OULUN AMMATTIKORKEAKOULU

Jere Muikku

Impact of quantization techniques on the performance and quality of AI models in AMD-based systems for local deployment of large language models

Impact of quantization techniques on the performance and quality of AI models in AMD-based systems for local deployment of large language models

How is AI quality affected by quantization on AMD-based platforms

Jere Muikku
Thesis
Fall 2024
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author(s): Jere Muikku
Title of the thesis: Impact of quantization techniques on the performance and quality of AI models in AMD-based systems for local deployment of large language models
Thesis examiner(s): Juha-Matti Huusko
Term and year of thesis completion: Fall 2024 Pages: 32

The performance and quality differences among various quantization techniques and degrees of quantization have long intrigued AI researchers. This thesis evaluates and compares the performance disparity and question-answering accuracy of k-quant GGUF models, focusing on text generation using locally run models.

The study began by selecting suitable models for text generation that were compatible with the GGUF format and capable of generating accurate, coherent text on Windows systems. After some challenges, the open-source Mistral NeMo and Llama 3.1 models were identified as meeting these criteria.

Testing revealed clear disparities in performance across quantization levels. At full precision (Mistral: 16-bit, Llama 3.1: 32-bit), both models ran slowly and exceeded GPU memory limits, requiring offloading to CPU and RAM, which significantly slowed generation (Mistral: 76s, 41 GB; Llama 3.1: 421s, 48 GB). Despite this, full weights answered 29 of 30 questions correctly.

The first quantization method tested, Q8_0 (legacy quantization), significantly reduced generation time (Mistral: 13s, 12 GB; Llama 3.1: 17s, 8 GB) and maintained accuracy, failing only 2 questions instead of 1. However, as quantization levels decreased, performance dropped, particularly in Llama 3.1, where Q3_K_L began producing incoherent answers and lower quantization's required multiple attempts to generate responses. Mistral NeMo faced similar issues, with its Q2_K model unable to answer questions at all.

Notably, once memory usage fit entirely within GPU VRAM, generation speed did not improve further; only memory consumption and question accuracy were affected. This behaviour might vary on hardware with less available memory, which would use offloading.

Keywords: Large Language model, Text generation, GPU, Deep learning, Artificial Intelligence, Quantization, ROCm, k-quants, CPU, RAM, VRAM

CONTENTS

VOCABULARY

LLM	Large language model, a model that is pre-trained on vast amounts of data.
GPU	Graphical Processing Unit, used to compute graphics and image process.
CPU	Central Processing Unit, control centre of computers, the brain.
CUDA	Platform for general computing on GPUs. Used in multiple many different purposes, including AI runtimes.
ROCm	Open software stack that includes a broad set of different things, including AI runtimes.
AI	Artificial intelligence, simulation of human intelligence.
NLP	Natural Language Processing, ability for AI to understand human languages.
GPT	Generative Pre-training Transformer, a type of LLM model
RLHF	Reinforcement learning from human feedback, type of learning and improvement method for AI models
GGUF	GPT-Generated Unified Format, a file type for created LLM models, format for running and to quantize models.
Token	AI:s way of classification for words, or characters, depending on the model. Used to calculate how many words a model can take and output.
GPTQ	Generalized Post-Training Quantization, is a layer-wise quantization technique: approach that quantizes each layer at a time to minimize errors.
FP	Floating-point arithmetic, method for representing and manipulating real numbers in a way that can support a wide range of values, FP64, FP32 (64-bit, 32-bit)
INT	Integer, in the context of LLMs, it's a fixed-point representation where values are integers instead of floating-point numbers, INT8, INT4 (8-bit, 4bit)
PTQ	Post-Training Quantization, technique for the quantization of already trained large language models
QAT	Quantization-Aware Training, technique used to quantize a large language model while training it.

1 INTRODUCTION

Artificial intelligence, or AI for short, is something that is becoming more common than ever before, be it in phones with their virtual assistants like iPhones with their Siri and Google assistant on Android devices, or in the cutting-edge technologies in healthcare, finances and transportation. However, has there been mention of the new emerging trend of AI-generated content? It represents a significant aspect of AI development in the current era, notably in the Natural Language Processing department, NLP for short. Prior to the year 2022, was there any awareness of AI-generated text content or stories? What about AI generated images or even videos? It is quite understandable if there was no prior awareness, as before the release of ChatGPT in the year 2022, the AI development was still in a stable phase. However, after the release and success, the field of text generation started to gain mainstream popularity and making the AI generated content to start booming (1).

One of the earlier, and more developed areas of NLP, is the AI text-generation. In this thesis, the focus will be on examining the performance and quality of text-generation on different methods of quantization.

2 ARTIFICIAL INTELLIGENCE

The realm of AI is vast, encompassing numerous learning methods and techniques (2). However, this thesis primarily focuses on three interconnected fields: deep learning, machine learning, and natural language processing, NLP for short. These domains hold particular significance as they are the most common groundwork for generating text-based content through AI methodologies, particularly NLP. Within these three fields AI is serving as the overarching concept, with machine learning and deep learning as a subset of it, further defining the specific areas of investigation (figure 1).

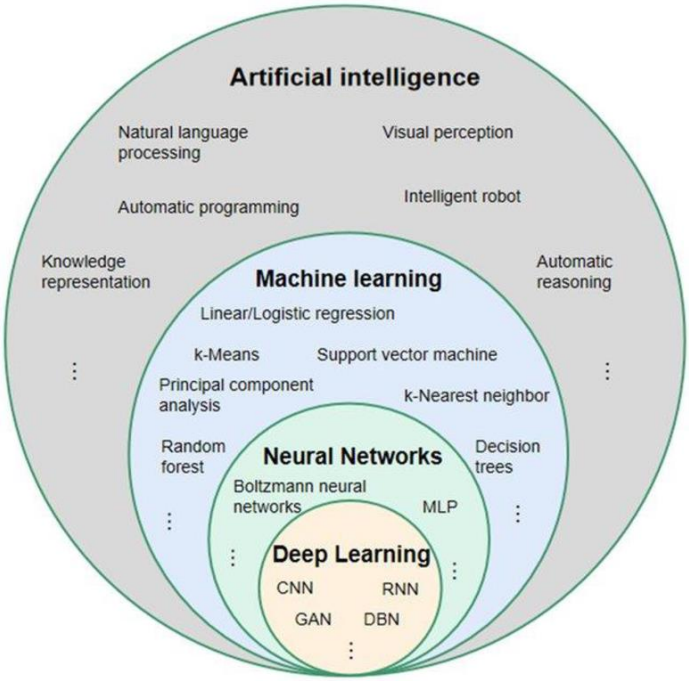

FIGURE 1: Relationship between artificial intelligence, machine learning, neural network, and deep learning. (3)

In the figure, multiple different words and terms can also be seen, but our sole focus, as I mentioned earlier, will be on NLP, Machine learning, and the deeper subcategory, deep learning, also a bit about neural networks. This thesis will not delve deeply into other concepts. The focus on the Deep learning will be about Large Language Models, LLMs for short, which will be discussed and explained on the next part. Before delving into the other topics, it's important to understand what NLP is. Briefly told it's, "The basics of natural language involve teaching computers to understand and generate human language in a similar way to humans. This enables computers to communicate

with users more naturally and perform language-based task more effectively" (4). This will be explored further in the latter part of the thesis.

2.1 Large Language Model

Large Language Model, or LLM for short, is essentially a sophisticated AI model designed to recognize and generate text, among other complex tasks. "LLMs trained on huge sets of data, hence the name large" (5). These datasets are larger than one might initially expect, ranging from as low as a few billion parameters to many trillions. They vary in size from a few gigabytes to tens of petabytes, depending on the scope and the purpose the model was trained for. (6)(7)

LLMs are based on machine learning and the underlying category of neural networks and deep learning techniques. The neural network architecture commonly used in the creation of these large models are known as the transformer model. But what is a transformer model? In a short sentence. "It's a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence" (8). The architecture of the transformer model can be seen on the figure below (figure 2).

