BACHELOR'S THESIS 2024

Quantization of Large Language Models and the Impact on Output Performance

Robin Baki Davidsson

ISSN 1650-2884
LU-CS-EX: 2023-79

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

KANDIDATARBETE
Datavetenskap

LU-CS-EX: 2023-79

Quantization of Large Language Models and the Impact on Output Performance

Kvantiseringspåverkan på stora språkmodeller

Robin Baki Davidsson

Quantization of Large Language Models and the Impact on Output Performance

Robin Baki Davidsson
ro5226ba-s@student.lu.se

October 16, 2024

Bachelor's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Examiner: Noric Couderc, noric.couderc@cs.lth.se

Abstract

Recent advancements in Large language models (LLMs) have demonstrated remarkable capabilities in natural language processing tasks. Their substantial size often presents challenges for deployment on resource-constrained devices. This necessitates efficient techniques for model compression, with quantization emerging as a prominent solution. Despite its benefits, the impact of quantization on the performance and accuracy of large language models remains a question. Specifically, how much does weight quantization impair these models?

This thesis investigates the performance of eight large language models at various quantization levels, focusing on tasks such as MMLU-Pro for knowledge and reasoning, CRUXEval for code comprehension, and MuSR for reading comprehension and coherence. The results demonstrate a consistent trend where higher bit precision (e.g., 8-bit Q8_0) yields improved performance, albeit with diminishing returns. Aggressive quantization (e.g., 2-bit Q2_K) usually retains acceptable accuracy, though some models may occasionally lose substantial performance and coherence.

The findings indicate that while lower bit precision generally reduces performance, the impact varies across models and tasks. Larger models show greater resilience to aggressive quantization but can still experience significant performance drops at lower precision levels. Mid-sized models in the 7-9 billion parameter range strike an optimal balance between capability, efficiency, and resource usage. These results provide valuable insights into the trade-offs between model size, quantization, and task performance in real-world applications.

Keywords: large language models, quantization, model performance, benchmarking, efficiency

Acknowledgements

I would like to express my sincere gratitude to my supervisor at LTH, Pierre Nugues. His expertise in guiding me through the writing and structuring of my thesis has been invaluable. The feedback and support he provided significantly enhanced both the clarity and coherence of my work. His guidance was crucial in shaping this project into its final form, and I am deeply thankful for his time and dedication.

Contents

1 Related Work **7**
1.1 Quantization . . . 7
1.2 Application . . . 8

2 Background **9**
2.1 Autoregressive Models . . . 9
2.1.1 Mathematical Definition . . . 9
2.1.2 Perplexity . . . 10
2.2 Tokens . . . 10
2.2.1 n-grams . . . 10
2.2.2 Context Window . . . 11
2.3 Weights . . . 12
2.3.1 Weight In Model Structures . . . 13
2.4 Decoders . . . 15
2.4.1 Positional Encoding . . . 16
2.4.2 Activation Functions . . . 16
2.4.3 Self-Attention . . . 16
2.4.4 Feedforward Neural Network . . . 17
2.4.5 Decoder Usage Of Tokens . . . 17
2.5 Float16 and bfloat16 . . . 17
2.5.1 Float16 . . . 18
2.5.2 Bfloat16 . . . 18
2.6 Weight Quantization . . . 18
2.6.1 Uniform And Non-Uniform Quantization . . . 19
2.6.2 Quantizing The Weights . . . 19
2.7 Llama.cpp & GGUF . . . 20

3 Models **21**
3.1 Llama . . . 21
3.2 Gemma . . . 22
3.3 Phi . . . 22
3.4 Mistral . . . 23

4 Datasets **24**
4.1 MMLU-Pro . . . 24
4.2 CRUXEval . . . 25
4.3 MuSR . . . 26
4.4 Wikitext . . . 26

5 Methodology **28**
5.1 Approach . . . 28
5.1.1 Dataset Accuracy . . . 28
5.1.2 Few-Shot Prompting . . . 28
5.1.3 Chain-of-Thought (CoT) . . . 29
5.1.4 8-bit as Baseline . . . 29
5.2 Implementation . . . 29
5.2.1 Code . . . 30
5.2.2 Perplexity Tool . . . 30
5.2.3 Settings . . . 30
5.2.4 Prompt Input . . . 30

6 Evaluation **32**
6.1 Results . . . 32
6.1.1 The Gemma Family . . . 32
6.1.2 The Llama 3 Family . . . 35
6.1.3 The Phi 3 Family . . . 40
6.1.4 Mistral . . . 46
6.1.5 Model Efficiency . . . 51
6.2 Discussion . . . 53
6.2.1 Outliers . . . 53
6.2.2 Model Specific Performance . . . 53
6.2.3 Task Specific Insights . . . 54
6.2.4 Perplexity and Task Performance Correlation . . . 55
6.2.5 Implications for Large-Scale Models . . . 55
6.2.6 Diminishing Efficiency . . . 55

7 Conclusion **56**

References **59**

Chapter 1
Related Work

This section aims to offer an overview of existing research efforts in the field of large language model (LLM) quantization, alongside related highlight in text generation techniques. Through this, we seek to highlight the current methodologies and challenges within LLM quantization, setting a solid foundation for our work. The focus will be on both the complexity involved in achieving effective quantization and their implications on model performance.

1.1 Quantization

The paper by Kim et al. (2024) introduces GPTQ, an innovative post-training quantization method that enables highly accurate compression of Transformer based models such as OPT-175B and BLOOM-176B down to 3 or 4 bits per parameter within approximately four GPU hours. This approach surpasses existing methods in efficiency and accuracy, allowing for the first execution of a 175 billion parameter model on a single high-end GPU, like NVIDIA A100, achieving up to 4.5x speedups compared to FP16 with more cost-effective GPUs (NVIDIA A6000). Kim et al. (2024) further demonstrates the robustness of their method under extreme quantization conditions, such as 2-bit, setting a new benchmark in the field. The work make these computationally intensive models more accessible without significant loss in performance, marking a substantial advancement towards wider adoption.

Gong et al. (2024) presents an analytical framework termed 'the lens of perturbation' which views quantization as the addition of small disturbances to the model weights and activations. Through this perspective, Gong et al. (2024) conduct experiments with various artificial disturbances on LLMs across different scales and families (LLAMA, BLOOM, and OPT), revealing that performance degradation varies significantly based on the magnitude of values within weights and activations, and highlights the robustness issues faced by uniform quantization due to its equal treatment of all elements without considering their sensitivity. Building upon these insights, the paper demonstrates the potential of non-uniform quantization techniques that can significantly reduce quantization error by leveraging an understanding of disturbances-friendly properties within LLMs.

Jin et al. (2024) discusses the advantages and disadvantages of LLMs, particularly focusing on their performance across various benchmarks. It highlights that increasing model size enhances task performance but also significantly increases computational and memory

requirements.

Quantization techniques, which reduce the precision needed for weights or activations with minimal loss in performance, are presented as a potential solution. However, Jin et al. (2024) states that previous studies have primarily focused on pre-trained models. The authors introduce a comprehensive evaluation framework that assesses knowledge, capacity, and efficiency to evaluate the impact of quantization on instruction-tuned LLMs.

Through experimentation with ten benchmarks, they discover that 4-bit quantized models can maintain performance comparable to their non-quantized counterparts. Additionally, larger parameter scales in quantized models tend to yield better outcomes (Jin et al. 2024). The study also identifies perplexity as a reliable indicator of quantized LLMs performance across various benchmarks.

1.2 Application

Nejjar et al. (2024) study the application of LLMs in scientific research, focusing on software engineering tasks such as generating code for applications and developing scripts for data analytics. Amidst the growing interest in generative AI tools like ChatGPT, which have been embraced by researchers for various purposes including refining text and troubleshooting code, the study presents how these LLMs can support core scientific activities. By examining a range of use cases related to coding, such as writing software prototypes, conducting data analysis, and visualizing data, Nejjar et al. (2024) highlights both the promise and limitations of current LLM-based tools. Notably, while these models show potential in enhancing productivity and providing support for scientific tasks, challenges such as dialog where models generate potentially incorrect information are identified as significant concerns that could compromise the integrity of research outputs. The findings underscore the need for further exploration into the effective integration of LLMs within the scientific process.

Tonmoy et al. (2024) explore the issue of hallucination, where models generate content that appears factual but is fundamentally flawed. This problem is particularly concerning for critical applications like medical or legal advice, where errors can have severe consequences. The authors provide a broad overview of more than thirty-two techniques aimed at reducing these hallucinations, emphasizing approaches such as Retrieval-Augmented Generation (RAG), Knowledge Retrieval, CoNLI, and CoVe. Tonmoy et al. (2024) also propose a detailed classification system to categorize these mitigation methods based on factors like dataset usage, common tasks, feedback mechanisms, and retriever types. This classification system highlights the diverse strategies designed to address hallucinations in LLMs, while acknowledging the challenges and limitations of existing techniques.

Chapter 2
Background

I first present the fundamental principles and techniques used, including how we utilize bits, represent text as tokens, define the context window, structure the underlying model, and apply k-means quantization. Throughout the chapter, illustrative examples using hypothetical values clearly present these concepts.

I then describe Llama.cpp and the GGUF file format tools used in LLM deployment. Python bindings for Llama.cpp enable efficient execution and interaction with LLMs. GGUF offers a compact format for model storage and quantization.

This chapter lays the groundwork for understanding the underlying structure of LLMs and their key components, including their essential components, quantization techniques, and associated software framework. Understanding the basics of the components is essential for comprehending how weights are distributed within the model, as some weights may be more or less important depending on their location and will be affected differently. It is also crucial to identify where information loss occurs.

2.1 Autoregressive Models

Autoregressive models are probabilistic machine-learning models designed specifically for sequence prediction. They function under the assumption that future elements in a sequence can be predicted based on an analysis of preceding elements.

2.1.1 Mathematical Definition

The predictability is mathematically represented by:

$$p(x_i|x_{i-1}, x_{i-2}, ..., x_1), \tag{2.1}$$

where $x_{i-1}, x_{i-2}, ..., x_1$ denotes the sequence of past elements. Essentially, these models are adjusted to map context into a probability distribution over all potential future values. Autoregressive models form the fundamental basis for text generation in most modern LLMs.

2.1.2 Perplexity

Building upon the predictive capabilities of autoregressive models, we introduce the concept of perplexity as a critical evaluation metric, as described by Zhong, D. Wang, and Miao (2018). Perplexity quantifies how effectively a language model predicts text corpora. It does this by calculating the uncertainty associated with the predictions given the preceding context (Kuribayashi et al. 2021).

A lower perplexity score indicates that the model more accurately anticipates the next token in a sequence, signifying stronger language understanding and generation abilities (Zhong, D. Wang, and Miao 2018). For N previous tokens, the formula for perplexity is expressed as:

$$\mathrm{PPL} = \prod_{i=0}^{N} p(w_i|w_{<i})^{-\frac{1}{N}} \tag{2.2}$$

Equation 2.2 is derived from entropy in information theory (Kuribayashi et al. 2021). This metric provides valuable insights into the model's predictive capabilities and overall performance.

2.2 Tokens

As presented in Section 2.1, the next prediction is dependent on the previous elements. Applying this to previous elements in natural language processing (NLP) involves dividing raw text input into smaller units called tokens (Mistral n.d.). These tokens can be individual words, parts of words, characters, subwords, non-English characters, or an arbitrary symbol. This conversion allows computers to 'understand' text by transforming it into integers that represent distinct elements from the original string.

2.2.1 n-grams

A crucial concept in NLP is n-grams. These are defined as a continuous sequence of n symbols in a given corpus (Shannon and Weaver 1949). The terms 'unigram', 'bigram', and 'trigram' is used to denote the first three n-grams (Shannon and Weaver 1949). Table 2.1 provides an illustration showing that a greater degree of characters (increased n values) requires fewer tokens for the same information. The lower n-grams are typically included as a singular set of tokens, as demonstrated in Table 2.2. Once identified, these n-grams are utilized as the tokens for LLMs.

When working with tokens as input to models, the tokenizer is typically predetermined unless a new tokenizer is required (Mistral n.d.). The selection of tokenizer or tokenization method is determined by the specific NLP task and the characteristics of the text being processed.

Figure 2.1 shows the segmentation of 'Hello, World!' into tokens at a subword level. In this example, "He", "ll", "o", ",", "Wor", "ld", "!" were split as individual token, each transformed into a numerical representation based on the decisions of the tokenizer.

	Unigram	Bigram	Trigram
Token 1	H	He	Hel
Token 2	e	el	ell
Token 3	l	ll	llo
Token 4	l	lo	
Token 5	o		

Table 2.1: Example of unigram, bigram, and trigram breakdown for the word *Hello* as three sets of tokens.

	n-gram
Token 1	H
Token 2	e
Token 3	l
Token 4	l
Token 5	o
Token 6	He
Token 7	el
Token 8	ll
Token 9	lo
Token 10	Hel
Token 11	ell
Token 12	llo

Table 2.2: Example of unigram, bigram, and trigram of the word *Hello* as one set of tokens.

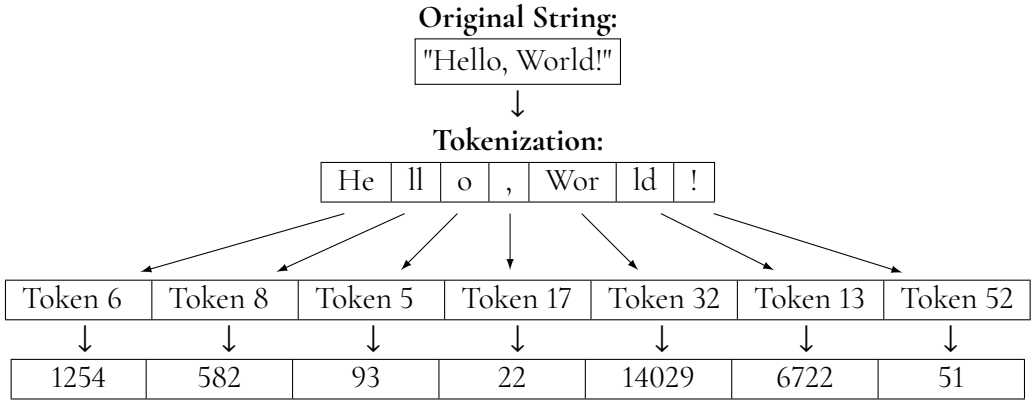


Figure 2.1: Example of the tokenization of a string.

2.2.2 Context Window

The concept of 'context window' refers to the count of preceding tokens utilized by the model for making future predictions (Dunn 2023). The calculation works by having text processed sequentially within a fixed-size context window. When the context exceeds the predefined limit set by the context window (maximum tokens for a specific model), the context window is shifted further along the text. As new tokens are processed, the oldest token within the window is dropped, and the most recently predicted token is added into it. This sliding window mechanism allows for longer text sequences than the context window size to be handled by the model (Dunn 2023).

Figure 2.2 provides a visual representation of the sliding window process. Three sequential steps are utilized by the figure to illustrate how a context window, with a capacity of **5**

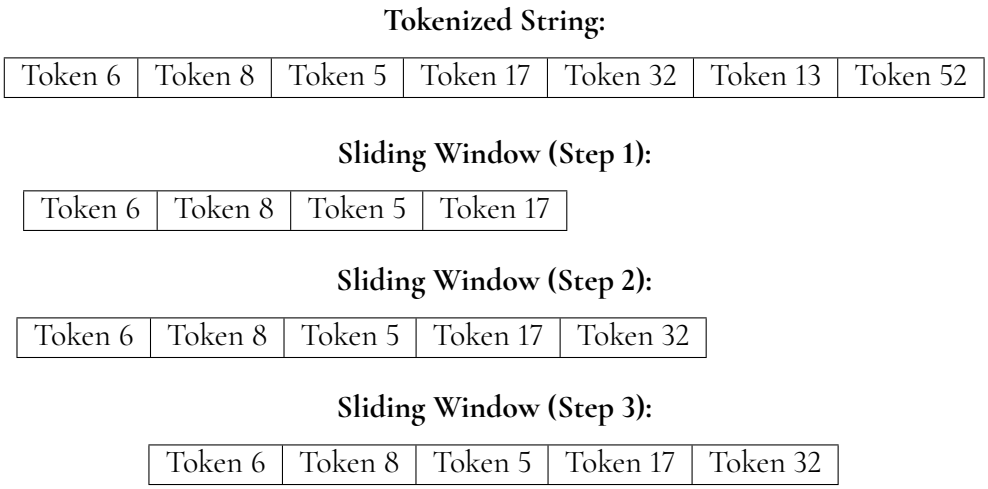


Figure 2.2: The sliding window process with a window context size of 5 Tokens.

tokens, is traversed across a sequence of tokens. The first step is shown where the window is positioned over the first four tokens of the sequence. One position to the right is shifted by the window in the second step, which capture the maximum number of tokens allowed within it. In the final step, the size of the window is maintained while its movement to the right continues, the leftmost token is discarded from the context, and a new token is appended to the right side of the sequence. This process demonstrates how sequential information is captured by the sliding window in relation to the context size.

2.3 Weights

Weights, also known as parameters, are the fundamental structure within autoregressive models, including LLMs. When discussing the size of a model, it specifically refers to the quantity of weights that are contained within the model. The model's output is determined by these weights, which respond to the provided input. Notably, 'open-weight models' are defined as those for which the weights have been made publicly available, allowing for local usage without dependence on an online service.

During the training process, the weights are iteratively updated to find the optimal function that fits the data. The weights are adjusted to minimize the difference between the model's predicted output and the target output based on a predetermined optimization algorithm. The adjustment process is responsible for refining the weights. These optimized weights are effective in capturing the underlying structure of languages, allowing the LLM to generate text that is both grammatically correct and semantically meaningful.

The notation for writing the probability, as seen in Equation 2.1, is rewritten using weights as the function

$$\hat{x}_i = f(x_{i-1}, x_{i-2}, ..., x_1|W), \tag{2.3}$$

the function is designed to take the past elements $x_{i-1}, x_{i-2}, ..., x_1$ as input and make the prediction $\hat{x}_i$, where W represents the weights.

2.3.1 Weight In Model Structures

State-of-the-art components have been increasingly adopted by various models such as newer activation functions and positional encodings. However, the overall concept has remained consistent with the descriptions provided in previous sections. The architectural specifications of a few model variants are detailed in the following subsections, including Gemma 2 and Llama 3, highlighting their similarities to earlier architectures. The original model by Vaswani et al. (2017) deviates from other models because it uses an encoder-decoder architecture, making direct comparisons difficult.

The Original

The original model is an encoder-decoder as described in *Attention is All You Need* by Vaswani et al. (2017). The authors presented multiple variations of the model, covering a range of parameter counts. Table 2.3 lists the decoder properties of the base model and top-performing model. For more details on encoders and these models, see Vaswani et al. (2017).

Model Architecture	**Base** (65M)	**Large** (213M)
Layers	6	6
Model Dimension	512	1024
Attention Heads	8	16
Key/Value Heads	-	-
Activation Function	ReLU	ReLU
Vocabulary Size	37000	37000
Positional Embeddings	Sine and cosine	Sine and cosine

Table 2.3: Encoder-Decoder model specifications from *Attention is All You Need*.

The base model, with a parameter count of 65M, consists of 6 layers and has a model dimension of 512. It employs 8 attention heads. The activation function used in this architecture is ReLU, and the vocabulary size is set to 37000. Positional embeddings are implemented using sine and cosine functions.

