Assignment 1

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

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

Contents

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

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

Challenges 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 generalizes well and was trained on the same dataset we are using, namely COCo-2017.

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 / 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 only limited ourselves to those which are compatible with LiteRT. Our first choice was Efficientnet, but we quickly stumbled upon 0-gradient bugs in the int8 quantified version – which we assume is because of the model depth. Additionally the int8 quantization tuning step frequently took over 2 hours to finish on our consumer machine.

We spent 2 full working days trying to mitigate these issues but ended up pivoting again, but this time to `mobilenet_v2` as our base model.

Given the experimental nature of LiteRT and the lack of both community and documentation especially the full integer quantization was very tedious. Both the quantization of weights and the inputs and outputs resulted in many complications, mostly because the authors auf this library didn't implement more fine granular error messages, causing shotgun debugging to be the only viable option whenever issues emerged.

Methodology For the experiment we used the MobileNetV2 model as our base. Our methodology involved quantizing this model using LiteRT with three different configurations: float32, float16, and int8. These represent different levels of quantization, with int8 being the most aggressive in terms of reducing model size and potentially inference speed.

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

For the int8 quantization, we used a representative dataset of 100 samples from the COCO 2017 training set. This step is crucial for calibrating the quantization process, ensuring that the reduced precision can still accurately represent the distribution of activations in the model.

We then evaluated these quantized models on a subset of 1500 images from the COCO 2017 validation set. For each image, we preprocessed it to match the input requirements of our model, including resizing to 300x300 pixels and normalizing the pixel values.

One of the more complex parts of our implementation is the computation of precision. We implemented a function to calculate the Intersection over Union (IoU) between predicted and ground truth bounding boxes. This IoU is then used to determine true positives and false positives, which are essential for computing precision. We set a very low IoU threshold of 0.01 to be more lenient in our evaluation, considering the potential loss in accuracy due to quantization.

We also implemented a normalization function for bounding boxes, ensuring that all coordinates are within the [0, 1] range. This normalization is important for consistent evaluation across different image sizes.

For each quantization configuration, we ran inference on our test set and measured both the inference time and precision. The inference time gives us an indication of the model's speed, while precision tells us about its accuracy in detecting objects.

One of the challenges we faced was dealing with the different output formats of the quantized models, particularly for the int8 model. We implemented a dequantization step to convert the int8 outputs back to floating-point values for consistent evaluation across all configurations.

We also implemented a filtering step based on a confidence threshold, allowing us to adjust the trade-off between precision and recall. However, for this experiment, we set the confidence threshold to 0.0, effectively considering all detections.

The results of our experiments are saved in a CSV file, allowing for easy analysis and comparison of the different quantization configurations. This approach enables us to assess the impact of quantization on both model size and performance, providing valuable insights into the trade-offs involved in model optimization for edge devices.

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.

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.

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 almost halved. This is 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 derived 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 and in the 1e-2 range, as visualized in the logarithmic boxplot of all inference times. The increase from a float16 to a fully integer quantized model however is substantial in the given tolerance range.

The IoU@0.01 precision however does show fluctuations as visualized in the scatterplot and boxplots.

However surprisingly, as we increase the model size and operation granularity the accuracy seems to worsen.

Given the counter intuitive properties we observed in the benchmarks there is a realistic chance that the IoU implementation is incorrect. However the same algorithm was also used for our PyTorch benchmark and lead to sane results. We were unable to find the cause of this disparity.

In conclusion: The results are inconclusive. Further inspection to is necessary to validate our findings. If these findings were to be correct, the int8 quantized model would be the best choice for any deployment scenario as it is the most efficient in terms of memory, inference speed and precision.