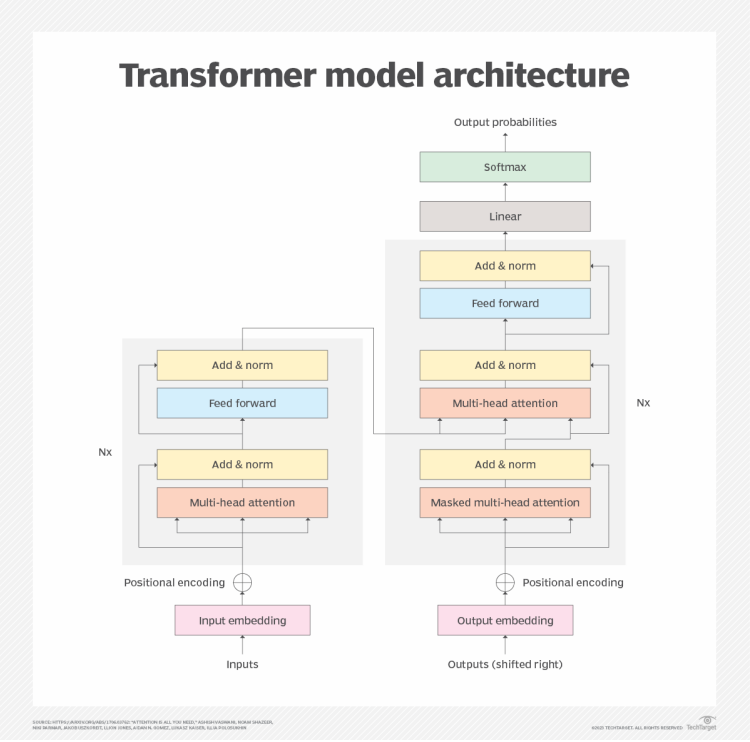

FIGURE 2: Architecture for a transformer Model (9)

These models use deep learning to understand how everything work together, whether it's characters, words or sentences. It involves the analysis of unstructured data, known as unsupervised type of learning in machine learning (5). As previously mentioned, unsupervised learning is a subset of machine learning, encompassing various learning types. However, in the case of LLMs, the primary method of learning is unsupervised learning (figure 3).

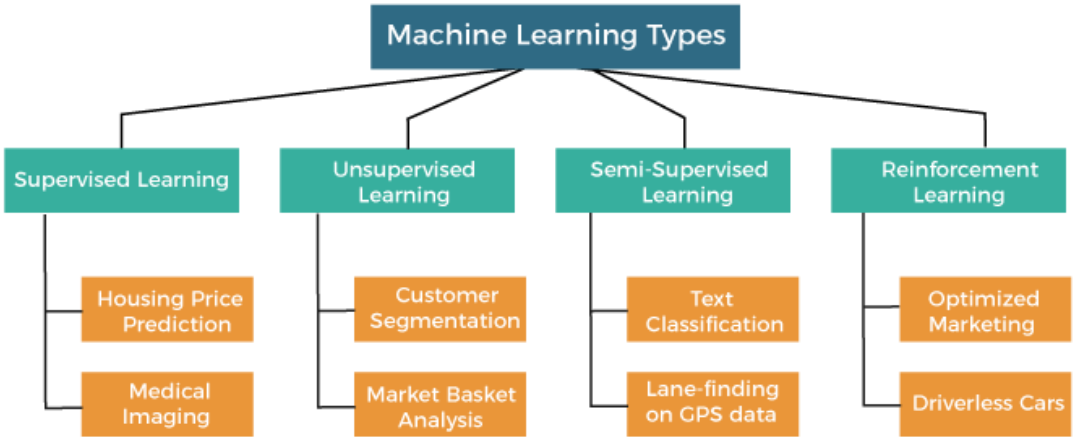

FIGURE 3: Mind map of Machine Learning Types (10)

After the model is created via unsupervised learning, it can be further fine-tuned using Reinforcement Learning through user input. This process is referred to as Reinforcement Learning from Human Feedback, or RLHF for short. This approach is demonstrated in the development and improvements of GPT 3.5, which evolved into ChatGPT (11). In short, model creation begins with unsupervised learning, where the model undergoes self-learning through user chosen and provided datasets. Subsequently, the model can be fine-tuned using Reinforcement Learning or/and supervised learning. The process will be revisited in the next chapter while the model(s) utilized for this thesis, Mixtral and LLAMA, are explained.

2.2 Quantization

Quantization, in both mathematics and physics, refers to the process of mapping an infinite range of continuous values to a finite set of discrete values (25). This principle is also applied in model quantization, a technique used to optimize large neural networks, including Large Language Models (LLMs). By reducing the precision of weights and activations, these models can achieve greater efficiency, minimizing their size and computational requirements while maintaining acceptable performance (26). Precision types are typically represented in bit-based formats, such as 64-bit floats. Quantization involves reducing values to lower precision levels, for example, converting from 64-bit to 32-bit and then to 16-bit floats (30).

A common application, particularly in local computing environments with limited graphical resources, involves quantizing 16-bit floats to 8-bit integers or lower. In this thesis, both fully non-quantized 16-bit float models and those quantized down to 1-bit integers will be used and analysed. Further details and testing will be provided in Chapter Three.

One prominent method for quantizing and deploying GPT-based LLMs is Generalized Post-Training Quantization (GPTQ). GPTQ is a layer wise quantization algorithm that optimizes and quantizes the weights of each layer while minimizing the error introduced by quantization (29). This approach enhances the efficiency of GPT-based models, making them more suitable for deployment on resource-constrained hardware. Several other quantization methods exist, each with its own advantages and limitations, and these will be explored in detail in a later chapter.

2.2.1 Weights and Biases

Weights are foundational to how neural networks, including Large Language Models (LLMs), process and learn from data. In a neural network, weights capture the strength of the relationships between inputs and outputs, acting as a bridge between neurons across different layers. These values determine how information flows through the network, shaping the model's ability to recognize patterns and make predictions (27).

In LLMs, weights and biases play critical roles. Weights, as discussed, establish relationships between inputs and outputs, determining the importance of each input in the model's decisions. Biases, on the other hand, are constants added to neurons to provide an offset, ensuring activation even when the weighted sum of inputs is insufficient. Together, they enable the model to learn complex patterns in data. This is where quantization, as mentioned earlier, comes into play — it reduces the precision of these weights and biases, which can affect how accurately the model learns and makes predictions (26). For instance, instead of using a value with several decimal places (like 1.32562122), a quantized model might round it to just one or two decimal places, such as 1.33 or even to just 1, depending on the degree of quantization applied to the model (figure 4).

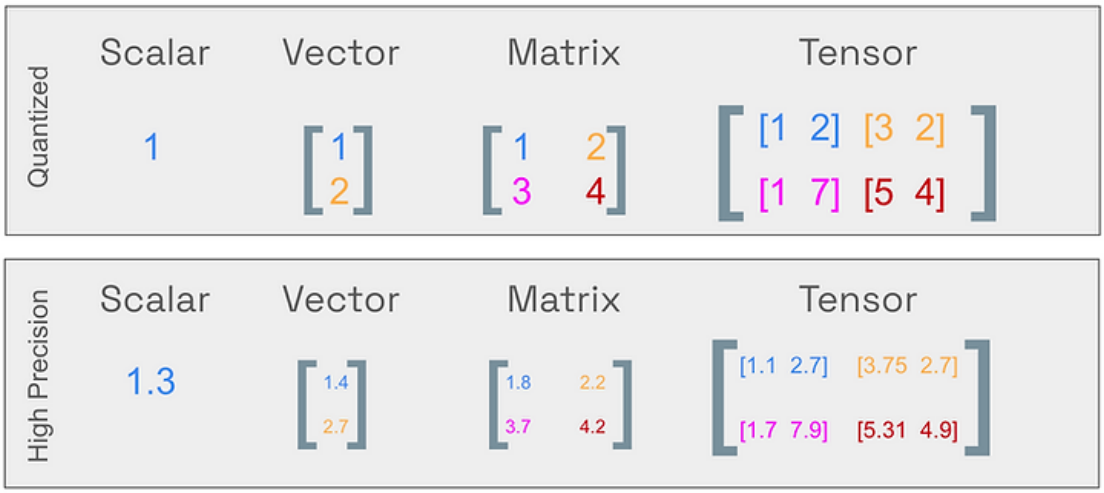

FIGURE 4: Quantized LLM data types (26)

When processing a sentence, the weights in an LLM determine how strongly the context of one-word impacts another. Biases allow the model to activate neurons even when the weighted sum of inputs is too small, ensuring the network's flexibility. Through multiple layers of transformations, weights help prioritize relevant information, while biases introduce a necessary threshold, enabling the model to adapt as it interprets data.

