	50.5	80.3	72.3	46.7	51.1	65.8	63.5	61.5	49.6	**80.5**	72.8	48.2	49.5	64.8	62.8	61.2
	Fisher	49.1	80.5	72.7	46.1	51.4	65.0	63.8	61.2	49.4	80.1	73.2	46.5	49.2	65.2	62.9	60.9
	CoopQ	45.3	79.1	71.7	43.6	47.5	61.8	59.4	58.3	46.1	78.5	69.1	40.9	43.1	53.8	52.2	54.8
	LIM	48.1	79.7	72.8	45.2	50.9	64.7	62.9	60.6	49.0	79.3	72.9	47.4	50.6	65.5	62.0	61.0
	MIXQUANT	**53.4**	80.0	73.1	**49.3**	**53.2**	**69.8**	**67.0**	**63.7**	**53.8**	80.4	**74.0**	**50.0**	**52.9**	**69.7**	**67.2**	**64.0**

Table 6: Downstream mixed-precision quantization results on Mistral-7B. Accuracies (%). **Bold** = best, underline = second best per column within each memory budget.

A.2 COMPLETE ALGORITHM

This section presents the complete offline and online procedures of MIXQUANT. The offline phase estimates budget-independent distortion scores, defines the tail-regularized upgrade rule, and calibrates quantizer parameters over representative plans from each memory interval. At runtime, the arriving memory budget is served by a priority-queue greedy allocator (Section 3.4; Algorithm 3), after which the precalibrated parameters associated with the corresponding memory interval are used to instantiate the base quantizer.

Let $\mathcal{L} = \{\ell\}_{\ell=1}^{L}$ index all linear modules in order of the forward pass, and let $\mathcal{B} = \{3, 4, 5, 6, 7, 8\}$ be the ordered set of candidate weight precisions. We write $b_{<\ell} = (b_1, \ldots, b_{\ell-1})$ for the bitwidths of modules preceding ℓ, and P_ℓ for the number of weights in module ℓ. Let $b_0 = \min \mathcal{B}$. All memory costs below refer to packed linear weights and exclude quantization metadata ($\mathrm{mem}(\mathbf{b}) = \frac{1}{8}\sum_\ell P_\ell b_\ell$).

A.2.1 STAGE 1: EXPECTED-DISTORTION ESTIMATION

A layer's distortion depends on the bit allocation of its upstream modules. Following Section 3.1, MIXQUANT estimates the expected distortion of module ℓ at bitwidth b by Monte Carlo averaging under the product prior

$$\pi(b_{<\ell}) = \prod_{m=1}^{\ell-1} \mathrm{Unif}_{\mathcal{B}}(b_m), \tag{5}$$

i.e., each preceding linear is assigned an independent bitwidth drawn uniformly from $\mathcal{B}$. We use $K = 10$ i.i.d. contexts per candidate pair (ℓ, b) (Section 4). For each sampled context $b_{<\ell}^{(k)}$, we calibrate the base-quantizer parameters $\mu_{b_{<\ell}}^{(k)}$ of technique $\mathcal{T}$ under that context; these context-specific parameters

are used only to score $d_\ell\left(b; b_{<\ell}^{(k)}\right)$ and are not deployed. Deployment parameters μ are estimated in Stage 3.

The context-conditional distortion matches the main-text definition

$$d_\ell(b; b_{<\ell}) = \frac{\mathbb{E}_{x\sim\mathcal{C}}\left\|y_\ell^{(b,b_{<\ell})}(x) - y_\ell^{\mathrm{FP16}}(x)\right\|_2^2}{\mathbb{E}_{x\sim\mathcal{C}}\left\|y_\ell^{\mathrm{FP16}}(x)\right\|_2^2}, \tag{6}$$

where $y_\ell^{(b,b_{<\ell})}$ is the output of module ℓ at bitwidth b with upstream modules quantized according to $b_{<\ell}$. The Stage 1 score is the equal-weight Monte Carlo average

$$D_\ell(b) = \widehat{d}_\ell(b) = \frac{1}{K}\sum_{k=1}^{K} d_\ell\left(b; b_{<\ell}^{(k)}\right), \qquad b_{<\ell}^{(k)} \sim \pi, \qquad K = 10. \tag{7}$$

Algorithm 1 gives the complete procedure.

