Empirical Evaluation of Post Training Quantisation Techniques for AI Models: A Comparative Study of Standard and Novel Approaches

Honours Thesis

Thomas Mikic

Bachelor of Software Engineering (Honours)
at **Macquarie University**

Faculty of Science and Engineering
School of Computing
Supervisor: Dr. Yuankai Qi

MACQUARIE
University
SYDNEY · AUSTRALIA

November 9, 2025

Contents

Declaration

This thesis is submitted to **Macquarie University** in fulfilment of the requirement for the degree of **Bachelor of Software Engineering (Honours)**.

The work presented in this thesis is, to the best of my knowledge and belief, original except as acknowledged in the text. I hereby declare that I have not submitted this material, either in full or in part, for a degree at this or any other institution.

Thomas Mikic
November 9, 2025

Generative AI Statement

I, **Thomas Mikic**, acknowledge the use of artificial intelligence (AI) tools in the research project and the writing of this thesis.

In this work:

I used ChatGPT and Gemini 2.5, [1, 2] to aid the process of gathering and researching papers.
I used ChatGPT, [1] to aid in the writing process, including grammatical and punctuation issues.

I am aware of the AI tools limitations and ultimately, I am responsible for the content, research and developing my ideas described in this document.

Thomas Mikic
November 9, 2025

Acknowledgements

I would like to express my sincere gratitude to everyone who has supported me throughout the course of this thesis.

First and foremost, I thank my supervisor, Dr. Yuankai Qi, for his invaluable guidance, patience, and encouragement. His expertise and insightful feedback greatly contributed to the success of this research.

I also extend my thanks to the faculty and staff of the School of Computing at Macquarie University for providing a supportive academic environment.

Thank you all.

Abstract

Artificial Intelligence models achieve exceptional performance across natural language and vision tasks but require substantial memory and computational resources that limit deployment on resource constrained devices. This thesis investigates post training compression techniques to reduce model size whilst preserving accuracy, comparing standard quantisation methods against custom approaches including attention based token pruning and outlier aware activation quantisation.

Comprehensive evaluation was conducted across five model architectures: Qwen2 0.5B (494M parameters), TinyLlama (1.1B parameters), Phi 2 (2.7B parameters), Mistral 7B (7.2B parameters), and LLaVA 1.5 7B (multimodal, 7.2B parameters). Models were tested using established benchmarks (HellaSwag, ARC Easy, and visual question answering) on NVIDIA A100 GPUs. Three categories of compression were tested: (1) standard weight only quantisation using BitsAndBytes (INT4/INT8), (2) attention based token pruning inspired by TRIM methodology, and (3) custom activation quantisation with outlier detection and layer wise dynamic precision.

The principal finding is that standard INT4 weight only quantisation substantially outperforms theoretically motivated custom techniques. Standard quantisation achieved 52 to 75 percent memory reduction with only 1 to 7 percent accuracy degradation on well trained models, whilst custom approaches caused 20 to 45 percent accuracy loss and introduced 2 to 5 times latency overhead. Specifically, attention based token pruning reduced average accuracy by 28.5 percent due to inadequate semantic preservation, outlier aware activation quantisation increased inference latency from 553 milliseconds to 1496 milliseconds (170 percent overhead), and layer wise dynamic quantisation showed worst performance at 21.5 percent average accuracy degradation.

Results demonstrated strong model quality dependence: larger models like Mistral 7B and LLaVA 1.5 7B maintained near baseline accuracy with INT4 quantisation (96 to 100 percent retention), whilst smaller models like Qwen2 0.5B exhibited higher sensitivity with 16 percent degradation. An optimised selective quantisation strategy maintained baseline accuracy for Phi 2 but failed catastrophically on Mistral 7B (30 percent vs 66 percent baseline), indicating that layer sensitivity is architecture specific and cannot be determined through heuristics alone.

This work contributes: (1) rigorous empirical baselines demonstrating standard INT4 quantisation is production ready for memory constrained deployment, (2) documentation of failure modes in custom techniques showing post training approaches have fundamental limits without training time integration, and (3) evidence that implementation maturity dominates practical effectiveness more than algorithmic novelty. For practitioners, standard INT4 quantisation using mature libraries (BitsAndBytes, GPTQ) represents the most reliable approach for deploying AI models on memory constrained hardware.

0.1 Background

Artificial Intelligence (AI) is a multidisciplinary field focused on building systems that perform tasks typically requiring human intelligence, such as reasoning, perception, and language understanding. The field's foundations were laid by Alan Turing's work on machine intelligence [3] and the formal establishment of AI at the 1956 Dartmouth Conference [4].

