Efficient LLMs via Switchable and Dynamic Quantization

Project Deliverables

Tsung-Huan Yang

Q1. What is the task accuracy of various quantization bit-width configurations?

Implementation note The original LLM-QAT workflow prompts GPT-2 with *random tokens* sampled from the vocabulary and trains quantized LLMs from the resulting completions. I instead initialize the prompt with the full {`context, question`} pair from SQuAD training set; GPT-2 continues the text, and those continuations become supervised fine-tuning targets for quantized GPT-2. Using task-aware prompts yields generations that better reflect SQuAD semantics and therefore train the quantized models to behave more like the full-precision models on SQuAD.

Training set-up. By training on a pre-defined set of bitmaps (including all-8-bit, stripe 8/4, all-4-bit, stripe 4/2, and all-2-bit) and aggregating the loss from every configuration in one backward pass, the optimizer updates GPT-2 once per batch while seeing every precision variant. This yields a **single checkpoint that can switch its quantization bitmap at inference time**. The task accuracy under each bitmap is reported below.

Table 1: Accuracy–efficiency of pre-defined bitmaps on SQuAD dev set.

Bitmap	F1	Weights (MB)
FP32 (Original)	10.8039	684.94
INT8	12.3534	442.25
Stripe 8 & 4	12.5233	418.63
INT4	12.3336	401.75
Stripe 4 & 2	1.7294	389.94
INT2	1.0986	381.50

Q2. How did you determine the optimal quantization bit-width configurations?

A. I applied a two-step procedure:

1. **Layer-wise sensitivity analysis.** For every linear layer ℓ and candidate bit-width $b \in \{8, 4, 2\}$ I compute the mean-squared error

$$\mathrm{MSE}(\ell, b) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left\| f_\ell^{\mathrm{FP}}(\mathbf{h}_{\ell-1}) - f_\ell^{(b)}(\mathbf{h}_{\ell-1}) \right\|_2^2,$$

where f_ℓ^{FP} is the full-precision layer, $f_\ell^{(b)}$ is the same layer quantized to b bits (with its corresponding LoRA adapter activated), and $\mathbf{h}_{\ell-1}$ denotes the activations emitted by the preceding full-precision sub-network. The expectation is approximated over 100 randomly sampled SQuAD training examples.

Layers exhibiting lower MSE are deemed more *robust* to quantization noise and are therefore prioritized for lower bit-width assignment.

2. **Greedy search with budget.** Starting from the FP32 model, layers were greedily quantized from least to most sensitive while the training-set F1 drop remained within a 30% budget. The sweep first tried 8-bit everywhere, then 4-bit, finally 2-bit.

The optimal quantization configuration contained **44 × 8-bit layers** and **4 × 4-bit layers**. Its metrics are in Table 2, and Figure 1 visualizes the greedy path versus the six pre-defined bitmaps. The optimal configuration offers a better accuracy-efficiency tradeoff compared to the six pre-defined bitmaps.

Table 2: Final mixed-precision checkpoint found by the greedy search.