Algorithm 1 Expected-Distortion Estimation

Require: FP16 model f_W; ordered linear modules $\mathcal{L}$; calibration data $\mathcal{C}$; base quantizer $\mathcal{Q}$ for technique $\mathcal{T}$; bitwidths $\mathcal{B}$; contexts $K = 10$
Ensure: Distortion table $D = \{D_\ell(b) : \ell \in \mathcal{L},\, b \in \mathcal{B}\}$

for all $\ell \in \mathcal{L}$ **do**
  for all $b \in \mathcal{B}$ **do**
    for all $k = 1, \ldots, K$ **do** ▷ Independent upstream sample for this (ℓ, b)
      for all $m = 1, \ldots, \ell - 1$ **do**
        $b_m^{(k)} \sim \mathrm{Unif}(\mathcal{B})$
      end for
      $b_{<\ell}^{(k)} \leftarrow (b_1^{(k)}, \ldots, b_{\ell-1}^{(k)})$
      $\mu^{(k)} \leftarrow$ CALIBRATEQUANTIZER$(f_W, \mathcal{C}, \mathcal{Q}, b_{<\ell}^{(k)})$ ▷ Context-specific; not deployed
      RESTOREFP16(f_W)
      APPLYQUANTIZER$(f_W, \mathcal{Q}, b_{<\ell}^{(k)} \cup \{(\ell \mapsto b)\}, \mu^{(k)})$
      $d_\ell^{(k)}(b) \leftarrow \mathrm{NMSE}(y_\ell^{(b,b_{<\ell}^{(k)})}, y_\ell^{\mathrm{FP16}}, \mathcal{C})$ ▷ Equation 6
    end for
    $D_\ell(b) \leftarrow \frac{1}{K}\sum_{k=1}^{K} d_\ell^{(k)}(b)$
    RESTOREFP16(f_W)
  end for
end for
return D

A.2.2 Stage 2: Tail Regularization

Low-bit assignments can introduce disproportionately large distortion. Stage 2 encodes a preference for upgrading modules that remain at the lowest bitwidths before spending spare budget on modules already at higher precision (Section 3.2). Let b^+ denote the successor of b in the ordered set $\mathcal{B}$. The unregularized efficiency of upgrading module ℓ from b to b^+ is the distortion reduction per unit additional memory:

$$E_\ell(b) = e_\ell(b) = \frac{D_\ell(b) - D_\ell(b^+)}{P_\ell\,(b^+ - b)}. \tag{8}$$

We discount this value by the distance from the minimum bitwidth $b_0 = \min \mathcal{B}$:

$$\widetilde{E}_\ell(b) = \widetilde{e}_\ell(b) = \frac{E_\ell(b)}{1 + \gamma\,(b - b_0)}, \qquad \gamma \geq 0. \tag{9}$$

We use $\gamma = 10$ for the online allocator. Stage 2 does not modify the distortion table or produce a separate calibration artifact; it defines the transition priority used by the greedy allocator.

Algorithm 2 Tail-Regularized Upgrade Efficiency

Require: Distortion table D; module ℓ; current bitwidth b; successor b^+; parameter count P_ℓ; minimum bitwidth b_0; penalty strength γ

Ensure: Tail-regularized efficiency $\widetilde{E}_\ell(b)$

1: $\Delta D \leftarrow D_\ell(b) - D_\ell(b^+)$
2: $E_\ell(b) \leftarrow \Delta D \,/\, \big(P_\ell\,(b^+ - b)\big)$
3: $\widetilde{E}_\ell(b) \leftarrow E_\ell(b)/(1 + \gamma\,(b - b_0))$
4: **return** $\widetilde{E}_\ell(b)$

A.2.3 Greedy Bit Allocation

Given a feasible budget M, the allocator starts from the uniform minimum-bitwidth plan $\mathbf{b} = (b_0, \ldots, b_0)$ and maintains one candidate transition per module—from its current bitwidth b_ℓ to its successor b_ℓ^+—scored by the penalized efficiency $\widetilde{E}_\ell(b_\ell)$ in Equation 9 (Section 3.4). At each iteration it applies the highest-scoring transition that fits within the remaining budget. After upgrading module ℓ, it inserts that module's next successor transition (if any). The procedure terminates when no remaining candidate fits.

