Quantization of ML Models

Noé CARINGI
INSA LYON - TU Wien
e12402501@student.tuwien.ac.at

Lucas GIRARDET
INSA LYON - TU Wien
e12404287@student.tuwien.ac.at

Table of contents

Introduction

AI is part of a major industrial revolution and will profoundly transform society over the coming decades. This technology promises major economic and societal benefits. But this digital transition is taking place at the same time as the advent of environmental issues linked to the climate crisis.

According to a projection by the International Energy Agency, the global data center and AI industries are set to double their electricity consumption by 2026, generating a surplus of 37 billion tonnes of CO2 in the atmosphere. As the use of machine learning explodes in all sectors of society and in all fields, it is vital to find solutions to make these technologies less energy and resource-intensive, and more sustainable.

One such solution is the quantification of ML models, making them less energy-intensive, albeit with slightly less precision or less relevant results.

The aim of this report is to understand the benefits of quantifying Machine Learning models to make them more efficient. We will first try to define all the notions and challenges that are useful for tackling this subject, before presenting our experiments and results concerning the quantization of an ODM model in the first part, followed by the quantization of an LLM model in the second part.

The goal of these experiments, then, is to apply the concepts of quantization to different pre-trained models, and to compare the impact of quantization on the efficiency of the models in their respective tasks, as well as on the demand for resources, be they energy, computational or temporal, using different metrics that we'll describe in detail later.

Differents ML models

Object Detection Models (ODM) :

Definition and use cases :

Object detection is a technique that uses neural networks to localize and classify objects in images. It is an instance of artificial intelligence that consists of training computers to recognize and classify objects according to categories. Object localization is a technique for determining the location of specific objects in an image by demarcating the object through a bounding box. Object classification is another technique that determines to which category a detected object belongs. The object detection task combines subtasks of object localization and classification to estimate the location and type of object at the same time. This computer vision task has a wide range of applications, from autonomous driving to medical imaging or security. Self-driving cars widely adopt object detection to recognize objects such as cars and pedestrians. Video surveillance may employ real-time object detection to track crime-associated objects, such as guns or knives in security camera footage. By detecting such objects, security systems can further predict and prevent crime.

How it works :

There are several approaches to object detection. The first methods are based on hand-crafted features and more recent methods are based on deep learning. Older methods involve extracting a set of hand-created features from the image, such as color, texture and shape, and using these to train a classifier to identify objects. More recent methods based on deep learning involve training a convolutional neural network (CNN) to learn features and detect objects. In these techniques, a CNN is often trained to identify and locate objects in an image or video. Single Shot Detector (SSD), You Only Look Once (YOLO) and fully convolutional region-based networks (R-FCN) are examples of object detection models.

Limits and bias :

Unbalanced datasets are a problem for object detection tasks, as negative samples (i.e. images without the object of interest) far outnumber positive samples in many domain-specific datasets. This is a particular problem for medical images, where it is difficult to obtain positive samples of disease.

Initially, object detection models focused largely on 2D images. Today, researchers have turned their attention to object detection applications for 3D images and videos. Motion blur and camera focus changes pose problems for object identification in video images. Researchers have explored lots of methods and architectures to help track objects from one frame to the next despite these conditions, such as recurrent neural network architecture with long-term memory (LSTM) and transformer-based models.

Large Language models (LLM) :

Definition and use cases :

LLMs are deep neural networks trained on large quantities of unlabeled text using self-supervised learning. They are mainly used for conversational agent implementation and excel at a wide range of tasks. Instead of being trained for a specific task, they are trained to predict a likely outcome to a given input. The quality of their output appears to be a function of the quantity of resources (parameter size, computing power, data) and the quality of the data supplied to them.

LLMs can be trained to solve a wide range of tasks, from generating text or responding to prompts, with ChatGPT being a well-known example, to recognizing, understanding and classifying various texts.

