Alg Coding Test 2 report

Han Houpu

September 26, 2025

1 Step 4: Quantization Accuracy and Insights

1.1 Task Accuracy under Different Bit-widths

Here is the table and result.

Table 1: Step 4 results full run

Config	EM	F1	Avg. bits
C1_all8	0.303	9.211	8.000
C8_mlpfc4_mlpproj8	0.170	7.356	5.000
C4_back8_front4	0.218	6.766	5.333
C3_front8_back4	0.085	6.711	5.333
C5_sandwich	0.104	6.640	5.333
C7_qkv4_proj8	0.246	6.592	5.000
C10_mixed_budget	0.170	6.486	4.667
C2_all4	0.151	6.387	4.000
C9_layernorm_fp32	0.151	6.387	4.000
config_4bit	0.151	6.387	4.000
C6_qkv8_proj4	0.246	6.295	5.000
test_2bit_6bit	0.104	5.348	4.083

1.2 Optimal Configuration and Observations

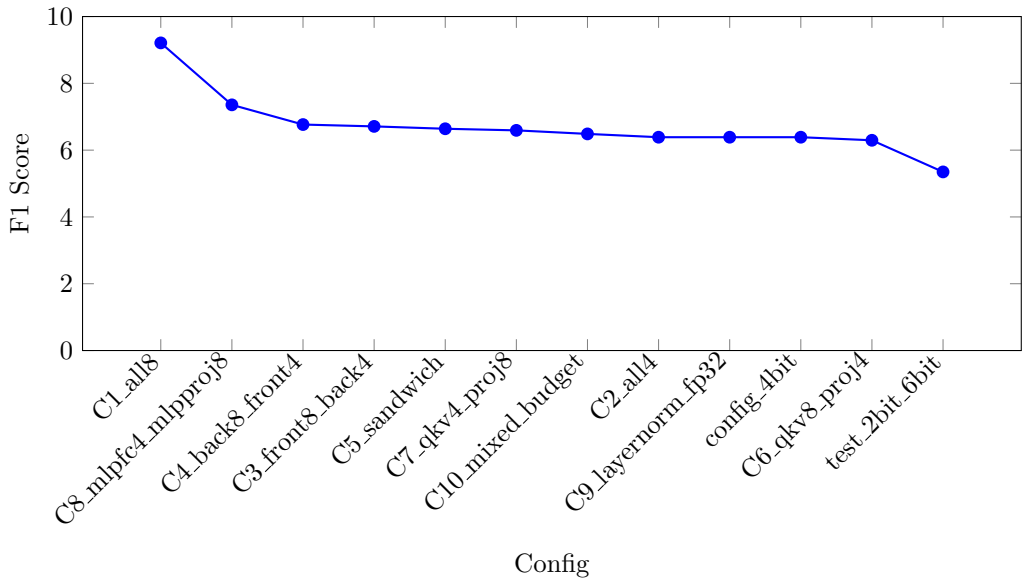

Figure 1: F1 score across different quantization configs

At very first, I crated 10 cfg files to do a rough sreach. based on the results in Figure1, we plan to fix `transformer.h.x.mlp.c_proj` at 8 bits. Although several configurations have a similar average bit-width, the specific configurations with `mlp.c_proj` quantized at 8 bits achieves the strongest F1 score. This indicates that the `mlp.c_proj` layer is especially important for getting more accuracy.

Figure2 presents additional experiments motivated by this observation.