Final Config	F1	Weights (MB)
44 × 8-bit + 4 × 4-bit	11.53	401.61

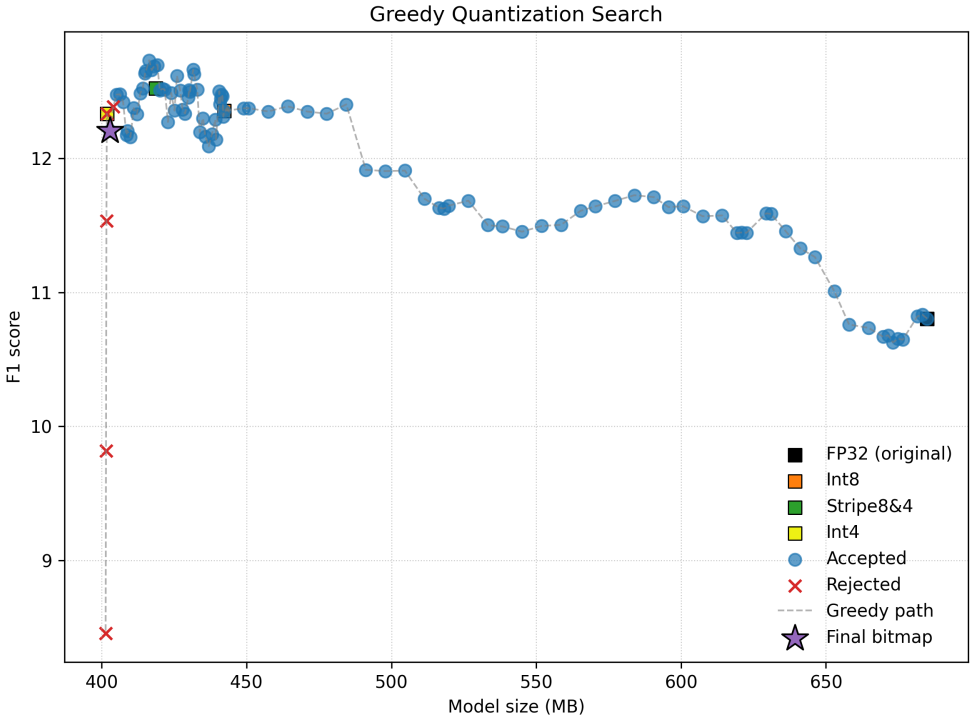

Figure 1: Greedy search trajectory: accuracy vs. model size. The final point narrowly missed the F1 budget, hence marked "rejected" during the sweep. It is still plotted for completeness.

Observation. Figure 2 exposes a consistent hierarchy of quantization robustness. Within every transformer block, `c_proj` exhibits the largest error once weights are quantized. One possible explanation is that MLP projections perform a high-gain, channel-expanding linear transformation with no residual mixing, so any weight perturbation is *amplified* before the activation layer.

Quantizing to two bits are especially destructive: for the same layer, the 2-bit error is often $10\times$–$100\times$ larger than the 4-bit counterpart. With signed symmetric quantization, the weight tensor collapses to $\{-1, 0, +1\}$; Such coarse quantizations, even with LoRA adapters, cannot recover the dense information learned by GPT-2. Consequently, the **stripe-4&2** and **all-2-bit** training branches failed to converge, and the greedy search ultimately assigns *few* layers to 2-bit.

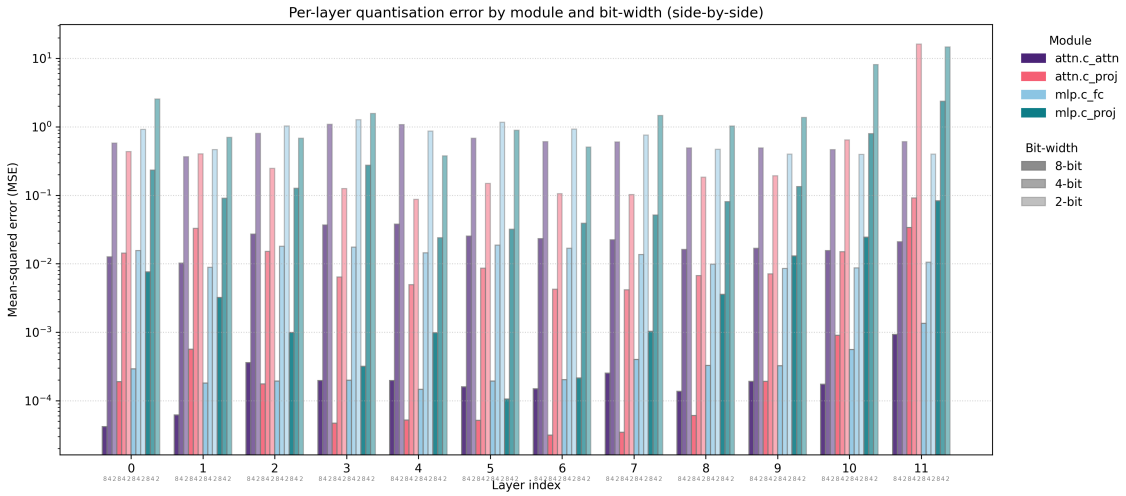

Figure 2: Per-layer MSE. Same types of modules are grouped by the same colors. Different bit-widths are highlighted with different opacity levels

Q3. Additional training objectives

A. Below are some objectives that can be added on top of the cross-entropy loss $\mathcal{L}_{\text{CE}}$ to facilitate switching quantization bit-widths.

