Methods of Reducing Computational Requirements for Large Language Models

Oleksandr Kononov *

* *South East Technological University, Cork Road, Waterford, Ireland (e-mail: 20071032@mail.wit.ie).*

Abstract: The increasing computational requirements of Large Language Models (LLMs) poses significant challenges to their deployment and accessibility, especially for consumers and small organisations with limited compute resources. This research aims to investigate viable and efficient LLM compression methods such as Post Training Quantization (PTQ) and pruning to reduce the hardware requirements of large LLMs.

Keywords: Artificial intelligence, Neural networks

1. INTRODUCTION

1.1 Background

An LLM is neural network model which is capable at working with natural language tasks such as text generation, text summarization, translation, and more. The novel Transformer architecture proposed in the "Attention Is All You Need" paper [27] has revolutionized the field by introducing more efficient multi-headed self-attention mechanism compared to Recurrent Neural Networks that came before. Following this, OpenAI have used this transformer architecture to design and develop their Generative Pre-Trained Transformer (GPT) LLMs in the following years. In particular, the release of GPT-3 in 2020, has sparked a global interest in the continued development of LLMs from various companies such as Meta, Google, Antropic and others.

The AI researchers at Meta have developed a series of open-weight LLMs called LLaMa, ranging from 7B parameters to 65B parameters [25]. Their research paper demonstrates that larger number of LLM parameters in their models correlates with higher scores on benchmark tests such as HellaSwag [30], WinoGrande [22], ARC [2] and OpenBookQA [20]. However, this presents a number of challenges, especially with regards to computational requirements necessary to inference these large LLMs,"the compute and memory requirements of state-of-the-art language models have grown by three orders of magnitude in the last three years, and are projected to continue growing far faster than hardware capabilities" [1, p. 97].

Quantization and pruning are some of the strategies that can be used to help reduce computational requirement and memory footprint of LLMs. However applying these strategies often comes at the cost of increasing the LLM Perplexity (PPL), a metric for evaluation the uncertainty of a model in predicting a sequence of tokens. Quantization methods involve reducing the numerical precision of the model's weights, such as allowing 32-bit value to be represented as an 8-bit value for example. Popular quantization methods include GPT-Generated Unified Format (GGUF) [6, 7], Activation-Aware Weight Quantization (AWQ) [15], Vector Post-Training Quantization (VPTQ) [17] and others. Pruning on the other hand, involves removing parts of the model that have little effect on the output, this process could involve removing blocks or entire layers and often requires some level of model recovery after this procedure.

1.2 Problem Statement and Motivation

As mentioned in the previous section, the growing hardware requirements for medium to large sized LLMs make it difficult for consumers or small organisations to run their own local LLMs, often requiring to use third-party providers for access to powerful LLMs. This has the potential to reduce their privacy, security and accessibility, which could be otherwise achieved by running LLMs locally on their own hardware. For large businesses who might already be hosting their own models, this could be an opportunity to potentially reduce their running costs with regards to LLMs.

If in the future, small sized LLMs become more capable than they are today, it still stands to reason that their larger counterparts would likewise become more capable. Therefore, finding efficient and cost effective methods of reducing hardware requirements for running large LLMs holds meaningful significance in helping to democratize access to powerful LLMs.

1.3 Research Scope and Limitations

This research will be using a select few foundational LLMs for testing and evaluation. The **Selected LLMs** will be Gemma2 9B from Google [23], LLaMa 3.1 8B from Meta [4] and Qwen2.5 7B from Alibaba [24]. These models were selected due to their research permissive licenses, community popularity and the reputability of parent companies that have trained them.

Due to time limitations, this research will not be using all available PTQ methods, the **Selected PTQ methods** will be GGUF, AWQ and VPTQ.

The hardware for conducting this research will be limited to a single Nvidia RTX 4090 GPU with 24GB of VRAM, which will be sufficient to run the **Selected LLMs** without any modifications. This will allow the researcher to establish baseline metrics and benchmark scores, that can be used to compare against the results of compressed models from the **Selected LLMs**.

The hardware used to demonstrate the effects and performance of LLM quantization will be a single Raspberry Pi 4b, which has quad-core Cortex-A72 @ 1.5GHz CPU and 8GB LPDDR4 RAM [21]. This device was chosen for it's limited hardware specifications, that should under normal circumstances be insufficient run any of the **Selected LLMs** previously mentioned. As such, it qualifies to be a test edge device for the purposes of this research and would serve as a baseline for future research evaluating more capable devices.

This research will be limited to exploring PTQ methods and not Quantization Aware Training (QAT) methods, as the latter requires significant computational resources and time in order to carry out such research. PTQ methods require significantly less computational resources and can be used on existing pre-trained LLMs.

With regards to benchmark and evaluation tests, there exists a large pool of datasets curated for various use cases. This research will select a subset of popular and often referenced datasets from categories such as **General Knowledge and Language Understanding**, **Reasoning Capabilities**, **Truthfulness** and **Instruction Following**.

1.4 Research Questions

This paper aims to answer the following Research Questions (RQs):

RQ1: How do the **Selected PTQ methods** compare in terms of reducing inference requirements for **Selected LLMs**, while retaining output quality, and which method ranks highest among them?