How it works :

The goal of a LLM is to discover patterns in data. Or more specifically, a pattern that describes the relationship between an input (a sequence of words) and an outcome (the next word for this sequence). This is just basically a classification task with many classes as there are words. Then, based on the data it analyzes during the pre-training phase, the LLM assigns a probability to each word to be the next and chooses one of them.

Limits and bias :

LLMs sometimes generate false assertions that don't seem to be justified by their training data, so we call them “ hallucinations ”. Bias is also a concern, as any complex model created by humans can reflect the biases of the teams who prepare and design LLMs, and of the data scientists who train and implement the models.

Challenges and goals of quantization

How quantization works

Quantization is a technique used to reduce the precision of numerical values in a model. Instead of using high-precision data types, such as 32-bit floating-point numbers, quantization represents values using lower-precision data types, such as int8 integers.

Quantization techniques

There are two main types of quantification for ML models : Post Training Quantization (PTQ) and Quantization Aware Training (QAT). We'll explain each of them in more detail below. In this report we will only focus on the first one.

Post Training Quantization :

In this method, quantization is applied after the model has been trained to standard accuracy. The model is not re-trained after quantization, making this technique quick and easy to implement. It can be dynamic or static. In the first case, the scope of each activation is calculated on the fly at runtime. Although this gives excellent results without too much work, it can be a little slower than the static method. In the second case, the range for each activation is computed in advance at quantization-time, typically by passing representative data through the model and recording the activation values.

Quantization Aware Training :

This method includes quantization in the training process, simulating these effects through quantization nodes that mimic the behavior of quantized weights and activations, enabling the model to adjust its parameters to compensate for the loss of precision. Once training is complete, the model can be converted to a quantized version.

Why using quantization

This process, in addition to significantly reducing memory usage, can speed up model execution while maintaining acceptable accuracy and saving energy resources. In fact, by reducing the precision of weights and activations, each parameter occupies less memory, thus reducing the overall size of the

model. In addition, calculations with integers or half-precision values are faster, enabling more data to be processed in less time. Finally, with less precise calculations, modern processors consume less energy, making the impact of ML models more acceptable. The idea is to try to reduce model power consumption by lowering its precision, while retaining relevance to the task in hand.

Its use can therefore be very useful in a number of scenarios, for example for experimentation or dimensioning purposes when optimum precision is not required but an ML model needs to be used to test certain things. Another practical application is to enable the use of ML models in embedded systems, which consequently have fewer resources at their disposal. Finally, we can also imagine meeting the needs of real-time applications through the use of quantization, as these require a short response time to be relevant and don't always need to respond to complex tasks.

Experiments on quantization

As explained above, the aim of our experiments is to understand the impact of quantization, both on model performance and on resource consumption. We will therefore detail here how we proceeded to measure the differences brought about by quantization on our LLM and ODM models, as well as presenting and analyzing our results.

The ODM experiments were carried out on a PC with the following technical specifications :

GPU : NVIDIA Quadro P520
CPU : i7
RAM : 16Go

For the LLM experiments, we used a cloud-based Kaggle notebook, which is very convenient for using for free a :

GPU : NVIDIA Tesla P100 GPU.

However, on Kaggle, GPU environments have lower CPU and main memory, but are a great way to achieve significant speed-ups for certain types of work such as ours like quantizing ML models.

ODM quantization experiments:

Tools used :

- To carry out this project we used the **Centernet** ODM with a **ResNet-v2-50** backbone pre-trained on the **COCO 2017 dataset** with 512*512 images. We retrieved this pre-trained model using **Kaggle**. The Centernet model was developed in 2019. It is a One Stage model, which means that it performs detection in a single stage by detecting the center of objects, unlike a model like Faster R-CNN, which uses several stages to define the position of an object.

- To quantify the model we use **LiteRT**. LiteRT is Google's high-performance execution environment for AI. We use LiteRT to quantify our model and obtain 3 configurations in **float32**, **float16** and **int8**. In our project, using quantification with TFLite RT allows us to study the accuracy and inference time obtained for each quantification of the model. This will enable us to find out whether or not reducing the size of a model via quantization still results in adequate accuracy and computation time.