2.2.2 Forward propagation

In the broader context of neural networks, the process of calculating outputs from inputs using weights and biases is referred to as forward propagation. In the context of LLMs, forward propagation involves processing input tokens (such as words or sub words) through multiple layers of the

model. Each layer has its own set of weights and biases, and during forward propagation, the model computes the weighted sum of inputs for each neuron, applies the activation function, and passes the output to the next layer (figure 5).

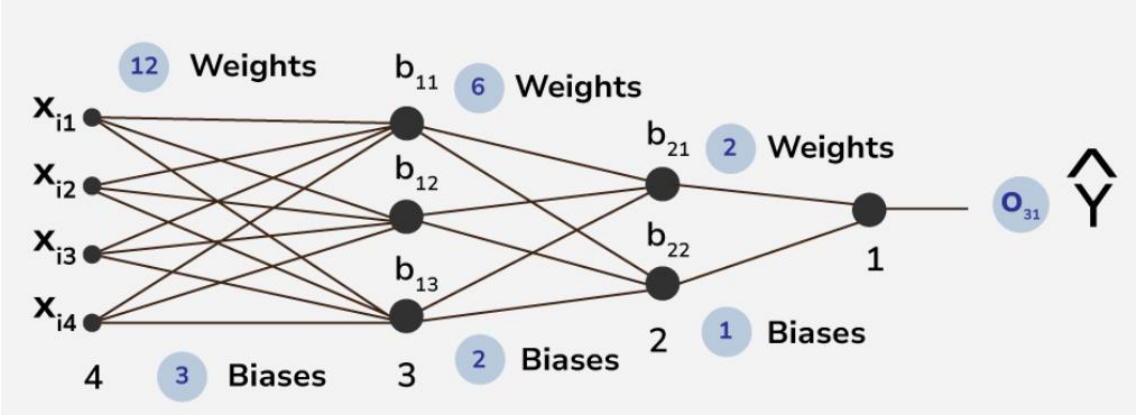

FIGURE 5: Forward Propagation (28)

At each layer, the model refines its understanding of the input by transforming the data based on learned weights. The output of the final layer corresponds to the model's prediction or, in the case of language generation, its next word or token. The importance of the weights in this process cannot be overstated, as they dictate the influence each token has on the generated output. The strength of these weights is what allows the model to generate contextually appropriate responses by understanding the relationships between words and concepts within the given text.

In Large Language Models, forward propagation occurs rapidly during inference (when the model is used to generate text or answer questions), as the weights have already been optimized during training. This enables the model to perform tasks such as generating text, completing sentences, or answering queries based on the vast amount of learned knowledge stored in the weights.

Alongside forward propagation there is also Backpropagation, but this topic will not be discussed further on this thesis as this goes more on the side of training Large Language Models.

2.2.3 Quantization types

Quantization methodologies for optimizing neural networks, particularly Large Language Models (LLMs), can generally be categorized into two types: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). Both approaches aim to improve model efficiency by reducing precision, yet they differ significantly in their implementation and computational demands (30).

Post-Training Quantization (PTQ) refers to techniques that quantize an already-trained model without requiring further training. This method is simpler and faster to implement compared to QAT, as it requires less training data and avoids the need for retraining the model. However, this simplicity comes at a cost: PTQ often results in a loss of model accuracy due to reduced precision in the weights and activations (31).

Quantization-Aware Training (QAT), by contrast, integrates quantization into the training process itself. During training, QAT incorporates weight conversion processes such as calibration, range estimation, clipping, and rounding. This allows the model to learn to operate effectively within the constraints of reduced precision, often yielding superior performance retention compared to PTQ. Nevertheless, QAT is more computationally intensive and requires additional training data, making it a more demanding approach (31).

Both PTQ and QAT serve as essential tools for quantization to run LLMs on resource-constrained hardware. While PTQ offers a quick and straightforward solution, QAT provides a robust alternative for scenarios where higher performance retention is critical at the cost of more training time.

2.2.4 Quantization methods

There are various methods for quantizing a model, with some of the most common being AQLM, AQW, GGUF/GGML, GPTQ, and EXL2 (32). This thesis focuses on the GGUF method, which is particularly effective for offloading computational tasks to the central processing unit (CPU) and random-access memory (RAM) rather than relying solely on the graphics processing unit's (GPU) video random-access memory (VRAM).

"GGUF is a file format for storing models for inference with GGML and executors based on GGML" (16). Other file formats for these models include GGML, GGMF, and GGJT. Previously, the GGLM format was used, but it has been replaced by the more efficient GGUF format, which is based on the GGJT format. Notably, executors designed for the earlier formats are fully compatible with GGUF. KoboldAI (17) is an example of a system built on this format. Later in this thesis, KoboldCPP, a distinct implementation from KoboldAI, will be discussed.

As of 2024, GGUF models support three primary methods of quantization:

The first method is Legacy Quantization, which includes techniques like q4_0 and q4_1. These were among the earliest approaches used for compressing model weights, primarily by reducing their precision to 4-bit integers. While effective at minimizing model size and enabling inference on devices with limited computational resources, legacy quantization methods may compromise model accuracy compared to more advanced techniques. Although these quants are fast and simple, they are largely outdated by 2024, as the more efficient k-quants have become the preferred standard.

The second method is K-Quantization, or k-quants, which represents a significant improvement over the legacy approach. This category includes formats such as q4_k_s, q4_k_m, q5_k_l, and q6_k. K-quants are specifically designed to preserve higher precision while still achieving substantial compression. By utilizing optimized algorithms, they minimize information loss and maintain model fidelity, making them particularly well-suited for resource-intensive tasks such as text generation and conversational AI.
K-quants are the most commonly used method for running large language models (LLMs) locally, offering excellent efficiency on both GPUs and CPUs with offloading capabilities. This method will be employed in the models tested for this study. Differences in quantization levels will be analysed

using k-quants rather than the other methods, providing a focused comparison within this framework.

The final method is I-Quantization, or i-quants, a newer SOTA method (33) which focuses on maximizing computational efficiency by converting model weights into integer formats. This approach is specifically tailored for integer-only computation hardware, such as CPUs with limited floating-point processing capabilities. I-quants are ideal for cost-effective inference scenarios and are especially advantageous for deployment on modern hardware accelerators optimized for integer arithmetic. They are the most storage-efficient method, producing smaller model sizes, which is beneficial for devices with limited VRAM. Additionally, i-quants can enhance computational speed; however, they are highly sensitive to memory bandwidth, making them less robust than legacy or k-quants on certain hardware.

2.3 Mistral

Mistral is an AI model developed by the Mistral AI Team, known for their provision of high-quality, free models for users. They stand out as one of the premier providers in this domain, alongside Meta, which offers their open-source Llama model. But why Mistral? According to their own benchmarking, their new model offers a better performance than their competitors, Meta with LLaMA 3 8B, Google with Gemma 2 9B. (figure 6)

	Context Window	HellaSwag (0-shot)	Winogrande (0-shot)	NaturalQ (5-shot)	TriviaQA (5-shot)	MMLU (5-shot)	OpenBookQA (0-shot)	CommonSense QA (0-shot)	TruthfulQA (0-shot)
Mistral NeMo 12B	**128k**	**83.5%**	**76.8%**	**31.2%**	**73.8%**	68.0%	**60.6%**	**70.4%**	**50.3%**
Gemma 2 9B	8k	80.1%	74.0%	29.8%	71.3%	**71.5%**	50.8%	60.8%	46.6%
Llama 3 8B	8k	80.6%	73.5%	28.2%	61.0%	62.3%	56.4%	66.7%	43.0%

FIGURE 6: Model benchmarks (12)

One of the models discussed and used in this thesis is their latest open-source model of Mistral NeMo 12B, their smallest open-source option. It is a model that can handle a context up to 128k tokens, as stated on their website (12). In the realm of Large Language Models, tokens refer to discrete units of text that the model processes or generates (13). The 128k token context capability implies that the model can analyse and generate content based on a substantial amount of text, effectively utilizing the provided information to generate smart responses.