Machine Learning (ML), a core subfield of AI, emerged to enable systems to learn from data rather than relying on hardcoded rules. Early models like support vector machines (SVMs) [5] and neural networks laid the groundwork for more advanced techniques. While neural networks were proposed in the 1950s, they gained traction in the 2000s with improvements in data availability and computational power.

A major breakthrough came in 2012 when a deep convolutional neural network known as AlexNet dramatically outperformed competitors in the ImageNet challenge [6], sparking the deep learning revolution. In natural language processing (NLP), the advent of Transformers [7] led to models like BERT and GPT, which leveraged self attention mechanisms to set new benchmarks in understanding and generation tasks.

Despite their success, large models like GPT 3 demand immense computational resources, prompting research into model compression techniques such as quantisation and pruning [8] to enable more efficient deployment.

0.2 Problem Statement

The core question addressed in this thesis is:

> *How can artificial intelligence models be optimised to operate efficiently on resource constrained embodied devices while preserving performance levels comparable to their original large scale counterparts?*

To be specific, this thesis will dive into a more specialised area of multi modal models:

> *How can AI models be optimised post training to operate efficiently on resource constrained embodied devices such as laptops while preserving performance levels comparable to their original large scale counterparts?*

Existing work on model optimisation or compression involves quantisation, pruning, low rank factorisation, knowledge distillation, and transfer learning, but no single technique provides a simple yet effective framework for such.

0.3 Research Objective

To address the problem above, this thesis will:

1. Survey and critically evaluate existing model compression techniques.
2. Implement and empirically test both standard and custom compression approaches.
3. Comprehensively evaluate compression effectiveness across multiple dimensions.
4. Compare custom techniques against standard quantisation baselines.
5. Document failures and provide deployment recommendations.

0.4 Scope and Limitations

This work focuses exclusively on post training compression techniques, specifically quantisation and token pruning, applied to existing transformer based AI models. The objective is to evaluate the trade offs between accuracy and resource efficiency without modifying the model architecture or performing any additional training.

This work does **not** investigate:

- Training or fine tuning of AI models.
- Design or implementation of new model architectures.
- Compression methods requiring retraining or iterative optimisation.
- Hardware specific acceleration or deployment to custom platforms.
- Use of cloud based computing resources, all tests are run on off the shelf GPU and CPU hardware.

Limitations of the Study

- Evaluations are conducted on a limited set of downstream tasks, and do not cover all multi modal or edge deployment scenarios.
- The interaction between quantisation and pruning may be sensitive to model size, layer depth, and specific task configurations.
- Performance is evaluated under a single shot post training compression pipeline, and may not represent the maximum gains achievable with hybrid or adaptive methods.

0.5 Research Questions

To guide this investigation, we pose the following questions:

1. **RQ1:** How do standard INT4/INT8 quantisation methods perform across different model architectures?
2. **RQ2:** Can custom compression techniques (activation quantisation and token pruning) improve upon standard quantisation approaches?
3. **RQ3:** What are the practical limitations of post training compression without retraining?
4. **RQ4:** How do theoretical compression gains translate to real world hardware performance?
5. **RQ5:** How do model scale and training quality affect quantisation tolerance?

To build up the knowledge to understand the solution, a literature review must be done. The next chapter goes into this in full detail.

Chapter 1

Literature Review

This chapter provides an overview of foundational concepts and recent progress in large scale AI models, the challenges posed by deploying these models on resource limited edge devices, and the main model compression techniques proposed to address these challenges. Modern AI systems, including large language models (LLMs) and vision transformers, rely on deep neural networks trained on massive datasets to achieve state of the art performance in language, vision, and multimodal tasks. However, their size and computational demands make them difficult to run efficiently on devices such as smartphones, drones, and IoT sensors.

Model compression methods including pruning, quantisation, low rank factorisation, knowledge distillation, and token reduction seek to reduce the memory footprint and computation requirements of these models whilst preserving accuracy. Each technique presents unique strengths and limitations, with significant questions remaining about their practical effectiveness when applied to modern transformer architectures. Recent research has proposed combining multiple compression techniques in single post training passes, claiming synergistic benefits. However, the gap between theoretical claims and practical deployment effectiveness remains poorly understood.

This review critically examines both established and recently proposed compression approaches to establish a foundation for empirical evaluation. By surveying standard quantisation methods (INT4/INT8), token pruning strategies (TRIM), and custom activation quantisation techniques, this chapter identifies key claims made in the literature regarding compression effectiveness. These claims are then subjected to rigorous empirical testing in subsequent chapters to determine which approaches are production ready for edge deployment and which remain research problems requiring further development.