The large model, with a parameter count of 213M, also consists of 6 layers but has a larger model dimension of 1024. This configuration utilizes 16 attention heads. Similar to the base model, it uses ReLU as the activation function and maintains the same vocabulary size of 37000. Positional embeddings are again implemented using sine and cosine functions.

Llama 3

Table 2.4 shows the Llama 3's 8B, 70B, and 405B model structures (Dubey et al. 2024). The table details various architectural components across these models.

The activation function used in all the architectures is SwiGLU, the vocabulary size is set to 128000, and the key/value heads is 8. Positional embeddings are implemented using RoPE (Su et al. 2023) with a base frequency hyperparameter $\theta = 500000$.

In the 8B model, the number of layers is set to 32, with a model dimension of 4096. 32 attention heads and 8 key/value heads are employed in this configuration.

Model Architecture	8B	70B	405B
Layers	32	80	126
Model Dimension	4096	8192	16384
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Activation Function	SwiGLU		
Vocabulary Size	128000		
Positional Embeddings	RoPE($\theta = 500000$)		

Table 2.4: The Llama 3 architecture. RoPE is Rotary Position Embeddings. The θ in RoPE is the base frequency hyperparameter.

For the 70B model, the number of layers increases to 80, and the model dimension expands to 8192. Additionally, the number of attention heads is doubled to 64.

The largest configuration, the 405B model, features 126 layers and a substantial model dimension of 16384. This structure utilizes 128 attention heads.

Gemma 2

Table 2.5 shows the Gemma 2's 2B, 9B, and 27B model structures (Gemma Team et al. 2024). The table details various architectural components across these models.

Model Architecture	2B	9B	27B
Layers	26	42	46
Model Dimension	2304	3584	4608
Attention Heads	8	16	32
Key/Value Heads	4	8	16
Activation Function	GeGLU		
Vocabulary Size	256128		
Positional Embeddings	RoPE($\theta = 10000$)[1]		

Table 2.5: The Gemma 2 architecture.

The activation function used in this architecture is GeGLU, and is shared across all sizes. Positional embeddings are implemented using RoPE and the value of $\theta = 10000$. The vocabulary size is set to 256128 for all sizes.

In the 2B model, the number of layers is set to 26, with a model dimension of 2304. 8 attention heads and 4 key/value heads are employed in this configuration.

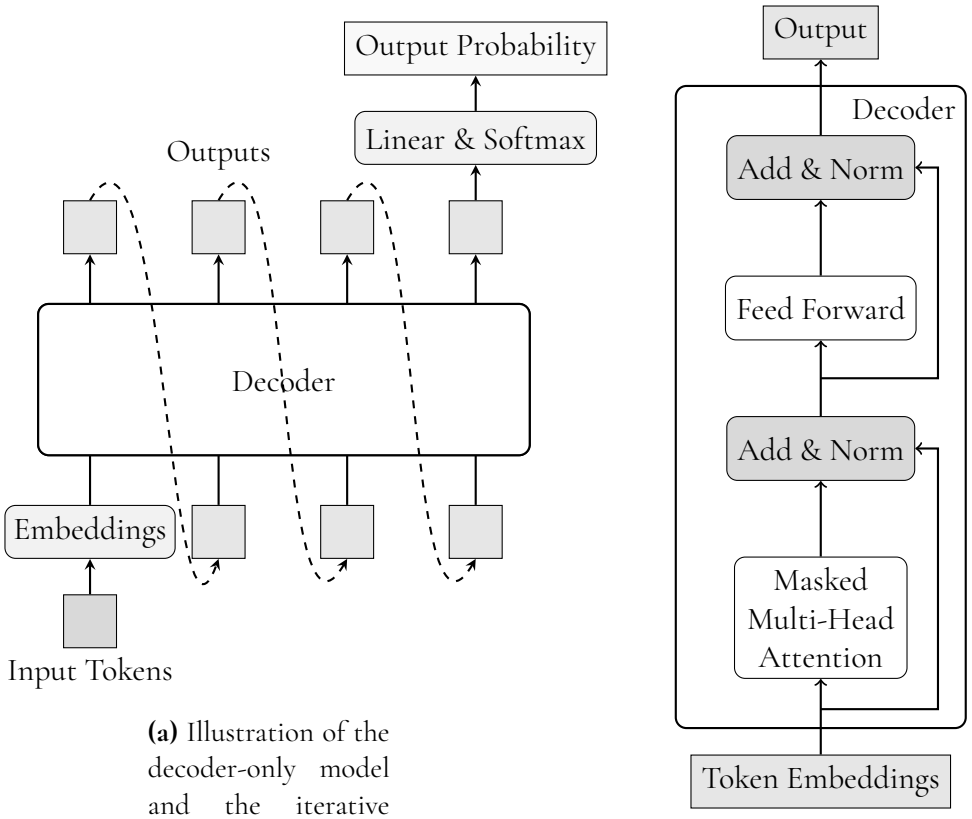
For the 9B model, the number of layers increases to 42, and the model dimension expands to 3584. Additionally, the number of attention heads is doubled to 16, while the key/value heads also increase to 8.

The largest configuration, the 27B model, features 46 layers and a substantial model dimension of 4608. This structure utilizes 32 attention heads and 16 key/value heads.

[1] https://huggingface.co/google/gemma-2-9b/blob/main/config.json

2.4 Decoders

All the large language models in this thesis have been built on the concept of transformers, an innovative architecture introduced by Vaswani et al. (2017). The transformer has been described as a combination of two main components, the encoder, and the decoder. The encoder processes the input data. However, it is through the decoder that the crucial role in generating the predicted output is made (Liu et al. 2018). This component is the autoregressive part of the transformer, where the predicted output is generated based on the previous tokens in the context window. LLMs built on the decoder architecture are known as decoder-only models (Liu et al. 2018). The decoder-only models are composed of multiple components in the form of dense vectors and matrices. Figure 2.3 is provided to illustrate the concept of the decoder-only model.



(a) Illustration of the decoder-only model and the iterative process of multiple decoder blocks.

(b) Illustration of the decoder block.

Figure 2.3: Illustration of the Decoder Model and Decoder Block. Inspired by the works of Vaswani et al. (2017). Each iterative input in 2.3a is processed as shown in 2.3b.

The mechanisms described in Sections 2.4.1, 2.4.2, 2.4.3, and 2.4.4 collectively enable the decoder to effectively generate contextually relevant outputs. These are all critical components in large language models.

2.4.1 Positional Encoding

Positional encoding (or embedding) is utilized to capture the ordering and sequential information within the input sequence (Vaswani et al. 2017). This is a crucial step for the LLM to understand the order of tokens in the input. Positional encoding is added to the input before it is fed into the transformer layers.

Positional encoding PE for a given position pos and dimension d_{model} is described by Vaswani et al. (2017) as

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \tag{2.4}$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \tag{2.5}$$

where pos is the index in the sequence, and i is a range from 0 to $\frac{d_{model}}{2} - 1$. Thus, each token in a sequence will receive a positional embedding vector of length d_{model}.

2.4.2 Activation Functions

Non-linearity is introduced into neural networks through the activation function (Vaswani et al. 2017). Functions such as the Rectified Linear Unit (ReLU) or the softmax function are typically used to accomplish this (Glorot, Bordes, and Bengio 2011). Equation 2.6 defines the ReLU function for $x \in \mathbb{R}$, while Equation 2.7 defines the softmax function.

$$\text{ReLU}(x) = \max(0, x) \tag{2.6}$$

$$\text{softmax}(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{2.7}$$

where $Z = XW^T$, W is a weight matrix, and the input vector X.

The crucial non-linear transformations are enabled by these functions, allowing models to learn complex patterns and relationships within the data (Glorot, Bordes, and Bengio 2011).

2.4.3 Self-Attention

The relationship and contextual information between different tokens in the sequence are captured by self-attention (Vaswani et al. 2017). Multiple self-attentions, known as multi-head attention, are usually present to capture various aspects of context. The attention matrix is calculated as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where the values of Q, K, and V are defined as the query, key, and value matrices, respectively. The matrices are calculated from the input X and the respective matrix, shown in Equation 2.8, 2.9, and 2.10, utilizing the weight matrices W_Q^T, W_K^T, W_V^T respectively.

$$Q = XW_Q^T \tag{2.8}$$
$$K = XW_K^T \tag{2.9}$$
$$V = XW_V^T \tag{2.10}$$

Multi-head attention allows for capturing a diverse range of contextual relationships by computing multiple attention heads in parallel, each providing different perspectives on the input data.

2.4.4 Feedforward Neural Network

A feedforward neural network (FFN) passes information sequentially from the input layer to the output layer without looping back (Vaswani et al. 2017). These networks are usually coupled with activation functions to model complex relationships (Glorot, Bordes, and Bengio 2011). The FFN utilizes the ReLU activation function in Equation 2.6, and is expressed as

$$\text{FFN}(X) = \max(0, XW_1^T + b_1)W_2^T + b_2,$$

where X is the input vector, and W_1, W_2, b_1, b_2 are weight and bias matrices.

2.4.5 Decoder Usage Of Tokens

In decoder-only models, special symbols such as 'Start Of Sequence' (SOS) and 'End Of Sequence' (EOS) are indicated by tokens that are utilized in a specific way (L. Wang et al. 2024). The beginning of a sequence is indicated by the SOS token, while the EOS token signifies its end. These special tokens provide information that guides the model's processing and generation. For example, the sequence 'I went to the store' is processed as '<SOS>I went to the store<EOS>'. Beside the indicators SOS and EOS, different use cases find utility in other indicators.

Different tokens might be defined uniquely by various models based on their specific tokenization schemes. For example, the start of a sequence might be denoted by a specific character string like '<SOS>', while another model such as Llama 3 might use '<|begin_of_text|>', or any other unique identifier that aligns with their specific training corpus.

2.5 Float16 and bfloat16

Floating-point numbers, or just floats, are used to represent all real numbers with a fixed number of bits. According to Ercegovac and Lang (2004), each floating-point number consists of three main parts: the sign bit, the exponent, and the mantissa (or fraction), with each part allocated a fixed number of bits. The sign bit indicates whether the number is positive or negative, with 0 representing positive and 1 representing negative. The exponent part determines the scale of the number by specifying a power of two to which the mantissa is multiplied. The mantissa holds the fractional digits of the number, effectively determining its precision. These components allow for an efficient and versatile bit representation of real numbers.

2.5.1 Float16

In the cases of float16 and bfloat16, 16 bits were used (Google 2024). The representation was achieved through the combination of an exponent and a mantissa, with the distribution of the bits illustrated in Figure 2.4. This distribution enables the representation of a wide range of both large and small numbers.

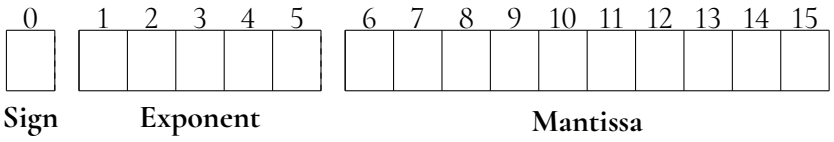


Figure 2.4: The bit distribution of the **16** bits in float16.

2.5.2 Bfloat16

The implementation of bfloat16, referred to as bfloat, aims to maintain low memory usage associated with 16-bit storage while preserving high exponential precision, albeit at the expense of reduced fractional precision. Figure 2.5 shows this bit distribution. The number of bits for the exponent is increased to 8 in the bfloat16 bit structure, aligning it with the number of exponent bits used in the larger float32 floating-point format.

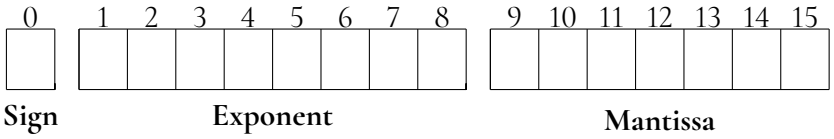


Figure 2.5: The bit distribution of the **16** bits in bfloat16.

2.6 Weight Quantization

Quantizing weights lowers the memory usage by reducing the number of stored bits. This conversion allows the models to fit into a smaller memory at the cost of lower precision. Lower precision weights also accelerate runtime by enabling faster computation and data transfer rates. This improvement is particularly beneficial for real-time applications where low latency is required.

Gong et al. (2024) state that the quantization of LLMs emerge as a crucial technique for addressing computational constraints. Furthermore, complexities such as wide parameter spaces demanding significant memory and processing power have been addressed by Gong et al. (2024). This requirement often necessitates high-end hardware for effective execution. Relying on high-end resources poses limitations to the deployment of models across diverse environments. Recognizing this challenge, quantization serves a crucial role in optimizing

models while preserving their essential functions (Gong et al. 2024). To ensure that LLMs become more adaptable and accessible, reducing the parameter precision is necessary.

When discussing the concept of quantization, post-training quantization (PTQ) has been specifically utilized as a straightforward process for quantization (Huang et al. 2024). Some advantages of PTQ include not requiring retraining and being computationally efficient, in comparison to Quantization-Aware Training (QAT) (Frantar et al. 2023).

2.6.1 Uniform And Non-Uniform Quantization

Uniform and non-uniform quantization are techniques used to reduce the precision of floating-point numbers by specifying how to distribute the bit representations. In uniform quantization, the weights are mapped to values, where the intervals between the values remain constant. This method is straightforward but may not efficiently represent the distribution of data if the data are not uniformly distributed.

In contrast, non-uniform quantization employs dynamically sized intervals that can better adapt to the distribution of the weights. By allocating more values where the density of weight values is higher and fewer where it is lower, this technique aims to minimize the quantization error. This approach results in a more accurate representation of the original data with reduced precision.

2.6.2 Quantizing The Weights

The process of quantizing LLM weights was found to be non-uniform and is based on the weight distribution within the layers. When n-bit quantization is performed, 2^n centroids are identified (Kim et al. 2024), which are the values that the weights will be mapped to.

A sensitivity-based k-means clustering method was employed to find non-uniform quantization, providing higher precision for sensitive weights (such as outliers). This resulting structure is termed a Dense-and-Sparse matrix, and this specific quantization method is referred to as k-quantization, also denoted as k-quant (Kim et al. 2024). The naming convention when using k-quantization is QN_K, where N represents the bit precision. For larger bit precisions, such as 8-bit and above, the k-means technique is not used.

To visualize the process, consider the following weight matrix in Equation 2.11

$$W = \begin{bmatrix} -2.5 & -1.8 & -0.5 & 0.0 & 0.3 & 1.2 & 1.9 & 2.4 & 2.7 \end{bmatrix}, \tag{2.11}$$

to quantize these weights to 2 bits. We have $n = 2$ and we determine 2^2 centroids through the application of k-means clustering. Figure 2.6 shows the clustering result.

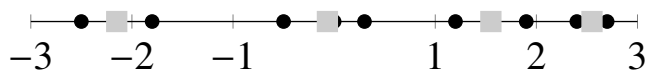


Figure 2.6: The weights are indicated as circles, and centroids as squares.

From Figure 2.6, the four centroids in Equation 2.12 are identified as

$$C = \begin{bmatrix} -2.15 & -0.066... & 1.55 & 2.55 \end{bmatrix}, \tag{2.12}$$

where $\boldsymbol{C}$ contains all the values that the weights can map to. Figure 2.7 shows the closest value for each weight in $\boldsymbol{W}$. The weights on the x-axis map to the closest centroid. After this process, we can store the weights as 2^n bit index values that map to the centroid values when used. In the previous example, using 4-bits, yields the indices $00, 01, 10, 11$ for the centroids in Eq. 2.12.

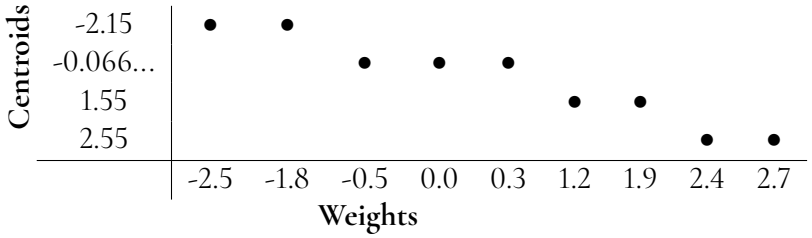


Figure 2.7: Mapping of the weights to the centroids.

2.7 Llama.cpp & GGUF

For the evaluation and execution of the LLMs, I used the llama.cpp software library (Gerganov 2024) along with the Python bindings provided by the llama-cpp-python library (Betlen 2024). This framework is particularly advantageous due to its utilization of the GGUF file format for model storage and quantization.

The GGUF format is a recently developed binary format made for efficient model representation and loading, particularly within llama.cpp and its library. The GGUF format offers several key benefits over previous formats such as GGML, GPTQ and AWQ. Some advantages are that GGUF encapsulates both tensor data (such as weight data) and metadata. Thus, delivering a thorough compact representation of the model. This simple storage enhances model management and usage. Furthermore, the binary nature of GGUF files ensures rapid model loading, which contributes to greater computational efficiency (HuggingFace 2024). GGUF is designed to address the limitations of previous formats, allowing for easier software extension without sacrificing backward compatibility. Lastly, GGUF supports various quantization types ranging from 1-bit to 8-bit (Gerganov 2024), and offers a flexible approach to balancing memory usage, processing speed, and model precision.

The adoption of GGUF enables the utilization of these benefits, particularly in terms of efficient model usage and the flexibility to run at different quantization levels. GGUF supports a wide variety of data types, including the standard floating-point formats and the specialized bfloat16 (HuggingFace 2024).

Chapter 3

Models

3.1 Llama

The release of the Llama 3 models by Meta (2024) marks a significant advancement in the field of open-weights LLMs. The models are available in two sizes, 8B and 70B. They are designed to be run locally, and to empower researchers and developers with powerful tools for exploring the capabilities of large language models. Smaller models like the 8B version can be deployed on less powerful hardware, making them more accessible for individual hobbyists, researchers or smaller institutions. The larger 70B model, while requiring significant computational power, offers enhanced performance and capabilities suitable for tasks demanding higher accuracy and complexity.

Table 3.1 shows the quantization bit sizes to be utilized, along with their corresponding file sizes. Additionally, the table presents the settings that will be used for the evaluation process.

Models	Parameters	Quantization (File Size)	Context Length	Max Tokens Out	Few-Shot
Llama 3 Instruct	8B	Q8_0 (8.5GB), Q6_K (6.6GB), Q4_K (5.0GB), Q3_K (4.0GB), Q2_K (3.2GB)	4096	2048	MMLU-Pro: 4 CRUXEval: 1 MuSR: 1
Llama 3 Instruct	70B	Q8_0 (75GB), Q4_K (42.5GB), Q2_K (26.4GB)	4096	2048	MMLU-Pro: 1 CRUXEval: 1 MuSR: 0

Table 3.1: The Llama Family specification.

3.2 Gemma

Google released one model at first containing 2B parameters (Banks and Warkentin 2024), named Gemma 2B, according to Banks and Warkentin (2024), the model is the state-of-the-art in its size bracket. Later, Farabet and Warkentin (2024) expanded the Gemma family by introducing two larger models: Gemma 2 9B and Gemma 2 27B (Farabet and Warkentin 2024). By selecting these recently released models, all originating from the same family, a comprehensive overview of their performance across a wide range of model sizes can be achieved. This approach allows for a more detailed comparison and understanding of how different model sizes perform under various conditions, providing valuable insights into their scalability and effectiveness.

