Assignment 1

194.125 – AI/ML in the Era of Climate Change 2024W

Code: `github.com/sueszli/byte-sized-gains/`

Contents

Introduction **2**

ODM Quantization **2**

LLM Quantization **6**

Introduction

In recent years, object detection models (ODMs) and large language models (LLMs) have made unprecedented progress in computer vision and natural language processing, respectively. These deep neural networks have found applications across various domains, from autonomous vehicles and surveillance systems to chatbots and content generation. However, the increasing complexity and size of these models have led to significant computational and energy demands, posing challenges for widespread deployment and raising concerns about their environmental impact.

As the global community grapples with climate change and the urgent need for sustainable technologies, the optimization of ODMs and LLMs has become a critical area of research. Quantization, a technique that reduces the precision of model parameters and activations, has emerged as a promising approach to address these challenges. By converting high-precision floating-point representations to lower-precision formats, quantization can substantially decrease the model's memory footprint and compute usage while compromising predictive performance.

Our project focuses on applying advanced quantization techniques to both ODMs and LLMs, with the dual objectives of **improving their efficiency and reducing their carbon footprint**. We aim to implement quantization methods that enable these models to run on resource-constrained devices while maintaining their accuracy and functionality. By optimizing these models for deployment on edge devices and less powerful hardware, we can reduce the need for energy-intensive cloud computing and data centers, contributing to a more eco-friendly computational landscape.

The quantization of ODMs presents unique challenges due to the spatial nature of visual data and the need for precise localization in detection tasks. For LLMs, the primary obstacles lie in preserving the nuanced relationships between words and maintaining coherence across long sequences. Our project these model-specific issues while exploring commonalities in quantization strategies that can be applied across both domains. Through this project, we seek to demonstrate that quantization can play a crucial role in making advanced machine learning models more accessible and environmentally sustainable. By reducing the computational and energy requirements of ODMs and LLMs, we aim to pave the way for their widespread adoption in eco-conscious applications, ultimately contributing to global efforts in mitigating climate change and promoting sustainable technological development.

ODM Quantization

Challenges In the first part of this project we quantize an object detection model.

We started this task off by aiming for the stars and comparing the best models we could find on the public "papers with code" leaderboard for the COCO 2017 dataset [1]. Then we ran our own experiments to find the most representative models from each architecture family [2]. We then noticed the DETR family to perform the best, particularly the "facebook/detr-resnet-101-dc5" model, as it also generalizes well across multiple datasets using the COCO vocabulary. This specific DETR model additionally was trained on COCO 2017 dataset - the exact one we are using - which should give it an advantage.

But after implementing the entire evaluation pipeline for our experiments in PyTorch we realized Torch XLA builds a shared library, `_XLAC.so` that needs to link to the version of Python it was built with (currently 3.10 or 3.11). And in order to ensure that `import _XLAC` can succeed, we had to update the `LD_LIBRARY_PATH` to the lib directory of our Python environment. This was a major blocker for us as we were unable to resolve this issue even within a docker container. This made us have to pivot to TensorFlow models instead as they are directly supported by LiteRT and do not need a seperate layer of abstraction and translation like PyTorch models do. Additionally the LiteRT compiled models return Tensorflow specific data types, making it more convenient to just write the whole pipeline in Tensorflow v2.

We started from scratch, but this time instead of looking for state-of-the-art performance we were solely looking for models compatible with LiteRT. We found that the models that are supported by LiteRT are very limited and the only models that are supported are the ones that are available in the TensorFlow model zoo. We initially started off by using Efficientnet but stumbled upon 0-gradient bugs in the int8 quantified version – which we assume is because of the depth of the model. This lead to the int8 quantization tuning steps wirh 100 samples to sometimes take over 2h a consumer machine.

We spent 2 full working days trying to mitigate these issues but ended up pivoting again, but this time to the SSD family of models, as they are even more lightweight and enable faster iteration for our demonstrative purposes. Finally we decided to use `mobilenet_v2` as our base model.

But given the experimental nature of LiteRT and the lack of both community and documentation especially the full integer quantization was very tedious, as both the quantization of weights and the inputs and outputs resulted in many complications. Additionally the authors auf this accelerator module didn't implement more fine granular error messages, causing shotgun debugging to be the only viable option.