In the context of story writing, this would mean providing the model with the equivalent of tens of pages of text and expecting it to generate a smart response while also remembering every single detail given. If there is curiosity regarding what happens if the input exceeds the amount the model can handle, it's quite simple: the model forgets the earlier data as it can only process the certain number of tokens at a time. Hence, it would be for the best to give and utilize concise and informative descriptions when giving an input to a model, even if the token amount is large.

	MT Bench	WildBench
Mistral 7B	6.48	25.55
Llama 3 8B	6.85	28.77
Mistral NeMo	**7.84**	**42.57**

FIGURE 7: Model benchmarks (Instruct) (12)

In figure 7, the key benchmark to focus on is the MT bench, specifically designed for Instruct Models. The term "MT" in the benchmark means: "multi-turn", indicating that it evaluates the conversational and instruction-following capabilities of LLMs across multiple turns of coherent, informative, and engaging conversations. "MT-Bench is a challenging multi-turn question set designed to evaluate the conversational and instruction-following ability of LLMs" (14).

An instruct model, as utilized in this thesis, refers to a "model that has been optimised through supervised fine-tuning and direct preference optimisation (DPO) for careful instruction following" (12). The MT Bench testing demonstrates efficacy, showcasing significant performance advantage over Meta's LLaMA 3 8B model and their own older Mistral 7B model (Figure 6). This performance of the model makes it well-suited for conducting the text-based testing in this thesis. And as it is open source, it's free to use.

The exact model used in this thesis for Mistral NeMo is called "dolphin-2.9.3-mistral-nemo-12b-gguf" (15). This model is formatted in GGUF (Mentioned in the earlier chapter) and is compatible with KoboldCPP, the back end used in this thesis. This topic will be revisited in later chapters.

2.4 Llama

Meta, formerly known as Facebook, has developed the Llama series of AI models as its flagship open-source offering. This thesis utilizes the 3.1 iteration of Llama, rather than the latest 3.3 version. The Llama lineup includes models with parameter sizes ranging from the largest at 405 billion (405B) parameters, to 70 billion (70B), and finally to 8 billion (8B). The 8B model was selected for this work to enable local deployment. As of 2024, a Llama 3.3 version of the 8B model has not yet been released, which necessitated the choice of Llama 3.1.

The largest Llama model (405B) is reportedly comparable to GPT-4 and Claude 3.5 Sonnet, according to Meta's own benchmarks (figure 8).

Category / Benchmark	Llama 3.1 405B	Nemotron 4 340B Instruct	GPT-4 (0125)	GPT-4 Omni	Claude 3.5 Sonnet
General MMLU (0-shot, CoT)	88.6	78.7 (non-CoT)	85.4	**88.7**	88.3
MMLU PRO (5-shot, CoT)	73.3	62.7	64.8	74.0	**77.0**
IFEval	**88.6**	85.1	84.3	85.6	88.0
Code HumanEval (0-shot)	89.0	73.2	86.6	90.2	**92.0**
MBPP EvalPlus (base) (0-shot)	88.6	72.8	83.6	87.8	**90.5**
Math GSM8K (8-shot, CoT)	**96.8**	92.3 (0-shot)	94.2	96.1	96.4 (0-shot)
MATH (0-shot, CoT)	73.8	41.1	64.5	**76.6**	71.1
Reasoning ARC Challenge (0-shot)	**96.9**	94.6	96.4	96.7	96.7
GPQA (0-shot, CoT)	51.1	-	41.4	53.6	**59.4**
Tool use BFCL	88.5	86.5	88.3	80.5	**90.2**
Nexus	**58.7**	-	50.3	56.1	45.7
Long context ZeroSCROLLS/QuALITY	**95.2**	-	**95.2**	90.5	90.5
InfiniteBench/En.MC	**83.4**	-	72.1	82.5	-
NIH/Multi-needle	98.1	-	**100.0**	**100.0**	90.8
Multilingual Multilingual MGSM (0-shot)	**91.6**	-	85.9	90.5	**91.6**

FIGURE 8: Model benchmarks, Llama 405B (34)

The model selected for this thesis is Meta-Llama-3.1-8B-Instruct-GGUF, converted and quantized by Bartowski (35). This version is the most widely used GGUF model of Llama 3.1 8B. According to Meta's benchmarks, the Llama 3.1 8B model performs marginally better than Mistral's 7B Instruct model (Figure 9).

Category / Benchmark	Llama 3.1 8B	Gemma 2 9B IT	Mistral 7B Instruct	Llama 3.1 70B	Mixtral 8x22B Instruct	GPT 3.5 Turbo
General MMLU (0-shot, CoT)	73.0	72.3 (5-shot, non-CoT)	60.5	86.0	79.9	69.8
MMLU PRO (5-shot, CoT)	48.3	-	36.9	66.4	56.3	49.2
IFEval	80.4	73.6	57.6	87.5	72.7	69.9
Code HumanEval (0-shot)	72.6	54.3	40.2	80.5	75.6	68.0
MBPP EvalPlus (base) (0-shot)	72.8	71.7	49.5	86.0	78.6	82.0
Math GSM8K (8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6
MATH (0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1
Reasoning ARC Challenge (0-shot)	83.4	87.6	74.2	94.8	88.7	83.7
GPQA (0-shot, CoT)	32.8	-	28.8	46.7	33.3	30.8
Tool use BFCL	76.1	-	60.4	84.8	-	85.9
Nexus	38.5	30.0	24.7	56.7	48.5	37.2
Long context ZeroSCROLLS/QuALITY	81.0	-	-	90.5	-	-
InfiniteBench/En.MC	65.1	-	-	78.2	-	-
NIH/Multi-needle	98.8	-	-	97.5	-	-
Multilingual Multilingual MGSM (0-shot)	68.9	53.2	29.9	86.9	71.1	51.4

FIGURE 8: Model benchmarks, Llama 8B (34)

3 TESTING

The testing process involves presenting the models, Mistral and Llama 3.1, with 30 questions simultaneously rather than one at a time. Both models must generate responses in a single uninterrupted session, and this process is repeated five times. The results are averaged across these repetitions to determine the final performance. The question set includes 20 reading comprehension questions and 10 basic math problems. The evaluation focuses on memory usage, the total time taken to generate the first output, memory utilization of the quantized model, and the average number of correct answers.

For the front-end interface to ask questions, SillyTavern is used due to its flexibility, with the specific settings being explained in detail later. On the back-end, KoboldCPP is used to run the GGUF models, specifically its ROCm version, designed for AMD-based GPUs.

The tests are conducted on a desktop computer equipped with an AMD Radeon 7900XTX GPU with 24GB VRAM, an AMD Ryzen 7 7800X3D CPU with PBO enabled, and 64GB of RAM running at 6000MHz CL30 with EXPO enabled. The system operates on Windows 10 Pro, version 10.0.19045 Build 19045. Additional details regarding the testing tools and methodologies are provided in the subsequent chapters.

3.1 KoboldCPP

KoboldCPP is described as “an easy-to-use AI text-generation software for GGML and GGUF models” (18). It is a distributable platform that extends from llama.cpp, incorporating versatile Kobold API endpoint and other front-end implementations. In this thesis, KoboldCPP is utilized as the backend server for loading our models. The frontend aspects will be covered by SillyTavern, a topic that will be elaborated on in the next chapter.

