AI in Climate Change — Assignment 1

Anna Stonek, Anton Lechuga, Yeongshin Park

November 2024

Contents

1 Introduction

In this report, the experiments of quantizing two different types of machine learning models, namely an object detection model and a large language model, will be described and analyzed.

The first section provides background information on object detection models and large language models in general and their respective applications.

In the second section, challenges regarding the deployment of large machine learning models will be covered, as well as why quantization is needed as a way of addressing the before-mentioned challenges.

In the third section, goal of the conducted experiments and the setup and approach to the experiments will be described.

In the fourth section, the results of the conducted experiments will be analyzed and discussed. Firstly, the results of the experiments with the object detection model regarding different configurations will be analyzed. The focus will be on the accuracy of the model, the inference time and the memory consumption. In the second part, the results of the experiments with the large language model will be analyzed. The focus of the analysis is on model accuracy, speed and memory consumption.

And lastly, conclusions will be drawn from the results of the experiments, and a prospect into the future of deploying large machine learning models.

1.1 About Object Detection Models

Object detection models are machine learning models that aim to detect objects in images. Usually, the objects are marked with a rectangle around them and the predicted label of the object is displayed as well as a confidence score, with which the model predicts the label.

Object detection is seen as a natural extension of object classification and it is classified as a supervised learning problem. This means, that the model learns predefined class-labels with the help of labelled training data. Though there have been many advances in the field of computer vision, there still remain some challenges in object detection models, for example the intra-class variation. Object detection is a trivial task for humans, but not for machines, and one reason for that is that within a class of objects, different instances can vary significantly. Objects could for example be rotated, scaled or blurry, which makes object detection significantly harder for the machine. [1]

Another challenge in object detection is the number of classes, which just seems endless. This requires a huge amount of training data and therefore con-

sumes a lot of energy. It also requires high-quality training data, which takes a lot of effort and expenses to create.[1]

Moreover, another challenge in object detection is the efficiency problem. Present day models require huge computational resources to get a high accuracy in predictions. Mobile and Edge devices are more and more incorporated and used in the field of object detection.[1]

Since object detection models are very versatile, possible applications include, but are not limited to self-driving cars, facial recognition, surveillance and video analysis.

1.2 About LLMs

Large Language Models are a new type of model in the field of natural language processing. A large language model is not defined by a specific architecture, but the definition rather lies in the capacity of the model to understand and generate human language on a level that was unreachable before.[2]

The first approach to using a transformer based architecture was "Attention is all you need" by Vaswani et al.[3], which triggered a big wave of research and development in the field of large language models. The architecture is still used in state-of-the-art models nowadays like BERT and GPT and is getting improved constantly, leading to new standards constantly.[2]

The most commonly used architecture is the encoder-decoder architecture as found in the BERT models. In this architecture, the input is first processed by the encoder, which uses self-attention to understand the context as a whole. This interims output is then emitted to the decoder, which generates output sequentially. This architecture can be used for translation tasks for example.

Next to the encoder-decoder architecture, the decoder can also be used separately as a stand-alone model. Architectures with a decoder only are also called causal decoder architectures and are commonly used for generative tasks like ChatBots, to ensure a coherent output of the model. Another architecture using decoders only is the non-causal decoder which offers a more flexible approach to attention. This model is not strictly limited to past information, but can include up to a certain point future context into the prediction.

LLMs can be applied in a variety of tasks like text generation, translation, summarization and automated customer service (e.g. a ChatBot). This is a non-exhaustive list and as research proceeds to continue, new ways of applying LLMs are found constantly.[2]

However, there are nevertheless limitations to LLMs, some of which will be mentioned here. For example, LLMs require a huge amount of training data,

which tends to be not domain-specific and rather general. This can lead to two problems: For one, the training data could contain known or unknown biases that influence the model, which could raise ethical concerns. For the other part, the performance of the LLM can decrease significantly on tasks that require domain-specific knowledge which is not included in the training data.

2 Background

In this section, background information on the challenges of deploying deep learning models like object detection models and large language models is discussed, which leads to why quantization is needed as an optimization technique. Lastly, an overview of quantization techniques is given.

2.1 Why Quantization?

With the swift growing of models in size, jumping from a couple of hundred parameters per model to billions of parameters, training times and also inference times of such models have increased hugely. Additionally, these models require enormous amounts of computational resources, which in turn consumes energy.

However, this energy necessary to train, re-train and run the models does not come from thin air, but it has to be produced and distributed. Energy is not a inexhaustible resource, and producing enough energy to run big present day models is becoming a challenge, that data scientists worry more and more about.