Table 3.2 provides a comprehensive overview of the quantization bit sizes, associated file sizes, and the settings that will be used.

Models	Parameters	Quantization (File Size)	Context Length	Max Tokens Out	Few-Shot
Gemma v1.1 Instruct	2B	Q8_0 (2.7GB), Q6_K (2.1GB), Q5_K (1.8GB), Q4_K (1.6GB), Q3_K (1.4GB), Q2_K (1.2GB)	4096	2048	MMLU-Pro: 4 CRUXEval: 1 MuSR: 1
Gemma 2 Instruct	9B	Q8_0 (9.8GB), Q6_K (7.6GB), Q4_K (5.8GB), Q3_K (4.8GB), Q2_K (3.8GB)	4096	2048	MMLU-Pro: 4 CRUXEval: 1 MuSR: 1
Gemma 2 Instruct	27B	Q8_0 (28.9GB), Q5_K (19.4GB), Q3_K (13.4GB), Q2_K (10.4GB)	4096	2048	MMLU-Pro: 1 CRUXEval: 1 MuSR: 0

Table 3.2: The Gemma Family specification.

3.3 Phi

Another recently released family of models is the Phi-3 Family, which comes in two sizes: Phi-3-small with 7B parameters and Phi-3-medium with 14B parameters (Bilenko 2024). According to Bilenko (2024), these models represent the state-of-the-art for small models within their respective size brackets. This makes them an excellent choice for comparison.

Table 3.3 presents the quantization bit sizes used and their respective file sizes.

Models	Parameters	Quantization (File Size)	Context Length	Max Tokens Out	Few-Shot
Phi 3 Mini Instruct	4B	Q8_0 (4.1GB), Q6_K (3.1GB), Q4_K (2.4GB), Q3_K (2.0GB), Q2_K (1.4GB)	4096	2048	MMLU-Pro: 1 CRUXEval: 1 MuSR: 0
Phi 3 Medium Instruct	14B	Q8_0 (14.8GB), Q5_K (10.1GB), Q3_K (6.9GB), Q2_K (5.1GB)	4096	2048	MMLU-Pro: 4 CRUXEval: 1 MuSR: 0

Table 3.3: The Phi Family specification.

3.4 Mistral

Since its release, the Mistral 7B model (Jiang et al. 2023) has become the benchmark for high-performance open-weight models. It is frequently selected as a preferred choice for tasks and research involving smaller language models due to its size and output quality. Therefore, incorporating the results from this model represents a significant contribution to the evaluation results.

Models	Parameters	Quantization (File Size)	Context Length	Max Tokens Out	Few-Shot
Mistral 7B v0.3 Instruct	7B	Q8_0 (7.7GB), Q6_K (5.9GB), Q4_K (4.4GB), Q3_K (3.5GB), Q3_K (2.7GB)	4096	2048	MMLU-Pro: 1 CRUXEval: 1 MuSR: 0

Table 3.4: The Mistral Family specification.

Chapter 4

Datasets

The datasets selected for the evaluations are new, having been released after the cutoff date of the data used by the LLMs. This separation is intentional and serves a crucial purpose to minimizes the probability of these datasets being accidentally included in the training dataset of the LLMs. This is essential for maintaining the integrity of the evaluation process, as it ensures that the models are being tested on unseen data, thus providing a more accurate assessment of their capabilities and performance.

The main reason of why the datasets MMLU-Pro, CRUXEval and MuSR is selected is to represent three different use cases. The first is MMLU-Pro which focuses on the model's knowledge capability. The second is CRUXEval which evaluates the model's ability to understand code and the logic behind the code. The last dataset, MuSR, assesses the model's capability to understand long context texts and have a coherent understanding.

4.1 MMLU-Pro

The MMLU-Pro (Massive Multitask Language Understanding Pro) dataset is designed with two purposes, to challenge LLMs in both knowledge and reasoning capabilities, and to reduce the probability of models arbitrarily guessing correct answers (Y. Wang et al. 2024). Each question in this dataset is structured with a Chain-of-Thought (CoT) explanation and ten possible answer choices, promoting a more comprehensive evaluation of the models cognitive processes.

STEM	Social Sciences	Humanities & Other
Math	Health	Law
Physics	Business	Other
Biology	Economics	History
Chemistry	Psychology	Philosophy
Engineering		
Computer Science		

Table 4.1: The MMLU-Pro categories used in the dataset.

We will employ a subset of the MMLU-Pro dataset, this is due to computational and time

constraints, we will utilize the first **100** questions from each of the **14** categories, resulting in a total of **1400** questions. The **14** categories can be seen in Table 4.1.

While this subset is not as exhaustive as the full dataset, it still provides a robust and diverse sample for evaluating the LLM's performance across multiple domains. The breadth of categories ensures a comprehensive assessment of the models' capabilities in various fields of knowledge and reasoning tasks. This approach allows us to maintain a balance between the depth of evaluation and the practicality of implementation, while still adhering to rigorous standards of model assessment in natural language processing research.

Question	Choices	Answer	Category	CoT
The concentration ratio for a monopoly is	["50", "5", "10", "90", "15", "100", "0", "25", "75", "N/A"]	F	economics	Let's think step by step. We refer to Wikipedia articles on microeconomics for help. The concentration ratio is calculated as the sum of market share of a specific number of largest companies. Monopoly means one company or entity controls the entire market, therefore, the concentration ratio is 100 percent. The answer is (F).

Table 4.2: An example of the MMLU-Pro dataset. The input choices to the model will have a letter, A to J, next to the choice which it will reply with.

4.2 CRUXEval

The utilization of LLMs in scientific domains has experienced a significant surge, particularly in the use of code generation (Nejjar et al. 2024). To address this growing application, the CRUXEval (Code Reasoning, Understanding, and eXecution Evaluation) dataset was specifically developed to assess code comprehension and reasoning capabilities of LLMs (Gu et al. 2024).

The CRUXEval dataset comprises **800** Python functions, each accompanied by corresponding input-output pairs. This structure makes it an ideal candidate for evaluating an LLM's ability to understand code, process inputs, and predict accurate outputs. The evaluation protocol involves presenting the LLM with both a function and an input, then tasking it to generate the expected output. The LLM's generated output is subsequently compared against the ground truth output provided in the dataset.

This methodology allows for a rigorous assessment of an LLM's code comprehension and execution simulation understandings, providing valuable insights into its potential for practical applications in software development and scientific computing.

Context	Input	Answer
def f(text): nums = list(filter(str.isnumeric, text)) assert len(nums) >0 return ''.join(nums)	'-123 \t+314'	'123314'

Table 4.3: Example from the CRUXEval dataset.

4.3 MuSR

The MuSR (Multistep Soft Reasoning) dataset was specifically designed to challenge the reasoning capabilities of LLMs in complex scenarios without relying on CoT prompting (Sprague et al. 2024). The dataset is split in three parts: Object placements, team allocation and murder mysteries. This evaluation technique contains narrative texts of approximately 1000 words each, accompanied by related questions and multiple-choice answers.

The dataset's structure makes it particularly suitable for studying the LLM's text and reading comprehension abilities. By presenting long-form narratives followed by questions, the MuSR dataset evaluates the model's capacity to:

- Retain and process substantial amounts of contextual information
- Draw inferences from complex narrative structures
- Formulate responses based on a comprehensive understanding of the text

This approach provides valuable insights into the limitations and capabilities of LLMs in handling multistep reasoning tasks without explicit intermediate steps. As such, the MuSR dataset serves as a crucial tool for studying the cognitive abilities of language models in more realistic and open-ended scenarios.

Context	Choices	Answer
In the shimmering opulence of a luxury restaurant, Roderick's life was abruptly ended by a pistol's merciless blow; now Detective Winston must untangle the web of…	['Oscar', 'Brianna']	Brianna

Table 4.4: MuSR caption. The input choices to the model will have a letter, A or B, next to the choice which it will reply with.

4.4 Wikitext

For the assessment of model perplexities, we utilize the 'wikitext-2-raw-v1' dataset (Merity et al. 2016). Two primary considerations drive our choice. It comprises clean, verified texts extracted from Wikipedia articles, ensuring high quality, and well-structured content. This is further amplified with a diverse and extensive vocabulary range, as well as coverage of various topics and writing styles. It also has a high compatibility with the llama.cpp library, and is

one of the recommended datasets (Georgi Gerganov 2024). The combination of these factors makes 'wikitext-2-raw-v1' an ideal choice for our perplexity assessments.

Due to computational resource constraints, we specifically use the 'test' partition of the dataset, which contains approximately 4000 text samples. While this subset is smaller than the full dataset, it still provides a substantial and diverse corpus for perplexity evaluation. And by using high quality and diverse text samples, and utilizing them within an efficient framework, such as llama.cpp, we can obtain reliable perplexity measurements. These measurements serve as a key metric in evaluating the model's ability to predict and generate coherent text across a wide range of topics. This provides a standardized base for comparing the performance of the quantization levels of the different models.

Chapter 5
Methodology

Large language models (LLMs) enhance natural language processing but face adoption hurdles due to high computational demands and data security issues. Quantization can ease these constraints without sacrificing performance. Local applications such as doctors need secure and reliable LLMs for patient data, researchers require them for usage under strict privacy rules, and educators seek them for personalized learning with minimal resources. Local LLMs is crucial for confidential applications, as online services often transmit user data to external servers, posing security risks. This study explores how quantization affects LLM's ability to handle complex queries and provide accurate responses in resource-limited settings.

In evaluating the performance of LLMs, the methodology will focus on analyzing response structure and information accuracy. The model's ability to understand complex queries and generate answers that precisely follow the provided instructions is assessed. This approach aims to determine how well LLMs interpret questions and produce relevant responses according to the instructions, ensuring they meet the specified criteria.

5.1 Approach

5.1.1 Dataset Accuracy

In our evaluation of large language models, accuracy on the dataset refers to the proportion of correct predictions made by the model relative to the total number of questions within the dataset. This metric quantifies how effectively the LLM can provide accurate responses, with higher accuracy scores indicating greater precision in handling a variety of tasks and queries. Furthermore, The accuracy is used as a way to measure the resource efficiency in terms of GB used by dividing the accuracy by the model size in GB.

5.1.2 Few-Shot Prompting

The comprehension of logical reasoning and the implementation of a systematic methodology for completing a task often necessitate an inherent understanding of cognitive processes and prioritization strategies to effectively achieve the objective. This foundational notion underlies the aim of few-shot prompting (Brown et al. 2020). In this context, a LLM is pro-

vided with one or more examples of how previous tasks have been completed along with comprehensive guidelines. Subsequently, the LLM is tasked to undertake a similar assignment. This approach has demonstrated excellent increased performance across various datasets, as evidenced by empirical research (Brown et al. 2020).

5.1.3 Chain-of-Thought (CoT)

Reasoning poses a significant challenge for LLMs, often leading to inaccurate responses, including instances of hallucination (Tonmoy et al. 2024). To address this issue, Wei et al. (2023) introduced the Chain-of-Thought (CoT) prompting technique. CoT's structured approach encourages LLMs to engage in a methodical, step-by-step reasoning process when generating responses by providing thoughts examples in the previous few-shot prompts (Wei et al. 2023). This technique has shown to significantly enhance performance across various reasoning tasks (Wei et al. 2023), as it facilitates more thorough and logical deliberation in the formulation of answers. Only if the datasets contain CoT will it be used.

5.1.4 8-bit as Baseline

In the evaluation on the quantization levels, 8-bit quantization will be the primary benchmark instead of the traditional FP32 or BF16. This choice is based on recent findings in 8-bit quantization, particularly the works of Dettmers et al. (2022) and Jin et al. (2024).

In both studies, Dettmers et al. (2022) and Jin et al. (2024) demonstrated that 8-bit quantization can achieve performance metrics close to those obtained with 32-bit precision across a wide range of tasks and model architectures. These findings suggest that 8-bit representation preserves enough information for task evaluation, and are unlikely to have a significant loss of precision in practical applications. Both studies also highlight the practical benefits of using 8-bit quantization, including reduced memory usage and speed improvements. These advantages allow for efficiently performing the evaluations and tasks needed with the limited amount of resources available.

It is important to acknowledge that while 8-bit weight quantization is highly effective in many scenarios, it is still a form of quantization and therefore not perfect. The reduction from 32-bit or 16-bit to 8-bit precision in model weights can, in some cases, lead to a loss of information that affects model performance. For instance, Jin et al. (2024) demonstrated a slight degradation in perplexity on certain benchmarks when using 8-bit quantization compared to the full precision. However, the impact of these limitations on the selected evaluations is expected to be minimal. The focus on lower bit quantization levels means that the 8-bit quantization still serves as a suitable baseline, providing a meaningful reference point for comparing the performance of the lower precision quantization levels.

5.2 Implementation

The experimental setup utilizes diverse model sizes, exploration of quantization techniques, and usage of multiple datasets. This promises a comprehensive evaluation of LLM performance across various task types. The utilization of the llama.cpp library underlines the practical work in deploying the large language models.

5.2.1 Code

I released the software code for both the quantization procedure of the models and the performance evaluation. This code can be retrieved from a dedicated repository hosted on GitHub, found in Appendix E.

5.2.2 Perplexity Tool

The Llama.cpp[1] library conveniently provides tools specifically designed for calculating perplexity, as elaborated by Georgi Gerganov (2024). The description and detailed implementation is further elaborated upon in the community discussion (Georgi Gerganov 2023) and documented by Gäßler (2023). Gäßler (2023) writes detailed about the usage of the llama.cpp, and the metrics it provides. This work has significantly contributes to understanding the documentation on tool utilization and its importance.

When calculating the perplexity, the command in Listing 5.1 is used. This runs on the 'wikitext-2-raw' dataset.

Listing 5.1: The command used for perplexity calculations.

```
llama.cpp/llama-perplexity \
    -m {full_model_path} \
    -ngl {ngl} \
    -t 16 \
    -f wikitext-2-raw/wiki.test.raw
```

5.2.3 Settings

The few-shot settings for Llama 3 are detailed in Table 3.1, Gemma and Gemma 2 in Table 3.2, Phi 3 in Table 3.3, and Mistral in Table 3.4. If the dataset contains CoT instructions, they are used. These tables also include the quantization levels assessed. Notably, models with a higher parameter count have fewer quantization levels available, due to constraints related to resources and time. The computational specifications and software details can be found in Appendix F.

5.2.4 Prompt Input

In the evaluation, we use the chat completion API (OpenAI 2024), which is replicated by the Python bindings for llama.cpp (Betlen 2024). This approach structures the input as a series of messages, each with a designated role and its content. Some benefits of this structure include providing a clear framework for instructions and context for the LLM, as well as separating human inputs from the LLM by using the 'user' and 'assistant' roles

To illustrate this structure, consider the following example:

Listing 5.2: Example of the Chat Completion API messages input structure.

[1] https://github.com/ggerganov/llama.cpp

```
[
    {"role": "system", "content": "You are a helpful
       assistant."},
    {"role": "user", "content": "Why is the sky blue?"},
    {"role": "assistant", "content": "The sky appears
       blue due to a phenomenon known as Rayleigh
       scattering. This effect occurs when the Earth's
       atmosphere scatters sunlight in all directions."},
    {"role": "user", "content": "Can the sky be green?"},
    ...
]
```

This structure allows for more nuanced and context-driven interactions with the LLM. The system prompt sets the overall behavior and capabilities of the model, while alternating user and assistant messages simulate a conversation, allowing the model to maintain context across multiple interactions. You can find the system inputs in Appendix A, Appendix B, Appendix C, and Appendix D.

In the evaluation, we use customized prompts according to the specific requirements of each task:

- MMLU-Pro includes task-specific instructions in the system prompt, few-shot examples as previous conversations, and CoT for complete understanding.
- CRUXEval includes task-specific instructions in the system prompt, code snippets and input data in the user prompts, and expects the model to return the output of the code as the response.
- MuSR includes task-specific instructions in the system prompt, longer narrative texts in the user prompts, followed by comprehension questions.

This input structure enables customized inputs for each task's specific needs while using a uniform interaction method across all models and quantization levels. Furthermore, this experimental setup performs a comprehensive evaluation of the LLMs across various quantization levels and task types, providing insights into their performance characteristics and the impact of different optimization techniques.

Chapter 6
Evaluation

6.1 Results

The results are organized by their respective architectures, showing both the perplexity and accuracy scores for the different quantization levels. This setup make it easy to compare how each architecture performs with its various quantization levels. Note that the goal of the thesis is only to study how the weight quantization affects the individual models, and the results of each model should not be compared to the result of the other models. The models might be evaluated under the different settings as presented in the model family tables in Section 3 for each model. This study evaluated language models belonging to the Gemma, Llama 3, Phi 3, and Mistral families, with a parameter range from 2 billion to 70 billion.

6.1.1 The Gemma Family

Beginning with the Gemma family, the Gemma 2B v1.1 model demonstrated a range of perplexity scores from 30.02 to 39.86 across quantization levels (Table 6.1). Notably, the Q8_0 quantization achieved the lowest perplexity, indicating superior language modeling performance. The accuracy scores for this model varied across datasets, with MMLU-Pro scores ranging from 11.92% to 15.42%, MuSR scores between 37.52% and 40.69%, and CRUXEval scores from 22.72% to 26.72% (Table 6.2). The mean accuracy across all datasets peaked at 27.26% for the Q5_K quantization.

Gemma 2B v1.1

In Table 6.1, the perplexity values are presented along with their margins of error for each quantization level, which range from scores of 30.02 to 39.86. This trend suggests that higher precision quantization preserves more of the model's predictive capabilities.

Figure 6.2 presents the plot of each quantization level's respective performance across the multiple subjects in the MMLU-Pro evaluation. Both the MuSR result and the CRUXEval accuracy scores can be found presented in Figures 6.1a and 6.1b. The model shows particular strength in subjects like biology and psychology, while struggling more with fields like law, engineering, and chemistry.

Gemma 2B Perplexity

Quantization	Perplexity
Q2_K	39.86 ± 0.43
Q3_K	32.11 ± 0.35
Q4_K	30.91 ± 0.34
Q5_K	30.25 ± 0.33
Q6_K	30.16 ± 0.33
Q8_0	**30.02 ± 0.33**

Table 6.1: The Perplexity for each quantization with margins of error. Lower is better.