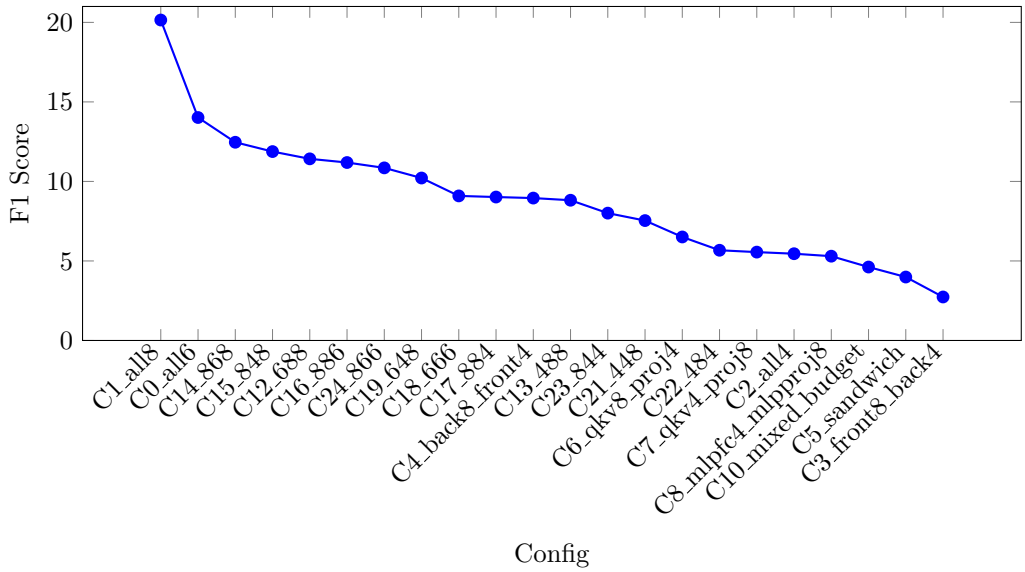

Figure 2: F1 scores across different quantization configs

Due to time constraints, the results in Figure2 are based on a subset ($N = 1500$). I made several improvements and additional changes after Figure1, 1) changed some evaluation strategies. 2) add 6bits configurations for curiosity(so the final iterations expand to 1500). 3) add more configurations files to find a more precise trend. The absolute F1 scores might differ from Figure1, but the total trend remains the same.

The results exhibit an approximately linear trend: as the average bit-width decreases, the F1 score also decreases. Based on this observation, one possible strategy is to keep the bit-width of `transformer.h.x.mlp.c_proj` at the maximum level(in this situation is 8), since it appears to be the most critical for performance, and then selectively increase the bit-width of other layers as needed.

1.3 Additional Training Objectives

After reviewing some papers, I believe that the distillation approach could add here to give us some additional training objectives. Here is the paper:

1)Mixed-precision Deep Neural Network Quantization With Progressive Bit-width Allocation and Joint Training (OpenReview 2023). From this, We may add joint training loss as an additional training objective, this will compute supervised losses across multiple bit-width configurations at the same time, so that different precision branches share the same backbone.

2)MBQuant: A Novel Multi-Branch Topology Method for Arbitrary Bit-width Network Quantization. From this, We may add multi-branch consistency loss as an additional training objective, the main idea here is, for the same input, require outputs from different bit-width branches to be close (e.g., using MSE or KL loss). This may help the model give stable predictions.

A possible risk is that it may decrease GPT-2's adversarial robustness in Step 6, since the high-precision and low-precision models become more similar, though this still needs further study.

2 Step 5: Relation to Cyclic Precision Training (ICLR'21)

Table 2: Step3 vs Step5 Comparison Summary

Bit-width	Step3	Step5	Δ
8-bit	20.046	24.274	+4.227
4-bit	5.355	21.080	+15.725

The results show that CPT can enhance the accuracy of our transformer model, with the 4-bit (low precision) branch gaining the most improvement, this aligns well with CPT (ICLR'21) paper. However, the magnitude of this increase for 4-bit may be somewhat suspicious. Due to time limit, I did not find this phenomenon in depth, and this needs future work.

3 Step 6: adversarial attacks

Table 3: Attack results summary

attack	model	clean_f1	adv_f1	Δ_f1	success_rate
PGD	fp32_lora	60.596	28.504	-32.092	0.29
PGD	static_8bit	35.998	29.611	-6.387	0.11
PGD	random_switch	23.450	15.343	-8.107	0.15
HotFlip	fp32_lora	60.596	34.055	-26.541	0.27
HotFlip	static_8bit	35.998	28.612	-7.387	0.12
HotFlip	random_switch	27.778	22.937	-4.841	0.13

Yes, the phenomenon broadly aligns with Double-Win Quant (ICML'21), but it is not a perfect match. Both 8-bit and random switch increase robustness, however some of the gain seems to come simply from the drop in clean accuracy (FP32 → 8-bit). I'm not sure how much the random precision switching itself contributes. From my latest observations, random switch behaves much like static 8-bit. I ran many tests on my own machine and the brief conclusion is that random switch tends to work better in more scenarios.

There are a few likely reasons for the uncertainty. One is the evaluation setup: how much F1 is used as the attack target and how much F1 is regarded as a successful attack might have a significant impact. I have not built a global large-scale testing pipeline yet, most of my runs are small-sample (these attacks are very expensive to run). So maybe statistical noise could explain why random precision switching sometimes looks not well. Another possibility is our switch configuration: I used a 4/6/8-bit random mix; maybe switching between just 4 and 8 (4/8-bit) would cause more noise and make the defense more effective. These are very early ideas, they need more future testing.

4 Promising Research Directions and Hypotheses

For me, what really matters is performance improvement or even breakthroughs, not just saving compute and usage. Efficiency alone does not make AI more intelligent. From this perspective through this project, I think there are 3 points worth mentioning.

The first point is, CPT seems in some cases to give real performance improvement. This makes me wonder how far such improvement can go (the upper bound), and what the actual source of the improvement might be (how to explain it).

Second, random bit-width switching at deployment could make the model harder to attack. It might also reduce the possibility of hallucinations and repetitions, since adding randomness can at least break repeated outputs, and maybe this kind of randomness could make the core important knowledge more stable.

Last thought, a very early idea is that current parameters feel too static, they are not really "alive." I wonder if we could design a dynamic LoRA that keeps running, doing small updates or perturbations to

weights even without new input. That way the model is always making some minimal internal iteration or "thinking". This might give extra performance. The idea may come from chain-of-thought, so it seems to be a workable direction.