Leveraging Efficient Transformer Quantization for CodeGPT: A Post-Training Analysis

Large Language Models are increasingly popular in machine learning for their natural language understanding abilities. Code auto-completion is a major use case, providing relevant suggestions to developers. However, the high costs and ethical concerns of deploying large models on resource-limited devices and online services are challenges.
We demonstrate that CodeGPT has high resilience to quantization noise, enabling the model to be compressed by four times its size with minimal accuracy loss using post-training quantization techniques.

1. Introduction

- Large Language Models' (LLMs) substantial size presents significant hurdles for their deployment.
- High running costs, limits LLMs' usage on resource-limited devices, and raise practical and ethical concerns when accessed through online services.
- Researchers have explored many compression techniques, including knowledge distillation, pruning, and quantization.
- Most of the research focuses on compressing BERT-like models. Almost no research has been done on the efficacy of these techniques on the CodeGPT model.

In this research, we strive to investigate the question:

"How effective are post-training quantization techniques (PTQ) for compressing a CodeGPT generation model?".

2. Background

But what is **post-training quantization**?

- Quantization involves reducing the precision of the model's weights and activations, thereby diminishing the memory footprint and enhancing inference speed
- PTQ is a quantization approach applied to pre-trained models without any additional training.
- The quantization process introduces noise to the network, which can potentially degrade performance.
- Calibration is a process used to mitigate noise and determine optimal quantization parameters for each tensor.
- Calibration involves estimating the scale factor and zero point using range estimators

$$\mathbf{x}^{\mathbb{Z}} = clip(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; 0, 2^b - 1)$$

Figure 1:
Asymmetric quantization function [1]

3. Methodology

Our methodology involves implementing, comparing, and evaluating different PTQ methods:

- **Naive PTQ:** applies simple per-tensor quantization on weights and activations without optimizations.
- **Mixed-precision (MP) PTQ:** uses different bit-widths for activation tensors to reduce accuracy loss while maintaining high compression rates.
- **Per-embedding-group (PEG) PTQ:** applies quantization to separate groups of embeddings based on their usage patterns.

We compare our best results, to the following other compression approaches:

- **De Moor** [3] uses hybrid in-training knowledge distillation, layer reduction, and quantization techniques.
- **Malmsten** [4] uses in-training knowledge distillation techniques.
- **Sochirca** [5] utilizes hybrid post-training pruning and quantization techniques on CPUs.

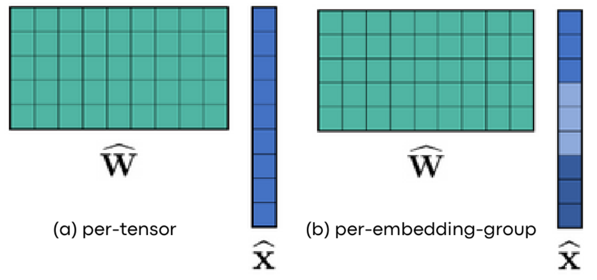

Figure 2:
An overview for several choices of activation quantization granularity. The color indicates quantization parameter sharing. In all cases we assume per-tensor weight quantization. [1]

4. Results

The CodeGPT model is fine-tuned and tested on the PY150 dataset of CodeXGLUE [2] on the Auto-Completion task.

We calculate the accuracy of our model on 1000 samples using the Edit Similarity and Exact Match metrics:

- **Edit similarity (ES):** quantifies the structural and semantic similarity between the model's output and the expected output.
- **Exact match (EM):** measures the number of times the model's output matches the expected output exactly.

We find that the best results are achieved with the following parameters:

- MSE for weight quantization Range Estimator
- >=8 bits for weights
- >=16 bits for activations
- 16-bit residual sum of block for MP PTQ
- 4 permuted groups for PEG PTQ

Method	ES	EM	Compression
Baseline	41	17	1x
W8A8	38	16	4x
W8A32	**40**	**17**	**4x**
W8A16	**40**	**17**	**4x**
W4A32	36	14	8x
W4A16	36	14	8x

Figure 3:
Naive PTQ Experiments:
WxAy means quantization with x bits for weights and y bits for activations

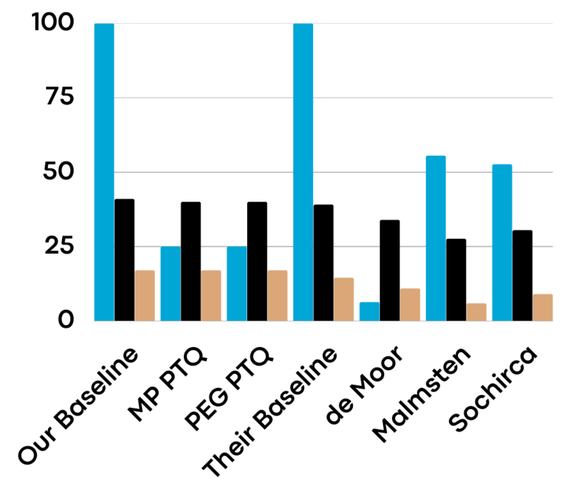

Figure 4:
Comparison of compression approaches:
Size % (blue), ES (black), EM (brown)

5. Conclusion

- Naive PTQ implementation achieves a compression rate of 4x with negligible accuracy loss.
- Advanced PTQ methods like mixed-precision, per-tensor, or per-embedding-group can virtually eliminate accuracy loss.
- CodeGPT is most susceptible to quantization of fewer than 8 bits for weights and less than 16 bits for activations.
- Performance loss could not be recovered for 4-bit quantization of weights using the implemented techniques.

6. Limitations

- The study simulated quantization, suggesting the need for future investigations using tensor quantization on different hardware.
- Further analyses can validate the performance of PTQ methods on CodeGPT, determine inference time speedup, and assess reduction in GPU size during runtime.

References

[1] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 7947–7969, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.
[2] Yiming Lu, Meng Zhang, Yuncong Li, and Xiaodong Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 6725–6736, 2021
[3] Aral D de Moor. Codegpt on xtc, 2023
[4] Emil Malmsten. Distilling code-generation models for local use, 2023
[5] Dan Sochirca. Compressing code generation language models on cpus, 2023

Authors	Mauro Storti | m.storti@student.tudelft.nl	**Supervisors**	Dr. Maliheh Izadi, ir. Ali Al-Kaswan
Responsible Professor	Prof. Dr. Arie van Deursen	**University**	EEMCS, Delft University of Technology

TUDelft