Gemma 2B v1.1

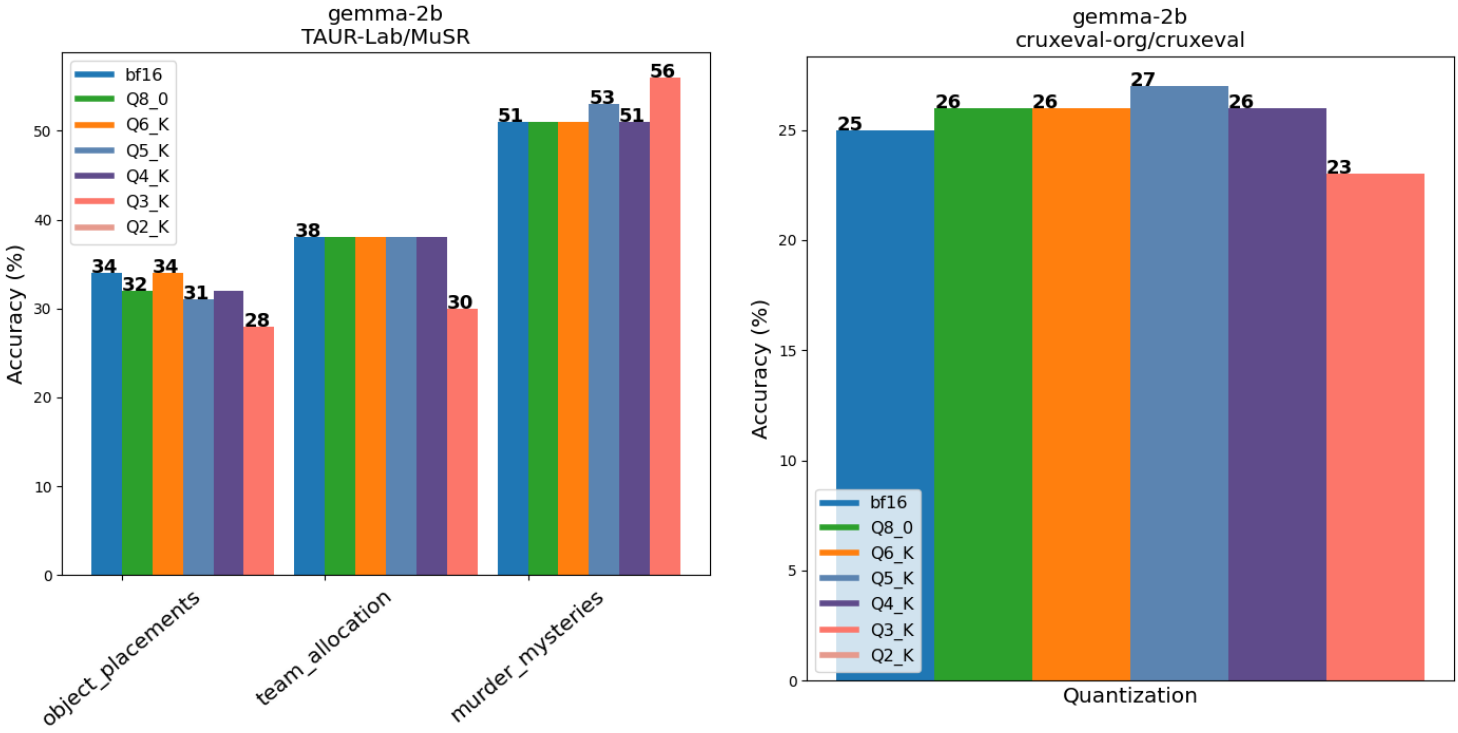
Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	11.92%	37.52%	22.72%	21.66%	**18.1**
Q3_K	13.78%	39.89%	26.47%	24.41%	17.4
Q4_K	14.49%	40.29%	**26.72%**	24.99%	15.6
Q5_K	15.13%	**40.69%**	25.97%	25.19%	14.0
Q6_K	**15.42%**	40.16%	25.97%	**25.24%**	12.0
Q8_0	14.78%	40.55%	25.22%	24.73%	9.2

Table 6.2: Comparison of performance metrics (in percentage) across the different quantization levels from multiple datasets, and their geometric mean values.

All the accuracy scores for this model varied across the datasets, with MMLU-Pro scores ranging from 11.92% to 15.42%, MuSR scores between 37.52% and 40.69%, and CRUXEval scores from 22.72% to 26.72% (Table 6.2). Interestingly, the mean accuracy across all datasets peaked at 25.24% for the Q5_K quantization, a slight improvement over Q8_0. This suggests that there may be an optimal quantization where the model maintains most of its performance while significantly reducing its size. The last column shows the performance efficiency of the model, it decreases from 18.1 for Q2_K to 9.2 for Q8_0.

Gemma 2 9B

The Gemma 2 9B model showed strong performance, with perplexity scores ranging from 8.82 to 10.16 (Table 6.3). This model showed more consistent performance across quan-



(a) The MuSR dataset.

(b) The CRUXEval dataset.

Figure 6.1: The rounded accuracy scores of the Gemma 2b v1.1 on the MuSR and CRUXEval datasets.

tization levels, with mean accuracy scores ranging from 39.8% to 41.93% (Table 6.4). The MMLU-Pro dataset saw particularly notable improvements, with scores reaching up to 45.61% for the Q4_K quantization.

In Figure 6.3, the accuracy scores are presented, MuSR in Figure 6.1a and CRUXEval in Figure 6.1b.

This model showed more consistent performance across the quantization levels, with mean accuracy scores ranging from 39.83% to 41.93% (Table 6.4). The MMLU-Pro dataset saw particularly notable improvements, with scores reaching up to 45.61% for the Q4_K quantization. This suggests that the larger model size allows for better retention of knowledge and reasoning capabilities even at lower quantization levels. The last column shows the performance efficiency of the model, it decreases from 10.5 for Q2_K to 4.3 for Q8_0.

In Figure 6.4, the accuracy scores are presented with the different quantization levels and different subjects of the MMLU-Pro dataset.

Gemma 2 27B

Gemma 2 27B, demonstrated the best performance in the Gemma family, with perplexity scores ranged from 7.19 to 9.17 (Table 6.5), with the Q8_0 quantization achieving the lowest perplexity.

The full table of all the accuracy evaluations across all the datasets can be seen in Table 6.6, with mean scores ranging from 42.92% to 48.91% (Table 6.6). The MMLU-Pro dataset saw scores up to 48.11%, while the CRUXEval dataset reached 54.56% accuracy for the

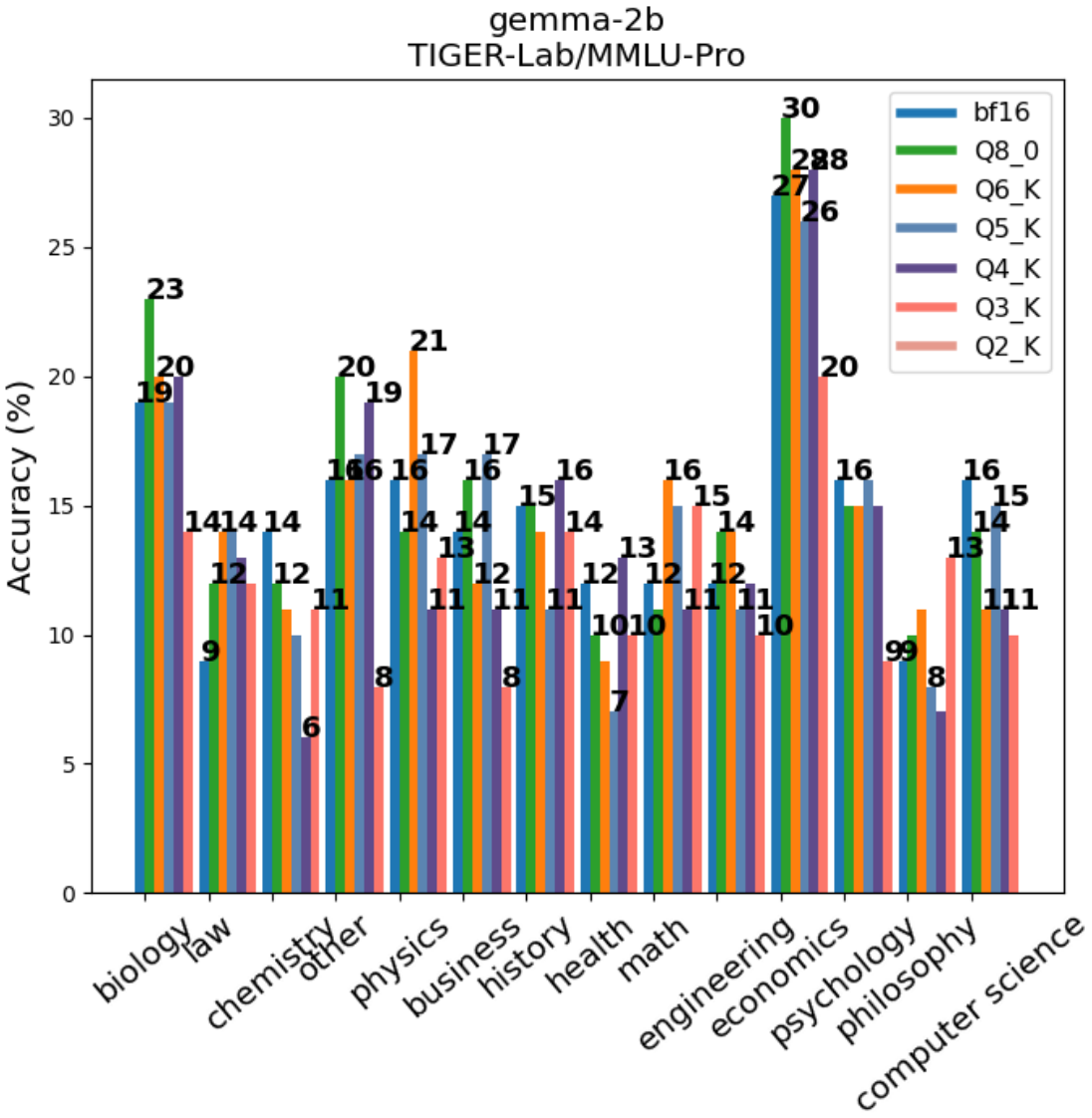


Figure 6.2: The rounded accuracy scores of the Gemma 2b v1.1 evaluation on the MMLU-Pro dataset.

Q5_K quantization. This performance leap underscores the significant impact of model size on task-specific capabilities. The last column shows the performance efficiency of the model, it decreases from 4.1 for Q2_K to 1.7 for Q8_0.

In Figure 6.5a, the evaluation scores are present on the MMLU-Pro dataset, and the graphs of the evaluation scores are presented for the MuSR and CRUXEval in Figure 6.5.

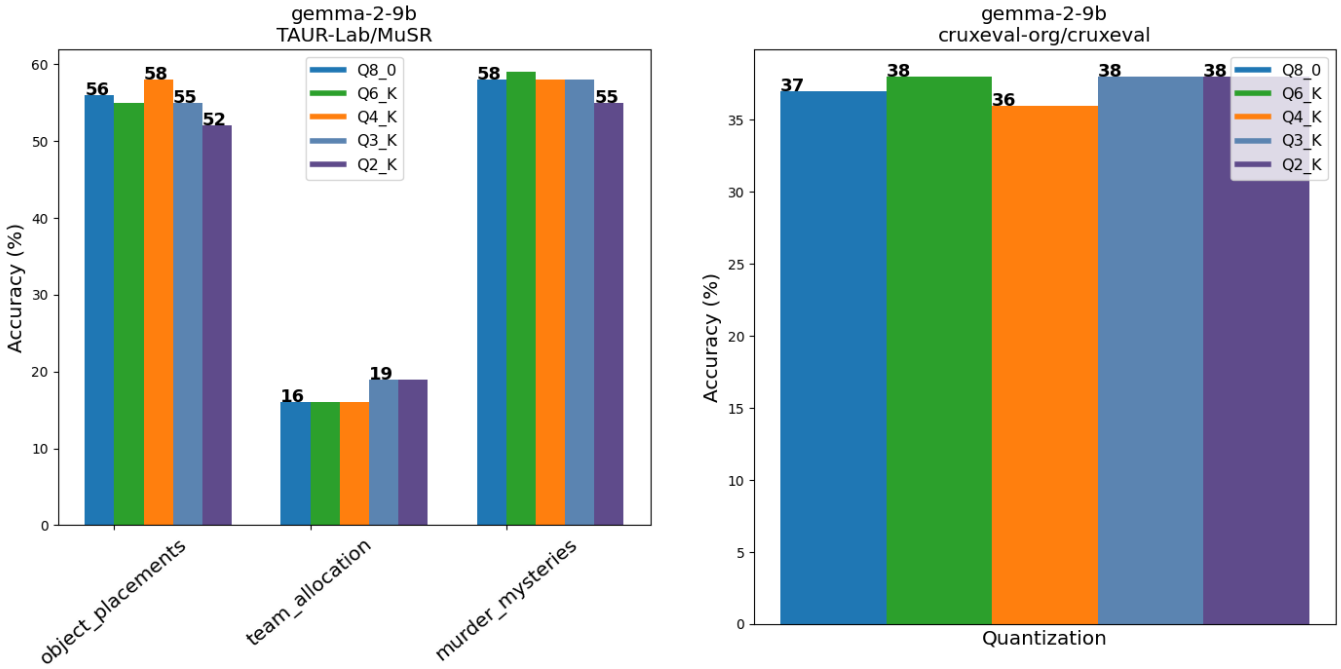
6.1.2 The Llama 3 Family

The Llama 3 family showed strong resilience against the impact of quantization across the various model sizes, demonstrating the effectiveness of its architecture.

Gemma 2 9B Perplexity

Quantization	Perplexity
Q2_K	10.16 ± 0.07
Q3_K	9.16 ± 0.07
Q4_K	8.92 ± 0.07
Q6_K	8.83 ± 0.07
Q8_0	**8.82 ± 0.07**

Table 6.3: The Perplexity score for each quantization level with the margin of error.



(a) The MuSR dataset.

(b) The CRUXEval dataset.

Figure 6.3: The accuracy scores of the Gemma 2 9B on the MuSR and CRUXEval datasets.

Llama 3 8B

The Llama 3 8B model showed competitive performance with perplexity scores ranging from 8.36 to 11.36 (Table 6.7). The accuracy scores for this model varied significantly across quantization levels, with a notable drop in performance for the Q2_K quantization.

The results for the MuSR and CRUXEval accuracy graphs are presented in Figure 6.7.

gemma 2 9B

Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	39.54%	42.01%	37.95%	39.80%	**10.5**
Q3_K	43.97%	**44.12**%	37.83%	41.87%	8.7
Q4_K	**45.61**%	**44.12**%	36.45%	41.86%	7.2
Q6_K	44.25%	43.33%	**38.45**%	**41.93**%	5.5
Q8_0	45.04%	43.20%	37.33%	41.72%	4.3

Table 6.4: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their geometric mean values.

Gemma 2 27B Perplexity

Quantization	Perplexity
Q2_K	9.17 ± 0.06
Q3_K	7.76 ± 0.05
Q5_K	7.25 ± 0.05
Q8_0	**7.19 ± 0.05**

Table 6.5: The Perplexity for each quantization with margins of error.

gemma 2 27B

Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	42.90%	42.54%	43.32%	42.92%	**4.1**
Q3_K	48.11%	**45.31**%	53.31%	48.80%	3.6
Q5_K	46.11%	45.44%	**54.56**%	48.53%	2.5
Q8_0	**49.86**%	44.78%	54.18%	**49.46**%	1.7

Table 6.6: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their geometric mean values.

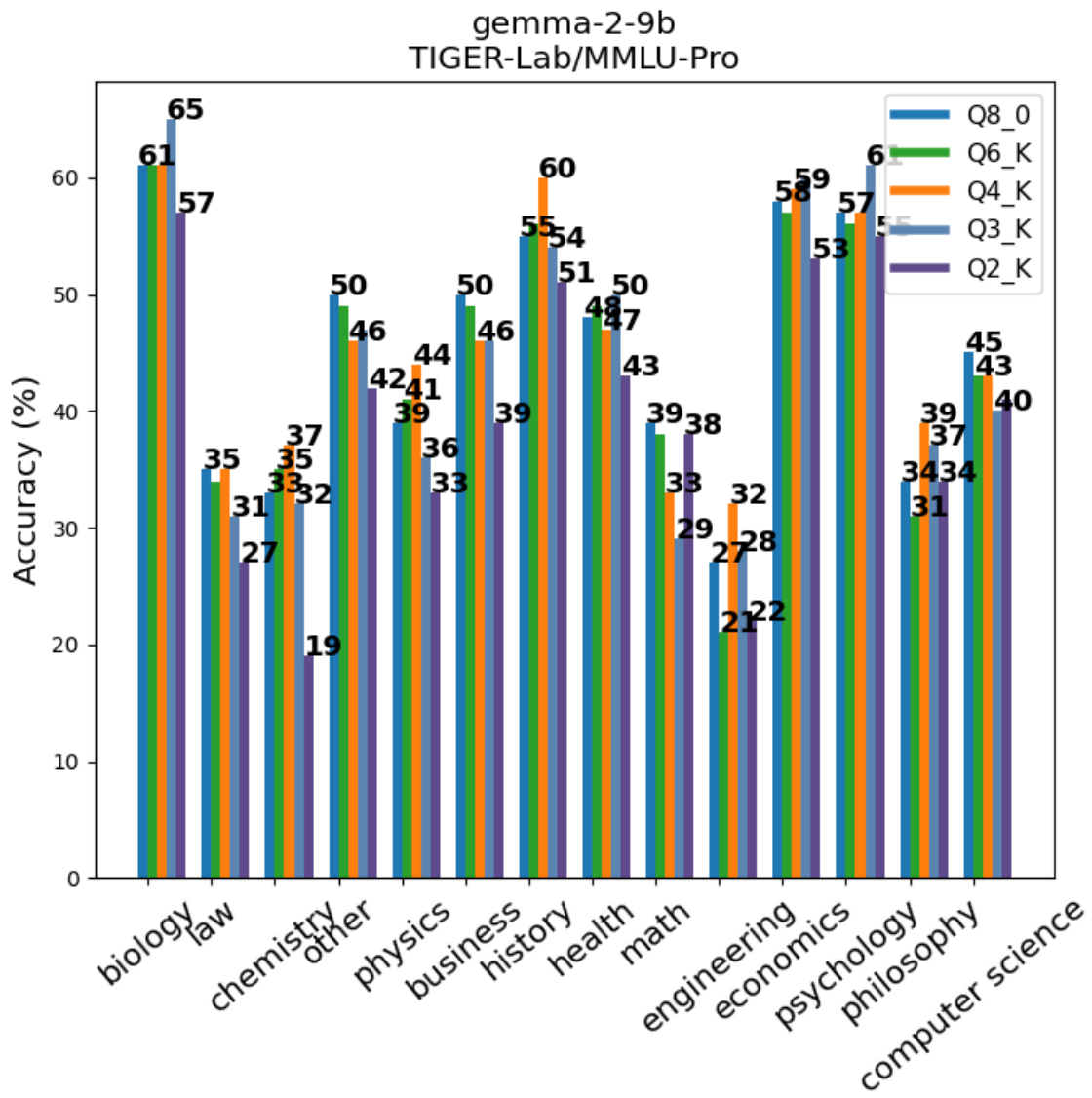


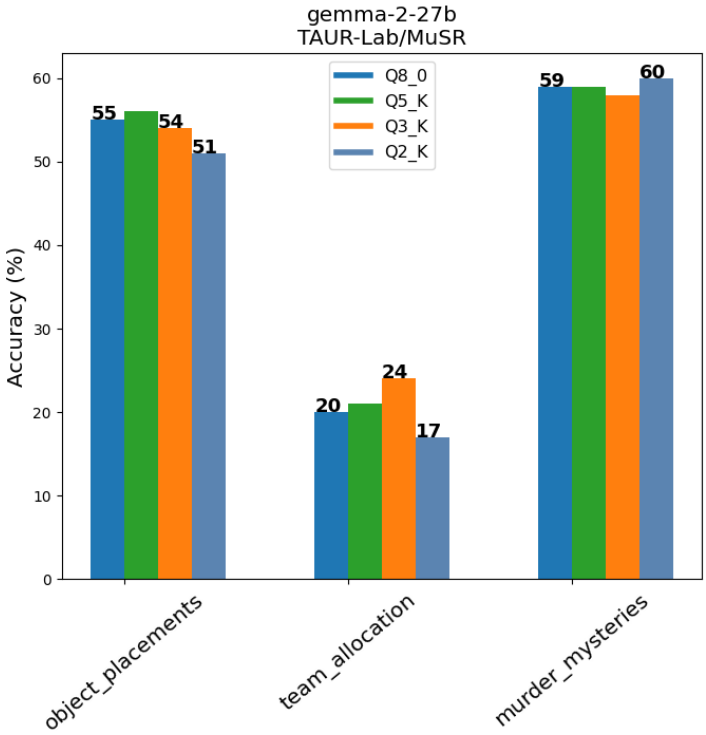
Figure 6.4: The accuracy scores of the Gemma 2b v1.1 evaluation on the MMLU-Pro dataset.