RQ2: What pruning strategies are the most effective for reducing **Selected LLMs** inference requirements while retaining most of it's output quality?

RQ3: Deriving the best performing method (or combination of methods) from the previous questions, what is the best achievable quality and performance of **Selected LLMs** on a Raspberry Pi 4b?

2. PRELIMINARY LITERATURE REVIEW

This section will examine the supporting research literature for the **Selected PTQ methods**, explore pruning methods and discuss the benchmark tests utilized in the industry for evaluating LLMs.

2.1 GGUF PTQ

One of the most well known LLM PTQ methods within the open-source/open-weight LLM community is GGUF. Developed by Georgi Gerganov in 2022, it was initially developed as a machine learning library written in C and C++ called GPT-Generated Model Language (GGML). Unlike other holistic machine learning libraries like PyTorch and TensorFlow, GGML primary focus was Transformer model inferencing while being minimal, lightweight and efficient [13]. It introduced the GGML quantization file format to facilitate easy sharing and execution of models, ensuring that it contains all necessary information required for loading in a single file. The successor to GGML is GGUF, a new format that is designed to address the limitations of the GGML format by incorporating support for versioning, model metadata, and greater compatibility with various LLMs architectures [8] (see figure 1).

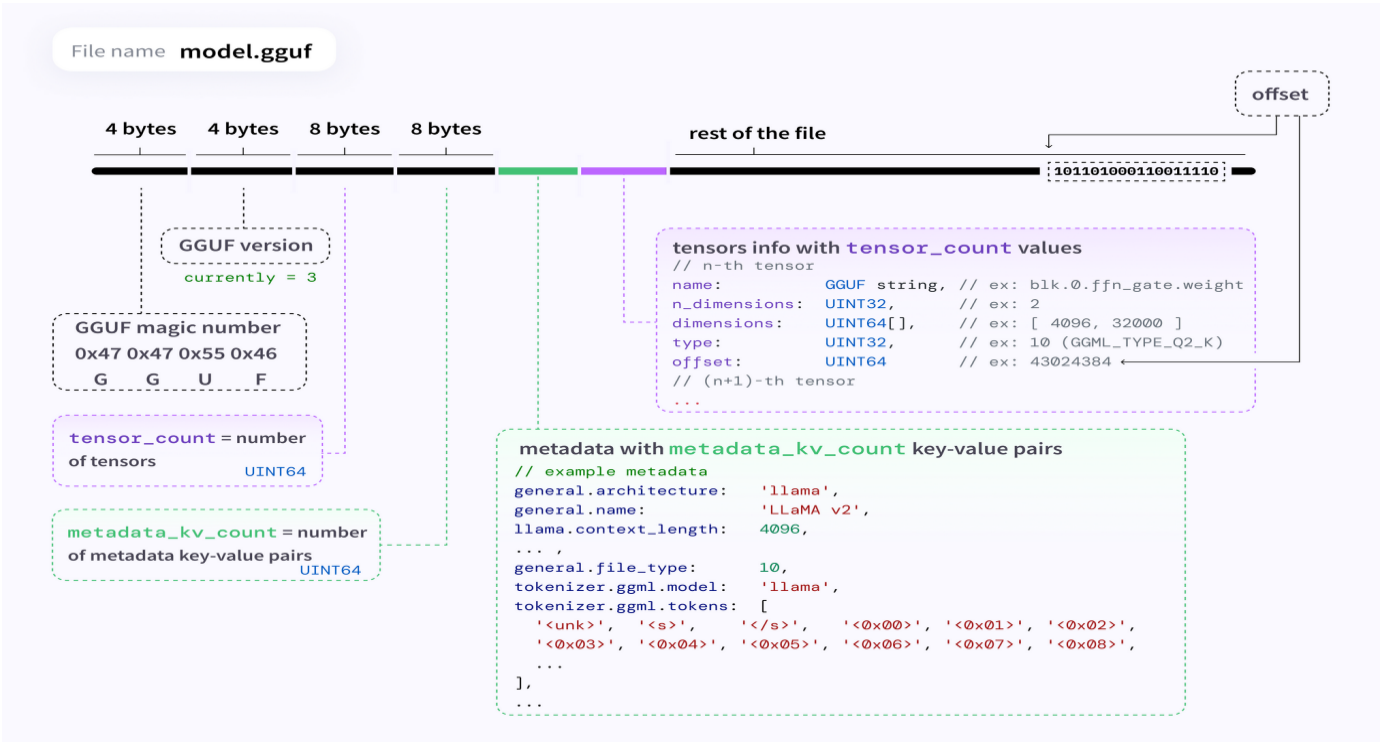

Fig. 1: GGUF Format Breakdown (source:[8]).