There are at most $L(|\mathcal{B}| - 1)$ adjacent transitions. Maintaining candidates in a priority queue costs $O(\log L)$ per upgrade, giving

$$O(L|\mathcal{B}| \log L) \tag{10}$$

total allocation cost, as stated in Section 3.4.

A.2.4 Stage 3: Solver-Guided Calibration

Stage 3 partitions the feasible memory range $[M_{\min}, M_{\max}]$ into $I = 3$ model-specific intervals and estimates a separate parameter set $\overline{\mu}_i$ for each interval (Section 3.3). Within interval i, we sample $J = 5$ memory budgets uniformly at random and, for each budget, generate a plan with Algorithm 3 using the **same** greedy solver and tail strength as online ($\gamma = 10$). Because the greedy solver is deterministic, each sampled budget yields the same plan offline as it would online.

Stage 3 is executed separately for AWQ and GPTQ. For AWQ, the calibrated parameters contain smoothing scales and clipping parameters; for GPTQ, the corresponding Hessian tensors. Parameters are averaged elementwise across the J plans:

$$\overline{\mu}_i = \frac{1}{J} \sum_{j=1}^{J} \mu_i^{(j)}, \qquad J = 5. \tag{11}$$

Algorithm 3 Tail-Regularized Greedy Bit Allocation

Require: Distortion table D; module sizes $\{P_\ell\}$; ordered bitwidths $\mathcal{B}$; memory budget M; tail strength γ
Ensure: Feasible bitwidth plan $\mathbf{b} = (b_1, \ldots, b_L)$

$b_0 \leftarrow \min \mathcal{B}$
$b_\ell \leftarrow b_0$ for every $\ell \in \mathcal{L}$
$M_{\text{used}} \leftarrow \frac{1}{8} \sum_{\ell \in \mathcal{L}} P_\ell b_0$
if $M_{\text{used}} > M$ **then**
    return INFEASIBLE
end if
Initialize an empty maximum-priority queue $\mathcal{H}$
for all $\ell \in \mathcal{L}$ **do**
    if $b_\ell < \max \mathcal{B}$ **then**
        $b_\ell^+ \leftarrow \textsc{Successor}(b_\ell, \mathcal{B})$
        $e \leftarrow \textsc{TailRegularizedEfficiency}(D, \ell, b_\ell, b_\ell^+, P_\ell, b_0, \gamma)$
        $\textsc{Push}(\mathcal{H}, (e, \ell, b_\ell, b_\ell^+))$
    end if
end for
while $\mathcal{H}$ is not empty **do**
    $(e, \ell, b, b^+) \leftarrow \textsc{PopMaximum}(\mathcal{H})$
    $\Delta M \leftarrow P_\ell \, (b^+ - b)/8$ ▷ Memory accounting uses bytes
    if $M_{\text{used}} + \Delta M > M$ **then**
        continue ▷ Try the next highest-scoring affordable transition
    end if
    $b_\ell \leftarrow b^+$
    $M_{\text{used}} \leftarrow M_{\text{used}} + \Delta M$
    if $b_\ell < \max \mathcal{B}$ **then**
        $b_\ell^+ \leftarrow \textsc{Successor}(b_\ell, \mathcal{B})$
        $e \leftarrow \textsc{TailRegularizedEfficiency}(D, \ell, b_\ell, b_\ell^+, P_\ell, b_0, \gamma)$
        $\textsc{Push}(\mathcal{H}, (e, \ell, b_\ell, b_\ell^+))$
    end if
end while
return $\mathbf{b}$

Algorithm 4 Solver-Guided Quantizer Calibration

Require: FP16 model f_W; calibration data $\mathcal{C}$; technique-specific distortion table D; module sizes $\{P_\ell\}$; bitwidths $\mathcal{B}$; memory intervals $\mathcal{R} = \{[M_i^-, M_i^+]\}_{i=1}^I$; plans per interval $J = 5$; online tail strength γ; base quantizer $\mathcal{Q} \in \{\text{AWQ}, \text{GPTQ}\}$
Ensure: Per-interval quantizer parameters $\{\overline{\mu}_i\}_{i=1}^I$