One part of the solution of the challenge of producing enough energy for the models is to reduce the models in size, so that they require less space and less training and inference time. The challenge here is to keep the model's accuracy on the same level or rather not letting in decrease significantly.

In this situation, quantization is an optimization technique that is becoming more popular. It addresses the issue decreasing the size of the model by reducing the size (and therefore the precision) of the model's parameters. It could be shown that with quantization, the training and inference times could be reduced significantly, while still keeping the model's prediction accuracy.

2.2 Challenges in ODMs and LLMs

The technical challenges of deploying large-scale machine learning models, like for example object detection models or large language models are manifold. Under a technical aspect, a big problem of these large models is the resource intensity and optimization problems. The bigger the model, the more resources like energy and water it consumes and this energy needs to be produced and distributed. [4]

Another challenge that one needs to deal with when deploying large scale model is the efficient management of memory and parallelization. The idea is here to distribute the computational load of a single model across multiple GPUs.[4]

And finally, even though there are many more, the last challenge mentioned here in this report is privacy preserving data publishing and model deploying. Specifically LLMs trained on huge datasets might inadvertently learn sensitive information, posing risks of data breaches. Additionally, it is important to ensure the compliance of the model with data protection regulations and privacy laws.[4]

2.3 Quantization Techniques

Quantization is the process of reducing the number of bits used to represent data, which can be done by rounding or truncating the data to a lower precision.[5]

One technique is to quantize the data and then convert it to a smaller configuration regarding bit-size. This can be done by training a model to learn the distribution of the data and how to scale it to a uniform distribution. Also, the data's quantiles are computed and used to quantize the data. Lastly, the quantized data is cast to the float32 and int32 data types. [5]

Another quantization technique is rounding to a smaller number of decimal places to reduce the amount of bits needed to represent data. This can be achieved by applying the Numpy.round function, which can round the data for example to 4 decimal places. The data is then represented with 12 bits of precision, instead of 64 bits of precision. [5]

3 Experiments

In this section, the setup for the experiments conducted with the object detection model and the large language model is described.

3.1 Object Detection

The object detection model was run in two different environments to check whether the results would differ. The first environment is the Google Colab platform using the T4 GPU offered by Google Colab. Additionally, per Google Colab Default the GPU acceleration was activated. Moreover, Python 3.10.12 and TensorFlow 2.18.0 was used in the configuration of Google Colab.

The second environment is a MacBook Pro 14 with an M3 Chip and macOS as the operating system. In this environment, Python 3.12.4 and TensorFlow

Table 1: Object Detection Experimental Setup

Category	Platform	Details
Environment	Google Colab	Linux-based(Google Colab default)
		Python 3.10.12
		TensorFlow 2.18.0
		T4 GPU
		GPU acceleration (Google Colab default)
	MacBook Pro 14	macOS
		Python 3.12.4
		TensorFlow 2.18.0
		Apple M3 Chip
		Optimized for M3 GPU and Neural Engine
Dataset and Model	Dataset	COCO Dataset 2017
	Model	Mask R-CNN with Inception ResNet V2 backbone

2.18.0 was used to conduct the experiments.

The model used in the experiments is the Mask R-CNN with an Inception ResNet V2 backbone and it was trained on the COCO dataset 2017. The pre-processing steps applied to the data include image decoding into an RGB tensor and resizing the image to 640x640 pixels, ensuring the input tensor is a tf.uint8 tensor with a shape of [1, height, width, 3] and values in the range [0, 255].

The pre-trained R-CNN-based model was converted into three formats: float32, float16, and int8. The original model is based on float32 operations, so the float32 version is simply the model converted to TensorFlow Lite (TFLite) format without any changes to its internal computations. The float16 and int8 models were created by performing weight-only quantization of the model.
The model and its quantized counterparts were evaluated on the bases of three metrics:

1. Avg Inference Time (seconds): The average time taken by the model to perform predictions was measured.
2. Model Size (MB): The size of each model expressed in megabytes (MB).
3. Accuracy (mAP): The mAP was calculated using COCO's Average Precision (AP) metric. AP is obtained by plotting a Precision-Recall curve and averaging the Precision values across different recall levels. The mAP is the average of the Average Precision values across all classes.

3.2 Text Generation

For the given task, we for the `Open Pre-trained Transformer Language Model` (OPT) [6] with 350 million parameters as a base model. The choice was made

since experiments were conducted with limited resources on Google Collab, see table 1 for the specific configuration.

We quantized the model with the `GPTQ`-approach [7] using the `AutoGPTQ` library. As the library requires the presence of a GPU, experiments were limited to the Google cloud platform for this task. The quantization process was based on the default dataset specified in the respective paper. All experiments were conducted using the `Pytorch` framework.