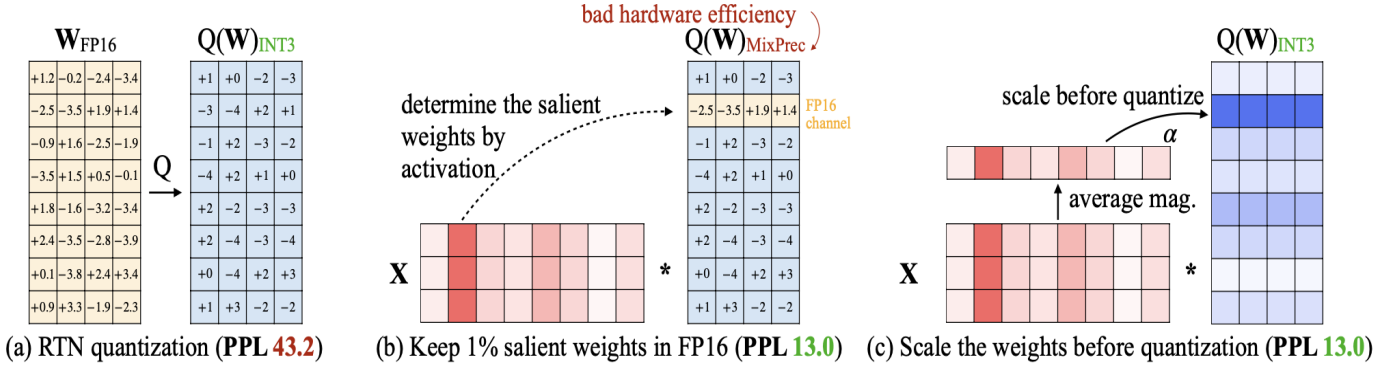

Fig. 2: AWQ method showing how scaling *salient* weights before quantizations shows comparable results to mixed precision weights but without the hardware inefficiencies mix precision introduces (source:[15]).

The `llama.cpp` software developed by Georgi Gerganov and the open-source community is used to load the GGUF format. After it is loaded and processed based on the stored metadata, the underlying GGML library can then begin inferencing the LLM [9]. The GGUF format offers various quantization options, originally the quantization method was to simply split each layer into blocks of 256 weights and each block is then converted into their 256 quantized values, this was denoted with the letter **Q**. Additionally, there are two quantization type, "type-0" (**Q4_0**, **Q5_0**, etc.) where weights w are calculated from quants q and block scale d using $w = d \times q$. While "type-1" (**Q4_1**, **Q5_1**, etc.) include an additional block minimum constant m such that $w = d \times q + m$ [10, 12].

Later the community have developed the **K** Quants, which expand the quantization range to include 2-bit, 3-bit and 6-bit quantization and prioritise certain weights over others (as denoted by suffixes like **Q3_K_S**, **Q3_K_M**, **Q3_K_L**) leading to smaller models with minimal PPL increase [12]. The newest form of GGUF quants are the **I** quants, which add very low bit quantization capability [11] and are implemented based on the QuIP# paper. According to this research paper, QuIP# achieves low-bit quantization through clever groupings of quants. For instance, even numbers of either positive or negative signed values are grouped into 8 quants, allowing sign information to be recorded using only 7 bits. This, combined with the use of an E8 lattice structure, enables very low-bit quantization [11, 26].

2.2 AWQ PTQ

Similar to the **K** Quants of GGUF, the AWQ method proposes a similar approach of selectively quantizing LLM weights depending on their importance by analysing the model activation patterns [15]. This method identifies *salient* weights in the LLM that hold more importance to the LLM performance and withhold quantization for those weights, thereby avoiding significant performance degradation when reducing the model size. However, having mixed precision weights is not hardware-efficient, it was found that scaling the weights before quantization mitigates this issue while still preserving the benefits (see figure 2).

2.3 VPTQ PTQ

Similar to the **I** Quants of GGUF, the VPTQ method is a low bit LLM quantization method that claims better accuracy for 1-2 bit LLM quantization compared to other conventional methods. It achieves this by compressing vectors into indices using lookup tables, with further refinement of weights using Channel-Independent Second-Order Optimization [17]. It differs from the previously covered AWQ method as instead of reducing the precision of weights, it builds an index that maps high-dimensional vectors to lower-dimensional vectors. According to the graph on figure 3, its performance is comparable to QuIP# method which the **I** Quants of GGUF are based on.

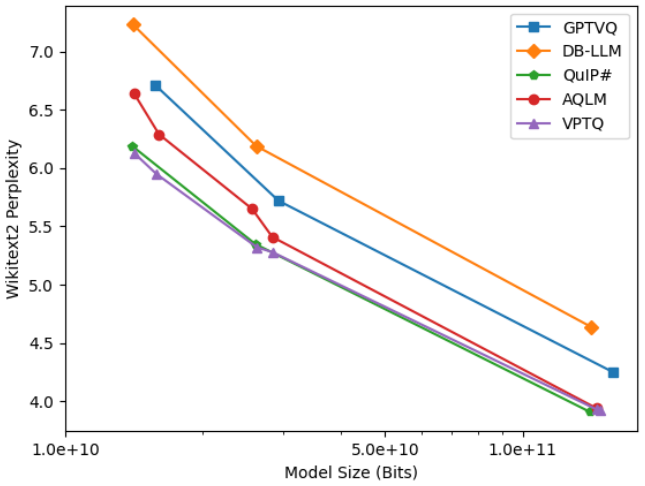

Fig. 3. Graph showing PPL of Wikitext2 dataset compared to model size using various PTQ methods (source:[19]).