Methodology We ran our benchmarks on a consumer MacBook M2 Pro and the latest Tensorflow Version 2.17.0 on macOS 14.6.1 23G93 arm64 hosted by a Mac14,10 with Kernel version 23.6.0 and 16384MiB of memory.

[1] https://paperswithcode.com/sota/object-detection-on-coco
[2] https://github.com/ETH-DISCO/advx-bench/tree/main/analysis/model-selection

All python dependencies were compiled using `pip-compile` and dumped out of a virtual environment for reproducibility. A docker container is provided for cross platform builds.

We quantized the mobilenet model with LiteRT and the configurations [float32, float16, int8].

The experiment was conducted as follows:

... insert some parts where perplexity explains code

Results The quantized models had the following memory footprints:

- original model: 31.88 MB
- float32 model: 23.72 MB
- float16 model: 12.17 MB
- int8 model: 7.21 MB

Meaning with each of our selected quantization steps the memory footprint halved. This os to be expected and also serves as a little sanity check for our model quantizer.

Next we want to gain a better understanding of inference speed vs. precision as the intersection over union (IoU@0.01). We observed the following correlations between predictive efficiency and effectiveness:

- int8 : -0.0236756526195283
- float16 : 0.0232883892068324
- float32 : -0.0365188366830796

This is because the changes in inference speed were really marginal andin the 1e-2 region, as visualized in the logarithmic boxplot of all inference times (add ref to plot).