1. **Switch dropout (precision dropout)** With probability p_{sd} (e.g. 0.1) randomly down-shift the bit-width of *each layer* by one level during training. This may resemble dropout and train the network to be robust under on-the-fly bit changes.

2. **Expected cost regularizer** A regularizer to reflect expected memory/latency cost. If c_b is the memory/latency cost of b-bit weights and $q_{\ell b}$ is the learned probability of assigning bit b to layer ℓ, we can add this objective

$$\mathcal{L}_{\text{cost}} = \lambda \sum_{\ell} \underbrace{\left(q_{\ell 8} c_8 + q_{\ell 4} c_4 + q_{\ell 2} c_2\right)}_{\text{expected resource}}$$

   The model is then regularized to *self-allocate* cheaper bit-widths while maintaining accuracy via $\mathcal{L}_{\text{CE}}$.

3. **Activation-stability penalty.** Beyond the cross-entropy term $\mathcal{L}_{\text{CE}}$, we can encourage hidden states at lower precision to mimic their 32-bit counterparts. Let $\pi(\ell) \in \{8, 4, 2\}$ denote the bit-width currently assigned to layer ℓ. For a mini-batch $\mathcal{B}$ we minimise

$$\mathcal{L}_{\text{act}} = \frac{1}{L} \sum_{\ell=1}^{L} \text{Var}_{x \in \mathcal{B}} \Big(f_{\ell}^{(32)}(x) \; - \; f_{\ell}^{\left(\pi(\ell)\right)}(x) \Big),$$

   where $f_{\ell}^{(b)}$ is the output of layer ℓ quantised to b bits.

Q4. Does cyclic-precision result match the findings of CPT (ICLR'21)?

A. *No.* In CPT, they reported that cycling the training precision yields accuracy gains on ImageNet-classifiers. My GPT-2 experiment (Table 3) showed no significant improvement over the "train-all-bits-at-once" baseline (Table 1)

Table 3: Accuracy and efficiency of pre-defined bitmaps on SQuAD dev set using Cyclic Precision Training

Bitmap	F1	Weights (MB)
FP	10.6562	684.94
INT8	12.1202	442.25
Stripe 8 & 4	12.0638	418.63
INT4	11.5034	401.75
Stripe 4 & 2	1.3555	389.94
INT2	1.1788	381.50

Potential reasons.

1. **Redundant supervision** "train-all-bits-at-once" already aggregates gradients from *all* bitmaps in each step; cyclic schedules may add little extra diversity.
2. **Architecture difference** CPT validated on image-classification CNNs where weight magnitudes vary wildly across layers. Transformer blocks, which come with LayerNorm and residual pathways, might be inherently more robust, narrowing the benefit margin.

Q5. Random Precision Switch v.s. Adversarial Robustness

To gauge robustness, I generated 200 suffixes with `nanoGCG` on AdvBench instruct–target pairs and measured attack success rate (ASR). Results are shown in Table 4. In random configuration, each layer's bit-width is randomly sampled from $4, 8, 32$.

Table 4: Attack success rate (ASR) vs. quantization level.

Bitmap	ASR
FP	0.955
INT8	0.950
Stripe 8 & 4	0.757
INT4	0.610
Stripe 4 & 2	0.488
INT2	0.407
Random (4/8/32)	0.421

ASR decreases as precision drops. In random configuration, the ASR is even lower compared to INT4. This observation is consistent with the finding in Double-Quant Win.