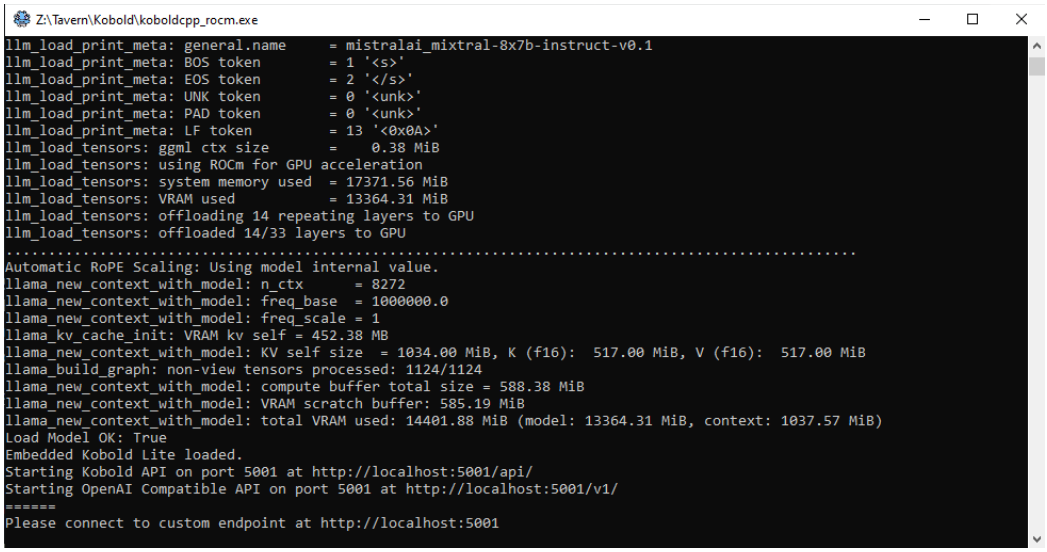

FIGURE 9: KoboldCPP-ROCm, model loaded.

Llama.cpp serves as the backend built on the Language Model Query Language, LMQL for short, a programming language specifically designed for Large Language Models (LLMs) (19). This backend enables the utilization of. GGUF model files, which are capable of running either exclusively on the CPU or in a mixed GPU/CPU environment, depending on the configured settings (20). (Figure 10)

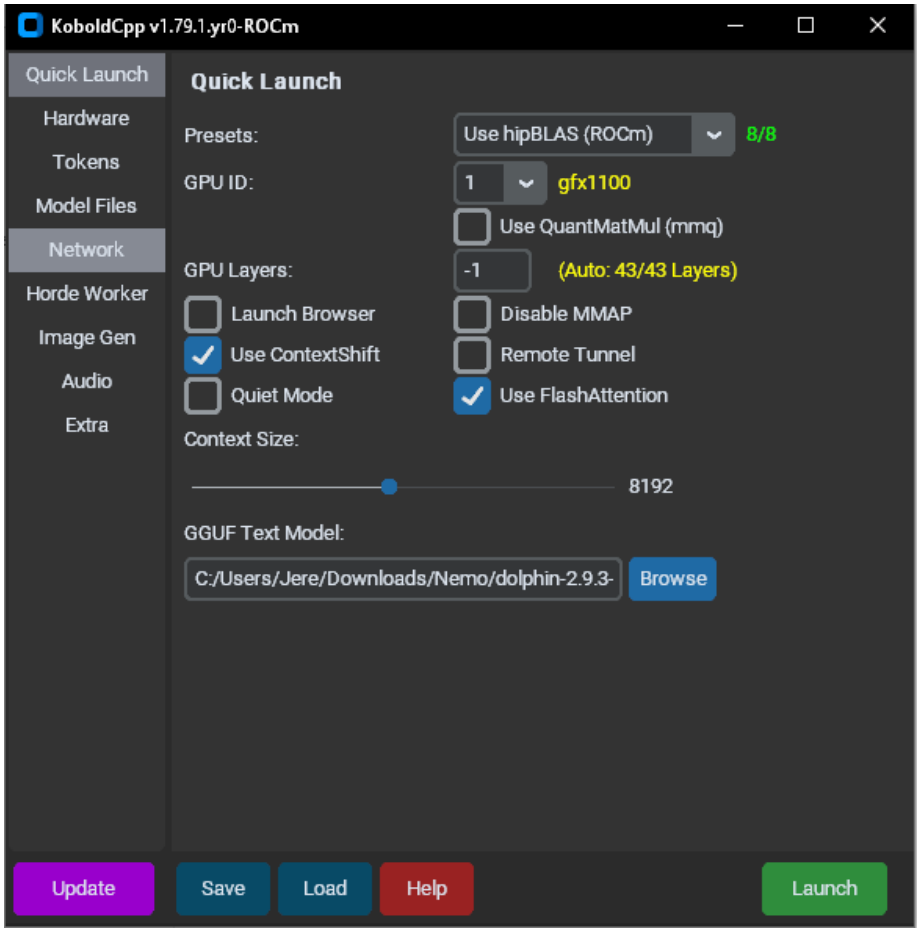

FIGURE 10: KoboldCPP-ROCm. V1.79.1

The figure depicts a modified version distinct from the original KoboldCPP. This iteration is a forked version capable of running on ROCm, the open software stack for AI learning developed and used by AMD and their GPUs (21; 22). Unlike the original version, which solely supports CUDA for NVIDIA-based GPUs. Both versions have support for only CPU run models, but this is very slow compared to using GPUs alongside CPUs.

For testing the models, the default settings were used, with three exceptions: the "launch browser" option was disabled, context size was set to 8192 (used to set how many tokens the AI looks at before generating a response) and Flash Attention was enabled.

Flash Attention is an algorithm that enhances the efficiency of the self-attention mechanism in transformer models by addressing memory and time bottlenecks. Instead of repeatedly transferring data between slower, larger HBM memory and faster, smaller SRAM, Flash Attention processes keys, queries, and values in a single step, reducing memory overhead and enabling faster training and inference (36).

In the case of KoboldCPP, Flash Attention speeds up text generation after the first output, streamlining the testing process as 30 questions are generated five times in total. Only the speed results from the first output are included in the testing data, as the effects of quantization are most noticeable there.

3.2 SillyTavern

"SillyTavern is a user interface designed for installation on computers and Android phones, enabling interaction with text generation AIs and facilitating chat or roleplay with characters created by users or the community" (23). This open-source, front-end application was selected for use in this thesis due to its superior customizability compared to the native GUI provided by KoboldCPP. SillyTavern's flexibility allows it to handle a range of tasks, including AI-driven chatting, story text generation, and AI-assisted applications similar to ChatGPT.

In addition to text generation, SillyTavern supports various extensions for tasks like image creation using Stable Diffusion speech recognition, and real-time voice cloning. While these extensions enhance its functionality, only the base text-generation features were utilized in this thesis (figure 11).

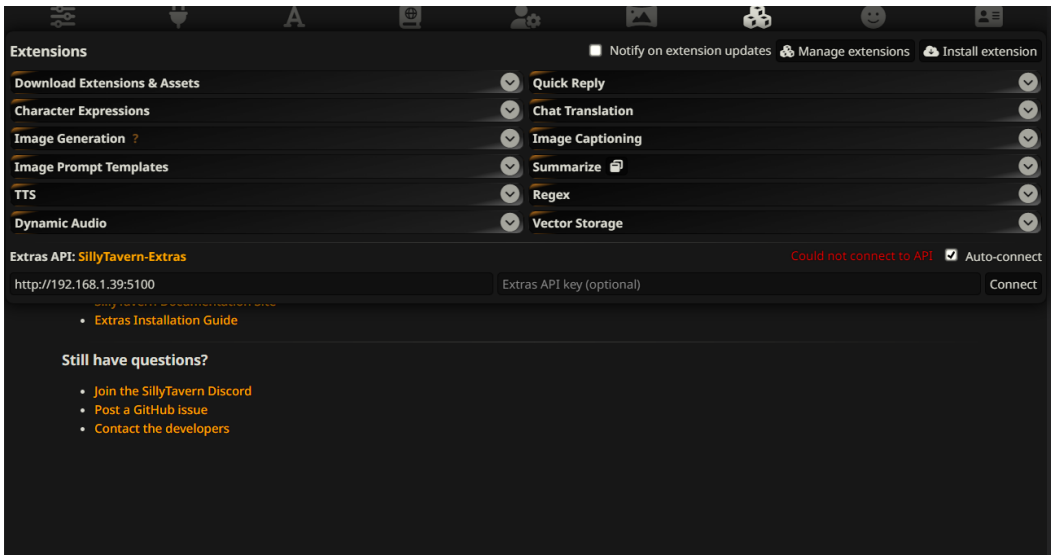

FIGURE 11: Extension possibilities.