- Following quantization, we obtain 3 models in .tflite format. We then perform object detection inference using our quantised TensorFlow Lite models (float32, float16, int8). Our program loads the images from the COCO 2017 validation dataset (not the test dataset because there was no annotations available), pre-processes them to have an input size adequate to the requirements of the **.tflite models**, then passes them to each model to obtain detection results, which are then saved in a COCO-compatible format in .json files. Each file contains the identifiers of each image and for each of them there are the identifiers of the objects detected, the coordinates of the box where the object is located and the score associated with this object. The score means how reliable the model believes its prediction is. For each type of quantification, the total inference time is calculated and displayed, making it possible to compare performance in terms of speed between the different models.

Factors and metrics :

The three factors tested in the various quantified versions of our Centernet ODM are accuracy, speed and memory.

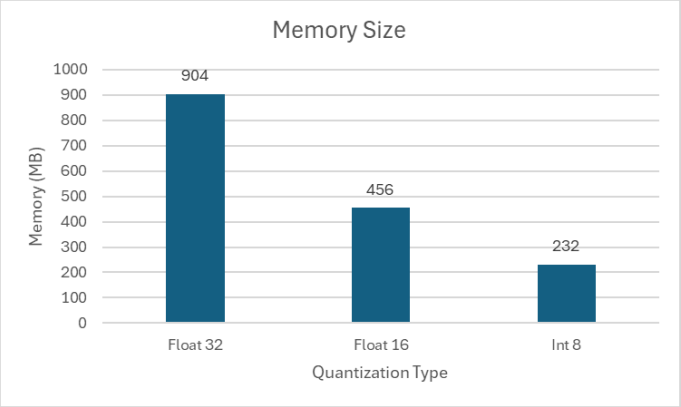
- **Accuracy** in object detection means how well the model can correctly identify and localize objects within images. Object detection involves both classifying the object and predicting its precise location using bounding boxes. The key metrics to measure accuracy in object detection are **Average Precision** (AP), and **Average Recall** (AR).

- **Speed** refers to how quickly the model processes an image and generates predictions, such as bounding boxes and class labels. The metric commonly used to measure this speed is the **inference time** per image, typically expressed in seconds per image (s/image).

- **Memory** factor is the space the model takes up on the machine running it. Here we measure the **size of the model in MB** to get an idea of the memory used to store our models.

Results and analysis :

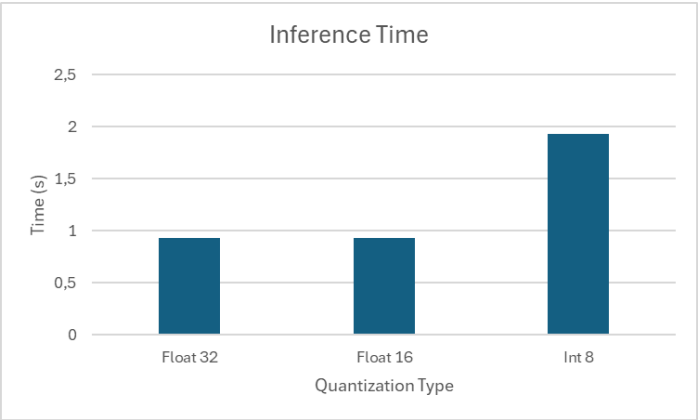
- **Memory :**



Float 32 is the most widely used configuration in Machine Learning and Deep Learning, offering the highest accuracy and therefore the largest model size. Post-training float16 quantification reduces TensorFlow Lite model sizes (up to 50%), while sacrificing very little precision.

Quantifying model constants (such as weights and bias values) from total floating point accuracy (32-bit) to a reduced precision floating point data type int8 quantization compared to typical FP32 models is allowing for a 4 times reduction in the model size and a 4 times reduction in memory bandwidth requirements. This quantization involves mapping floating-point numbers (typically in float32) to an int8 integer representation.