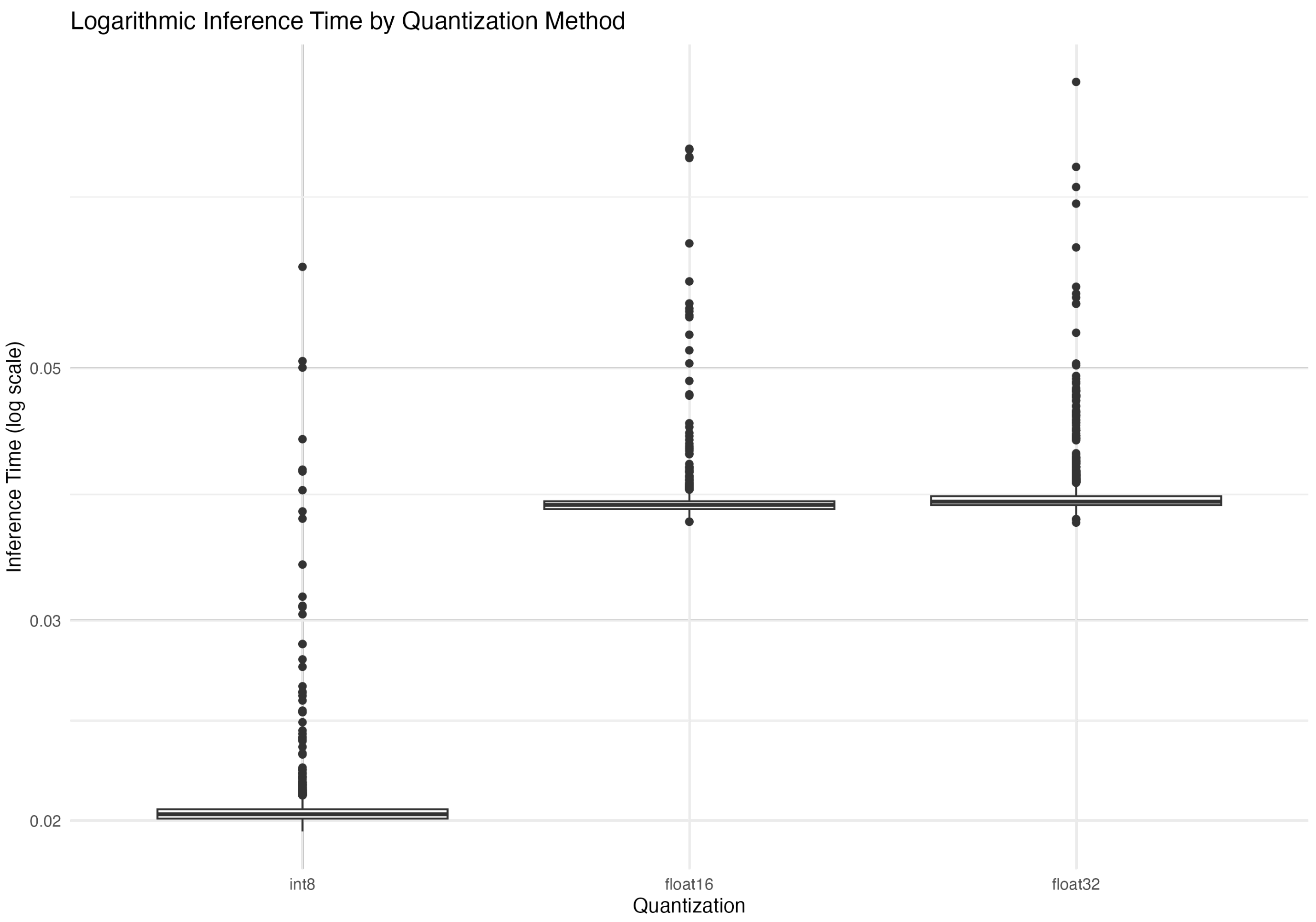

Figure 1: ODM: Logarithmic Inference Time box plot

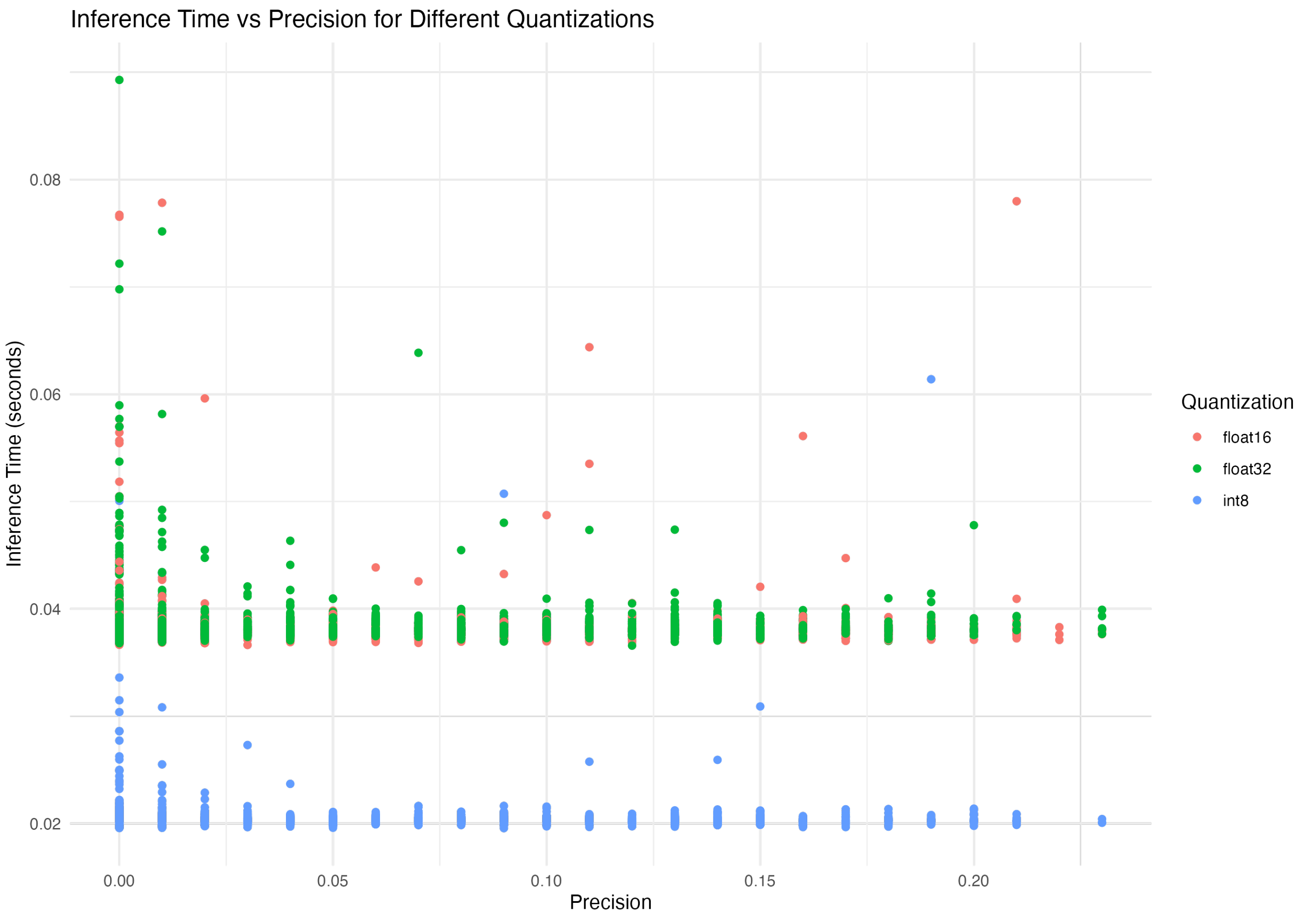

Figure 2: ODM: Inference Time vs. IoU Precision scatter plot

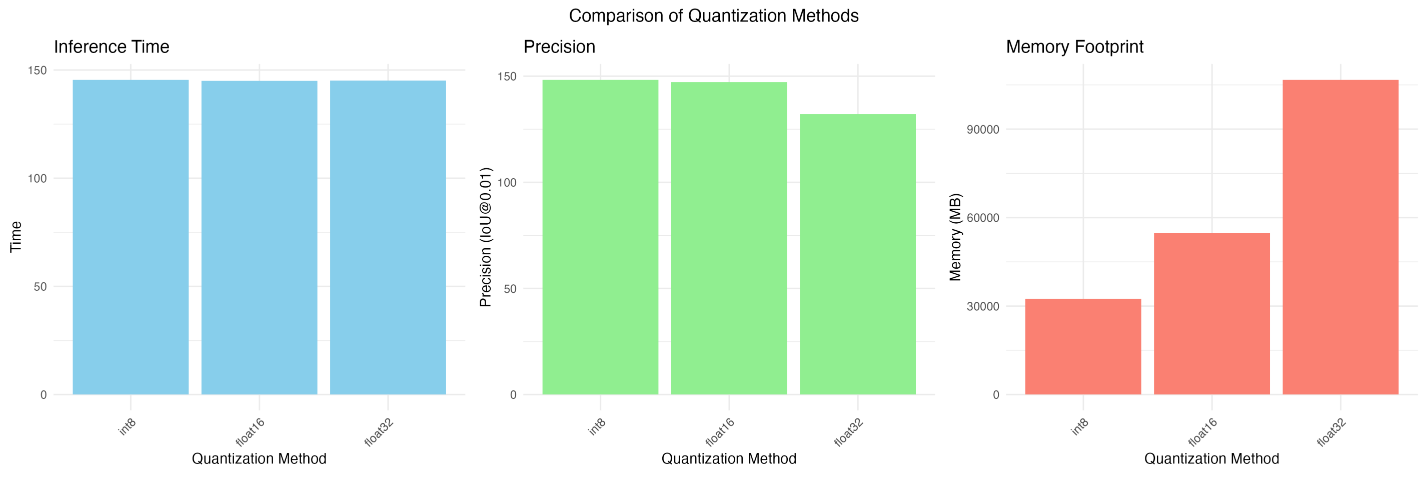

Figure 3: ODM: Inference Time vs. IoU Precision Bar plots

LLM Quantization

In the second part of our project, we focused on quantizing a large language model (LLM). We selected the SmolLM-135M model, a smaller variant of LLMs that is more suitable for edge devices and resource-constrained environments.

Methodology Our methodology involved quantizing this model using AutoGPTQ with three different configurations: int8, int4, and int2, representing increasingly aggressive levels of quantization. For the quantization process, we utilized the GPTQ (Generative Pre-trained Transformer Quantization) method, which is implemented in the AutoGPTQ library. We configured the quantization process with a group size of 64 and used a subset of 100 samples from the LAMBADA dataset as a representative dataset for calibration. This step is crucial for ensuring that the quantized model can accurately represent the distribution of activations in the original model.