Looking at the figure 12, the Kobold preset used when generating AI responses was “Assistant”, this gave the best results when generating responses when asking questions. Response was adjusted to 2048 tokens to allow the AI to write every answer in one sitting, and the context was changed to 8192.

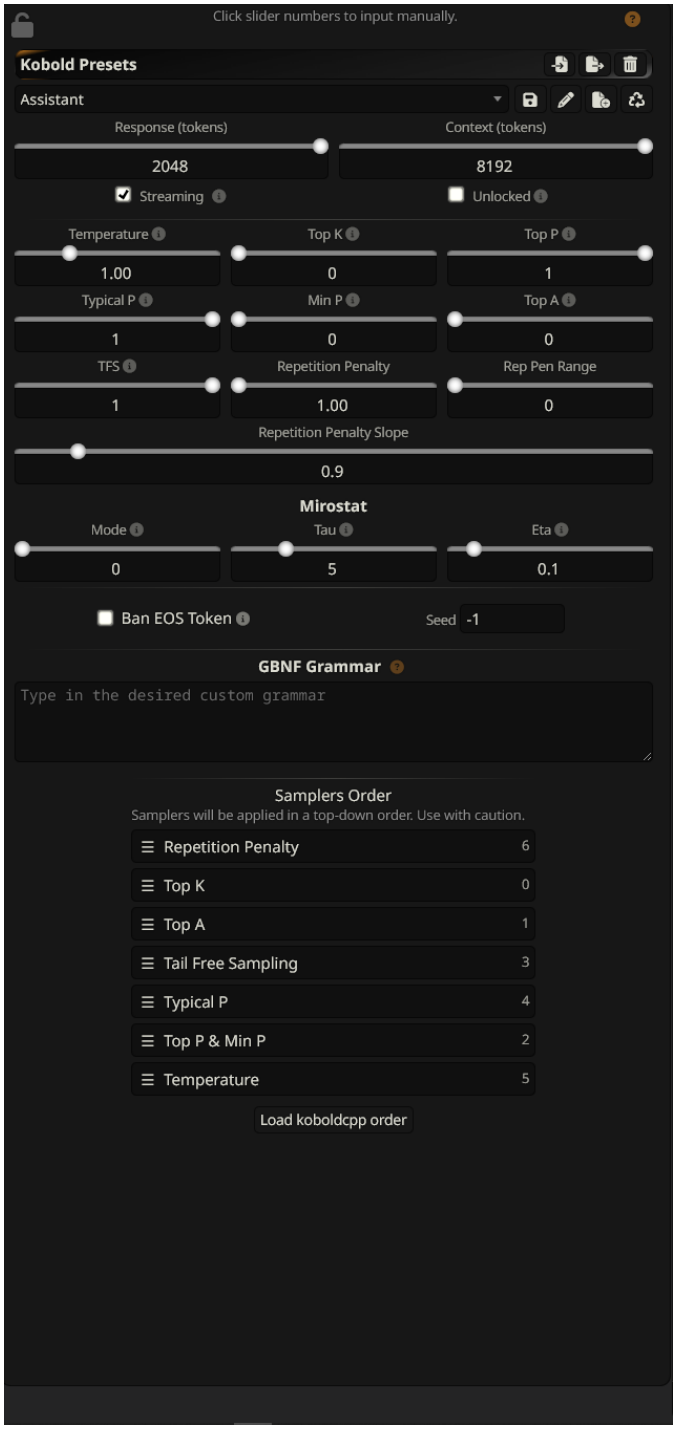

FIGURE 12: AI Response Configurations.

Looking at the figure 13, we can see the settings for the formatting of the AI responses. This one is for Mistral NeMo as it uses these formats to get the input and give an output of it.

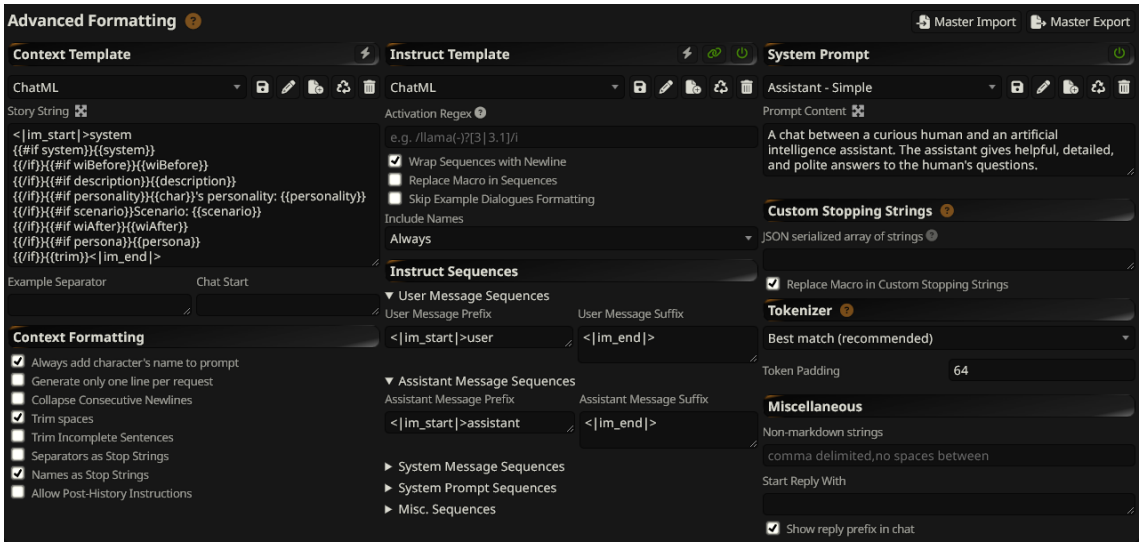

FIGURE 13: AI Response Formatting (Mistral).

The Mistral NeMo can be used with the context template of "ChatML" that is by default (at least as of 2024) on the SillyTavern interface.

For Llama3.1 settings we can look at figure 14. It uses the default template of Llama 3 instruct.

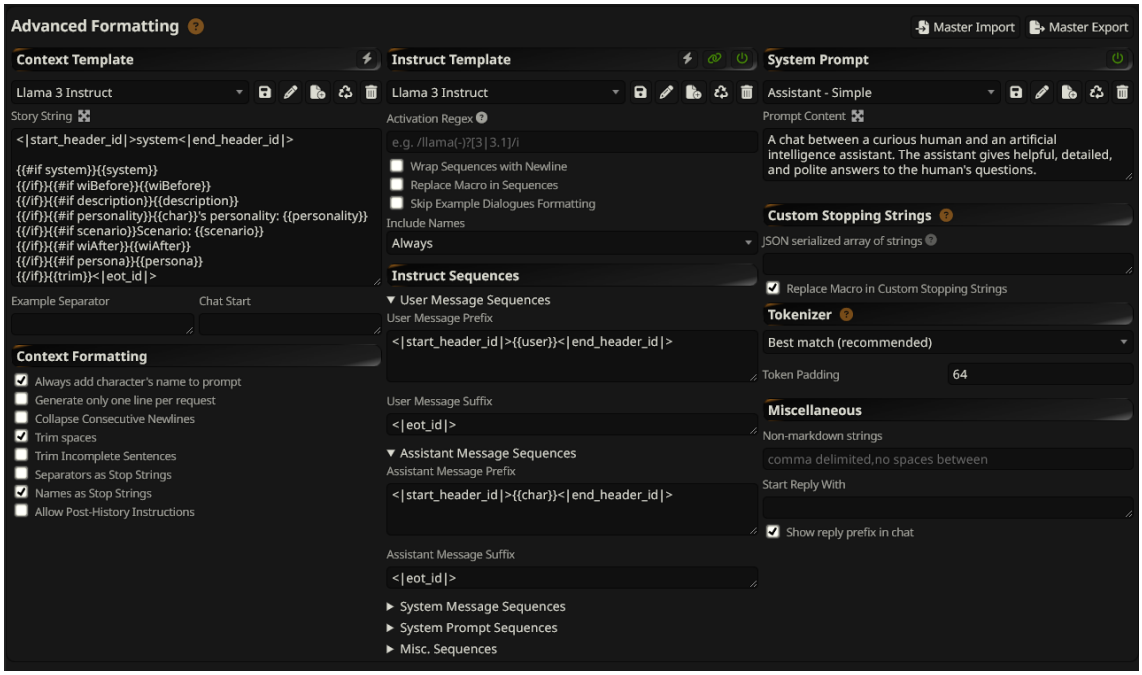

FIGURE 14: AI Response Formatting (Llama 3.1)

For the AI assistant (card) that was used in the testing can be seen in figure 15.