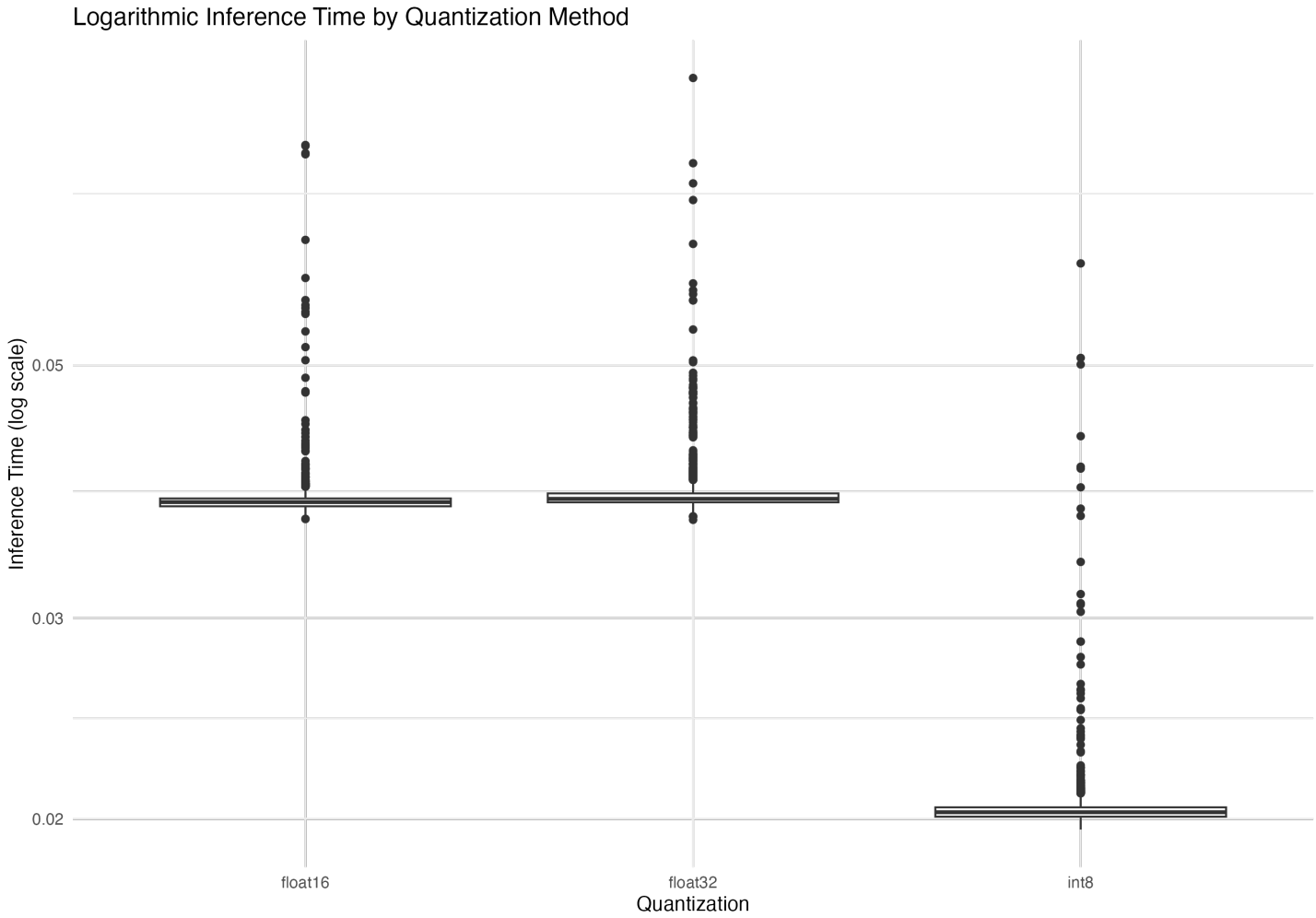

Figure 1: ODM: Logarithmic Inference Time Boxplot

The IoU@0.01 precision however does show fluctuations as visualized in the scatterplot and boxplots (add ref to plot).

However very counter intuitively, as we increase the model size and granularity the performance seems to worsen.