The accuracy scores for Llama 3 8B did not vary much across most of the quantization levels, with a notable drop in performance for the Q2_K quantization. Mean accuracy scores ranged from 26.38% to 36.02% (Table 6.8), with the Q6_K quantization achieving the highest overall performance. Q3_K scored close to the highest mean score, suggesting that it might offer the optimal balance between model size reduction and performance retention. The last column shows the performance efficiency of the model, it decreases from 9.0 for Q3_K to 4.2 for Q8_0.

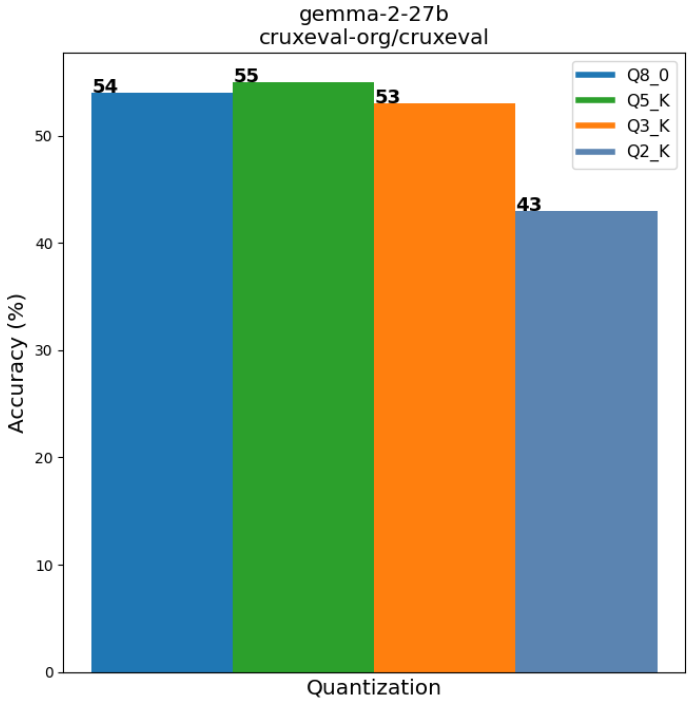
The graph of the accuracy score of Llama 3 8b on MMLU-Pro is presented in Figure 6.8.

Llama 3 70B

Llama 3 70B showed strong performance across the various quantization levels, with perplexity scores ranging from 5.18 to 6.87 (Table 6.9). The Q8_0 quantization achieved the lowest



(a) The MuSR dataset.



(b) The CRUXEval dataset.

Figure 6.5: The rounded accuracy scores of the Gemma 2 27B on the MuSR and CRUXEval datasets.

Llama 3 8B Perplexity

Quantization	Perplexity
Q2_K	11.36 ± 0.08
Q3_K	9.02 ± 0.07
Q4_K	8.49 ± 0.06
Q6_K	**8.36 ± 0.06**
Q8_0	**8.36 ± 0.06**

Table 6.7: The Perplexity for each quantization with margins of error.

perplexity of 5.18.

The full table of all the accuracy evaluations across all the datasets can be seen in Table 6.10. The accuracy scores showed improvement with higher quantization levels, with mean scores ranging from 41.92% to 51.08% (Table 6.10). The CRUXEval dataset reached 55.43% accuracy for the Q8_0 quantization, demonstrating the model's strong code comprehension. The consistency in MuSR scores across quantization levels (ranging from 41.08% to 43.86%) suggests that the model's text comprehension capabilities are quite resilient to quantization effects. The MMLU-Pro dataset saw scores up to 55.32% for Q8_0 and dropping to 38.83%

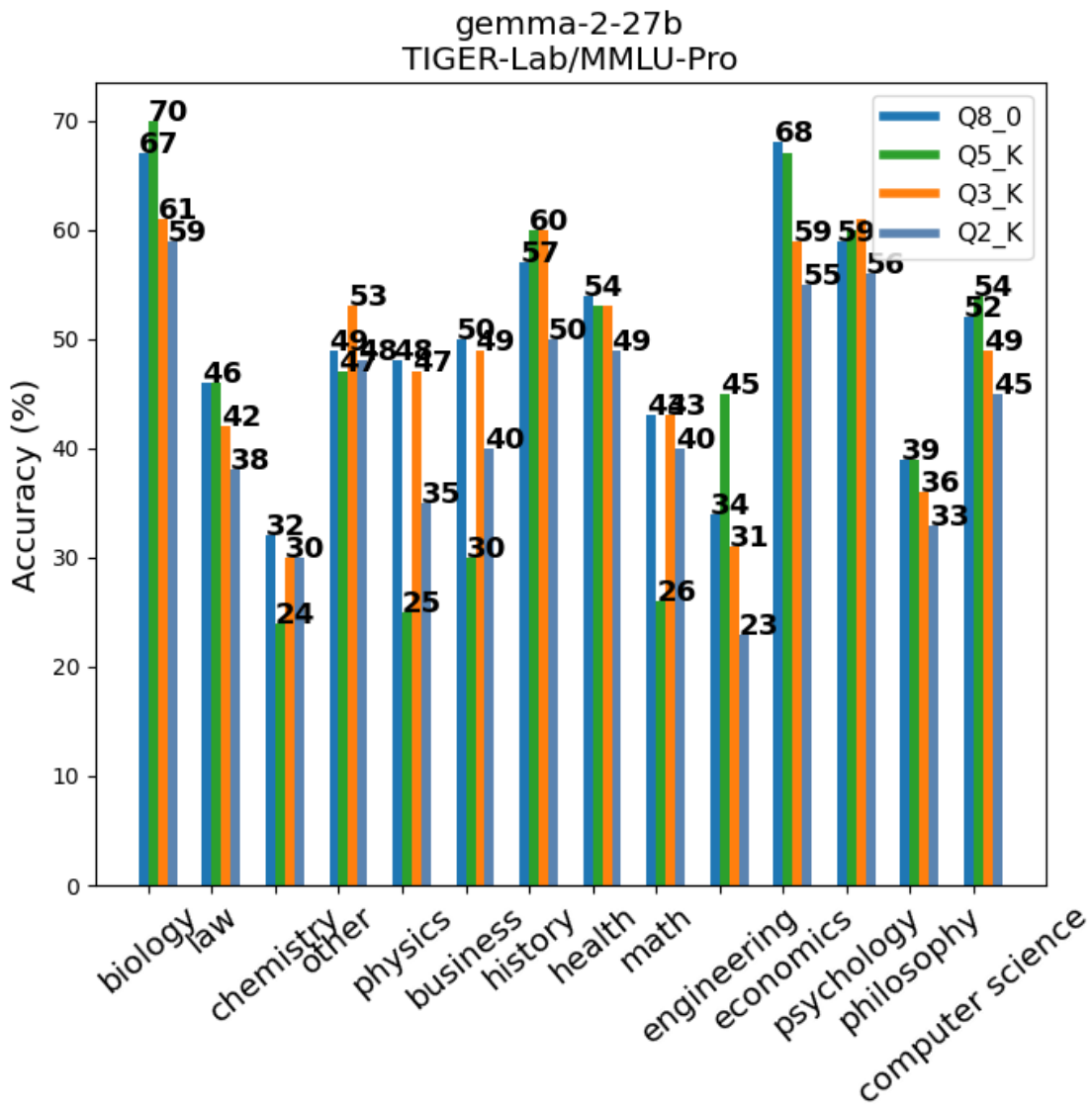


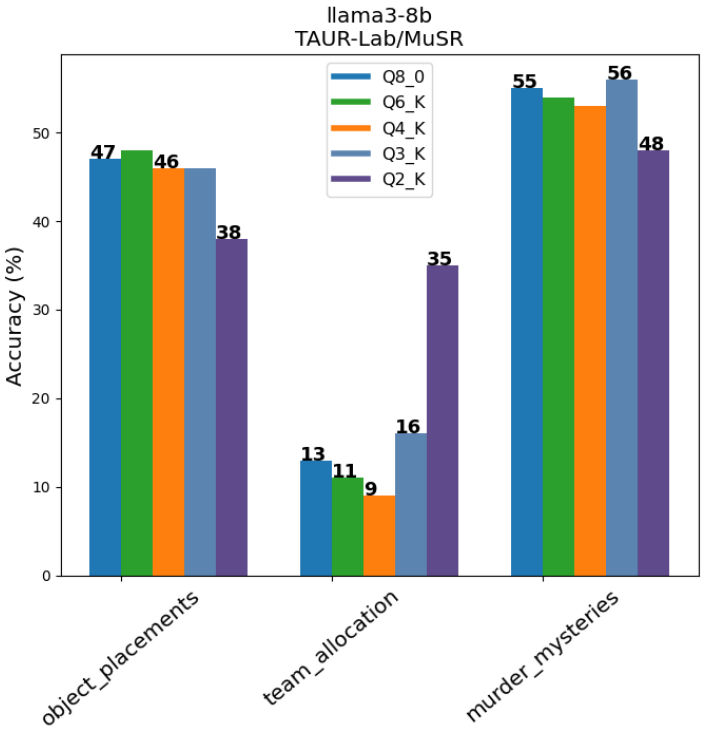
Figure 6.6: The rounded accuracy scores of the Gemma 2 27b evaluation on the MMLU-Pro dataset.

for Q2_K, indicating a significant loss of knowledge. The last column shows the performance efficiency of the model, it decreases from 1.6 for Q2_K to 0.7 for Q8_0.

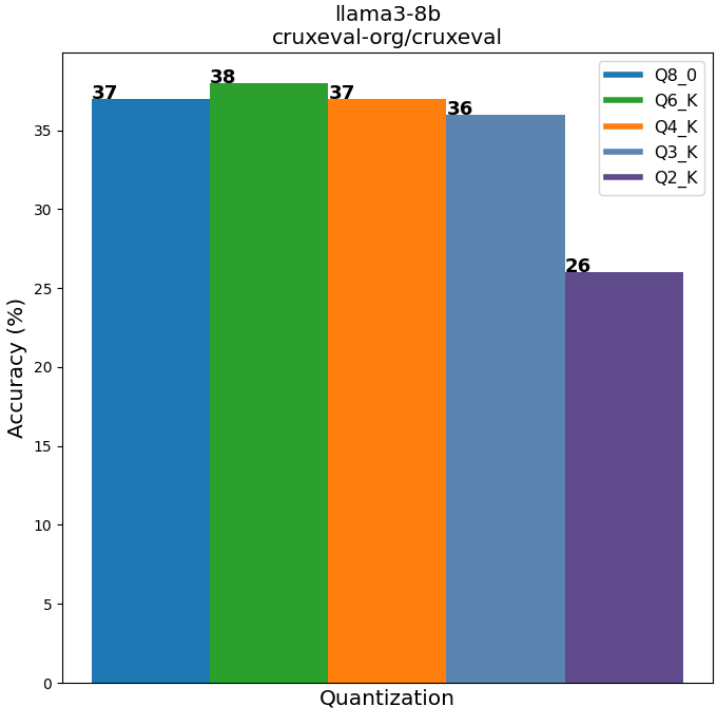
In Figure 6.9, the graphs evaluation scores are presented for the MuSR and CRUXEval datasets. In Figure 6.10, the graphs evaluation scores are presented on the MMLU-Pro dataset.

6.1.3 The Phi 3 Family

The Phi 3 family presented interesting results, particularly in terms of the impact of 2-bit quantization on performance.



(a) The MuSR dataset.



(b) The CRUXEval dataset.

Figure 6.7: The rounded accuracy scores of the Llama 3 8B on the MuSR and CRUXEval datasets

Llama 3 8B

Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	17.84%	**40.03%**	25.72%	26.38%	8.2
Q3_K	32.62%	39.37%	35.96%	35.88%	**9.0**
Q4_K	32.33%	36.20%	**36.95%**	35.10%	7.0
Q6_K	32.69%	37.91%	37.70%	**36.02%**	5.5
Q8_0	**38.31%**	37.08%	31.55%	35.52%	4.2

Table 6.8: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their geometric mean values.

Phi 3 Mini 4B

The Phi 3 Mini model showed the widest range of perplexity scores, from 6.41 to an unusually high 195.79 for the Q2_K quantization (Table 6.11). This extreme variation was reflected in the accuracy scores, where the Q2_K quantization performed poorly across all datasets,

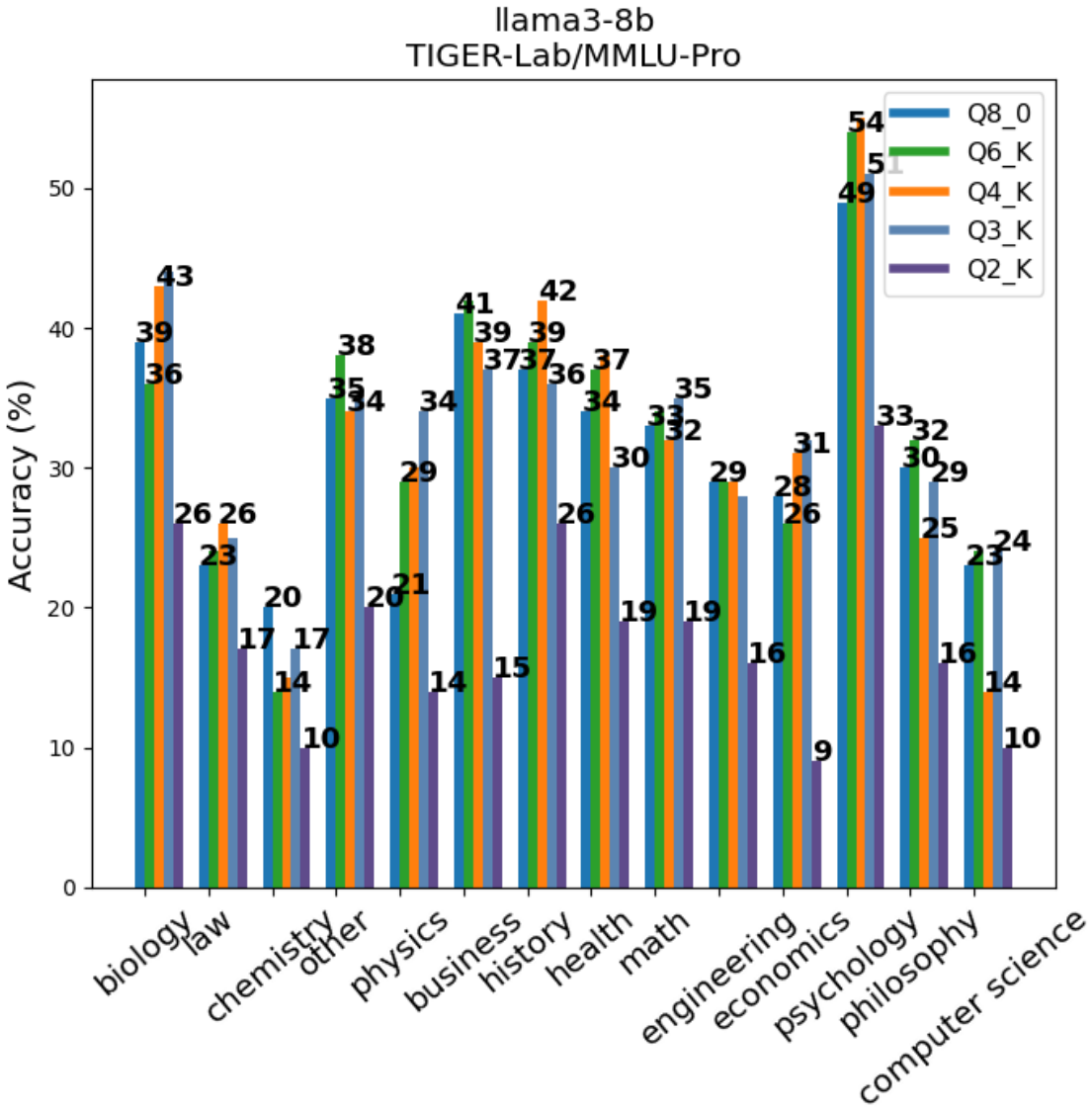


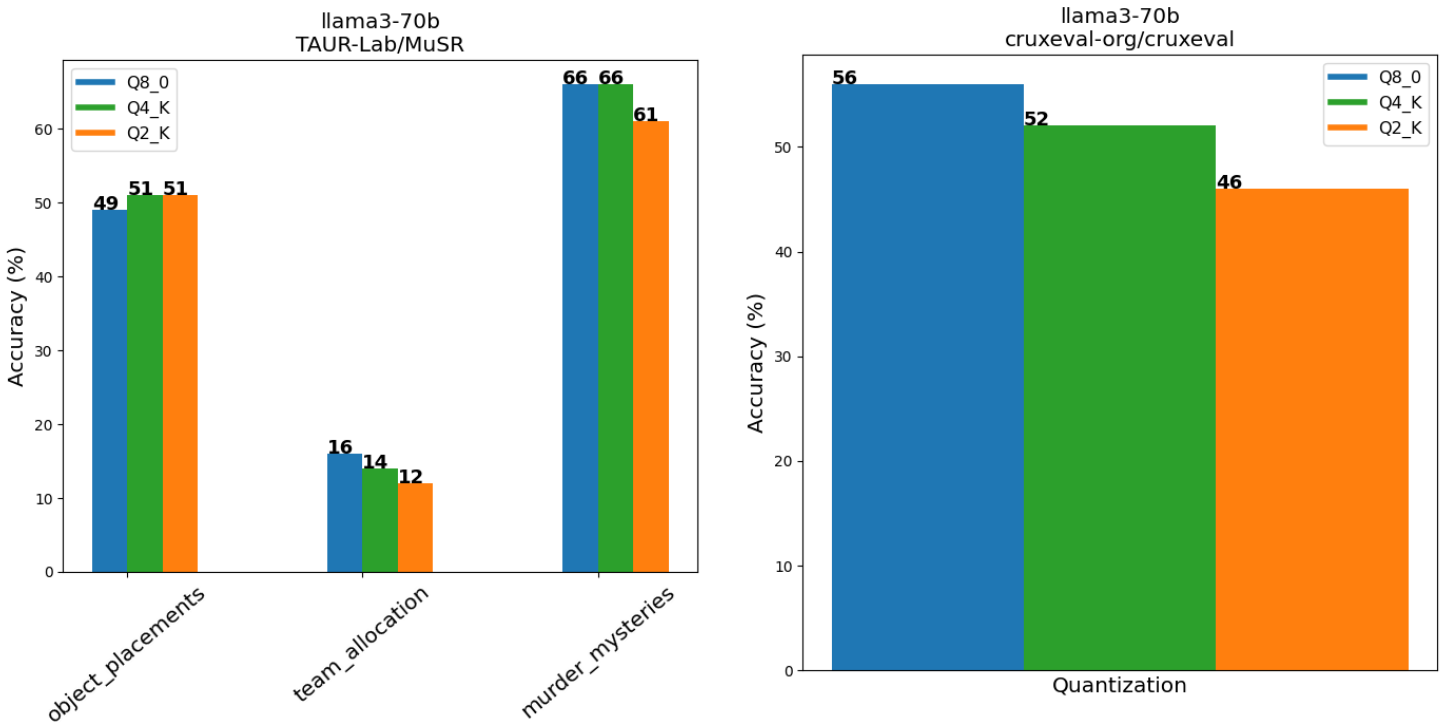
Figure 6.8: The rounded accuracy scores of the evaluation on the MMLU-Pro dataset.