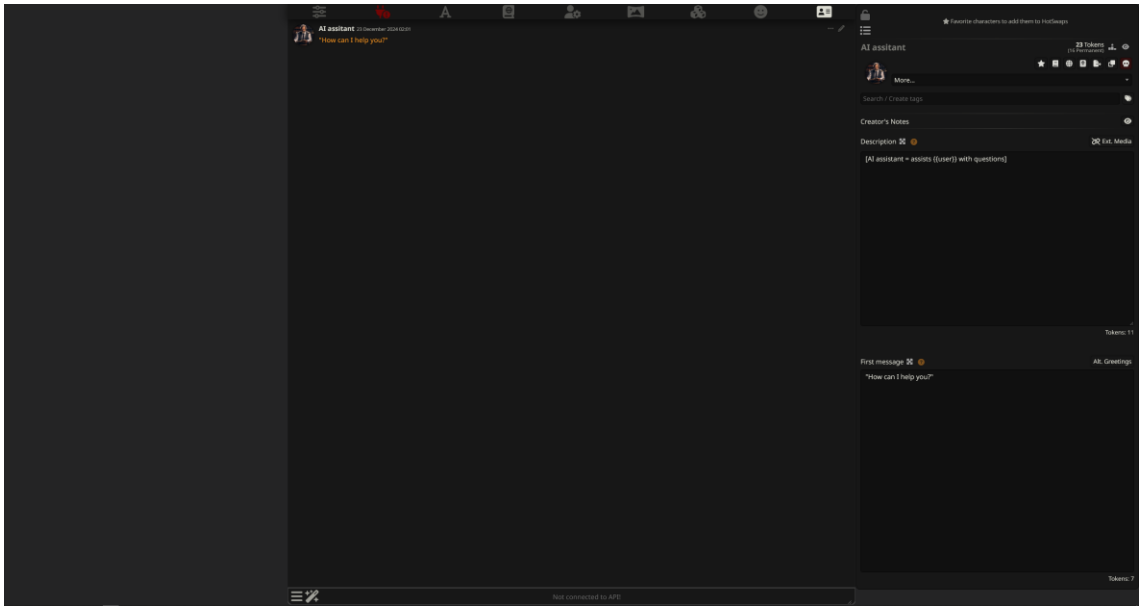

FIGURE 15: AI Assistant card

3.3 Questions

The learning comprehension questions were sourced from the web, while the math questions were basic ones specifically created for this testing. The learning comprehension questions were selected from grades 6, 7, and 8 (37).

The exact questions used can be checked from references (38).

4 CONCLUSION

Based on the test results, we can conclude that quantization has a significant impact on large language models, both positively and negatively. The positive effects include faster generation times and reduced memory usage, as seen with the testing system. On the negative side, there is a noticeable decline in the models' comprehension and coherence as the quantization levels decrease. These observations are summarized in the tables below:

Quants	Mistral NeMo 12B	Speed (Mistral)	Memory (Mistral)
F32 (LLAMA) and F16 (NeMo)	100 %	Total:75.60s (8.40T/s)	RAM = 23361.62MiB VRAM = 18201.37 MiB
Q8_0	97 %	Total:13.72s (25.00T/s)	RAM = 680.01 MiB VRAM = 11731.59 MiB
Q6_K	90 %	Total:13.00s (27.16T/s)	RAM = 525.01 MiB VRAM = 9057.84 MiB
Q5_K_M	90 %	Total:12.51s (27.18T/s)	RAM = 440.01 MiB VRAM = 7875.81 MiB
Q5_K_S	93 %	Total:9.01s (39.97T/s)	RAM = 440.01 MiB VRAM = 7676.59 MiB
Q4_K_M	83 %	Total:30.30s (34.52T/s)	RAM = 360.01 MiB VRAM =6763.31 MiB
Q4_K_S	83 %	Total:12.45s (28.04T/s)	RAM = 360.01 MiB VRAM = 6422.84 MiB
Q3_K_L	79 %	Total:19.32s (31.99T/s)	RAM = 275.00 MiB VRAM = 5975.03 MiB
Q3_K_M	86 %	Total:12.71s (28.17T/s)	RAM = 275.00 MiB VRAM = 5518.78 MiB
Q3_K_S	83 %	Total:29.26s (40.09T/s)	RAM = 275.00 MiB VRAM = 4995.34 MiB
Q2_K	0 %	No answers	RAM = 210.00 MiB VRAM = 4351.59 MiB

Quants	Llama 3.1 8B	Speed (LLAMA)	Memory (LLAMA)
F32 (LLAMA) and F16 (NeMo)	100 %	Total:421.60s (3.21T/s)	RAM = 30633.02 MiB VRAM = 18304.69 MiB
Q8_0	97 %	Total:17.37s (40.08T/s)	RAM = 532.31 MiB VRAM = 7605.33 MiB
Q6_K	93 %	Total:8.57s (35.57T/s)	RAM = 410.98 MiB VRAM = 5871.99 MiB
Q5_K_M	90 %	Total:8.18s (35.06T/s)	RAM = 344.44 MiB VRAM = 5115.49 MiB
Q5_K_S	90 %	Total:15.02s (52.34T/s)	RAM = 344.44 MiB VRAM = 4987.99 MiB
Q4_K_M	86 %	Total:8.03s (54.41T/s)	RAM = 281.81 MiB VRAM = 4403.49 MiB
Q4_K_S	83 %	Total:5.21s (54.71T/s)	RAM = 281.81 MiB VRAM = 4185.99 MiB
Q3_K_L	83 %	Total:8.44s (36.28T/s)	RAM = 215.27 MiB VRAM = 3609.99 MiB
Q3_K_M	76 %	Total:5.76s (54.33T/s)	RAM = 215.27 MiB VRAM = 3609.99 MiB
Q3_K_S	76 %	Total:6.40s (55.45T/s)	RAM = 215.27 MiB VRAM = 3271.99 MiB
Q2_K	55 %	Total:11.23s (59.59T/s)	RAM = 164.39 MiB VRAM = 2859.99 MiB

Testing revealed clear disparities in performance across quantization levels. At full precision—Mistral (16-bit) and Llama 3.1 (32-bit)—both models ran slowly and exceeded GPU memory limits, requiring offloading to CPU and RAM, which significantly slowed generation times (Mistral: 76 seconds, 41 GB of memory; Llama 3.1: 421 seconds, 48 GB of memory). Despite these issues, the full-precision models answered 29 out of 30 questions correctly.

The first quantization method tested, Q8_0 (legacy quantization), significantly reduced generation time while maintaining accuracy. Mistral took 13 seconds and used 12 GB of memory, while Llama 3.1 took 17 seconds and used 8 GB. Both models only failed 2 questions instead of 1. However, as the quantization levels decreased further, performance dropped, especially in Llama 3.1. At Q3_K_L, the model started producing incoherent answers, and lower quantization levels required

multiple attempts to generate responses. Mistral NeMo faced similar issues, with its Q2_K model unable to provide any answers at all.