2.4 LLM Pruning

Model pruning is a way of compressing the size of a model by removing certain components from the model while avoiding severe damage to how the model functions. This is achieved by removing redundant or unimportant singular or groups of neurons [14]. There are two type

of pruning methods that can be performed on a model, structured pruning and unstructured pruning, the former involves simplifying the model by removing entire structural weights such as channels or layers while maintaining the network structures, while the latter focuses on removing redundant neurons or links [14]. Between the two approaches, structured pruning has much better hardware compatibility compared to unstructured pruning which maybe require additional software or hardware treatment to complete the task [14].

LLM-Pruner is a tool developed to perform structural pruning on an LLM using gradient-based optimization processes for structure selection [18]. The LLM-Pruner uses an algorithm to detect dependencies within a model, this allows them to select optimal structured groups for pruning based on their importance estimation. Like with most pruning methods, a post pruning re-training is required, which can be called a "recovery" phase, where it is trained on a small dataset in order to help maintain its performance [18].

Unlike LLM-Pruner, the Bonsai pruning method does not use a gradient-based structured pruning and is designed for more typical consumer grade hardware to execute [3]. It instead generates sub-models and evaluates their performance to give a more holistic view, which claim to outperform State Of The Art (SOTA) "gradient-based structured pruning methods like LLM-Pruner" [3, p. 2] on 4/6 evaluation tests. Similar to LLM-Pruner, the Bonsai method performs post pruning operations to help recover some lost performance, but unlike LLM-Pruner, it uses distillation from the original model to the pruned model. However, in some cases this post pruning adaptation may not be necessary due to the robustness of the LLM as well as redundancy of its modules [3].

2.5 LLM Benchmarking

There are many benchmark dataset available that are designed to test various capabilities of LLMs, and as mentioned in the research scope section, this research will focus on a subset of datasets from three categories of **General Knowledge and Language Understanding**, **Reasoning Capabilities**, **Truthfulness** and **Instruction Following**. Starting with Massive Multitask Language Understanding (MMLU) Pro dataset which is an iterative improvement over the original MMLU dataset by removing trivial and noisy questions [28]. It is a dataset with a diverse fields such as mathematics, computer science, physics, chemistry, biology, business and more to produce over 12,000 questions [28]. The dataset is constructed as series of multiple choice questions with up to 10 possible response options, compared to only 4 in the original MMLU dataset, which should reduce random guessing score [29].

Next dataset to look at is HellaSwag, which similar to MMLU part of the **General Knowledge and Language Understanding** category. This dataset evaluates an LLMs capability of answering questions in a contextually appropriate and common-sense manner, referred to as *common-sense natural language inference* [30]. Similar to MMLU the dataset is structured as multiple choice questions.

Moving onto the **Reasoning Capabilities** category, AGIEval dataset uses various human-centric exams to evaluate a models reasoning capabilities [31]. The dataset is constructed with a particular focus on human-level cognitive tasks and real-world scenarios following a wide range of exam papers like mathematics exams, lawyer qualification tests, college admission tests and other qualification exams [31, p. 5]. The format of the dataset is comprised of multiple choice with an addition of some fill-in-the-blank questions that employ exact matching strategy during evaluation [31, p. 6].

For the **Truthfulness** category there exists a TruthfulQA dataset that evaluates a model for providing accurate and unbiased information. The benchmark consists of over 800 questions that covers 38 categories which include law, health, finance, and politics [16]. This dataset was designed to help address the concerns of accidental and/or malicious LLM misuse. It relies heavily on true or false questions and uses Wikipedia to source the factual information. The questions in the dataset were designed to be adversarial such that "questions test a weakness to imitative falsehoods: false statements with high likelihood on the training distribution" [16, p. 4]

Finally, for the **Instruction Following** there is the Instruction-Following Eval (IFEval) dataset which was designed to evaluate how well a model follows a set of instructions in the prompt using verifiable instructions. A verifiable instruction is a specific, measurable and objective directive that, upon completion, can be checked and confirmed for adherence [32]. The IFEval has identified 25 types of verifiable instructions, with the dataset comprising 500 prompts based on these types. For example, an instruction prompt might ask an LLM to generate 25 sentences, the output of which can be automatically and objectively verified [32]. This dataset is highly valuable as it evaluates a key LLM capability: instruction following. This skill is crucial for performing well on multiple-choice questions, a common evaluation method in many other datasets. Additionally, these questions require structured responses for correct evaluation, making instruction-following a foundational skill in LLM evaluations.

2.6 Summary

This preliminary literature review section has explored some of the existing LLM PTQ methods such as GGUF, AWQ and VPTQ. All of the discussed methods operate based on similar principals of reducing the precision of the numeric values within the model weights. The GGUF, being a community driven open-source project has more options for quantization available to it in the form of **K** quants (which selectively quantize certain weights over others) and **I** quants. Following from this, the LLM pruning methods LLM-Pruner and Bonsai function by removing parts of the model which has little impact on its quality. The differing feature between the two approaches lies with its selection process for the structured groups (gradient-based versus sub-model evaluation). Finally, benchmarking review covered various datasets that evaluate specific model capabilities such as **General Knowledge and Language Understanding**, **Reasoning Capabilities**, **Truthfulness** and **Instruction Following**.