- **Speed :**



To measure inference time, we used python's time module. We measure the inference time for each quantization of the model. Our results above highlight an equivalence in inference time between float32 and float16 quantization.. The float16 approach does not reduce the latency as much as other techniques though, also this is not optimized for CPU based inference. The inference time for the Int8 quantization is more than double that of the other two quantizations.

However, we would expect it to decrease as the model is supposed to be lighter. This contradiction is due to the fact that we are using dynamic int8 quantization. This means that the weights are saved in Int8 but the computations are performed in float. The conversion required at this stage explains the significant increase in inference time. Another source of error in our inference time readings comes from the fact that we use standard CPU timing and do not use synchronized timing. Asynchronous execution can therefore lead to

measurement errors. To obtain a more reliable average inference time and with more efficient equipment we could have performed more iterations and check our results.

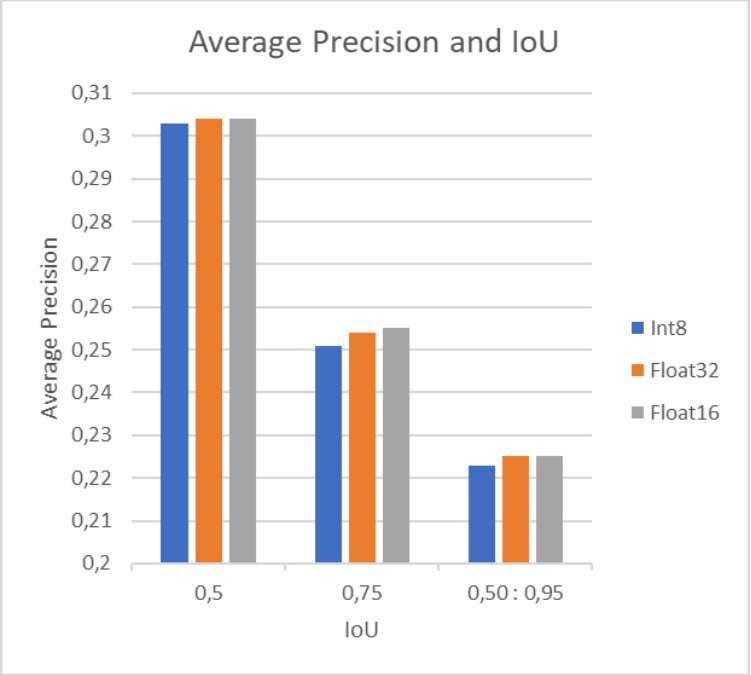
- **Accuracy :**

By reducing the precision of the calculations, these quantizations make it possible to reduce the size of the model and speed up inference times, but can potentially affect the accuracy of the model. This analysis aims to understand how each level of quantification influences the performance of the object detection model.

Average Precision

- **IoU parameter**

IoU thresholds are used to define a limit beyond which a detection is considered correct. For example, a threshold of 0.5 means that the detection is considered correct if the IoU between the predicted box and the actual box is greater than or equal to 0.5 (i.e. 50% overlap). The IoU is calculated by dividing the area of intersection of the two boxes by the area of their union.