We evaluated the quantized models on the LAMBADA dataset, which is specifically designed to test the ability of language models to understand and generate coherent text. For our experiments, we used a sample size of 5000 examples from the test split of the dataset. The LAMBADA task involves predicting the last word of a given context, which aligns well with our evaluation metrics. Our evaluation pipeline involved several key steps. First, we loaded the quantized model and tokenizer for each bit configuration (8, 4, and 2 bits). We then iterated through the dataset, processing each example by separating the context (all words except the last) and the target (the last word). We tokenized the context and passed it through the model to generate predictions.

We measured two primary accuracy metrics: Top-1 accuracy and Top-5 accuracy. Top-1 accuracy represents the proportion of examples where the model's highest probability prediction matches the target word. Top-5 accuracy measures the proportion of examples where the target word is among the model's top 5 predictions. These metrics provide insights into the model's precision at different levels of strictness. To assess the computational efficiency of the quantized models, we measured the inference speed in tokens per second. We calculated this by dividing the total number of tokens processed (including both input tokens and the single generated token for each example) by the total elapsed time for the evaluation. We also tracked the memory footprint of each quantized model using the `get_memory_footprint()` method, which provides an estimate of the model's size in memory. This metric is crucial for understanding the trade-off between model size and performance, especially for deployment on edge devices with limited resources.

All experiments were conducted on a CUDA-enabled NVIDIA GeForce RTX 3090 using a Red Hat Linux Distribution to ensure consistent and efficient processing. We implemented safeguards to clear GPU memory and collect garbage between different quantization configurations to prevent memory-related issues and ensure fair comparisons.

Results Starting with accuracy, we observe a clear trend of improved performance as we increase the number of bits used for quantization. The 2-bit model shows the lowest accuracy, with a top-1 accuracy of 0.18% and a top-5 accuracy of 0.82%. There is a significant jump in performance when moving to 4-bit quantization, with top-1 accuracy increasing to 5.24% and top-5 accuracy to 8.88%. The 8-bit model performs slightly better, achieving 5.48% top-1 accuracy and 9.8% top-5 accuracy. This trend aligns with our expectations, as higher bit precision generally allows for more accurate representation of the model's weights and activations.