The goal of this literature review is not to assume that combining compression techniques will succeed, but rather to understand the theoretical foundations and reported results sufficiently to design fair empirical comparisons. By examining both the promises and limitations reported in prior work, this review motivates the comprehensive evaluation methodology presented in Chapter 3, which tests whether theoretically motivated custom compression approaches deliver practical benefits over mature standard methods.

1.1 Basics of neural networks

At the core of large scale AI models lies artificial neural networks (ANNs), computational structures inspired by the human brain. An Artificial Neural Network (ANN) or DNN is a computational model composed of layers of nodes (or neurons), where each node performs these mathematical function and passes its output to the next layer. To be specific, an ANN is composed of an input layer with N nodes, corresponding the number of input features (e.g 10 numbers). One or more hidden layers and an output layer with M nodes similarly corresponding to the number of output features.

This thesis will focus solely on inference, that is, executing or "running" a pre trained model. During inference with an artificial neural network (ANN), a series of mathematical operations are performed layer by layer. Starting from the input layer, a vector of input values is provided. For each subsequent layer, the output is computed by first taking the activation vector from the previous layer (which, for the first hidden layer, is the input layer). Each neuron in the current layer receives this input vector and performs a weighted sum of its elements, each value is multiplied by a corresponding weight, the results are summed, and a bias term is added. This intermediate result, known as the pre activation, is then passed through a non linear activation function (such as ReLU or sigmoid) to produce the neuron's final output.

Each neuron in an ANN must store a bias term and a set of weights, and perform these computations during inference. While this process is computationally manageable for small networks, the resource requirements grow rapidly with model size. As networks scale, for example, in the case of large language models like GPT 4, the number of neurons, layers, and parameters becomes so large that inference requires significant computational power and memory. This has led to the deployment of such models on high performance computing infrastructure and large scale server clusters.