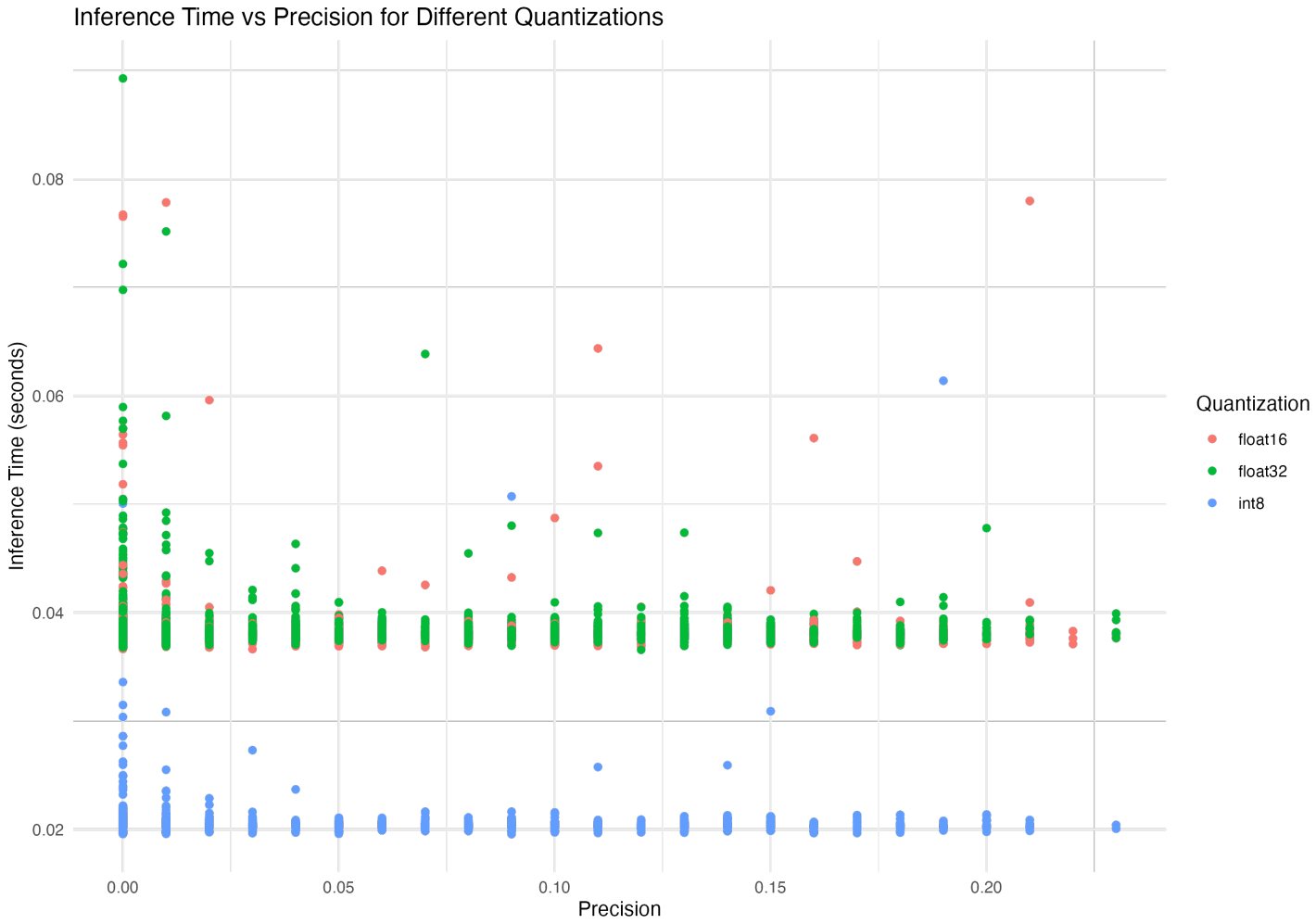

Figure 2: ODM: Inference Time vs. IoU Precision Scatterplot

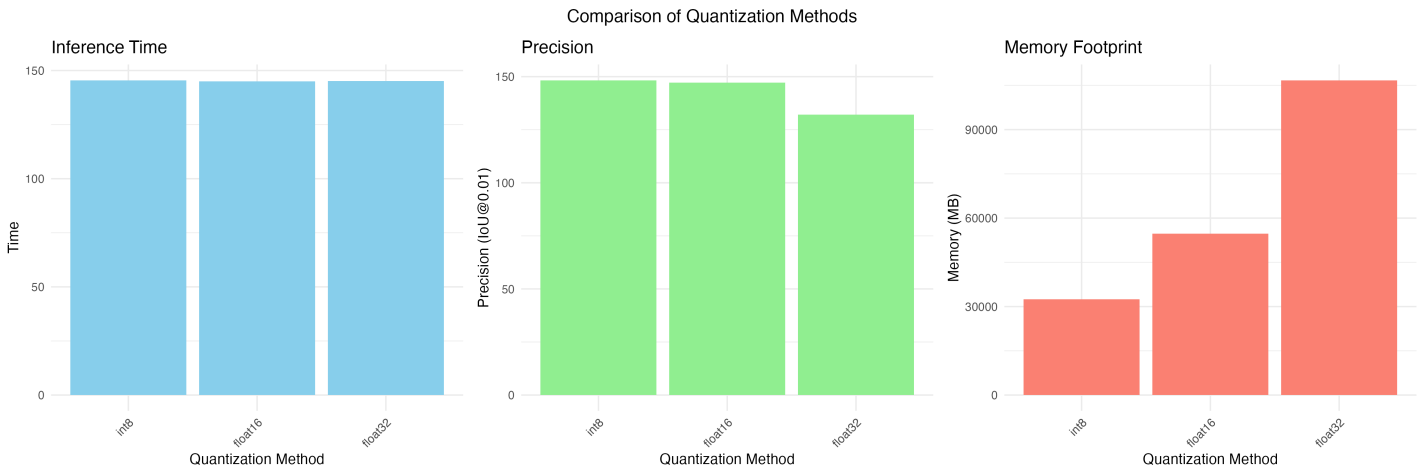

Figure 3: ODM: Inference Time vs. IoU Precision Boxplots

Given the counter intuitive results we observed in the benchmarks there is a realistic chance that the IoU implementation is faulty, given how bad the models performed and how low we set the area of intersection to be to count as a hit. However the same algorithm was also used for our PyTorch benchmark and lead to sane results. We were unable to find the cause.

In conclusion: The results are inconclusive. Further inspection to is necessary to validate our counter intuitive findings.

LLM Quantization

LAMBADA leaderboard:

- https://paperswithcode.com/sota/language-modelling-on-lambada