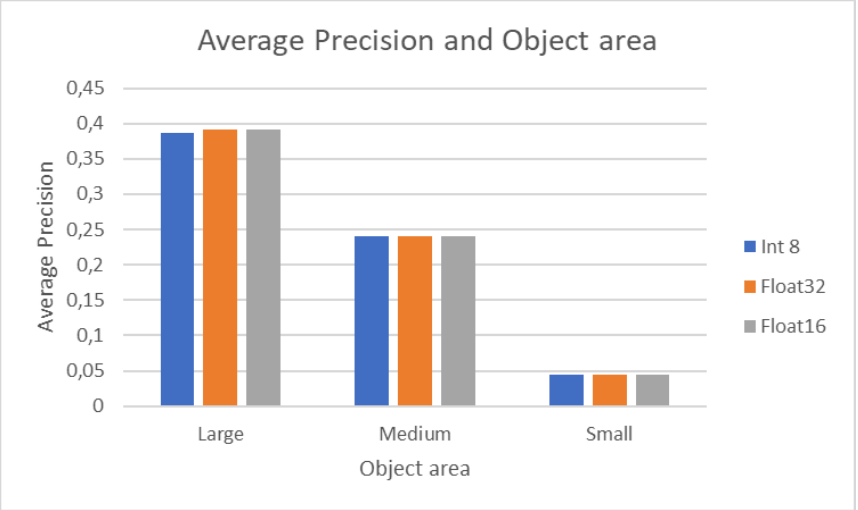


This graph shows the evolution of the accuracy of our different quantizations depending on the “Intersection over Union" parameter. In view of the definition of this parameter, it is consistent that the maximum accuracy values for every quantization are with IoU = 0.5 . It is also logical for the accuracy to be lower with IoU = 0.75, as it is more restrictive to consider a detection as correct. The last value of IoU, 0.50 : 0.95, means calculating the average precision for each value of IoU from 0.50 to 0.95 with a step of 0.05 and averaging these 10 values to obtain a mean of the average precision. This approach provides a more complete assessment of the model's performance, as it takes into account approximate detections (with a lower IoU, such as 0.50), and very accurate detections (with a

higher IoU, such as 0.95). Returning to the 3 types of quantization, we note that for each value of IoU the average accuracies of the three types are equivalent for float 32 and float 16 and very slightly lower for Int 8.

- **Area parameter**

Area is measured as the number of pixels in the segmentation mask. In COCO, there are more small objects than large objects. Specifically: approximately 41% of objects are small (area < 322), 34% are medium (322 < area < 962), and 24% are large (area > 962). We will therefore compare the accuracy of the different quantizations as a function of object size.



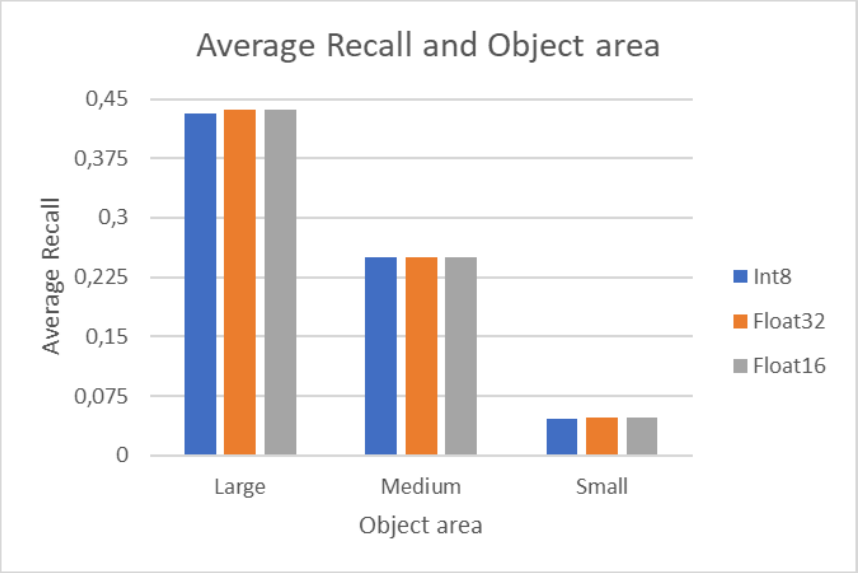
Logically, the results show that the smaller the object is, the harder it is for the model to detect it. An object considered small will only be detected in 0.05% of cases. This demonstrates some of the limitations of the model. Once again, we can see that the average precision values are equivalent for Float 32 and 16 and very slightly lower for Int 8.