for $i = 1, \ldots, I$ **do**
    for $j = 1, \ldots, J$ **do**
        $M_{i,j} \sim \text{Unif}(M_i^-, M_i^+)$
        $\mathbf{b}^{(i,j)} \leftarrow \textsc{GreedyAllocate}(D, \{P_\ell\}, \mathcal{B}, M_{i,j}, \gamma)$ ▷ Same greedy as online (Section 3.4)
        $\mu_i^{(j)} \leftarrow \textsc{CalibrateQuantizer}(f_W, \mathcal{C}, \mathcal{Q}, \mathbf{b}^{(i,j)})$
    end for
    $\overline{\mu}_i \leftarrow \frac{1}{J} \sum_{j=1}^{J} \mu_i^{(j)}$
    $\textsc{SaveParameters}(\overline{\mu}_i, [M_i^-, M_i^+], \mathcal{Q})$
end for
return $\{\overline{\mu}_i\}_{i=1}^I$

A.2.5 Complete Offline and Online Procedures

Algorithm 5 summarizes the complete offline phase. It is run separately for AWQ and GPTQ. The resulting package contains the distortion table, memory-interval definitions, averaged quantizer parameters, module sizes, candidate bitwidths, and online tail-penalty strength.

Algorithm 5 MixQuant Offline Phase

Require: FP16 model f_W; calibration data $\mathcal{C}$; base quantizer $\mathcal{Q}$; bitwidths $\mathcal{B}$; module sizes $\{P_\ell\}$; memory intervals $\mathcal{R}$; contexts $K = 10$; plans per interval $J = 5$; online tail strength $\gamma = 10$
Ensure: Offline package $\mathcal{P}$
1: $D \leftarrow \textsc{EstimateExpectedDistortion}(f_W, \mathcal{C}, \mathcal{Q}, \mathcal{B}, K)$
2: $\{\overline{\mu}_i\}_{i=1}^{I} \leftarrow \textsc{SolverGuidedCalibration}(f_W, \mathcal{C}, D, \{P_\ell\}, \mathcal{B}, \mathcal{R}, J, \gamma, \mathcal{Q})$
3: $\mathcal{P} \leftarrow \big\{D, \{\overline{\mu}_i\}_{i=1}^{I}, \mathcal{R}, \{P_\ell\}, \mathcal{B}, b_0, \gamma, \mathcal{Q}\big\}$
4: $\textsc{SerializeOfflinePackage}(\mathcal{P})$
5: **return** $\mathcal{P}$

When a memory budget arrives, the online phase first generates a bitwidth plan using the tail-regularized greedy allocator with $\gamma = 10$. It then selects the parameter cache associated with the budget's memory interval. No interpolation is performed between intervals, and no calibration, optimization, or gradient computation is required online.

Algorithm 6 MixQuant Online Phase

Require: FP16 model f_W; arriving memory budget M; offline package $\mathcal{P} = \{D, \overline{\mu}, \mathcal{R}, P, \mathcal{B}, b_0, \gamma, \mathcal{Q}\}$
Ensure: Quantized model $\widehat{f}$ and bit-width plan $\mathbf{b}$
1: $\mathbf{b} \leftarrow \textsc{GreedyAllocate}(D, P, \mathcal{B}, M, \gamma)$
2: $i \leftarrow \textsc{FindMemoryInterval}(M, \mathcal{R})$
3: $\mu \leftarrow \overline{\mu}_i$ ▷ No interpolation between intervals
4: $\widehat{f} \leftarrow \textsc{ApplyQuantizer}(f_W, \mathcal{Q}, \mathbf{b}, \mu, \text{activation bits} = 8)$
5: **return** $\widehat{f}, \mathbf{b}$

A.3 Baselines Formulation

We compare against four sensitivity baselines, all scored under an FP16 prior. Unlike MixQuant, no baseline estimates an expected distortion over upstream contexts: each candidate (ℓ, b) is scored once, with module ℓ quantized at bitwidth b by the deployed base quantizer $\mathcal{Q} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$ and every other module held at FP16. Two are curvature-based (HAWQ-V2 and Fisher norm) and score curvature-weighted weight perturbations; two are activation-based (quantization error and KL divergence) and score the error in the module's output activations. Each linear module inside a transformer block is one allocation unit; embeddings, normalization layers, biases, and the language-model head are excluded. Candidate weight precisions are $\mathcal{B} = \{3, 4, 5, 6, 7, 8\}$. All four score tables are allocated with the same ILP (Section A.3.3), without tail regularization.