Interestingly, the inference speed, measured in tokens per second, remains relatively consistent across all three quantization levels. The 4-bit model shows the highest throughput at 1422.68 tokens per second, followed closely by the 2-bit model at 1422.49 tokens per second. The 8-bit model is slightly slower at 1414.73 tokens per second. These minor differences suggest that the quantization level has a minimal impact on inference speed for this particular model and hardware configuration.

The memory footprint of the model, as expected, increases with the number of bits used for quantization. The 2-bit model has the smallest footprint at 87.57 MB, the 4-bit model occupies 114.53 MB, and the 8-bit model requires 168.44 MB. This linear increase in memory usage is consistent with the theoretical expectations of quantization, where doubling the number of bits roughly doubles the memory requirements.

It's worth noting that all models processed the same number of samples (5000) and tokens (416,476) during the evaluation, ensuring a fair comparison across different quantization levels. The elapsed time for each evaluation was also similar, ranging from 292.74 to 294.39 seconds, which corroborates the consistency in inference speed.

The trade-off between model size and accuracy is evident in these results. While the 2-bit model offers the smallest memory footprint, its accuracy is significantly lower than the 4-bit and 8-bit versions. The jump from 2-bit to 4-bit quantization provides a substantial boost in accuracy with a relatively modest increase in memory usage. The improvement from 4-bit to 8-bit is less pronounced in terms of accuracy but comes with a more significant increase in memory requirements.

These findings suggest that for deployment scenarios where memory is extremely constrained, the 4-bit model might offer the best balance between accuracy and resource usage. It provides a significant accuracy improvement over the 2-bit model while still maintaining a relatively small memory footprint. The 8-bit model, while offering the highest accuracy, may be more suitable for scenarios where memory constraints are less severe and maximum accuracy is prioritized.

It's important to note that while the accuracy figures may seem low in absolute terms, they should be considered in the context of the LAMBADA dataset, which is known to be challenging even for larger language models. The task of predicting the last word given a context is particularly difficult and often requires a deep understanding of long-range dependencies and complex semantics.

In conclusion, our quantization experiments demonstrate the viability of running compressed versions of the SmolLM-135M model on resource-constrained devices. The results provide valuable insights into the trade-offs between model size, accuracy, and inference speed, which can guide decisions in deploying language models for various applications, especially in edge computing scenarios where energy efficiency and reduced computational requirements are crucial.