Average Recall

Average Recall (AR), also known as sensitivity or true positive rate, measures the ratio between the number of true positives and the total number of positive samples. Recall focuses on the model's ability to correctly identify positive samples from the entire pool of positive instances.

- **Area parameter**

In the same way as for the average precision, we will evaluate the evolution of the average recall as a function of the size of the objects in the images for each quantization.



Despite the difference between large and small objects, the maximum recall remains limited to 0.45, even for large objects and float32 quantization. This suggests that the model generally has difficulty identifying all the objects present, regardless of their size and the quantization used. This result may indicate a limit in the model's capacity.

- **Conclusion :**

Having analyzed all these parameters and metrics from the experiments on our quantified ODM model, we can conclude that quantization is particularly interesting for this model. In fact, int8 quantization reduces the size of the model by a factor of 4 while maintaining accuracies very close to those of the float32 quantized model. If the fact that the dynamic int8 quantization of our model doubles the inference time becomes a problem in use, then float16 quantization would be a good compromise between accuracy, speed and model size.

LLM quantization experiments :

Tools used :

- The LLM model used to carry out our experiments is **Llama-3.2-3B**, an open source model developed by Meta and released in July 2023. As its name suggests, this is the third version of the model, with 3 billion parameters. We also tried to use the Llama-2-7B model, but it took up too much space on our system when we saved its quantised version.

- With regard to the dataset used to test the performance of our models, we used the **LAMBADA Dataset**, a database developed in 2016 by researchers at the University of Trento and University of Amsterdam from English-language novel extracts of all genres. It asks the AI to predict the last word of the last sentence of a literary excerpt and is designed so that a human can easily guess the last word as long as they have the context sentences provided previously. It is therefore a perfect test for evaluating text understanding through word prediction.

- To quantify our model, we used the **AutoGPTQ** tool, which allowed us to choose between different configurations: int8, int4 and int2. We also tried using **BitsAndBytes** however after many unsuccessful attempts due to conflict between libraries and with the impossibility of using a GPU or even quantifying in int2 we had to give up with this tool.

- As mentioned earlier, we decided to turn to **Kaggle** so that we could use a powerful GPU to run our quantisation. However, this came at the expense of system memory, so we had to make a concession by switching to a lighter version of LLama.

- We used the Franco-American **HuggingFace** platform to easily download the models, training data and quantization libraries used here.

Factors and metrics :

The three factors tested in the various quantified versions of our LLM Llama are accuracy, speed and memory.

- **Accuracy** is a factor that evaluates the degree of congruence of the model's results with the correct answers or expected results. The metric used here to measure it is **Top-k Accuracy**. This metric examines the k-words with the highest prediction probabilities when the model calculates the accuracy score for sentence completion. If one of the first k words matches the word actually written in the extract, then the prediction is considered accurate. It then gives the percentage of correct answers out of the total number of answers. In our experiments, we set k = 5.

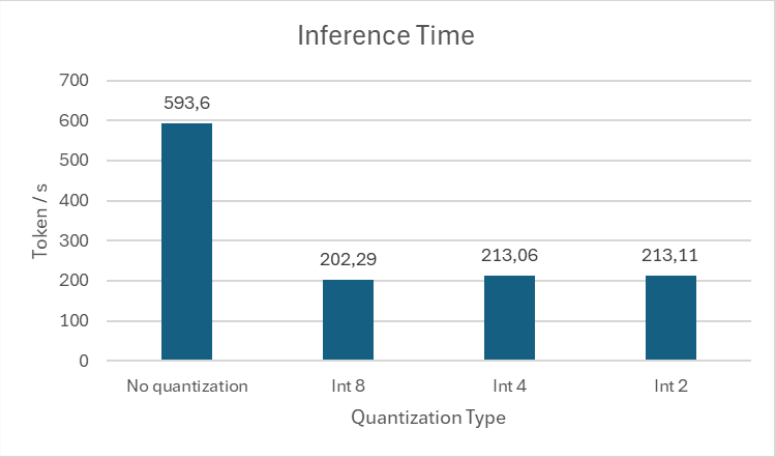
- **Speed** is simply the rate at which our model is able to process the information and generate the missing word. In our case, the metric for measuring this speed is the number of tokens processed plus the token generated by the model every second, **Token/s**. It is also possible to measure this in responses per second, but whatever the unit chosen, the trends in the results remain the same.