Llama 3 70B Perplexity

Quantization	Perplexity
Q2_K	6.87 ± 0.05
Q4_K	5.33 ± 0.04
Q8_0	**5.18 ± 0.03**

Table 6.9: The Perplexity for each quantization with margins of error.

achieving a mean accuracy of only **2.59%** (Table 6.12). However, the other quantization



(a) The MuSR dataset.

(b) The CRUXEval dataset.

Figure 6.9: The rounded accuracy scores of the Llama 3 8B on the MuSR and CRUXEval datasets

Llama 3 70B

Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	38.83%	41.08%	46.19%	41.92%	**1.6**
Q4_K	53.53%	**43.86%**	51.69%	49.51%	1.2
Q8_0	**55.32%**	43.46%	**55.43%**	**51.08%**	0.7

Table 6.10: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their geometric mean values.

levels performed more consistently, with mean accuracy ranging from 10.66% to 11.76%. The last column shows the performance efficiency of the model, it decreases from 6.7 for Q3_K to 3.4 for Q8_0.

All the accuracy results on the MMLU-Pro dataset for each of the subjects can be found in Figure 6.11, and the graphs of the evaluation scores are presented in Figure 6.12 of the MuSR and CRUXEval datasets.

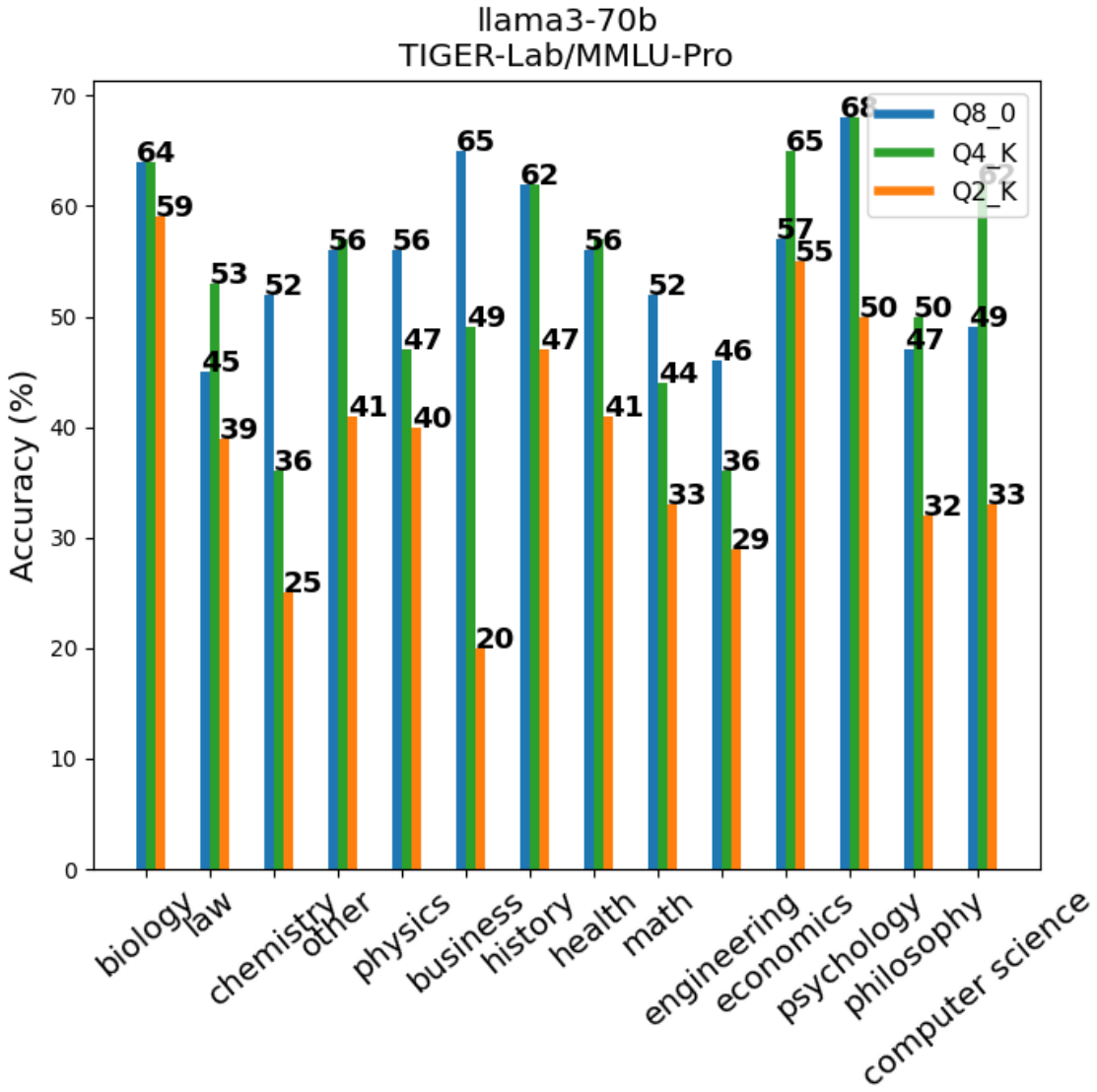


Figure 6.10: The rounded accuracy scores of the evaluation on the MMLU-Pro dataset.

Phi 3 Medium 14B

The Phi 3 Medium 14B model demonstrated the same issue at lower quantization. Perplexity scores ranged from 4.57 to 57.07 (Table 6.13), with the Q2_K quantization again showing significantly worse performance. Accuracy scores for this model were more consistent across the Q3_K, Q5_K, and Q8_K quantization levels, with mean accuracy ranging from 33.20% to 33.41% (Table 6.14). The MMLU-Pro dataset saw particularly strong performance, with scores reaching up to 47.39% for the Q5_K quantization.

Accuracy scores for this model were more consistent across the Q3_K, Q5_K, and Q8_K quantizations, with mean accuracy ranging from 31.84% to 32.57%, however, Q2_K saw an drastic (Table 6.14). The MMLU-Pro dataset saw particularly strong performance, with scores reaching up to 47.39% for the Q5_K quantization. This suggests that the Phi 3 Medium model is capable of maintaining its performance even at lower quantization lev-

Phi 3 Mini Perplexity

Quantization	Perplexity
Q2_K	195.79 ± 1.69
Q3_K	7.23 ± 0.04
Q4_K	6.69 ± 0.04
Q5_K	6.48 ± 0.04
Q6_K	6.42 ± 0.04
Q8_0	**6.41 ± 0.04**

Table 6.11: The Perplexity for each quantization with margins of error.

Phi 3 Mini 3B

Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	6.14%	**7.66%**	0.37%	2.59%	1.9
Q3_K	20.56%	5.81%	**13.61%**	**11.76%**	**5.9**
Q4_K	21.48%	7.00%	9.49%	11.26%	4.7
Q6_K	**27.69%**	6.74%	6.49%	10.66%	3.4
Q8_0	27.48%	7.74%	6.61%	11.20%	2.7

Table 6.12: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their geometric mean values.

Phi 3 Medium Perplexity

Quantization	Perplexity
Q2_K	57.07 ±0.44
Q3_K	5.43± 0.03
Q5_K	4.65± 0.03
Q8_0	**4.57 ± 0.03**

Table 6.13: The Perplexity for each quantization with margins of error.

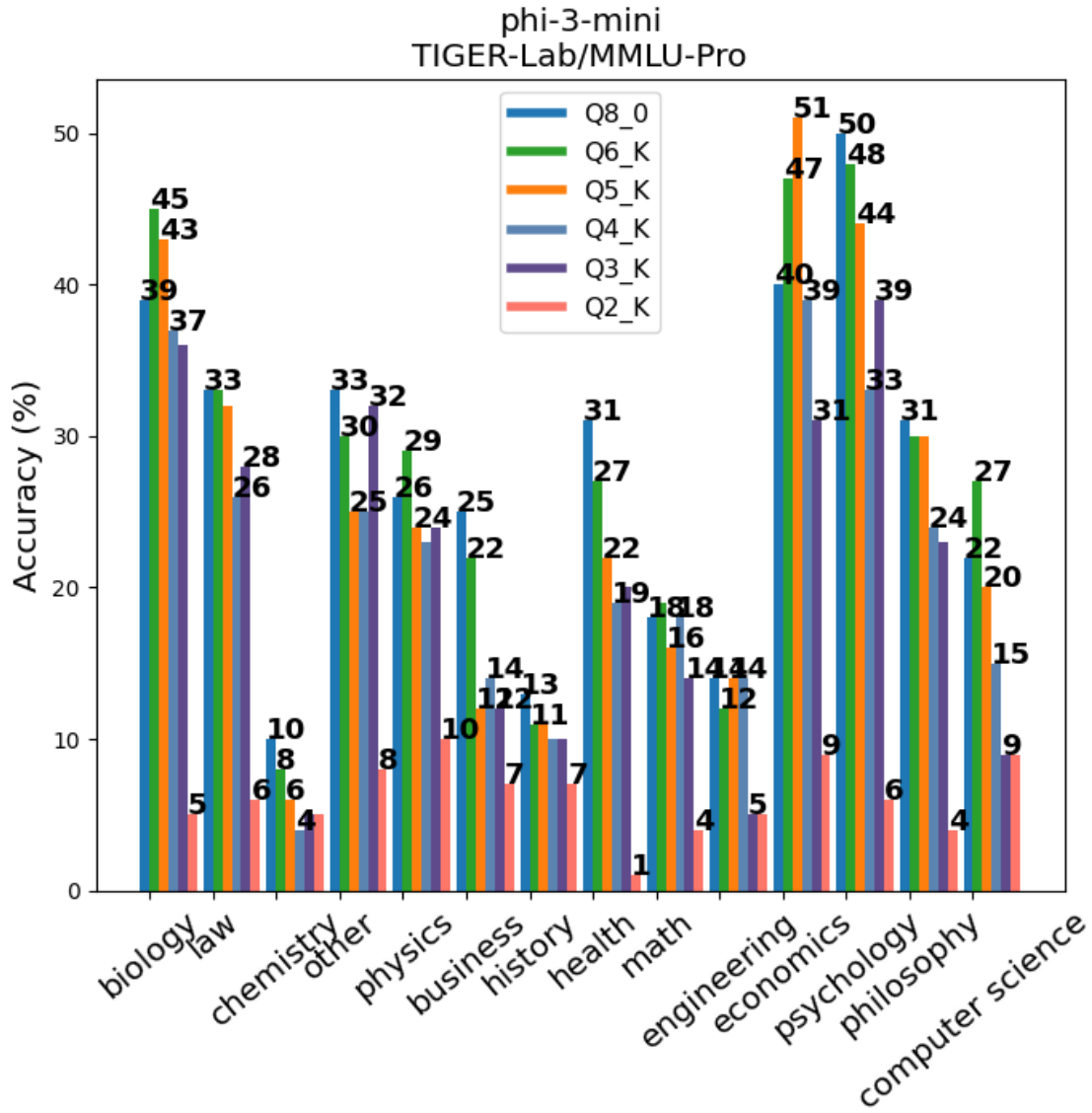


Figure 6.11: The rounded accuracy scores of the Phi 3 mini evaluation on the MMLU-Pro dataset.

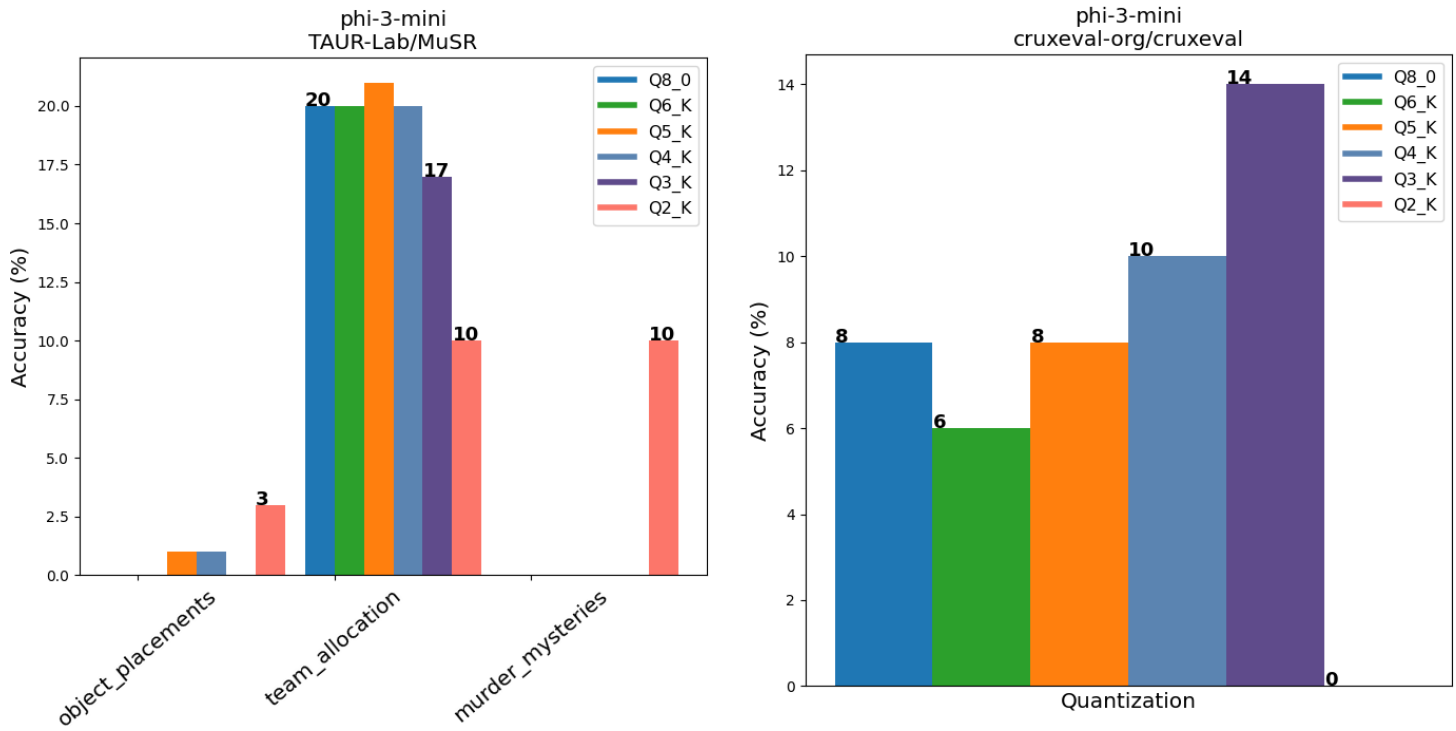
els, provided they are not too aggressive. The last column shows the performance efficiency of the model, it decreases from 4.1 for Q3_K to 2.2 for Q8_0, Q2_K is notably at 0.2.

In Figure 6.14, the graphs for the evaluation scores are presented for the MMLU-Pro dataset, and the graphs for the evaluation scores for the MuSR and CRUXEval datasets is seen in Figure 6.13.

6.1.4 Mistral

Mistral 7B v0.3

As anticipated, the Mistral 7B v0.3 model demonstrates a clear increase in the perplexity values as the model is further quantized. However, this increase was not as substantial as seen in some other models. The range of the perplexity score are all between 6.18 at the



(a) The MuSR dataset.

(b) The CRUXEval dataset.

Figure 6.12: The rounded accuracy scores of the Phi 3 Mini on the MuSR and CRUXEval datasets.

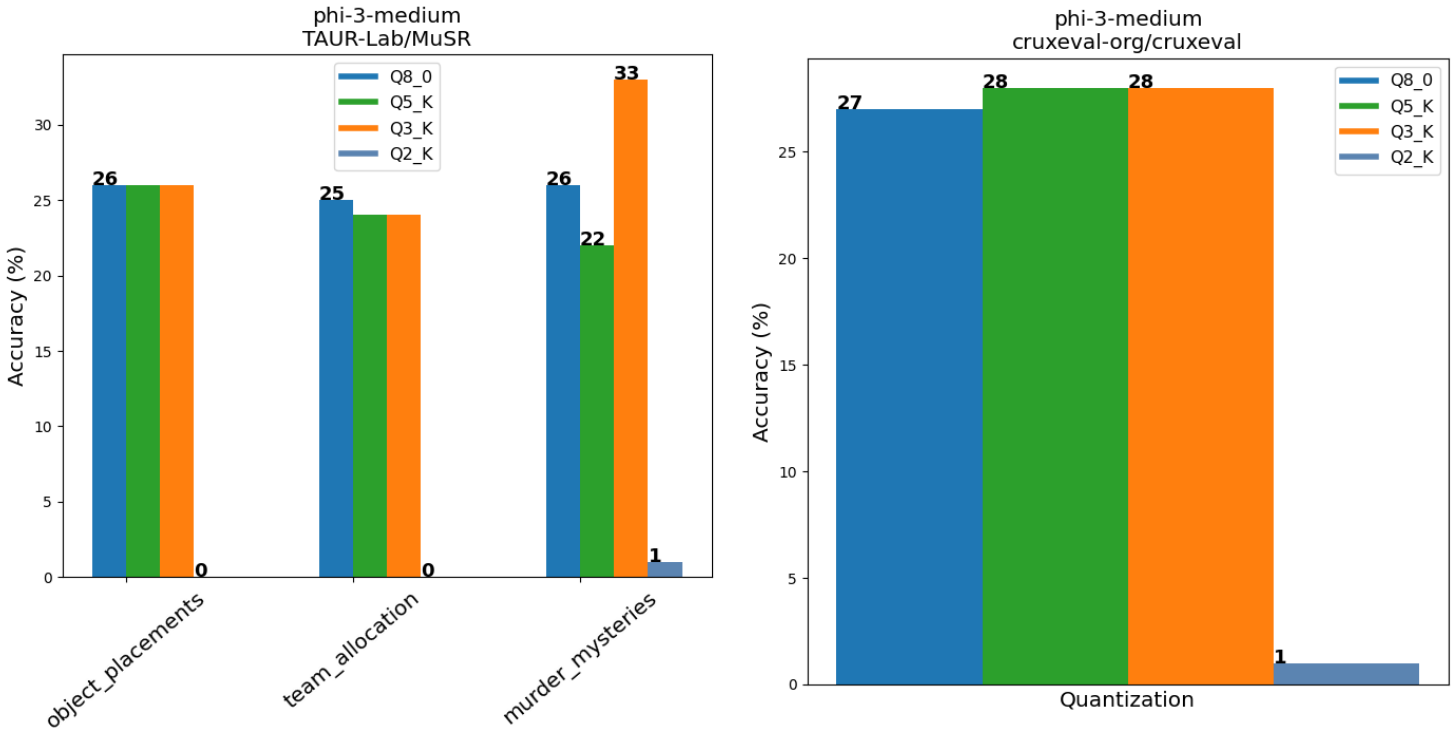
Phi 3 Medium 14B

Quantization	MMLU-Pro	MuSR	CRUXEval	Mean	Mean / GB
Q2_K	3.21%	0.26%	0.62%	0.80%	0.2
Q3_K	44.54%	**27.61%**	28.09%	**32.57%**	**4.1**
Q5_K	**47.39%**	24.04%	**28.34%**	31.84%	3.2
Q8_0	47.11%	25.76%	26.72%	31.89%	2.2

Table 6.14: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their mean values.