A.3.1 Curvature-Based Baseline Formulations

Both baselines use a shared curvature profile computed once from the clean FP16 model using the calibration set described in the experimental setup. Let $W_\ell \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$ denote the weight matrix of linear module ℓ. We collect its input activations during an FP16 calibration pass and flatten the batch and token dimensions to obtain $X_\ell \in \mathbb{R}^{n_\ell \times d_{\mathrm{in}}}$. The input-curvature diagonal is

$$h_{\ell,j} = \frac{2}{n_\ell} \sum_{t=1}^{n_\ell} X_{\ell,tj}^2, \qquad j = 1, \ldots, d_{\mathrm{in}}. \tag{12}$$

The sums are accumulated in FP32. This is a diagonal input-side Gauss–Newton/Fisher proxy; no full Hessian is materialized.

For each $b \in \mathcal{B}$, we quantize module ℓ alone at bitwidth b with the deployed base quantizer $\mathcal{Q} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$, calibrated with all other modules at FP16, and define the bit-width-specific perturbation as the difference between the effective dequantized weight and the FP16 weight:

$$\Delta W_{\ell,b} = \widehat{W}^{\mathcal{Q}}_{\ell,b} - W_\ell. \tag{13}$$

For AWQ, the per-input-channel scales $\mathbf{s}_\ell$ are folded back in, $\widehat{W}^{\mathrm{AWQ}}_{\ell,b} = Q_b\big(W_\ell \,\mathrm{diag}(\mathbf{s}_\ell)\big)\,\mathrm{diag}(\mathbf{s}_\ell)^{-1}$; for GPTQ, $\widehat{W}^{\mathrm{GPTQ}}_{\ell,b}$ is the error-compensated quantized weight. Every candidate is quantized directly from the clean FP16 weights, so quantization error from one candidate is not propagated into another. Because $\Delta W_{\ell,b}$ depends on the base quantizer, AWQ and GPTQ receive separate curvature score tables; the curvature profile $\mathbf{h}_\ell$ is shared.

HAWQ-V2. For the HAWQ-V2 baseline, we collapse the input-curvature diagonal into a single scalar trace proxy for each module:

$$\tau_\ell = \sum_{j=1}^{d_{\mathrm{in}}} h_{\ell,j}. \tag{14}$$

We then multiply this module-level sensitivity by the squared Frobenius norm of the candidate's static weight perturbation:

$$d^{\mathrm{HAWQ}}_\ell(b) = \tau_\ell \,\|\Delta W_{\ell,b}\|_F^2 \,. \tag{15}$$

A smaller value indicates a less damaging assignment. The curvature term captures differences in sensitivity across modules, while $\|\Delta W_{\ell,b}\|_F^2$ captures the error associated with the candidate bit-width.

Our implementation follows the trace-weighted perturbation objective of HAWQ-V2, but estimates the trace using the diagonal input-curvature proxy in Equation 12, rather than materializing the full Hessian. Unlike the original formulation, which scores uniform quantization noise, $\Delta W_{\ell,b}$ here is the error of the deployed quantizer, so the score reflects the error AWQ or GPTQ actually leaves. The raw, unnormalized score in Equation 15 is passed to the allocator.

Algorithm 7 summarizes the construction of the HAWQ-V2 score table from the shared FP16 curvature profile.

Algorithm 7 HAWQ-V2 Score Generation

Require: FP16 model f_W; FP16 weights $\{W_\ell\}_{\ell\in\mathcal{L}}$; curvature vectors $\{\mathbf{h}_\ell\}_{\ell\in\mathcal{L}}$; bit-widths $\mathcal{B}$; calibration data $\mathcal{C}$; base quantizer $\mathcal{Q} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$
Ensure: HAWQ-V2 score table D^{HAWQ}

for all $\ell \in \mathcal{L}$ **do**
  $\tau_\ell \leftarrow \sum_j h_{\ell,j}$
  for all $b \in \mathcal{B}$ **do**
    $\widehat{W}^{\mathcal{Q}}_{\ell,b} \leftarrow \textsc{QuantizeModule}(f_W, \mathcal{C}, \mathcal{Q}, \ell, b)$ ▷ Module ℓ only; others FP16
    $\Delta W_{\ell,b} \leftarrow \widehat{W}^{\mathcal{Q}}_{\ell,b} - W_\ell$
    $D^{\mathrm{HAWQ}}_{\ell,b} \leftarrow \tau_\ell \|\Delta W_{\ell,b}\|_F^2$
  end for
end for
return D^{HAWQ}