- **Memory** factor is the space the model takes up on the machine running it. Here we measure the **size of the model in Ko** to get an idea of the memory used to store our models.

Results and analysis :

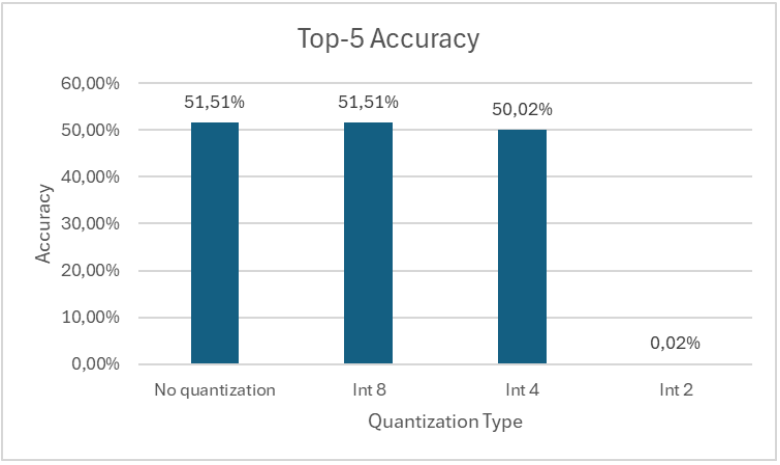
Here we present the results of our experiments on LLM. After showing the various graphs concerning evolution of memory, speed and precision, we will try to analyze the similarities and differences between all the versions of the model and to understand the positive and negative aspects.

- **Speed :**



One of the first conclusions we can draw from these models is that, given its lightness (only 3 billion parameters), the version of LLama chosen is generally quick to run. It therefore seems quite logical not to see any major differences in inference time between our quantized models, especially when the power of the machine used is taken into account. In our specific case, therefore, we don't seem to have gained any particular time resources through quantification and, on the contrary, the token time per second even decreases between the model without quantifications and the quantified versions. This is probably due to the fact that the input data is not in the same format as our weights, which may require the model to take some time to convert. Indeed with quantisation, the model weights are represented in a reduced format, which means more complex data conversion and transfer. However, it is interesting to note that as quantization increases, the processing time decreases, allowing more tokens to be processed each second in int2 than in int8.

- **Accuracy :**

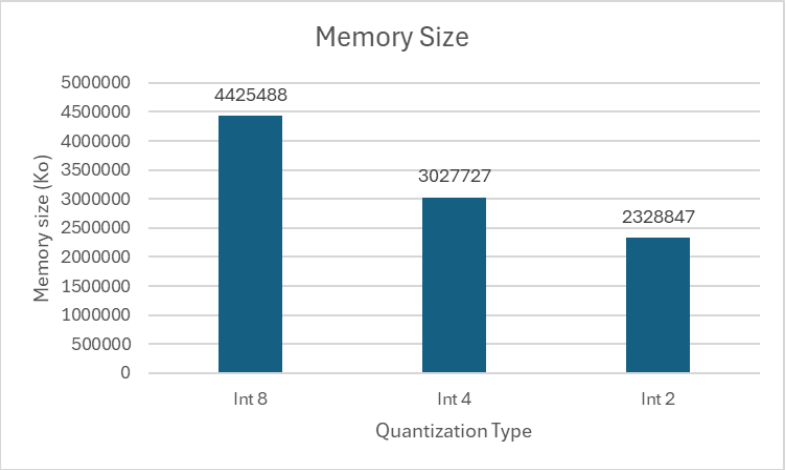


It goes without saying that the basic model does not produce impressive results on the LAMBADA dataset, as the half of correct responses can be considered to be only fair, but this can be explained by the small number of model parameters. Putting that aside, we can see that the differences in accuracy between the int8 model and the original model are fairly negligible when we take into account the size saved by quantisation. What's more, it's quite remarkable to

see that even an int4 quantification still gives results close to those of the unquantified model. It can therefore be estimated that int8 and int4 quantization may be excellent compromises for certain applications of this model.