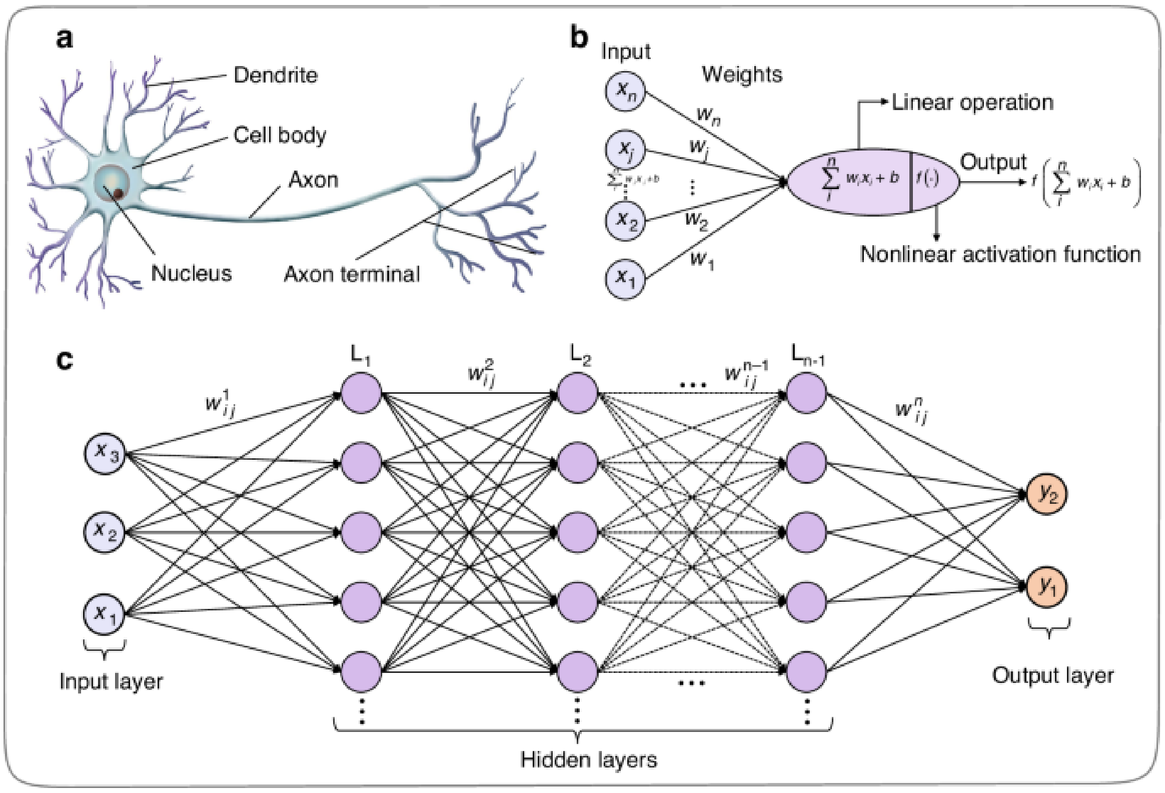

Figure 1.1: Example of a small ANN

1.2 Network Architectures

1.2.1 Large Language Models (LLMs)

LLMs are DNNs trained on vast amounts of text data to understand and generate human like language. They leverage the transformer architecture, which utilises self attention mechanisms to process input data in parallel, enabling efficient handling of long range dependencies in text.

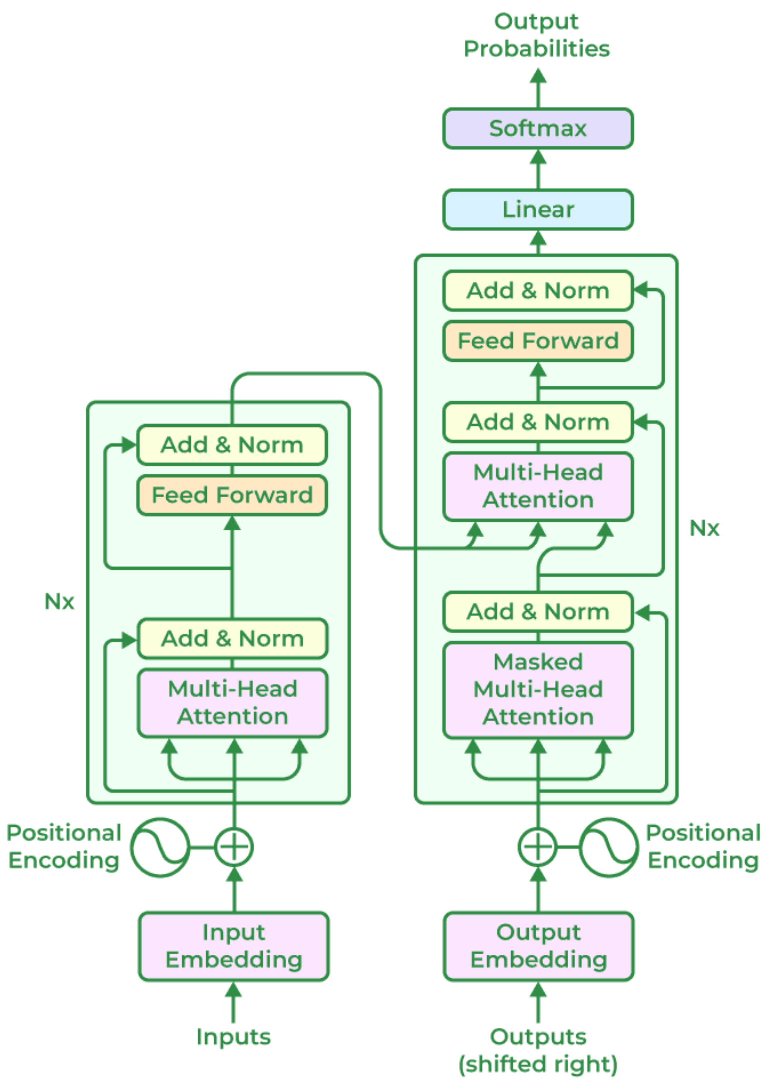

Figure 1.2: Example of a transformer architecture LLM

Before textual data is fed into an LLM as illustrated by the green box in Figure 1.2, it is first tokenised. Tokenisation involves breaking down the input into smaller units, such as words or subwords, each of which is mapped to a unique index in the model's vocabulary. These indices are then converted into continuous valued vectors via an embedding layer. The resulting input is a sequence of dense vectors, typically with values ranging between 1 and 1, which capture both the semantic and syntactic properties of the original text.

Following this, each vector V_i will have a value P_i added based on its position within the given input, such that $V_i = V_i + P_i$ because the model has no inherent way to know the original positions of the tokenised input.