The model and its quantized counterparts were evaluated on the bases of three metrics:

1. Memory: Total memory size of all model parameters
2. Speed: The number of tokens per second the model was able to generate using the `mode.generate()` function. To ensure stable results, we measured the speed a total of $n = 50$ times after warm up and report the average. Note that due to unstable results, we compare this to the total time, the model took for evaluation using the `model.foward()` function.
3. Accuracy: The accuracy was measured using the top-k accuracy fixing $k = 7$. The lambada dataset [8] used for evaluation initially required to predict the last word of a sequence of sentences. Due to the fact that the OPT-model operates on tokens smaller than the word dimensions and results were already moderate on a token level, we decided to frame the task to predict the last token given the entire context before.

To deal with the limitations of using free GPU resources in google colab, we evaluated the model on batched inputs with a batch size of 64 and limited the dataset to the test set. The goal of the experiments will be to assess the effectiveness of current quantization techniques in standard `LLM` workflows.

4 Results

In this section, the results of the experiments will be described and discussed. The first part of the section will focus on the results for the object detection model, whereas the second part of the section will focus on the results for the large language model.

4.1 Object Detection

Table 2: Object Detection Quantization Results

	Average Inference Time (s)	Model Size (mb)	Accuracy (mAP)
Base Model	35.630931	277.3	0.000425
float32	38.653304	249.55072	0.000413
float16	39.742096	250.296082	0.000413
int8	10.808577	67.99968	0.000421

If we focus on the average inference time, we can see that it decreases significantly with smaller bit-representation of the data. This seems trivial, since the smaller the data, the faster the computation. Interestingly, the average inference time for the float32 configuration is even higher than the one of the base model. Furthermore, we can see that the biggest step in decreasing the average inference time is between configurations float16 and int8. This could be due to the fact that the integer configuration takes up even less space than the float configuration.

If we look at the model size, we can see that it is nearly the same for both the float32 and the float16 configurations. So in terms of model size, we can see that there is no gain in using the float16 configuration over the float32 configuration. However, where we can see the biggest step again, is the int8 configuration, where the model size drops from 250MB to only 67MB.

And finally, if we look at the accuracy, we can immediately see that it is very low in general. Unfortunately, it was not possible nor the objective of this exercise to improve the accuracy of the models. So we can observe that the accuracy is equally low for all 3 different quantization configurations including the baseline model.

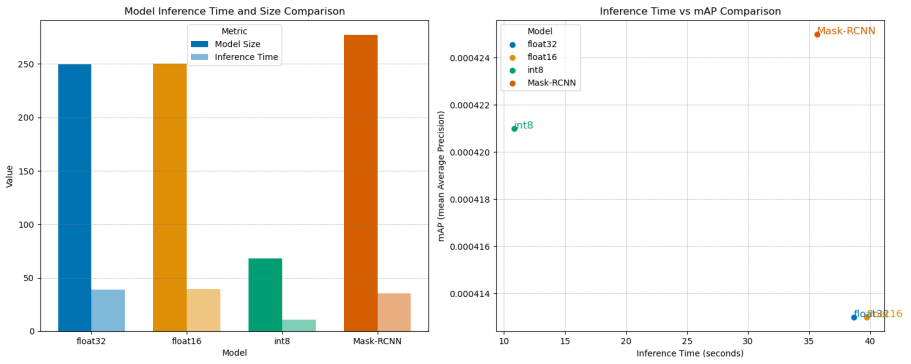

Figure 1: Quantizing an object detection model makes it faster and lighter, but it may result in accuracy loss, especially with the Int8 model.

(a) Float32 Results (b) Float 16 Results (c) Int8 Results

Figure 2: Comparison of model detection results across three different Quantization

4.2 Text Generation

Table 3: LLM Quantization Results Overview

	Memory (mb)	Speed (token/s)	Total Evaluation Time (s)	Accuracy (top 7)
Base	1263.414	51.124	219.946	0.177
8 bits	351.301	20.222	57.331	0.007
4 bits	206.176	47.169	53.112	0.006
2 bits	133.613	33.423	55.381	0.000

If we look at the top-7-accuracy of the model, we can also see, like in the object detection model, the accuracy is very low in general. As already mentioned above, unfortunately it was not possible to solve this problem, as it was not the task to improve the accuracy of the model. But with the LLM, we can definitely see a trend for the accuracy to go down, the smaller the quantization configuration is. For example, the baseline model has an accuracy of 0.177, whereas the 4-bit configuration has an accuracy of 0.006. We can hence observe a clear trade-off between model size and model accuracy.