It's important to note that Top-k accuracy initialized at 5 changes a lot the results we obtain with this kind of experiment. Obviously, if k were greater, the probability that the correct word would be among the predictions would increase, even though k = 5 is a fairly common value for this type of experiment. In the same way, we can say that the small difference between the int8 and int4 quantification and the basic model is perhaps due to this fairly large k. In fact, we can suppose that if we reduce it, the differences might be a little greater because the non-quantised system places the correct words with more accurate probabilities than the other models.

- **Memory :**



Generally speaking, it's easy to see that the size of the model is a parameter that decreases drastically with quantization. This is one of the reasons why this technique is so widely used. We can then see a fairly impressive gap at each quantification. The int2 quantised model is almost 2 times smaller than the int8 quantised model. As we couldn't measure the size of the non-quantised model ourselves, we preferred not to include it on the graph, but when you compare the estimated size of this model available on HuggingFace, it takes up more than twice as much memory space as the quantised int8 version.

Concerning the int2 quantization and according to the model we have chosen, which is an extremely light version of Llama, it seems normal that a quantization as heavy as int2 on this type of model obtains such mediocre results. **We can also see below** that the majority of predictions are hallucinations, punctuation or only syllables, which clearly proves the difficulty of the model in carrying out the required task. This is probably due to the fact that the weights are too degraded by quantization and no longer contain enough information to be relevant.

```
counter : 523
true word : tree
[' top', ',', ' that', ' ', ' now']
0
total_tokens : 42447
time : 198.63017535209656
counter : 524
true word : beth
[' also', ' a', ' but', ' -', ' del']
0
```

```
counter : 525
true word : milk
['row', 'ish', ' full', 'po', ' personal']
0
total_tokens : 42610
time : 199.38691568374634
counter : 526
true word : receivers
['?', ' min', ':', ' ', ',']
0
```

Conclusion

Insights and future recommendations

We have learned a lot from this project and here are the areas where we would have liked to improve our experiments :

Firstly, it would have been interesting to compare different competing models of LLM and ODM and their respective quantified versions to see if all the models reacted equally well. In addition, it would have been interesting to test the quantification and evaluation operations of our models on different machines, but due to lack of time we were unable to do this. Secondly, we also wanted to investigate the impact of training and fine-tuning the models after quantification in order to improve the results of our models. Finally, although it may seem obvious, we would have liked to test other quantification methods such as Quantization Aware Training.

Summary and conclusion

In conclusion, our experiments on quantization applied to Machine Learning models, and more specifically to LLM and ODM, clearly demonstrate the value of quantization.

First of all, quantization is particularly practical in our case, where our personal machines don't have a great deal of computing power. Indeed, while the non-quantized versions of the models were particularly painful to run on our computers, it was much simpler afterwards to continue our measurements on the lightened versions of these same models thanks to quantization.

Then, in addition to a substantial computing saving, it is also a substantial energy saving that can be brought by the use of quantization to the field of artificial intelligence. In our example, this may not have been meaningful, but it's easy to imagine that on the scale of industry giants like Google, Meta or OpenAI, quantified versions of conversational agents could save precious resources, particularly in the current environmental context.

Finally, we mustn't lose sight of the fact that quantization is always accompanied by a decline in the accuracy and relevance of models, which can in some cases render them obsolete. It is therefore a tool to be used wisely and with precision, insofar as it is capable of resolving many of the challenges facing AI today