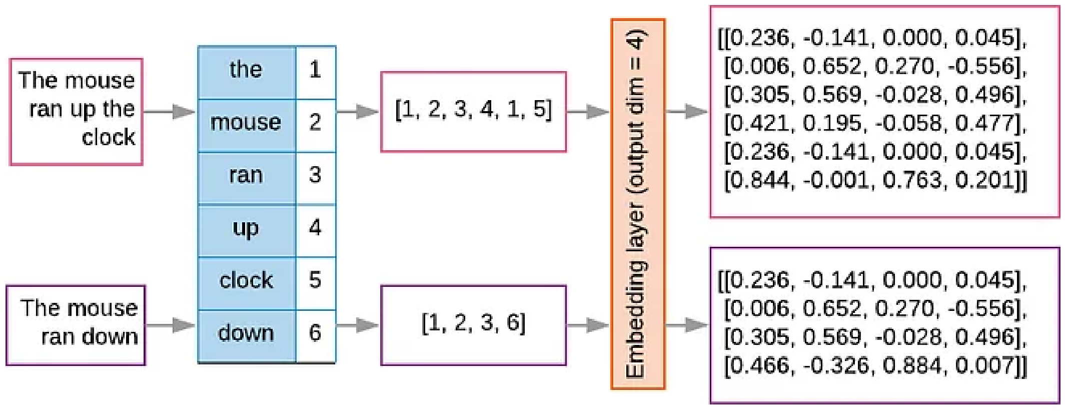

Figure 1.3: Example of Embeddings Matrix

Most large scale models adopt this transformer architecture, using encoders, decoders or both. Although different, in principle they act similar. In decoder only models such as ChatGPT, only the decoder component of the original Transformer architecture [7] is used.

Each Transformer block performs a series of mathematical computations to update and transform the input. These include:

- Attention mechanisms that help the model focus on relevant parts of the input.
- Feedforward operations that refine the information for each word.
- Layer normalisation and residual connections that improve stability and learning.

Throughout these layers, the model uses matrix multiplications and activation functions powered by learned weight matrices and biases. These learned values are critical they allow the model to encode knowledge, patterns, and relationships between words and phrases.

Once the input has passed through all layers, the final output is passed into a simple mathematical function that produces a probability distribution over possible next words. The word with the highest score is typically chosen, and the process continues for generating responses.

In essence, while the system is deeply mathematical under the hood, the key idea is that it uses structured computations and vast amounts of learned parameters to model and generate human like language.

Notable LLMs include:

- **BERT (Bidirectional Encoder Representations from Transformers)**: Employs a bidirectional approach to understand the context of words in a sentence, leading to significant improvements in various NLP tasks.
- **GPT (Generative Pre trained Transformer)**: Autoregressive language models that generate coherent and contextually relevant text, demonstrating remarkable capabilities in text generation and understanding.
- **T5 (Text To Text Transfer Transformer)**: Frames all NLP tasks as a text to text problem, enabling a unified approach to various tasks.

1.2.2 Vision Models (VM)

Vision models, and subsequently hybrid visual language models, work similarly to large language models (LLMs). To demonstrate, the architecture of a vision only model will be explored.

Like text, an input image is represented as a 3D tensor corresponding to its height, width, and RGB channels. For example, an input image of shape $224 \times 224 \times 3$ (height $\times$ width $\times$ RGB channels).

The image is split into non overlapping patches, typically of size 16×16. For a 224×224 image, this results in $\frac{224}{16} \times \frac{224}{16} = 14 \times 14 = 196$ patches.

Each patch is flattened into a vector (e.g $16 \times 16 \times 3 = 768$ values) and projected into a lower dimensional embedding space using a learned linear transformation, producing an embedding matrix. This learned linear function produces a 2D matrix like the text token embeddings. Like before, learned positional embeddings $E_{pos} \in \mathbb{R}^{196 \times d}$ are added to retain spatial information. These embeddings are then fed into transformer blocks and processed similarly to LLMs to generate outputs.

In contrast, convolutional neural networks (CNNs) process images by applying convolutional filters that scan across the image to capture local spatial patterns. CNNs build hierarchical features layer by layer by combining nearby pixel information. ViTs, on the other hand, treat image patches as tokens and use self attention mechanisms to capture global relationships between patches from the start, allowing them to model long range dependencies more effectively.

This fundamental difference in architecture leads to distinct strengths, CNNs excel at learning local features with inductive biases like translation invariance, while ViTs leverage the flexibility of attention to learn global context without such biases [9, 10].