Fisher Norm. The Fisher-norm baseline retains the input-channel resolution of the curvature diagonal. Since input channel j is shared across all output rows, $h_{\ell,j}$ is broadcast over the output dimension. The score for candidate bit-width b is

$$d_\ell^{\mathrm{Fisher}}(b) = \sum_{o=1}^{d_{\mathrm{out}}} \sum_{j=1}^{d_{\mathrm{in}}} h_{\ell,j} \big(\Delta W_{\ell,b;oj}\big)^2 = \Big\| \Delta W_{\ell,b} \,\mathrm{diag}\left(\sqrt{\mathbf{h}_\ell}\right) \Big\|_F^2, \tag{16}$$

where $\mathbf{h}_\ell = (h_{\ell,1}, \ldots, h_{\ell,d_{\mathrm{in}}})$. As with HAWQ-V2, a smaller value indicates a less damaging assignment.

Unlike the HAWQ-V2 score, which uses one scalar curvature value for the entire module, the Fisher-norm score gives greater weight to perturbations in input channels with larger calibration curvature. It is therefore a bit-width-specific, curvature-weighted perturbation score, rather than a pure module-importance score. The raw score in Equation 16 is used for allocation.

Algorithm 8 shows how the shared input-curvature diagonal is retained at input-channel resolution when constructing the Fisher-norm scores.

Algorithm 8 Fisher-Norm Score Generation

Require: FP16 model f_W; FP16 weights $\{W_\ell\}_{\ell\in\mathcal{L}}$; curvature vectors $\{\mathbf{h}_\ell\}_{\ell\in\mathcal{L}}$; bit-widths $\mathcal{B}$; calibration data $\mathcal{C}$; base quantizer $\mathcal{Q} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$
Ensure: Fisher-norm score table D^{Fisher}

```
for all ℓ ∈ L do
    for all b ∈ B do
        Ŵ^Q_{ℓ,b} ← QuantizeModule(f_W, C, Q, ℓ, b)        ▷ Module ℓ only; others FP16
        ΔW_{ℓ,b} ← Ŵ^Q_{ℓ,b} − W_ℓ
        D^Fisher_{ℓ,b} ← Σ_{o=1}^{d_out} Σ_{j=1}^{d_in} h_{ℓ,j} (ΔW_{ℓ,b;oj})^2
    end for
end for
return D^Fisher
```

A.3.2 Activation-Based Baseline Formulations

The activation-based baselines measure how quantizing module ℓ distorts its output activations. For each candidate (ℓ, b), only module ℓ is quantized at bitwidth b with base quantizer $\mathcal{Q} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$; all other modules remain FP16. Because the input to module ℓ is identical in the FP16 and quantized runs, the score reflects only module ℓ's own quantization. AWQ and GPTQ therefore receive separate score tables.

For calibration sample x, let $a_\ell^{\mathrm{FP16}}(x) \in \mathbb{R}^{N_\ell}$ denote the flattened FP16 output of module ℓ (tokens $\times$ channels), and $a_\ell^{(b)}(x)$ the corresponding output with module ℓ quantized at bitwidth b. We retain only the coordinates whose FP16 magnitude lies at or above the p-th percentile ($p = 0.95$, i.e., the top 5%), writing $S_\ell(x) \subset \{1, \ldots, N_\ell\}$ for this index set and $a_{\ell,S}$ for the restriction of a to $S_\ell(x)$. The same index set is applied to both activations.

Quantization Error. The score is the normalized, unsquared ℓ_2 distortion on the retained coordinates, averaged over calibration samples:

$$d_\ell^{\mathrm{QE}}(b) = \mathbb{E}_{x\sim\mathcal{C}} \left[\frac{\big\| a_{\ell,S}^{(b)}(x) - a_{\ell,S}^{\mathrm{FP16}}(x) \big\|_2}{\big\| a_{\ell,S}^{\mathrm{FP16}}(x) \big\|_2 + \varepsilon} \right], \qquad \varepsilon = 10^{-12}. \tag{17}$$

Unlike the NMSE of Equation 6, which is a ratio of expected squared norms under sampled upstream contexts, this score averages a per-sample unsquared ratio under the FP16 prior.