3. WORKING THEORY

Following from the preliminary literature review, this section will define Theoritical Propositions (TPs) based on the previously reviewed literature which will help guide this paper's RQs.

TP1: *The effectiveness of PTQ techniques in reducing computational requirements is influenced by the LLM architecture, where certain architectures benefit more than others.*
Just like how the architecture of the model must be known in order to load it for inferencing, it must be known in order to perform PTQ operations. Given that there are many emerging architectures for LLMs, it stands to reason that some architectures would have greater support for quantization compared to others. For example, since Meta's LLaMa has a strong community following, it is reasonable to expect it would receive better support from the community compared to more obscure models.

TP2: *The combination of PTQ and pruning can achieve significant reduction in hardware requirements compared to using only one approach over the other.*
As covered earlier, quantization operates primary relies on reducing numerical precision of the models' weights, and does not alter the structural behaviour of the model, unlike pruning methods. Therefore, it is reasonable to consider that the two methods are non-mutually exclusive and in theory can be applied to the same model and thereby greatly reduce its hardware requirements. There has been limited research done on this to date, and it is an area worth exploring further, even if the effects end up being destructive.

TP3: *The impacts of PTQ and pruning on LLM performance can vary across different types of tasks (i.e., Instruction Following, Knowledge).*
During LLM training, instruction following capabilities are fine-tuned on a base model, which has been trained solely to predict the next token in a sequence, unlike its "knowledge" capabilities which would be instilled throughout the training process. Considering how non-uniform the training process is with respect to different LLM capabilities, it stands to reason that impacts of model compression techniques would not be distributed uniformly.

TP4: *LLMs reasoning tasks will be more strongly impacted compared to knowledge retrieval tasks after applying a PTQ or pruning method.*
Following from the previous TP, reasoning tasks such as problem-solving and logical inference would require a model to be able to integrate information from multiple parts of its learned knowledge base, while knowledge retrieval is a more straightforward process. Given that model compression has some impact to quality, it is likely that reasoning related tasks would be more heavily impacted compared to knowledge retrieval tasks.

4. RESEARCH DESIGN

4.1 Introduction

This section will detail the methodology for determining the most effective method to reduce hardware requirement for running LLMs. The process will involve systematic evaluation and comparison of various compression methods described in this paper, applied to the **Selected LLMs**. To evaluate various aspects of post-compression LLMs, the previously discussed benchmark tests will be used and compared to their pre-compression evaluations. The research will also use a Raspberry Pi 4b as a constant constraint to determine the most effective method or combination of methods from those previously outlined, thereby demonstrating their feasibility and practicality in real-world applications.

4.2 Design

Initially, it is necessary to establish a baseline against which the experiments can be compared against. To do this, it is necessary to be able to run the unmodified **Selected LLMs** and gather various metrics from it such as PPL (see appendix A), Tokens Per Second (TPS) output speed and the results of the benchmark tests for all the categories. The LLM sampling parameters must be deterministic for all tests, this can be done by setting the temperature to 0 and top_k to 1, additionally the seed should be kept constant for reproducibility. This must be done for all the **Selected LLMs** recorded to be used as baseline for each model.

After the baseline metrics have been recorded, each of the **Selected LLMs** must be quantized using each of the **Selected PTQ methods**. Following this, like with the baseline metrics, the sampling parameters must be deterministic, and all the same metrics must be recalculated for the quantized models and recorded. For certain quantization method like VPTQ which are new and have less community support, some coding might be required to support certain LLM architectures.

After all metrics for the **Selected PTQ methods** have been recorded, the same process must be done for the pruning methods using LLM-Pruner and Bonsai, likewise setting deterministic sampling parameters.

In order to satisfy **RQ1**, it's important to have a formula for scoring the models based on the gathered metrics. We can use the expected decrease in benchmark score to compare against the uncompressed model score.

$$D_i = \left(1 - \frac{B_i^c}{B_i^u}\right) \times 100$$

$$D_{\text{overall}} = \frac{1}{n} \sum_{i=1}^{n} D_i$$

Where D_i is the percentage decrease in benchmark i score of compressed model B_i^c over uncompressed model B_i^c.

Additionally, to get a singular quality score value for a model, we can use the following formula to combine the weighted benchmark scores, PPL and TPS:

$$Q = w_B \left(\sum_{i=1}^{n} w_i B_i\right) + w_{PPL} \left(\frac{1}{PPL}\right) + w_{TPS}(TPS)$$

Where Q is the quality score of a model based on the weighted result of n benchmarks, weighted PPL and weighted TPS.

Similarly, to satisfy **RQ2**, the same can be repeated for pruned models using the same formulas to compare uncompressed LLMs against their compressed counterpart.

For the final **RQ3**, it may be necessary to combine quantization and pruning methods. We can select the best scoring pruning method and attempt to use the best scoring PTQ method on the **Selected LLMs**. The resulting models will be attempted to be loaded onto a Raspberry Pi 4b and, if successful, be evaluated using the previously defined formulas.