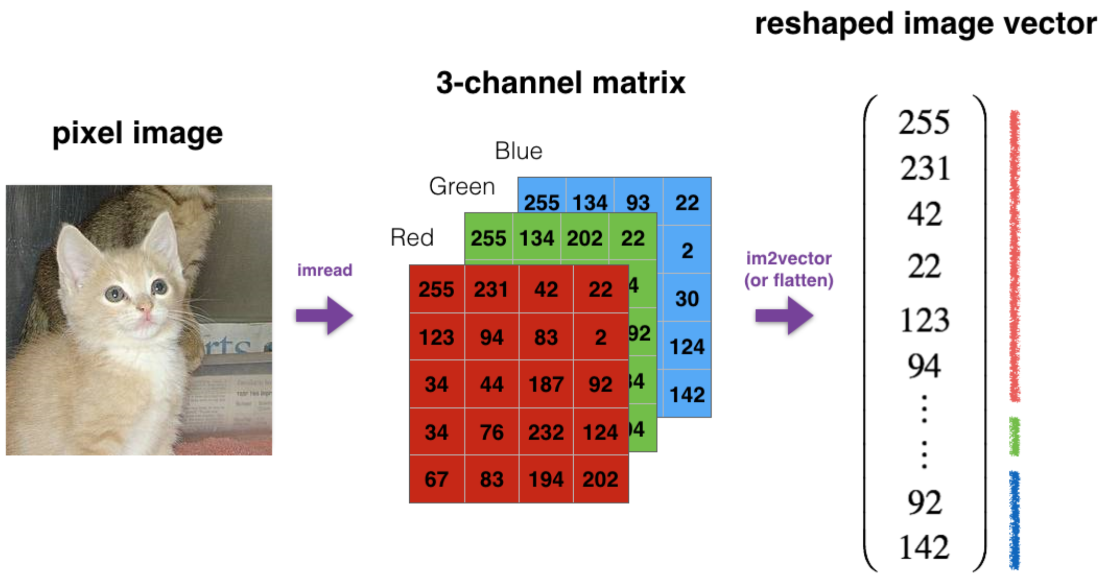

Figure 1.4: Image to vector

1.2.3 Multimodal Models

Multimodal models integrate and process multiple data modalities such as text, images, audio, and video to achieve a richer, more holistic understanding of complex inputs. In particular, state of the art systems like Google's Gemma 3 [11] combine visual and language processing pipelines into a single decoder only Transformer, yielding a unified textual output.

Multimodal models such as *Gemma 3* adopt a ”vision and language” architecture that extends the traditional transformer pipeline. The process begins by independently encoding each input modality, text is tokenised and mapped into a sequence of embedding vectors using a standard text embedding matrix, while images are processed by a vision encoder which is normally a linear function that transforms them into a sequence of image patch embeddings.

To enable joint reasoning across modalities, the resulting text and image embeddings, each a sequence of vectors in the same embedding dimension are concatenated into a unified input sequence. This two dimensional tensor has shape (X, d), where $X = X_{\text{text}} + X_{\text{image}}$, and d is the shared embedding size. Positional encodings are added to retain order information across the combined sequence. Since both modalities are now represented as compatible vector sequences, the transformer can process the fused input as it would any other sequence, enabling cross modal interactions via self attention.

1.3 Need for Model Optimisation

Transformer based architectures, such as GPT 4, have demonstrated exceptional performance across natural language, vision, and multimodal tasks. These models often contain billions of parameters and require high precision arithmetic, resulting in memory footprints of tens or hundreds of gigabytes and inference latencies measured in seconds even on modern GPUs. In contrast, embodied edge devices such as smartphones, drones, wearable cameras, and IoT sensors are constrained by limited RAM, compute cores and power draw. As a result, directly deploying large scale AI models on these platforms is infeasible. This tension between model capacity and device capability has motivated research into methods that can bridge the gap and enable on device intelligence without relying on cloud connectivity.

1.3.1 Real Time Robotic Interaction with Video Input

Advanced multimodal models, such as Gemma 3, have been developed to address the challenges of deploying AI on resource constrained devices. Gemma 3 is designed to run efficiently on a single GPU, delivering high performance in processing multimodal inputs, including text and high resolution images, with a context window of up to 128K tokens [12]. Notably, Gemma 3 can process 4K video at 30 frames per second, achieving 2.5 times the real time speed of GPT 4V's vision module [12].

In robotic systems, the ability to process a continuous stream of video frames alongside textual instructions or prompts is crucial for real time perception and interaction with dynamic environments. Gemma 3's architecture enables robots to interpret complex visual scenes, make context aware decisions, and respond appropriately. This capability is essential in applications such as autonomous navigation, human robot collaboration, and intelligent manipulation in unstructured environments.