KL Divergence. The KL baseline compares magnitude distributions on the retained coordinates. For restricted activations $u = a_{\ell,S}^{\mathrm{FP16}}(x)$ and $v = a_{\ell,S}^{(b)}(x)$, we form discrete distributions by L1-normalizing absolute values,

$$p_i = \frac{|u_i|}{\sum_j |u_j| + \varepsilon}, \qquad q_i = \frac{|v_i|}{\sum_j |v_j| + \varepsilon}, \tag{18}$$

and average the symmetric KL divergence over calibration samples:

$$d_\ell^{\mathrm{KL}}(b) = \mathbb{E}_{x\sim\mathcal{C}}\left[\tfrac{1}{2}\big(\mathrm{KL}(p\,\|\,q) + \mathrm{KL}(q\,\|\,p)\big)\right]. \tag{19}$$

This is a distributional mismatch between activation magnitudes, not a divergence between output-token distributions. For both metrics, smaller values indicate less damaging assignments.

Algorithm 9 summarizes score-table construction for both metrics.

Algorithm 9 Activation-Based Score Generation

Require: FP16 model f_W; calibration data $\mathcal{C}$; base quantizer $\mathcal{Q} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$; bitwidths $\mathcal{B}$; percentile p; metric $\phi \in \{\mathrm{QE}, \mathrm{KL}\}$
Ensure: Score table D^ϕ

```
Run f_W on C; cache a_ℓ^FP16(x) and S_ℓ(x) for all ℓ ∈ L, x ∈ C
for all ℓ ∈ L do
    for all b ∈ B do
        μ ← CalibrateQuantizer(f_W, C, Q, {ℓ ↦ b})
        ApplyQuantizer(f_W, Q, {ℓ ↦ b}, μ)            ▷ All other modules FP16
        D^φ_{ℓ,b} ← (1/|C|) Σ_{x∈C} φ(a_{ℓ,S}^FP16(x), a_{ℓ,S}^(b)(x))
        RestoreFP16(f_W)
    end for
end for
return D^φ
```

A.3.3 ILP Allocation for the Baselines

All four score tables are passed separately to the same multiple-choice knapsack formulation. For $m \in \{\mathrm{HAWQ}, \mathrm{Fisher}, \mathrm{QE}, \mathrm{KL}\}$, let $x_{\ell,b} \in \{0,1\}$ indicate whether linear module ℓ is assigned bit-width b. The allocation is

$$\begin{aligned} \min_{x} \quad & \sum_{\ell\in\mathcal{L}} \sum_{b\in\mathcal{B}} d_\ell^m(b)\, x_{\ell,b} \\ \text{s.t.} \quad & \textstyle\sum_{b\in\mathcal{B}} x_{\ell,b} = 1 \quad \forall \ell \in \mathcal{L}, \\ & \tfrac{1}{8}\textstyle\sum_{\ell,b} P_\ell\, b\, x_{\ell,b} \le M, \\ & x_{\ell,b} \in \{0,1\} \quad \forall \ell, b. \end{aligned} \tag{20}$$

Here, $\mathcal{L}$ is the set of linear modules inside the transformer blocks, P_ℓ is the number of weights in module ℓ, and M is the packed linear-weight memory budget in bytes. The budget excludes scales, zero-points, activations, embeddings, normalization parameters, the language-model head, and padding overhead.

For numerical conditioning, the implementation divides all scores by their maximum absolute value before optimization. This common positive scaling does not change the minimizing allocation. We solve the ILP using PuLP with HiGHS when available and CBC otherwise, using a relative optimality gap of 0.5%.

All four baselines use identical allocation constraints and differ only in their score tables, each computed once under the FP16 prior and reused across all memory budgets. All four scores are measured under the deployed base quantizer, so AWQ and GPTQ each receive their own set of score tables. After solving the ILP, the base quantizer is calibrated from scratch on the selected bit-width

plan and applied with 8-bit activations. Baseline plans therefore receive plan-specific calibration, whereas MIXQUANT reuses parameters averaged over several plans per memory interval (Stage 3; Section A.2.4).