4.3 Timeline

Below is an expected timeline for this research, showcasing the stages and projected time-frames for the work to be done. The *Quantization* and *Pruning* phases will involve applying the previously discussed model compression method against the **Selected LLMs**. It is likely that some form of troubleshooting and potential code implementation might be required to support the **Selected LLMs**. Once all the data has been gathered and documented, *Evaluation & Edge Device Test* phase will be used to interpret the results and apply the findings in a practical manner by using a Raspberry Pi 4b as a test edge device.

Throughout this research, new discoveries such as LLMs and new PTQs methods would be investigated in parallel to the other stages, as part of the *Revising & Updating* process. Findings from these discoveries will be documented and reflected in this research.

5. RESEARCH

*5.1 Establishing a Baseline for **Selected LLMs***

In ordered to evaluate and demonstrate the effects of compression on a model, it is first necessary to establish a baseline to which we can compare the results. As covered in the previous sections, a set of recognized academic benchmark tests will be used to give us a baseline set of scores.

Using *lm-evaluation-harness*, a unified framework for testing LLMs on a large number of evaluation tasks [5], the benchmark tests **Hellaswag**, **AGIEval**, **TruthfulQA** and **IFEval** can be run in a consistent manner against our **Selected LLMs**.

Setup The installation, setup and usage of lm-evaluation-harness is documented on their Github page, which simply involves cloning the repository and installing the package by running `pip install -e .` in the project root directory. This will install all necessary dependencies including the `transformers` package, which would be necessary to run uncompressed full-precision models. Table 1 shows the baseline tested scores, from running `lm_eval` commands as shown in figure 4.

```
lm_eval --model hf \
--model_args pretrained=google/gemma-2-9b-it \
--tasks hellaswag,agieval,truthfulqa,ifeval \
--device cuda:0 \
--batch_size 1
```

Fig. 4. lm-evaluation-harness command for Gemma2 9B

Table 1: GGUF Baseline Tested Scores and Official Scores

	Gemma2 9B		**LLama3.1 8B**		**Qwen2.5 7B**	
	Tested Score	Official Score	Tested Score	Official Score	Tested Score	Official Score
Hellaswag	81.05	81.9	79.17	No Official Score	80.47	80.2
AGIEval	49.65	52.8	42.33	47.1	59.08	No Official Score
TruthfulQA	60.18	50.27	54.04	No Official Score	64.77	56.4
IFEval	76.02	No Official Score	61.75	76.8	73.02	71.2

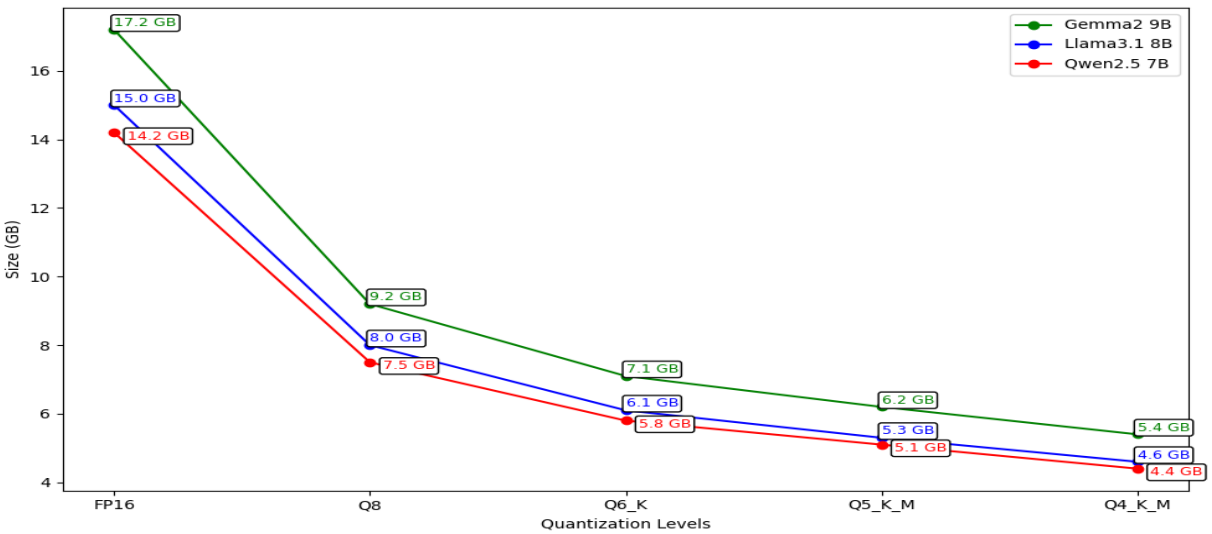

Fig. 5: GGUF Sizes

5.2 GGUF Quantization

Starting with GGUF quantization method, which requires the LLamaCPP project on Github [7]. In order to quantize a model, it is first necessary to convert (i.e. package) the model into a GGUF format, which is achieved using the `convert_hf_to_gguf.py` Python script within the LLamaCPP repository and produces a full precision FP16 GGUF model file.