To facilitate deployment on edge devices, Gemma 3 incorporates several optimisation techniques:

- **Model quantisation**
- **Knowledge Distillation**
- **Efficient Architecture Design**

Deploying the 27B parameter version of Gemma 3 for real time video processing remains challenging on consumer grade GPUs. While the model's VRAM requirements have been reduced to approximately 14.1 GB, this figure pertains solely to the model's weights. In practice, additional memory is required for components like the key value (KV) cache, especially when handling high resolution video streams and large context windows. Consequently, achieving real time inference speeds (e.g. 30 frames per second) necessitates substantial computational throughput, which can lead to latency issues on consumer GPUs such as the NVIDIA RTX 4090 or 5090.

Furthermore, deploying the 27B model on smaller edge devices, such as smartphones, drones, or IoT sensors, is currently impractical due to their limited computational and memory resources. These devices typically lack the necessary VRAM and processing power to handle the model's requirements, even in its quantised form [13].

For applications requiring real time video processing on resource constrained hardware, developers might consider using smaller variants of Gemma 3, such as the 12B or 4B models, which have lower computational and memory requirements. Alternatively, deploying the 27B model on more powerful hardware, like the NVIDIA A100 or H100 GPUs, can better meet the demands of real time processing [14].

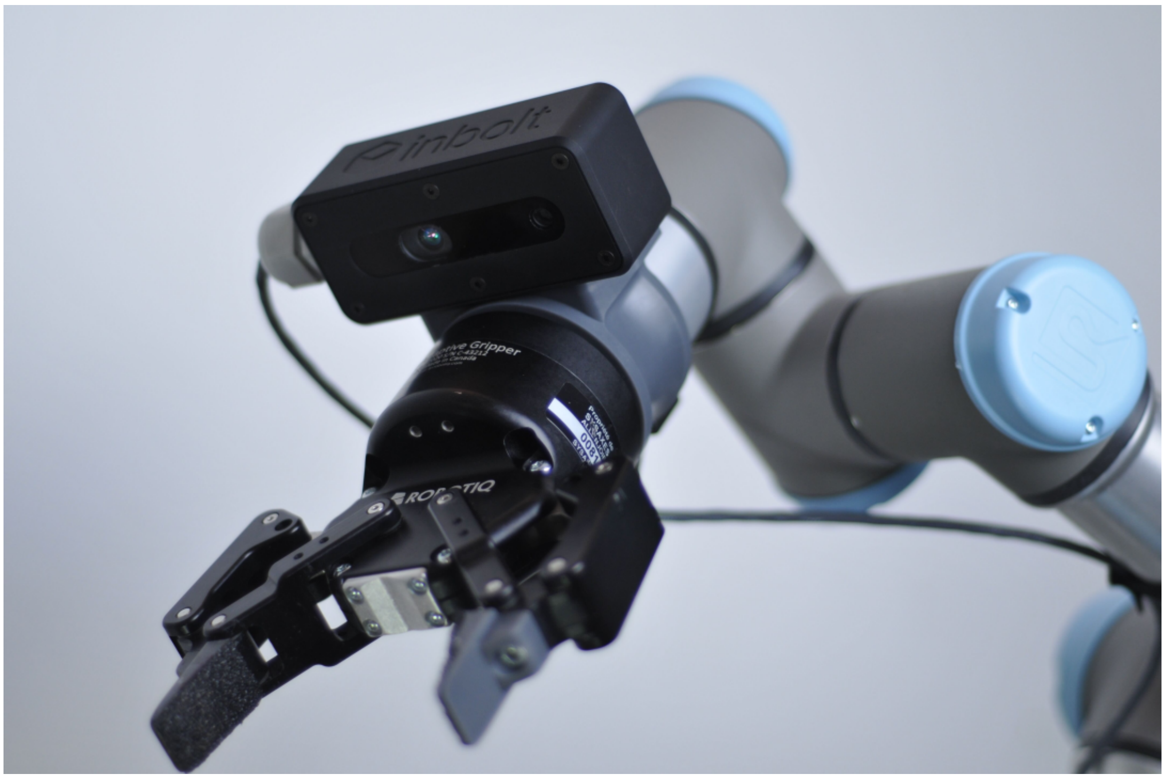

Figure 1.5: Example of Robotic Application

1.3.2 Implications for On Device Intelligence

The deployment of optimised multimodal models like Gemma 3 on edge devices has significant implications for the development of intelligent robotic systems. By enabling real time processing of multimodal inputs without reliance on cloud connectivity, robots can achieve greater autonomy, responsiveness, and privacy. This advancement opens up new possibilities for deploying AI in environments where network access is limited or latency is critical, such as in disaster response, remote exploration, and personal assistive technologies.