Q8_0 quantization level, and the highest at 6.98 for the Q2_K quantization level, as shown in Table 6.15.

When analyzing performance scores, which can be found in Table 6.16, the MMLU-Pro scores remained relatively consistent across most quantization levels. However, the Q2_K quantization level experienced a drop to 24.29% from 28.57% at the Q3_K quantization level. The MuSR scores showed minimal variation across all quantization levels and stayed



(a) The MuSR dataset.

(b) The CRUXEval dataset

Figure 6.13: The rounded accuracy scores of the Phi 3 Medium on the MuSR and CRUXEval datasets

Mistral 7B v0.3 Perplexity

Quantization	Perplexity
Q2_K	6.98 ±0.04
Q3_K	6.32± 0.04
Q4_K	6.21± 0.04
Q6_K	6.19± 0.04
Q8_0	**6.18 ± 0.04**

Table 6.15: The Perplexity for each quantization with margins of error.

relatively the same. Interestingly, the CRUXEval score saw a significant increase in the score from the lowest at the Q4_K quantization level with a score of 10.74% and the highest at 19.60% for the Q2_K quantization. Overall, the mean score remained stable through all of the quantization levels.

In Figure 6.15, the graphs of the evaluation scores of the MuSR and CRUXEval datasets are presented, and the MMLU-Pro score graphis presented in 6.16.

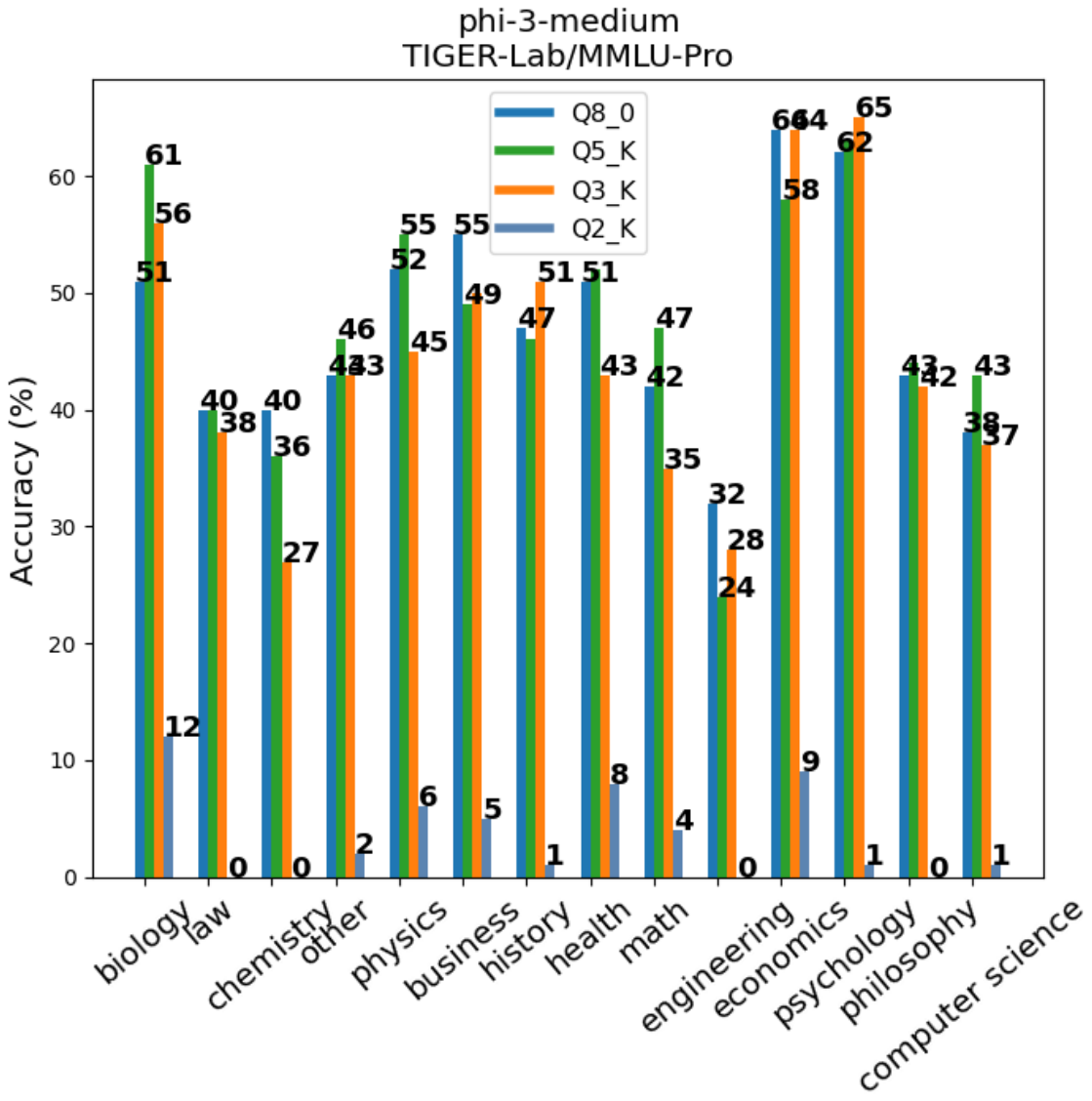
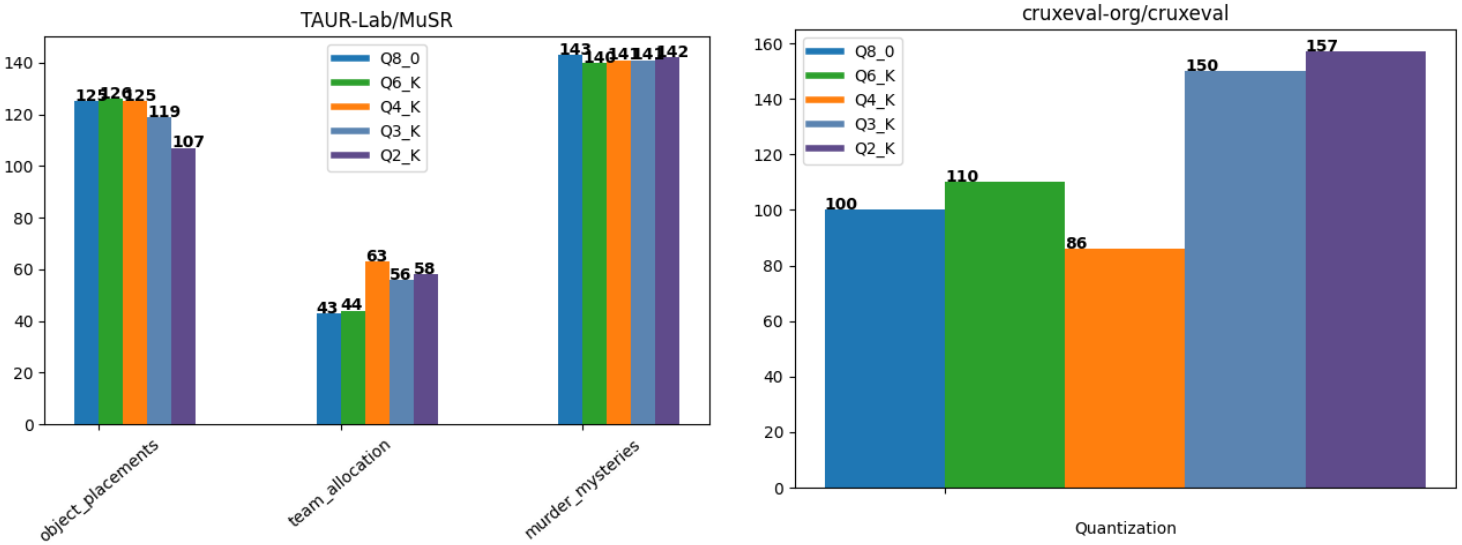


Figure 6.14: The rounded accuracy scores of the Phi 3 medium evaluation on the MMLU-Pro dataset.



(a) The Mistral 7B scores on the MuSR dataset.

(b) The Mistral 7B scores on the CRUXEval dataset.

Figure 6.15: The rounded accuracy scores of the Mistral 7B v0.3's on the MuSR and CRUXEval datasets.

Mistral 7B v0.3

Quantization	MMLU-Pro	MuSR	CRUXEval	Geometric Mean	Mean / GB
Q2_K	24.05%	40.56%	**19.60%**	26.74%	**9.9**
Q3_K	29.84%	41.74%	18.73%	**28.57%**	8.2
Q4_K	30.69%	**43.46%**	10.74%	24.29%	5.5
Q6_K	30.69%	40.95%	13.73%	25.84%	4.4
Q8_0	**31.41%**	41.08%	12.48%	25.25%	3.3

Table 6.16: Comparison of performance metrics (in percentage) across different quantization levels using multiple datasets, and their geometric mean values.

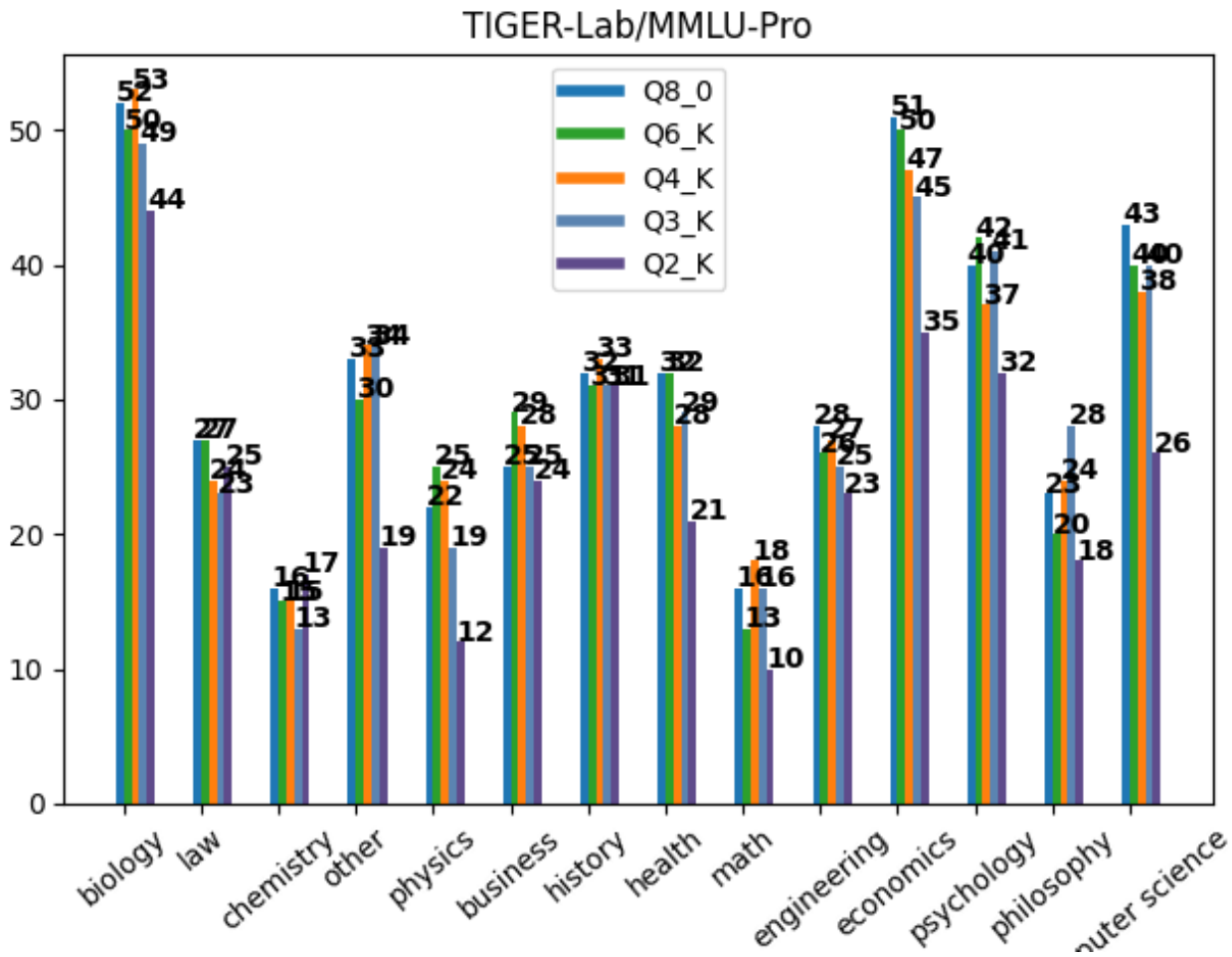


Figure 6.16: The rounded accuracy scores of the Mistral 7B v0.3 evaluation on the MMLU-Pro dataset.

6.1.5 Model Efficiency

Displayed in Figure 6.17 is the efficiency score of different language models in relation to their respective file sizes. It reveals that as the size of these languages increases, generally speaking, efficiency decreases. The trend suggests larger models are less efficient in terms of file size usage per performance unit.

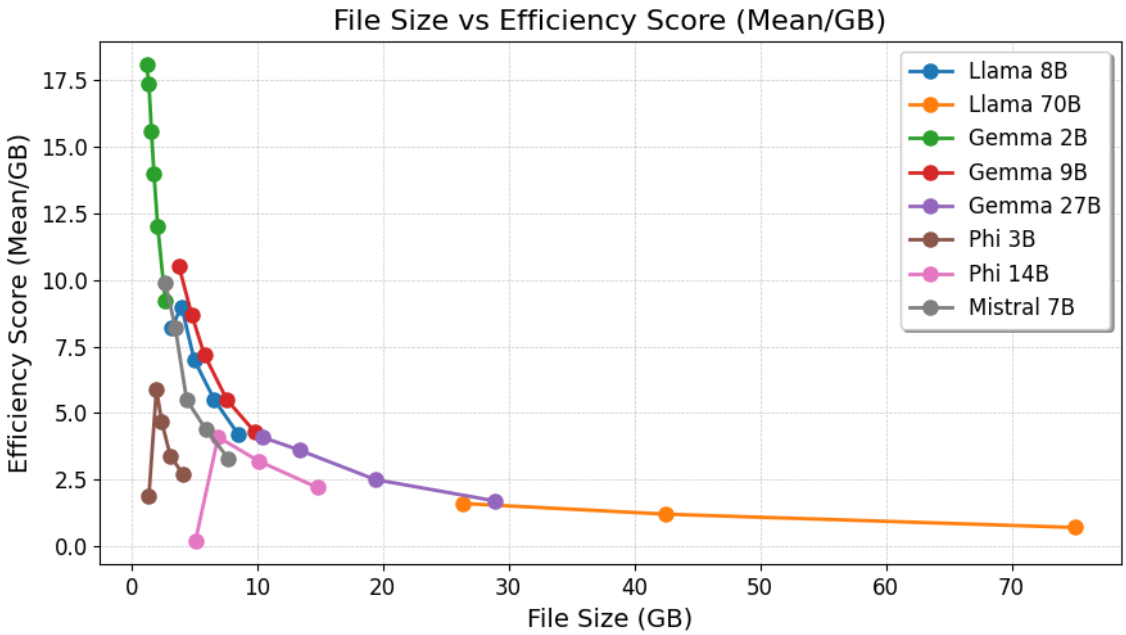


Figure 6.17: The efficiency score of each models quantization against the file sizes.

6.2 Discussion

Evaluating large language models (LLMs) across various quantization levels and task types has provided valuable insights into their performance and the relationship between quantization precision and model capabilities. The investigation included the models Gemma 2B, Gemma 2 9B, Gemma 27B, Llama 3 8B, Llama 3 70B, Phi 3 Mini, Phi 3 Medium, and Mistral 7B v0.3. These were evaluated across multiple quantization levels on tasks such as MMLU-Pro, CRUXEval, MuSR, and perplexity measurements.

6.2.1 Outliers

The evaluation results show a few interesting outliers, particularly in Llama 3 8B Figure 6.7a, All scores of Phi 3 mini and Phi 3 medium, and Mistral 7B Figure 6.15a.

The team allocation score in Figure 6.7a presents a significant inrease in accuracy, with the Q4_K only receiving an accuracy score of 9 while Q2_K saw an accuracy score of 35. The rest of the Llama 3 8B scores remain as expected. The reason for an increase in some scores at Q2_K requires further research.

The scored of Phi 3's medium and mini Q2_K scores show a significant regresson in every single dataset, getting few to no accuracy scores at all. This indicates that the loss of information is too significant causing degradation in the performance. Additionally, Studying the scores in Figure 6.12a we find that the object placement and murder mysteries scores are close to 0 while team allocation performs as expected. Taking a look at the input length distribution, they are all shorter than the context length of 4096. The input length in the murder mysteries has a shortest length of $\sim$ 3800 characters, and the longest length is $\sim$ 7200 characters. Thus, the possible reason for the low scores is the large input lengths causing confusion and hallucination which results in the model output being gibberish and unrelated text.

6.2.2 Model Specific Performance

Quantization levels have significantly to little impact on model performance, with higher levels (Q8_0) generally outperforming lower ones (Q2_K) across all models. This was evident in both task-specific accuracy and perplexity scores. For example, the Gemma 2B model's perplexity ranged from 30.02 at Q8_0 to 39.86 at Q2_K (Table 6.1), indicating that higher quantization levels maintain better predictive capabilities. This trend was consistent across all models, underlining the importance of precision in maintaining performance. However, it is important to note that while mean accuracy performance was decreasing for the lower quantizations, the scores for specific tasks did sometimes vary quite a bit. For example, the CRUXEval score for the Mistral 7B v0.3 model showed a notable difference.

The impact of quantization on perplexity varied across models. The Llama 3 70B model had a smaller range of perplexity scores, from 5.18 at Q8_0 to 6.87 at Q2_K (Table 6.10), suggesting that larger models may be more robust to quantization effects. This suggests that an increased parameter count allows models to better withstand information loss during quantization.

The Llama 3 8B model showed competitive performance with perplexity scores ranging from 8.36 to 11.36 (Table 6.7). Notably, the highest accuracy was achieved at Q6_K quantiza-

tion (36.10%), highlighting that a balance can be struck between model size and performance (Table 6.8).

The Phi 3 family models showed a dramatic difference in perplexity, with Phi 3 mini having a perplexity of 195.79 at Q2_K and as low as 6.41 for Q8_0 (Table 6.11). Phi 3 medium experienced a similar result when quantized at Q2_K (Table 6.13) ranging from 4.57 (Q8_0) up to 57.07 (Q2_K). This highlight the critical importance of the sensitivity when quantization at lower bit counts, and the stark contrast shows the impact of lower precision quantization in some models. This seems only to be an issue for the Phi 3 family, further research is needed to find the root cause. One should use caution when applying low bit quantization as the performance degradation can be severe.