In order to quantize this GGUF file to a desired quant, it's necessary to compile LLamaCPP project using `cmake`. The resulting binaries include `llama-quantize` which is responsible for quantizing the full precision GGUF file outputted by the Python script earlier.

For I-Matrix quantization types, due to their very low precision, it's required to compute the importance matrix from the full precision model in order to determine important weights, this can be done using another compiled binary `llama-imatrix` and training file (e.g. wiki.train.raw) which will be used for inferencing the model and identifying important weights. When quantizing a GGUF file with imatrix support, it's necessary to provide it the output of `llama-imatrix` so that quantization is aware of important weights.

```
# Convert gemma2-9B-it model to GGUF format
python convert_hf_to_gguf.py \
--outfile Gemma2-GGUF/ gemma-2-9B-it/

# Compile LLamaCPP binaries
cmake -B build -DGGML_CUDA=ON
cmake --build build --config Release -j 8

# Compute importance matrix
./llama.cpp/build/bin/llama-imatrix \
-m Gemma2-GGUF/gemma-2-9B-it-F16.gguf \
-f wikitext-2-raw/wiki.train.raw \
-o Gemma2-GGUF/imatrix_gemma2.dat \
--chunk 100 \
--n-gpu-layers 99

# Quantize to Q8
./llama.cpp/build/bin/llama-quantize \
./Gemma2-GGUF/gemma-2-9B-it-F16.gguf \
./Gemma2-GGUF/gemma-2-9B-it-IQ1_S.gguf \
IQ1_S

# Quantize to IQ1_S
./llama.cpp/build/bin/llama-quantize \
--imatrix Gemma2-GGUF/imatrix_gemma2.dat \
./Gemma2-GGUF/gemma-2-9B-it-F16.gguf \
./Gemma2-GGUF/gemma-2-9B-it-IQ1_S.gguf \
IQ1_S
```

Fig. 6. GGUF quantization command for Gemma2 9B

Gemma2 9B The table 2 shows the Gemma2 9B model quantized using GGUF quantization and it's performance on various benchmarks.

Table 2. Gemma2 9B GGUF Scores

Benchmark Test	Q8	Q6_K	Q5_K_M	Q4_K_M
Hellaswag	65.45	65.21	65.08	64.79
AGIEval	-	-	-	-
TruthfulQA	62.25	62.15	62.07	62.37
IFEval	34.41	35.49	35.61	35.97

LLama3.1 8B The table 3 shows the LLama3.1 8B model quantized using GGUF quantization and it's performance on various benchmarks.

Table 3. LLama3.1 8B GGUF Scores

Benchmark Test	Q8	Q6_K	Q5_K_M	Q4_K_M
Hellaswag	79.19	79.09	78.99	78.74
AGIEval	-	-	-	-
TruthfulQA	54.01	53.78	53.37	52.56
IFEval	59.95	59.83	59.59	56.35

Qwen2.5 7B The table 4 shows the Qwen2.5 7B model quantized using GGUF quantization and it's performance on various benchmarks.

Table 4. Qwen2.5 7B GGUF Scores

Benchmark Test	Q8	Q6_K	Q5_K_M	Q4_K_M
Hellaswag	80.46	80.34	80.37	80.19
AGIEval	-	-	-	-
TruthfulQA	64.72	65.23	65.37	64.28
IFEval	71.34	73.50	71.10	71.46

6. CONCLUSION

In conclusion, this research proposal has suggested various popular LLM compression methods that can be used to reduce hardware requirements for inferencing LLMs. The proposal has laid out its working theory and design methods for how this research will be conducted.

REFERENCES

[1] Rishi Bommasani, Drew A. Hudson, et al. On the opportunities and risks of foundation models, 2022. https://arxiv.org/abs/2108.07258.

[2] Peter Clark, Isaac Cowhey, et al. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. https://arxiv.org/abs/1803.05457.

[3] Lucio Dery, Steven Kolawole, Jean-François Kagy, Virginia Smith, Graham Neubig, and Ameet Talwalkar. Everybody prune now: Structured pruning of llms with only forward passes, 2024. https://arxiv.org/abs/2402.05406.

[4] Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. https://arxiv.org/abs/2407.21783.

[5] Leo Gao, Jonathan Tow, et al. lm-evaluation-harness: A framework for few-shot language model evaluation, 07 2024. https://zenodo.org/records/12608602.

[6] Georgi Gerganov. ggml, 2022. https://github.com/ggerganov/ggml.