Algorithm 10 summarizes how each score is converted into a feasible bit-width plan and instantiated using AWQ or GPTQ.

Algorithm 10 ILP Allocation for Baselines

Require: Score table D^m for $m \in \{\mathrm{HAWQ}, \mathrm{Fisher}, \mathrm{QE}, \mathrm{KL}\}$; module parameter counts $\{P_\ell\}$; bit-widths $\mathcal{B}$; memory budget M; base quantizer $\mathcal{T} \in \{\mathrm{AWQ}, \mathrm{GPTQ}\}$

Ensure: bit-width plan $\mathbf{b}$ and quantized model $\widehat{f}$

$\alpha \leftarrow \max_{\ell,b} |D^m_{\ell,b}|$

$\overline{D}^m_{\ell,b} \leftarrow D^m_{\ell,b}/\alpha$ ▷ Numerical conditioning

for all $\ell \in \mathcal{L}$ and $b \in \mathcal{B}$ **do**

  Create binary variable $x_{\ell,b} \in \{0, 1\}$

end for

Set objective: $\min_x \sum_{\ell,b} \overline{D}^m_{\ell,b} x_{\ell,b}$

for all $\ell \in \mathcal{L}$ **do**

  Add constraint: $\sum_{b \in \mathcal{B}} x_{\ell,b} = 1$

end for

Add memory constraint: $\frac{1}{8} \sum_{\ell,b} P_\ell b \, x_{\ell,b} \leq M$

$x^\star \leftarrow \textsc{SolveILP}(x, \text{relative gap} = 0.005)$

for all $\ell \in \mathcal{L}$ **do**

  $b_\ell \leftarrow$ the unique b such that $x^\star_{\ell,b} = 1$

end for

$\mu \leftarrow \textsc{CalibrateQuantizer}(f_W, \mathcal{C}, \mathcal{T}, \mathbf{b})$ ▷ Fresh, plan-specific calibration

$\widehat{f} \leftarrow \textsc{ApplyQuantizer}(f_W, \mathcal{T}, \mathbf{b}, \mu, \text{activation bits} = 8)$

return $\mathbf{b}, \widehat{f}$

A.4 TAIL-PENALTY ANALYSIS

Figure 4 shows how WikiText-2 perplexity changes as the tail-penalty strength γ increases.

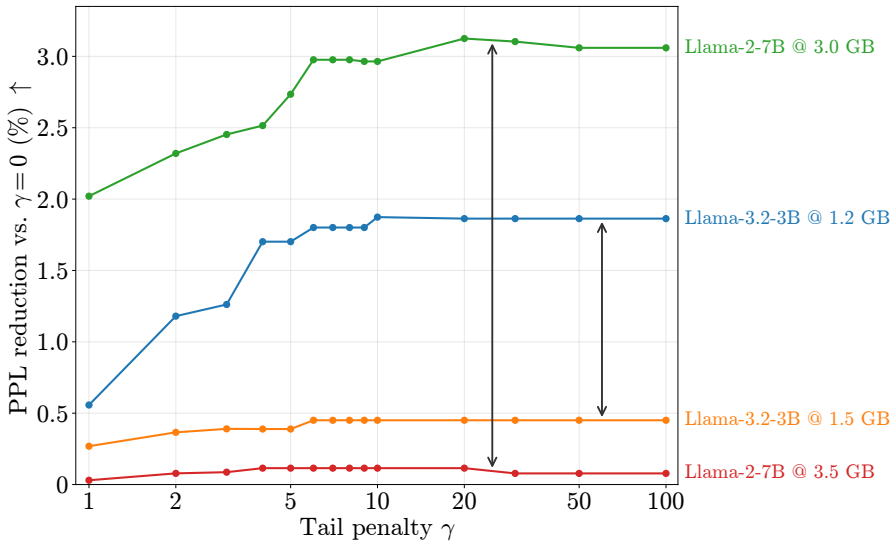


Figure 4: Effect of the tail penalty. Relative change in WikiText-2 perplexity versus the unregularized allocator ($\gamma = 0$) as γ increases across four (model, budget) settings. Here PPL_0 is the $\gamma = 0$ perplexity for each curve.