Regarding the speed of the model, measured in tokens per second, we can see that there are mixed results when applying quantization techniques. When the baseline model has a mean speed of 51 tokens per second, the smallest configuration with 2 bits has a speed of only 33 tokens per second. However, the model with the 4-bit-configuration has a speed of 47 tokens per second, which is almost as fast as the baseline model. Compared to the much faster evaluation times of the quantized models, as further shown in Figure 3, it seems that the speed is very dependent on the support of quantization for different functionalities. In the case of Pytorch, the `generate()` function used to autoregressive generate new tokens still lacks support and possibly need to convert the smaller weights to a supported size. In contrast, the total time needed to evaluate the

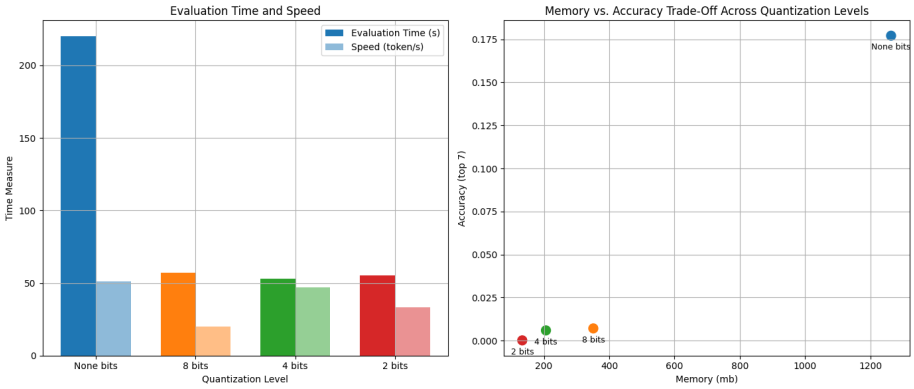

Figure 3: LLM Quantization lead to smaller models with less accuracy. The speed depended greatly on the support for quantized models.

models drastically shrinks when using the `foward()` method, which computes the tokens in parallel. We opted for leaving the measures as they are, to highlight the key differences in speed.

5 Conclusion

To conclude, it can be said that big machine learning models in general and object detection models and large language models specifically have made a shift in modern technology, enabling performance and efficiency at a scale unattained before.

However, these large models also come with certain challenges, since they consume huge amounts of resources in terms of energy and water, for example for cooling the data centers, where the models are deployed. Since the models are constantly being developed and are getting bigger and bigger, resources though are limited at a certain point, new techniques and strategies have to be found to improve efficiency and reduce the resource-intensity of models.

One technique to enhance efficiency of large machine learning models is quantization, where the models are configured to reduce the size of the bit-representation of the data.

In two experiments with one object detection model and one large language model it could be shown that for the object detection model, the inference time significantly decreases with smaller configurations, as well as the model size. The accuracy was equally low for all configurations.

In the experiment with the large language model, it could be shown that the model size and the evaluation speed significantly decreases with the reduction of the configuration size, however, also the accuracy decreases with smaller configuration sizes. So what we can conclude here, is that quantization techniques always bring a trade-off with them, between the model accuracy and reduction of e.g. model size and evaluation speed.

Quantization is a promising technique to improve a model's efficiency and contribute to the sustainable usage of resources like energy and water. However, applying quantization techniques always brings the trade-off between inference time speedup as well as reduction of the model size and the reduction of model performance. So this certainly introduces an interesting approach for further research.

References

[1] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital signal processing*, vol. 126, p. 103514, 2022.

[2] T. Oroz, *Comparative Analysis of Retrieval Augmented Generator and Traditional Large Language Models*, Wien, 2024.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[4] A. Menshawy, Z. Nawaz, and M. Fahmy, "Navigating challenges and technical debt in large language models deployment," in *Proceedings of the 4th Workshop on Machine Learning and Systems*. New York, NY, USA: ACM, 2024, pp. 192–199.

[5] M. Goswami, S. Mohanty, and P. K. Pattnaik, "Optimization of machine learning models through quantization and data bit reduction in healthcare datasets," *Franklin Open*, vol. 8, p. 100136, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2773186324000665

[6] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022.

[7] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2023. [Online]. Available: https://arxiv.org/abs/2210.17323

[8] D. Paperno, G. Kruszewski, A. Lazaridou, N. Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernandez, "The LAMBADA dataset: Word prediction requiring a broad discourse context," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, August 2016, pp. 1525–1534. [Online]. Available: http://www.aclweb.org/anthology/P16-1144