From these results, it is clear that lower quantization levels improve speed and reduce memory usage. These factors are closely correlated—less memory usage allows more data to fit into the GPU, preventing offloading to the slower CPU and RAM. However, this improvement in speed and efficiency comes at the cost of the models' ability to generate coherent and accurate responses.

The more detailed results can be seen check from references (39).

REFERENCES

1. Chiancone C 2023. A year of AI Revolution: Celebrating ChatGPT's First Anniversary. LinkedIn. Paragraph 12. Available in: https://www.linkedin.com/pulse/year-ai-revolution-celebrating-chatgpts-first-chris-chiancone-fimuc. Search date 20.2.2024.
2. The Upwork Team 2024. The Top 4 AI Techniques to Know in 2024. Article. Upwork. Available in: https://www.upwork.com/resources/ai-techniques. Search date 20.2.2024
3. Li S, Deng YQ, Zhu ZL, Hua HL, Tao ZZ 2021. A Comprehensive review on Radiomics and Deep Learning for nasopharyngeal carcinoma imaging. Diagnostics. 11(9), article n o. 1523, MDPI. Available in: https://www.mdpi.com/2075-4418/11/9/1523. Search date 20.2.2024.
4. Sarrion S 2023. ChatGPT for Beginners. E-Book. Chapter 4: The Foundations of ChatGPT. Apress. Available in: https://learning.oreilly.com/library/view/chatgpt-for-beginners/9781484298046/ Search date 24.2.2024.
5. Cloudflare. What is a large language model (LLM). Cloudflare. Available in: https://www.cloudflare.com/en-gb/learning/ai/what-is-large-language-model/ Search date 24.2.2024.
6. Amazon. What are Large Language Models (LLM). Amazon. Available in: https://aws.amazon.com/what-is/large-language-model/ Search date 24.2.2024.
7. Yan L, Jiuhaun C, Chonguy L, Kai D, Lianwen J 2024. Datasets for Large Language Models: A comprehensive Survey. Diagnostics. Arxiv Available in: https://arxiv.org/abs/2402.18041. Search date 25.2.2024.
8. Merrit R 2022. What is a Transformer Model. Blogs. Nvidia. Available in: https://blogs.nvidia.com/blog/what-is-a-transformer-model/ Search date 25.2.2024.
9. Kerner S. Large language models. Tech-target. Available in: https://www.techtarget.com/whatis/definition/large-language-model-LLM. Search date 26.2.2024
10. Javatpoint. Types of Machine Learning. Javatpoint. Available in: https://www.javatpoint.com/types-of-machine-learning. Search date 26.2.2024.
11. Abideen Z 2023. Reinforcement learning from Human Feedback (RLHF): Empowering ChatGPT with User Guidance. Blogs. Medium. Available in: https://medium.com/@zaiinn440/reinforcement-learning-from-human-feedback-rlhf-empowering-chatgpt-with-user-guidance-95858592fdbb. Search date 29.2.2024.

12. Mistral Team 2024. Mixtral of experts, A high quality Sparse Mixture-of-Experts. Mistral AI Team. Available in: https://mistral.ai/news/mistral-nemo/. Search date 15.10.2024.
13. Humor M 2023. Understanding "tokens" and tokenization in large language models. Blogs. Medium. Available in: https://blog.devgenius.io/understanding-tokens-and-tokenization-in-large-language-models-1058cd24b944. Search date 1.3.2024.
14. Walker S. MT-Bench (Multi-turn Benchmark). Klu. Available in: https://klu.ai/glossary/mt-bench-eval. Search date 1.3.2024.
15. Cognitivecomputations 2024. dolphin-2.9.3-mistral-nemo-12b-gguf. Hugging Face. Available in: https://huggingface.co/cognitivecomputations/dolphin-2.9.3-mistral-nemo-12b-gguf. Search date 17.10.2024.
16. ggerganov 2024. Ggerganov/ggml. docs/gguf.md. GitHub. Iteration v3. Available in: https://github.com/ggerganov/ggml/blob/master/docs/gguf.md. Search date 3.3.2024.
17. KoboldAI 2022. KoboldAI-Client. Iteration 1.19.2. Github. Available in: https://github.com/KoboldAI/KoboldAI-Client. Search date 4.3.2024.
18. Lostruins 2024. KoboldCPP. Iteration 1.60.1. Github. Available in: https://github.com/LostRuins/koboldcpp. Search date 4.3.2024.
19. Mansurova M. LMQL – SQL for Language Models. 2023, Medium. Available in: https://towardsdatascience.com/lmql-sql-for-language-models-d7486d88c541. Search date 5.3.2024.
20. Lmql.ai. Llama.cpp. lmql. Available in: https://lmql.ai/docs/models/llama.cpp.html. Search date 5.3.2024.
21. YellowRoseCx 2024. Koboldcpp-rocm, Iteration: 1.59.1. Github. Available in: https://github.com/YellowRoseCx/koboldcpp-rocm. Search date 5.3.2024.
22. AMD. ROCm Software, AMD. Available in: https://rocm.docs.amd.com/en/latest/what-is-rocm.html. Search date 5.3.2024.
23. SillyTavern 2024. SillyTavern – LLM frontend for power users. SillyTavern. Available in: https://sillytavernai.com/. Search date 6.3.2024.
24. SillyTavern 2024. SillyTavern. Iteration: 1.11.5. Github. Available in: https://github.com/SillyTavern/SillyTavern. Search date 6.3.2024.
25. MathWorks. What is Quantization? MathWorks. Available in: https://www.mathworks.com/discovery/quantization.html. Search date 24.11.2024.
26. Miguel N 2023. What are Quantized LLMs. TensorOps. Available in: https://www.tensorops.ai/post/what-are-quantized-llms. Search date 24.11.2024.

27. Stephen W II. What are weights and Biases. Klu.ai. Available in: https://klu.ai/glossary/weights-and-biases. Search date 24.11.2024.
28. Geeksforgeeks. What is Forward Propagation in Neural Networks 2024. Geeksforgeeks. Available in: https://www.geeksforgeeks.org/what-is-forward-propagation-in-neural-networks/. Search date 26.11.2024.
29. Picovoice 2023. What is GPTQ? Picovoice. Available in: https://picovoice.ai/blog/what-is-gptq/. Search date 3.11.2024.
30. Kartik T 2024. A guide to Quantization in LLMs. Symbl.ai. Available in: https://symbl.ai/developers/blog/a-guide-to-quantization-in-llms/. Search date 11.12.2024.
31. Jahid H 2024. Optimizing Large Language Models through Quantization: A Comparative Analysis of PTQ and QAT Techniques, Page 4. Arxiv.org. Available in: https://arxiv.org/abs/2411.06084. Search date: 14.12.2024.
32. Hugging face. Quantization. Hugging Face. Available in: https://huggingface.co/docs/transformers/main/quantization/overview. Search date: 15.12.2024
33. Ikawrakow 2024. SOTA 2-bit quants. Github. Available in: https://github.com/ggerganov/llama.cpp/pull/4773. Search date: 15.12.2024
34. Meta 2024. Introducing Llama3.1: Our most capable models to date. Meta. Available in: https://ai.meta.com/blog/meta-llama-3-1/. Search date: 22.12.2024
35. Bartowski 2024. bartowski/Meta-Llama-3.1-8B-Instruct-GGUF. Hugging Face. Available in: https://huggingface.co/bartowski/Meta-Llama-3.1-8B-Instruct-GGUF. Search date: 23.12.2024
36. Hugging Face. Flash attention. Hugging Face. Available in: https://huggingface.co/docs/text-generation-inference/conceptual/flash_attention. Search date: 23.12.2024
37. EnglishForEveryone. Reading comprehension worksheets. EnglishForEveryone. Available in: https://englishforeveryone.org/Topics/Reading-Comprehension.html. Search date: 23.12.2024
38. Jere Muikku (Fobban12). Thesis questions. Github. Available in: https://github.com/Fobban12/quantization_thesis_tests/tree/main/Thesis%20questions.
39. Jere Muikku (Fobban12). Thesis results. Github. Available in: https://github.com/Fobban12/quantization_thesis_tests/tree/main/Thesis%20testing%20results