[7] Georgi Gerganov. llama.cpp, 2023. https://github.com/ggerganov/llama.cpp.
[8] Georgi Gerganov. Ggml, 2024. https://github.com/ggerganov/ggml/blob/master/docs/gguf.md.
[9] Georgi Gerganov. Gguf, 2024. https://github.com/ggerganov/ggml.
[10] Georgi Gerganov. Gguf quantize, 2024. https://github.com/ggerganov/llama.cpp/tree/master/examples/quantize.
[11] Georgi Gerganov and Kawrakow. Gguf i quants pull request, 2024. https://github.com/ggerganov/llama.cpp/pull/4773.
[12] Georgi Gerganov and Kawrakow. Gguf k quants pull request, 2024. https://github.com/ggerganov/llama.cpp/pull/1684.
[13] Georgi Gerganov and Xuan Son Nguyen. Introduction to ggml, 2024. https://huggingface.co/blog/introduction-to-ggml.
[14] Hanjuan Huang, Hao-Jia Song, and Hsing-Kuo Pao. Large language model pruning, 2024. https://arxiv.org/abs/2406.00030.
[15] Ji Lin, Jiaming Tang, et al. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. https://arxiv.org/abs/2306.00978.
[16] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022. https://arxiv.org/abs/2109.07958.
[17] Yifei Liu, Jicheng Wen, et al. Vptq: Extreme low-bit vector post-training quantization for large language models, 2024. https://arxiv.org/abs/2409.17066.
[18] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models, 2023. https://arxiv.org/abs/2305.11627.
[19] Microsoft. Vptq: Extreme low-bit vector post-training quantization for large language models, 2024. https://github.com/microsoft/VPTQ/blob/main/README.md.
[20] Todor Mihaylov, Peter Clark, et al. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. https://arxiv.org/abs/1809.02789.
[21] Raspberry Pi Ltd. *Raspberry Pi 4 Model B*, 4 2024. https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.
[22] Keisuke Sakaguchi, Ronan Le Bras, et al. Winogrande: An adversarial winograd schema challenge at scale, 2019. https://arxiv.org/abs/1907.10641.
[23] Gemma Team, Morgane Riviere, et al. Gemma 2: Improving open language models at a practical size, 2024. https://arxiv.org/abs/2408.00118.
[24] Qwen Team. Qwen2.5: A party of foundation models, September 2024. https://qwenlm.github.io/blog/qwen2.5/.
[25] Hugo Touvron, Thibaut Lavril, et al. Llama: Open and efficient foundation language models, 2023. https://arxiv.org/abs/2302.13971.
[26] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024. https://arxiv.org/abs/2402.04396.
[27] Ashish Vaswani, Noam Shazeer, et al. Attention is all you need, 2017. https://arxiv.org/abs/1706.03762.
[28] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark (published at neurips 2024 track datasets and benchmarks), 2024. https://arxiv.org/abs/2406.01574.
[29] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, et al. Mmlu-pro huggingface, 2024. https://huggingface.co/datasets/TIGER-Lab/MMLU-Pro.
[30] Rowan Zellers, Ari Holtzman, et al. Hellaswag: Can a machine really finish your sentence?, 2019. https://arxiv.org/abs/1905.07830.
[31] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, et al. Agieval: A human-centric benchmark for evaluating foundation models, 2023. https://arxiv.org/abs/2304.06364.
[32] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, et al. Instruction-following evaluation for large language models, 2023. https://arxiv.org/abs/2311.07911.

GLOSSARY

Selected PTQ methods GGUF, AWQ, VPTQ. 1, 2, 5
Selected LLMs Gemma2B 9B, LLaMa 3.1 8B, Qwen2.5 7B. 1, 2, 5, 6

AWQ Activation-Aware Weight Quantization. 1, 3, 4, 9

GGML GPT-Generated Model Language. 2, 3
GGUF GPT-Generated Unified Format. 1–4, 6, 7, 9
GPT Generative Pre-Trained Transformer. 1

IFEval Instruction-Following Eval. 4

LLM Large Language Model. 1–7, 9

MMLU Massive Multitask Language Understanding. 4

PPL Perplexity. 1, 3, 5, 6
PTQ Post Training Quantization. 1–6, 9

QAT Quantization Aware Training. 2

RQ Research Question. 2, 5, 6

SOTA State Of The Art. 4

TP Theoritical Proposition. 5
TPS Tokens Per Second. 5, 6

VPTQ Vector Post-Training Quantization. 1, 3–5, 9

Appendix A. PERPLEXITY CALCULATION CODE

```
# gist for the blog post on https://safjan.com
import tiktoken
import math

# Exemplary text
text = "Another vital aspect to consider in the realm of text generation is perplexity. Essentially, perplexity refers to the amount of information contained within a text. Natural language possesses a remarkable redundancy that enables effective communication, even in noisy environments such as a crowded bar or an intimate dinner. This redundancy allows for the overall message to be comprehended, even if certain parts of the text are missing or obscured."

# Tokenize the text using tiktoken
tokens = tiktoken.tokenize(text)

# Calculate the total number of tokens
total_tokens = len(tokens)

# Create a dictionary to count the frequency of each token
token_freq = {}
for token in tokens:
if token in token_freq:
token_freq[token] += 1
else:
token_freq[token] = 1

# Calculate the probability of each token
token_probabilities = {token: freq / total_tokens for token, freq in token_freq.items()}

# Calculate the text probability by multiplying the probabilities of each token
text_probability = 1.0
for token in tokens:
text_probability *= token_probabilities[token]

# Calculate the perplexity using the formula
perplexity = math.pow(text_probability, -1/total_tokens)

print(f"Perplexity of the text: {perplexity:.2f}")

```

Listing 1: https://gist.github.com/izikeros/e97a9d3359f3872b5e74cb36380c46ae