The performance of the Mistral 7B v0.3 model saw the MMLU-Pro scores (Table 6.16) remained relatively consistent across most quantization levels, a noticeable drop occurred at Q2_K (24.05%) from the previous quantization level Q3_K (29.84%), indicating a loss of knowledge. MuSR scores showed minimal variation and was stable across all quantization levels. CRUXEval showed a more interesting trend, the score increased from the lowest score of Q4_K (10.74%) to Q2_K (19.60%), suggesting a potential benefit of lower quantization for this specific task. Overall, Mistral 7B v0.3's mean performance remained relatively stable across quantization levels.

In task-specific performance, lower quantization levels (Q3_K and Q2_K) mostly offer a substantial reduction in model size while maintaining acceptable performance, which is crucial for applications with limited resources. For example, the Gemma 2 9B model's mean accuracy was 39.83% at Q2_K and 41.85% at Q8_0 (Table 6.4), demonstrating that even at lower quantization levels, the model retained significant capabilities. Performance degradation across quantization levels was not uniform across all tasks and models. For instance, the larger models showed stronger resilience to quantization in each of the tasks.

Each LLM showed unique strengths and weaknesses across different tasks. Gemma 2B performed well in the tasks, demonstrating its ability to reason effectively across diverse domains despite its relatively small size. This suggests that model architecture and training approach can sometimes compensate for limitations in parameter count. The Phi 3 Mini model, despite its smaller parameter count, was competitive in the MMLU-Pro task, emphasizing the importance of task-specific tuning and specialized architectures.

It is worth noting that the most efficient model is the Gemma 2B v1.1 (Table 6.2) due to the small parameter count, but Gemma 2 9B (Table 6.4) and Mistral 7B v0.3 (Table 6.16) performed quite a bit better and scored a efficiency score of 10.5 and 10.4, respectively. This shows that models should not be compared with each other, as suggested. Generally, the efficiency increased as the bit quantization is reduced until the models lost too much information, then started to decrease in efficiency.

6.2.3 Task Specific Insights

The variety of task types employed in the evaluation revealed significant insights into the model's diverse capabilities. CRUXEval tasks highlighted differences in code comprehension and execution simulation abilities, with performance being quite constant across models and their quantization levels. This suggests that code-related tasks may be an excellent category for quantization. MuSR tasks emphasized text and reading comprehension capabilities, revealing interesting patterns in how different models handle long-form narratives

and multi-step reasoning. Performance on these tasks did not seem affected by the quantization precision, except for the Phi 3 family which failed to have a coherent understanding on long texts at the lower quantization levels and the Mistral model which saw an increased CRUXEval score at Q_2.

MMLU-Pro tasks offered a comprehensive assessment of the LLMs' reasoning capabilities across multiple fields of knowledge. The results on this dataset showed that even heavily quantized models could maintain a significant portion of their reasoning abilities. This resilience suggests that the knowledge in model weights may be quite robust to quantization. In general, it showed the expected results, a lower bit precision cause worse performance, but it is not linear. In some cases, models showed similar performance at Q4_K and Q8_0, suggesting a potential "sweet spot" where further precision would get a diminishing return.

6.2.4 Perplexity and Task Performance Correlation

The perplexity measurements provided valuable insights into the models fundamental language capabilities across the quantization levels. Interestingly, the correlation between perplexity and task-specific performance was found to be as expected. A lower perplexity score generally meant a higher accuracy score for the tasks, but was not always the case. For example, comparing the Gemma 2 9B's Q2_K and Q8_0 scores, the Q2_K had a 15% increase in perplexity, but only a 5% lower relative accuracy score. However, the Mistral's Q2_K and Q8_0 scores saw a insignificant change in the mean scores, but a 13% increased perplexity. This highlight the limitations of using perplexity alone for overall model quality and emphasizes the importance of task-specific evaluations.

6.2.5 Implications for Large-Scale Models

The performance of the largest model in the study, Llama 3 70B, was particularly noteworthy. Its ability to maintain high performance even at Q2_K quantization levels, even though it saw a performance drop, suggests that lower quantization levels of larger models might be the key to deploying powerful AI capabilities on resource-constrained devices, or for general user interactions. However, the computational requirements for training, fine-tuning and running such large models present their own challenges.

6.2.6 Diminishing Efficiency

Figure 6.17 highlights an interesting relationship between the accuracy score and memory usage, as the model size increases. It appears that larger models lead to a decline in efficiency, suggesting either a disproportionate increase in resource requirements and diminishing efficiency as models grow larger. We can also see that employing quantization techniques shows that the efficiency can be greatly increased for smaller model like Mistral and Gemma. This pattern emphasizes the need for further research with more diverse model samples and parameter counts to better understand these relationships.

Chapter 7
Conclusion

This thesis has shown the complex connections between the quantization precision, computational efficiency, and task specific performance in large language models. Using the three datasets, we saw that for the most part, across the models, that a lower quantization level results in both a worse accuracy score and a higher perplexity value. But there are outliers, as discussed in Section 6.2.1, model families such as Phi 3 could not handle the 2-bit quantization in any of the results.

The CRUXEval tasks, which involve code comprehension and execution simulation, in general saw consistent performance across different models and quantization levels. This suggests that code related tasks might be well suited for quantized models. The same can be said about the MuSR tasks, text comprehension capabilities were generally not significantly affected by the quantization precision. However, low level quantizations (e.g., Q2_K) notably impacted specific models like Phi 3 Mini in terms of handling long narratives. We also found that perplexity is a well suited metric for assessing how well the language models handle quantization and serves as a indicator of the models overall performance and robustness. Perplexity does come with the trade-off that evaluating perplexity necessitates multiple calculations for each quantization level to determine determine relative performance.

The findings emphasize that there is no one-size-fits-all solution for the quantization level to use for large language models. However, each of the models showed diminishing return as the precision increased from Q2_K to Q8_0, as seen in Figure 6.17. The optimal choice of model and quantization level depends heavily on the specific use case, available resources, and performance requirements.

It is important to acknowledge the limitations of this study. The scope of models and tasks examined, while comprehensive, represents only a fraction of the vast usage of LLMs and potential applications. Constraints on computational resources limited the ability to test every possible configuration or to explore extremely large models larger than 70B. Additionally, the number of few-shot learning scenarios varied across different models due to computational limitations, necessitating adjustments in configurations when certain settings were not feasible. Further studies on the accuracy to size ratio would be prefered as it shows a quantified measurment of the efficiency. Furthermore, given the rapid advancements in AI research, some findings may become obsolete relatively quickly and should be interpreted with this in mind.

Future work could focus on exploring dynamic quantization techniques, such as impor-

tance matrix quantization[1] that allow models to adjust their precision levels dynamically based on data characteristics. Running evaluation benchmarks using a judge large language model without question choices can provide more nuanced assessments of model quality, revealing aspects of performance not captured by traditional metrics. Post-quantization fine-tuning might be useful for recovering accuracy lost during quantization while maintaining high efficiency. Additionally, investigating 1-bit to 2-bit quantization[2] techniques could provide better efficiency, and new applications in resource constraint scenarios. Understanding the effects of quantization on Mixture of Experts[3] (MoE) models is also relevant given their growing usage in large language modeling. Developing task-specific benchmarks that are sensitive to quantization effects can provide valuable insights for improving these techniques.

Conclusively, The results highlight the critical importance of considering the entire picture of model development, from architecture design and training to quantization and deployment, in creating efficient and effective language models. As AI continues to be included in various aspects of technology and society, the ability to deploy powerful language models in diverse computational environments will be crucial in using the full potential of this transformative technology.

[1]https://github.com/ggerganov/llama.cpp/pull/4861
[2]https://doi.org/10.48550/arXiv.2402.17764
[3]https://doi.org/10.48550/arXiv.1701.06538

References

Banks, Jeanine and Tris Warkentin (Feb. 2024). *Gemma: Introducing new state-of-the-art open models*. https://blog.google/technology/developers/gemma-open-models. [Accessed 12-07-2024]. URL: https://blog.google/technology/developers/gemma-open-models/.

Betlen, Andrei (2024). *Python Bindings for llama.cpp*. https://github.com/abetlen/llama-cpp-python/blob/main/README.md. [Accessed 11-07-2024].

Bilenko, Misha (Apr. 2024). *Introducing Phi-3: Redefining what's possible with SLMs*. https://azure.microsoft.com/en-us/blog/introducing-phi-3-redefining-whats-possible-with-slms. [Accessed 12-07-2024]. URL: https://azure.microsoft.com/en-us/blog/introducing-phi-3-redefining-whats-possible-with-slms/.

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

Dettmers, Tim et al. (2022). *8-bit Optimizers via Block-wise Quantization*. arXiv: 2110.02861 [cs.LG]. URL: https://arxiv.org/abs/2110.02861.

Dubey, Abhimanyu et al. (2024). *The Llama 3 Herd of Models*. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

Dunn, Craig (Aug. 2023). *Infinite Chat using a sliding window*. URL: https://devblogs.microsoft.com/surface-duo/android-openai-chatgpt-16/.

Ercegovac, Milos D. and Tomás. Lang (2004). *Digital Arithmetic*. Morgan Kaufmann; Elsevier Science.

Farabet, Clement and Tris Warkentin (June 2024). *Gemma 2 is now available to researchers and developers*. https://blog.google/technology/developers/google-gemma-2. [Accessed 12-07-2024]. URL: https://blog.google/technology/developers/google-gemma-2/.

Frantar, Elias et al. (2023). *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. arXiv: 2210.17323 [cs.LG]. URL: https://arxiv.org/abs/2210.17323.

Gäßler, Johannes (Aug. 2023). *GitHub - JohannesGaessler/johannesgaessler.github.io: llama.cpp Quantization Metrics*. https://github.com/JohannesGaessler/johannesgaessler.github.io. [Accessed 15-09-2024].

Gemma Team et al. (2024). *Gemma 2: Improving Open Language Models at a Practical Size*. arXiv: 2408.00118 [cs.CL]. URL: https://arxiv.org/abs/2408.00118.

Georgi Gerganov, Community Contributors (2023). *GitHub - ggerganov/llama.cpp: Perplexity (Quality of Generation) Scores — github.com*. https://github.com/ggerganov/llama.cpp/discussions/406. [Accessed 13-08-2024].

Georgi Gerganov, Community Contributors (2024). *GitHub - ggerganov/llama.cpp: Perplexity — github.com*. https://github.com/ggerganov/llama.cpp/tree/master/examples/perplexity. [Accessed 15-07-2024].

Gerganov, Georgi (2024). *GitHub - ggerganov/llama.cpp: LLM inference in C/C++ — github.com*. https://github.com/ggerganov/llama.cpp. [Accessed 08-07-2024].

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (Apr. 2011). "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

Gong, Zhuocheng et al. (2024). *What Makes Quantization for Large Language Models Hard? An Empirical Study from the Lens of Perturbation*. arXiv: 2403.06408 [cs.LG]. URL: https://arxiv.org/abs/2403.06408.

Google (2024). *The bfloat16 numerical format*. [Accessed 5-July-2024]. URL: https://cloud.google.com/tpu/docs/bfloat16.

Gu, Alex et al. (2024). *CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution*. arXiv: 2401.03065 [cs.SE]. URL: https://arxiv.org/abs/2401.03065.

Huang, Wei et al. (2024). *BiLLM: Pushing the Limit of Post-Training Quantization for LLMs*. arXiv: 2402.04291 [cs.LG]. URL: https://arxiv.org/abs/2402.04291.

HuggingFace, Community Contributors (2024). *hub-docs/docs/hub/gguf.md at main · huggingface/hub-docs — github.com*. https://github.com/huggingface/hub-docs/blob/main/docs/hub/gguf.md. [Accessed 08-07-2024].

Jiang, Albert Q. et al. (2023). *Mistral 7B*. arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825.

Jin, Renren et al. (2024). *A Comprehensive Evaluation of Quantization Strategies for Large Language Models*. arXiv: 2402.16775 [cs.CL]. URL: https://arxiv.org/abs/2402.16775.

Kim, Sehoon et al. (2024). *SqueezeLLM: Dense-and-Sparse Quantization*. arXiv: 2306.07629 [cs.CL]. URL: https://arxiv.org/abs/2306.07629.

Kuribayashi, Tatsuki et al. (Aug. 2021). "Lower Perplexity is Not Always Human-Like". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong et al. Online: Association for Computational Linguistics, pp. 5203–5217. DOI: 10.18653/v1/2021.acl-long.405. URL: https://aclanthology.org/2021.acl-long.405.

Liu, Peter J. et al. (2018). *Generating Wikipedia by Summarizing Long Sequences*. arXiv: 1801.10198 [cs.CL]. URL: https://arxiv.org/abs/1801.10198.

Merity, Stephen et al. (2016). *Pointer Sentinel Mixture Models*. arXiv: 1609.07843 [cs.CL].

Meta, AI at Meta (Apr. 2024). https://ai.meta.com/blog/meta-llama-3/. [Accessed 13-07-2024]. URL: https://ai.meta.com/blog/meta-llama-3/.

Mistral, AI (n.d.). *Tokenization*. https://docs.mistral.ai/guides/tokenization/. [Accessed 14-09-2024]. URL: https://docs.mistral.ai/guides/tokenization/.

Nejjar, Mohamed et al. (2024). *LLMs for Science: Usage for Code Generation and Data Analysis*. arXiv: 2311.16733 [cs.SE]. URL: https://arxiv.org/abs/2311.16733.

OpenAI (2024). *Chat Completion API*. https://platform.openai.com/docs/guides/gpt/chat-completions-api, https://platform.openai.com/docs/guides/text-generation. [Accessed 11-07-2024].

Shannon, C.E. and W. Weaver (1949). *The Mathematical Theory of Communication*. Illini books v. 1. University of Illinois Press. ISBN: 9780252725487. URL: https://books.google.se/books?id=dk0n_eGcqsUC.

Sprague, Zayne et al. (2024). *MuSR: Testing the Limits of Chain-of-thought with Multistep Soft Reasoning*. arXiv: 2310.16049 [cs.CL]. URL: https://arxiv.org/abs/2310.16049.

Su, Jianlin et al. (2023). *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv: 2104.09864 [cs.CL]. URL: https://arxiv.org/abs/2104.09864.

Tonmoy, S. M Towhidul Islam et al. (2024). *A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models*. arXiv: 2401.01313 [cs.CL]. URL: https://arxiv.org/abs/2401.01313.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

Wang, Liang et al. (2024). *Improving Text Embeddings with Large Language Models*. arXiv: 2401.00368 [cs.CL]. URL: https://arxiv.org/abs/2401.00368.

Wang, Yubo et al. (2024). *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. arXiv: 2406.01574 [cs.CL]. URL: https://arxiv.org/abs/2406.01574.

Wei, Jason et al. (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903.

Zhong, Peixiang, Di Wang, and Chunyan Miao (2018). *An Affect-Rich Neural Conversational Model with Biased Attention and Weighted Cross-Entropy Loss*. arXiv: 1811.07078 [cs.CL]. URL: https://arxiv.org/abs/1811.07078.

Appendices

The general system prompt template in Appendix A was handwritten. The value inserted in '{}' are inspired by the paper and code provided. Those are seen in Appendix B, Appendix C, Appendix D.

Appendix A

Listing 1: The unformatted system prompt

```
### IDENTITY and PURPOSE

- **Your Task:** Derive Your Final Answer {}.

### INSTRUCTION STEPS:

* {}.
* Identify key phrases or words that can guide your decision.
* Eliminate any options that are clearly incorrect based on your knowledge.
* Compare the remaining options to determine which one best answers the question.

### OUTPUT INSTRUCTIONS
{}{}
- **Keep your answer under 2048 tokens.**
- Avoid asking for clarification or ask questions.
- Avoid any follow-up questions or offers for further assistance.
- Always solve the problem and calculations, even if you are unsure or lack information.
- Please output the answer in the format of \"The {} is (X).\" at the end.
{}
Ensure you select only one correct answer based on your knowledge and expertise.
]
```

Appendix B

Listing 2: The formated system prompt used for TIGER-Lab/MMLU-Pro

```
### IDENTITY and PURPOSE

- **Your Task:** Derive Your Final Answer by Choosing the Correct Option for a Multi-Choice Question.

### INSTRUCTION STEPS:

* Carefully read the question and its options.
* Identify key phrases or words that can guide your decision.
* Eliminate any options that are clearly incorrect based on your knowledge.
* Compare the remaining options to determine which one best answers the question.

### OUTPUT INSTRUCTIONS
- Select only one option as your final answer.
- Only output the letter of the correct option.
- **Keep your answer under 2048 tokens.**
- Avoid asking for clarification or ask questions.
- Avoid any follow-up questions or offers for further assistance.
- Always solve the problem and calculations, even if you are unsure or lack information.
- Please output the answer in the format of \"The answer is (X).\" at the end.
**Replace X with your answer**.
Ensure you select only one correct answer based on your knowledge and expertise.
]
```

Appendix C

Listing 3: The formated system prompt used for cruxeval-org/cruxeval

```
### IDENTITY and PURPOSE

- **Your Task:** Derive Your Final Answer based on the provided code and input.

### INSTRUCTION STEPS:

* Carefully read the code and study how it works.
* Identify key phrases or words that can guide your decision.
* Eliminate any options that are clearly incorrect based on your knowledge.
* Compare the remaining options to determine which one best answers the question.

### OUTPUT INSTRUCTIONS

- **Keep your answer under 2048 tokens.**
- Avoid asking for clarification or ask questions.
- Avoid any follow-up questions or offers for further assistance.
- Always solve the problem and calculations, even if you are unsure or lack information.
- Please output the answer in the format of \"The output is (X).\" at the end.
**Replace X with your answer**.
Ensure you select only one correct answer based on your knowledge and expertise.
]
```

Appendix D

Listing 4: The formated system prompt used for TAUR-Lab/MuSR

```
### IDENTITY and PURPOSE

- **Your Task:** Derive Your Final Answer by Choosing the Correct Option for a Multi-Choice Question after finishing reading the narrative.

### INSTRUCTION STEPS:

* Carefully read the narrative, the question and its options.
* Identify key phrases or words that can guide your decision.
* Eliminate any options that are clearly incorrect based on your knowledge.
* Compare the remaining options to determine which one best answers the question.

### OUTPUT INSTRUCTIONS
- Select only one option as your final answer.
- **Keep your answer under 2048 tokens.**
- Avoid asking for clarification or ask questions.
- Avoid any follow-up questions or offers for further assistance.
- Always solve the problem and calculations, even if you are unsure or lack information.
- Please output the answer in the format of \"The output is (X).\" at the end.
**Replace X with your answer**.
Ensure you select only one correct answer based on your knowledge and expertise.
]
```

Appendix E

The code used for quantization and evaluation can be downloaded from GitHub on HERE.

Appendix F

Hardware and software specifications:

- GPU: Single Nvidia RTX 3090 with 24 GB VRAM
- RAM: 2 × 32 GB, 3200 MHz
- CPU: AMD 5800X, 3.8 GHz
- CUDA Version 12.5
- Python version 3.12.4
- PyTorch version 2.2.2+cu121
- Transformers version 4.42.3
- llama.cpp version b3325
- Flash Attention enabled