194.077 Applied Deep Learning
Post-training int8 Weight Quantization of a small autoregressive Transformer

Philipp Sepin
11918494

1 Introduction

1.1 Problem Statement

Large language models (LLMs) have demonstrated impressive capabilities. However, their millions to billions of parameters/weights consume a large amount of GPU memory and computational resources for inference, limiting their application in resource-constrained settings, such as edge devices.

1.2 Proposed Solution

Post-training quantization reduces the number of bits per weight, significantly lowering memory usage. This enables deployment on edge devices and paves the way for accelerated inference on GPUs that support low-precision operations, while preserving most of the model's performance. All of this is achieved without the need for retraining the model.

2 Methodology

I re-implemented a small 2-3 M parameter autoregressive transformer based on NanoGPT [**?**], trained it once on a 10 MB subset of the TinyStories V2 text dataset [**?**], and implemented two post-training integer (int8) weight quantization techniques.

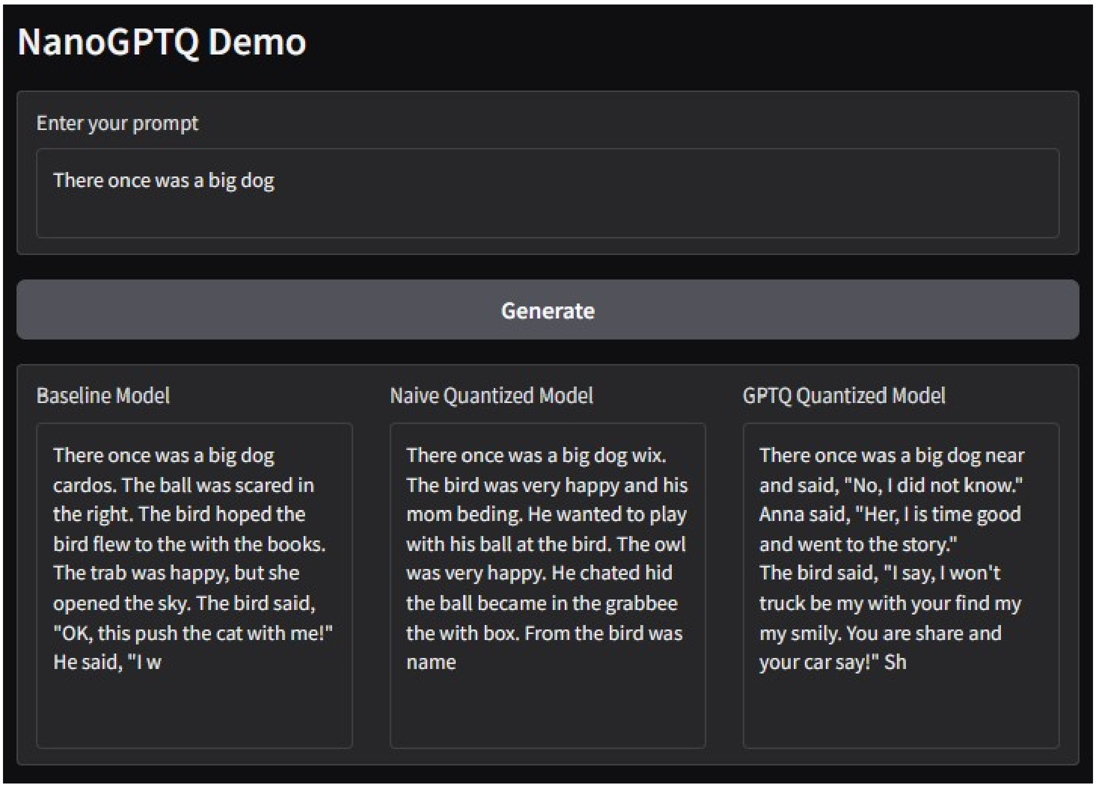
- **Naive Rounding Quantization:** Rounds each weight to the nearest representable int8 value.
- **Hessian-aware Quantization (GPTQ-like):** Iteratively rounds weights one by one and adjusts all remaining weights using a Hessian approximation computed on a small calibration set [**?**, **?**].

Both quantized model variants were then compared to the floating-point (fp32) baseline model. All computations were done on a Nvidia GTX 960M (4 GB VRAM). I evaluated the models using perplexity and developed a local Python web app with Gradio, which generates and displays responses from the three model variants side by side for easy comparison given a user prompt.

3 Results and Insights

3.1 Results

Model Variant	Perplexity	Model Size
Baseline (FP32)	2.76	31.9 MB
Naive Quantization (INT8)	2.78	2.9 MB
GPTQ Quantization (INT8)	2.76	2.9 MB



There once was a big dog

There once was a big dog cardos. The ball was scared in the right. The bird hoped the bird flew to the with the books. The trab was happy, but she opened the sky. The bird said, "OK, this push the cat with me!" He said, "I w

There once was a big dog wix. The bird was very happy and his mom beding. He wanted to play with his ball at the bird. The owl was very happy. He chated hid the ball became in the grabbee the with box. From the bird was name

There once was a big dog near and said, "No, I did not know." Anna said, "Her, I is time good and went to the story."
The bird said, "I say, I won't truck be my with your find my my smily. You are share and your car say!" Sh

The baseline model was able to predict full and correct words, often generating even half-sentences that are accurate, and it uses punctuation marks appropriately. In contrast, the naively quantized model frequently predicted incorrect or nonexistent words and misused symbols, while the GPTQ version eliminated these errors and produced outputs similar to the baseline. This qualitative difference is also reflected in the perplexity results. Integer quatization reduced the model size by 90%, from 31.9 MB to 2.9 MB. The naive quantization led to a slightly worse perplexity than the baseline, while the Hessian-aware quantization was able to match the baseline perplexity.

3.2 Insights

Re-implementing NanoGPT from scratch was harder and more time-consuming than expected, but a very valuable learning experience, as it deepened my understanding of autoregressive transformers. Balancing the training of a 2-3 M parameter model on such a weak GPU was also quite challenging, but pretty interesting, as It tought me valuable lessons about managing expensive training runs.

4 Retrospective

4.1 What would be done differently?

I am quite satisfied with the overall project outcome. If I would have had more time, I would have been interested in also quantizing the activations during inference, which would enable accelerated inference on compatible GPUs. It also would have been very interesting to go to lower-bit datatypes such as int4, where GPTQ-sytle quantization really shines, but this would have required a custom PyTorch datatype. Additionally, it would have been benefitial for such a project to have better computational resources, as the benefits of quantization become more apparent with larger models. And those larger models would have required better GPUs.

4.2 Time Management

The project took approximately 42.5 hours to complete, compared to the initial estimate of 55 hours.

Task	**Estimated Time (h)**	**Actual Time (h)**	**Difference (h)**
Repo setup	3	0.5	2.5
Data collection	3	1.5	1.5
NanoGPT Implementation	8	11	-3
Baseline Training	12	11	1
Naive Quantization	4	2	2
Hessian-aware Quantization	8	8.5	-0.5
Evaluation & Demo	7	4.5	2.5
Report & Presentation	10	3.5	6.5
Total	**55**	**42.5**	**12.5**

5 References

References

[1] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[2] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[3] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[4] Andrej Karpathy. nanogpt. `https://github.com/karpathy/nanoGPT`, 2023.