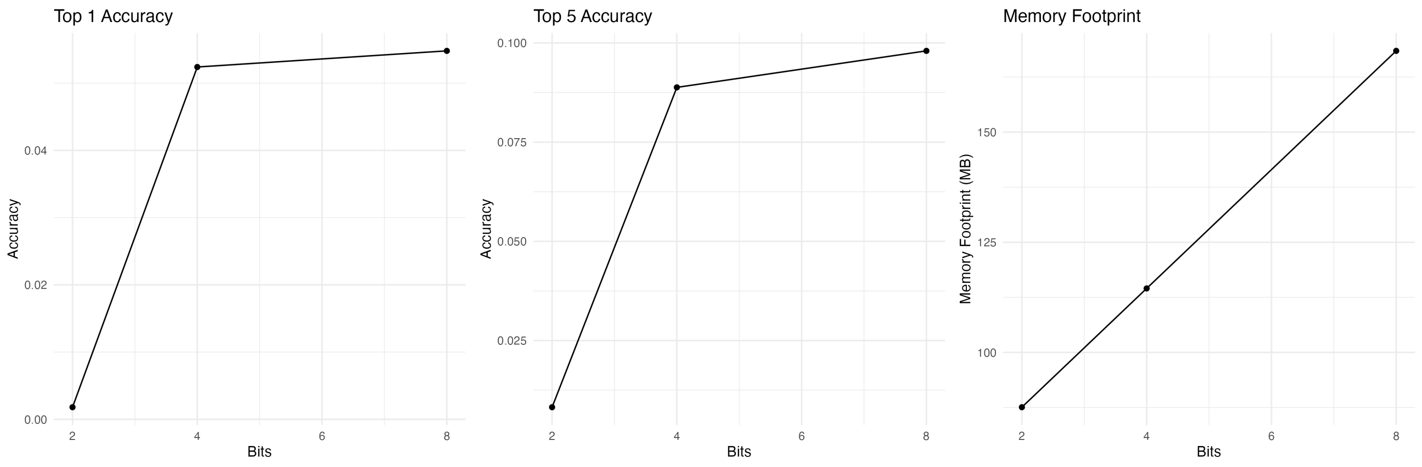

Figure 4: Accuracy vs. Memory Footprint

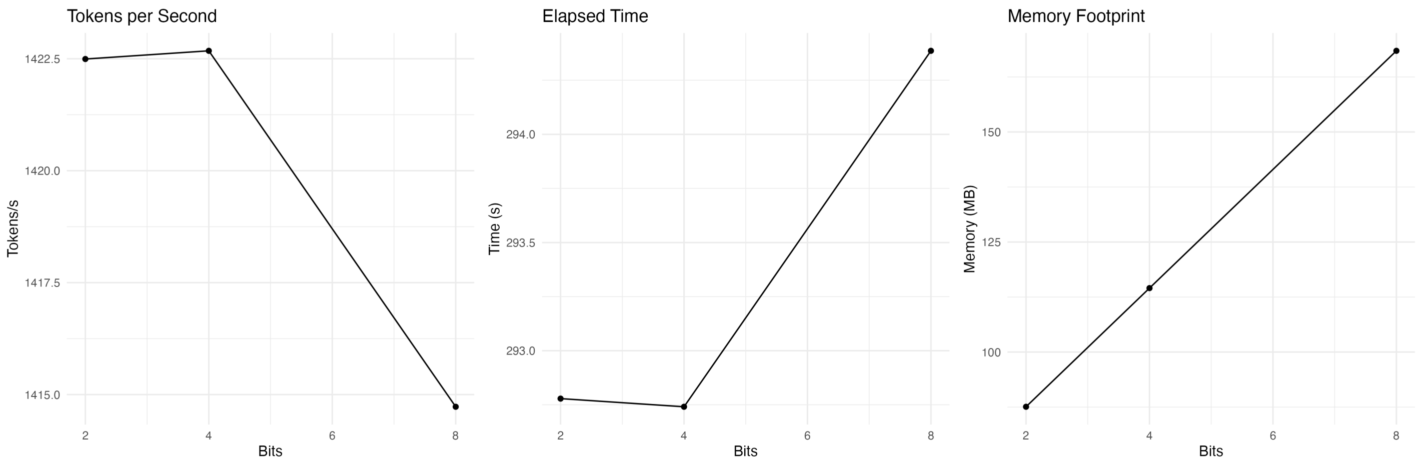

Figure 5: Inference Speed vs. Memory Footprint

Final Thoughts

The experiments we conducted are not representative of data-driven performance analytics research because they fall into several common pitfalls highlighted in the article "Always Measure One Level Deeper"[3]. Firstly, the experiments seem to rely heavily on superficial measurements, such as overall model size and basic precision metrics, without getting into deeper system behaviors or the underlying factors affecting performance. This superficiality can lead to incomplete and potentially misleading conclusions about system performance.

Moreover, there appears to be a confirmation bias in the methodology. The experiments focus on quantization techniques that reduce model size and memory footprint but do not sufficiently explore how these changes impact other critical aspects of performance, such as computational efficiency or energy consumption. This selective reporting can skew results to favor certain outcomes without a comprehensive analysis of all potential impacts. Additionally, the experiments might suffer from hasty execution, as indicated by the reliance on a small set of configurations and datasets without extensive validation or exploration of alternative scenarios. This haste can result in overlooking important system behaviors or potential improvement. Lastly, there's a lack of deeper measurement to validate top-level findings, such as understanding why certain quantization configurations lead to specific performance outcomes or identifying bottlenecks within the system.

In conclusion this project is a good starting point for further research into quantization techniques for ODMs and LLMs. However, to draw more robust conclusions and make informed decisions about model optimization, future work should address the limitations highlighted in this analysis.

[3]Ousterhout, J. (2018). Always measure one level deeper. Communications of the ACM, 61(7), 74-83.