1.4 Model Compression Techniques

1.4.1 Pruning

Pruning eliminates redundant parameters in a neural network to reduce model size and computational cost, making it one of the earliest and most widely studied model compression techniques. The core idea is to identify weights, neurons, or entire structures that contribute minimally to the model's predictions and remove them without significantly degrading performance. Unstructured pruning, individual low magnitude weights are zeroed out, resulting in sparse matrices. Structured pruning, on the other hand, removes entire filters or channels, yielding thinner layers. Han et al. demonstrated that iterative pruning and retraining can reduce parameters by 9 to 13× without accuracy loss [15]. Structured pruning methods produce dense, slimmer models that are more amenable to acceleration on hardware. However, unstructured pruning often leads to irregular sparsity patterns that require specialised libraries for speed up, offering limited benefits on typical hardware. Structured pruning, while more hardware friendly, may reduce model capacity and often necessitates costly retraining. Both approaches can suffer accuracy loss if over pruned, and iterative pruning pipelines add training complexity. For example, within the LLM transformer block explained in 1.2.1, it refers to multiple weight matrices to be applied to the token embedding matrix. Values within the weight matrix could be zeroed out or entire rows or layers could be removed or pruned to reduce computation.

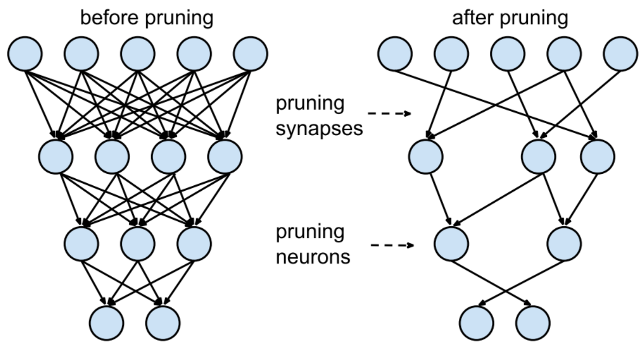

Figure 1.6: Pruning Example

1.4.2 Quantisation

Quantisation reduces the numerical precision of weights and activations in neural networks. Post training quantisation (PTQ) maps a pre trained floating point model to low bit width (e.g., int8) using calibration, where as quantisation aware training (QAT) simulates low precision during fine tuning to preserve accuracy. Uniform quantisation uses fixed step sizes, which is hardware friendly, while non uniform schemes allow adaptive step sizes for different ranges or groups of weights. Jacob et al. proposed an integer only quantisation scheme that achieves 8 bit inference on mobile CPUs with minimal accuracy drop [16]. More recent work, such as Learned Step Size quantisation (LSQ), trains quantisation scales end to end [17]. Activation aware quantisation (AWQ) has been proposed specifically for large language models (LLMs) it protects a small fraction of salient weights by scaling them based on activation statistics, enabling 3-4× speedups of 70B models on edge GPUs [18]. Similarly, AgileQuant automatically finds mixed 4 bit quantisation configurations for LLMs, balancing task accuracy and device latency.

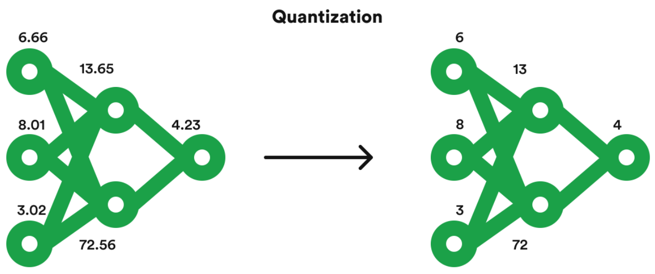

Figure 1.7: Example of Quantisation

1.4.3 Low Rank Factorisation

Low rank factorisation exploits linear algebra to compress weight matrices. For convolutional layers, filters can be reshaped into matrices and decomposed using techniques like singular value decomposition (SVD) or tensor decomposition. Denton et al. showed that low rank SVD of convolutional kernels can yield significant speedup with small accuracy loss. Lebedev et al. used CP decomposition to break a 4D convolutional kernel into a sequence of smaller convolutions. Generally, a convolutional layer with a tensor of shape (C_{in}, C_{out}, k, k) can be factored into multiple smaller convolutions, dramatically reducing computation if the chosen rank is small. However, low rank techniques assume weight redundancy and may underperform if layers are already compact. They often introduce extra layers, adding overhead, and typically require fine tuning after decomposition to recover accuracy. On edge hardware, the irregular multi step kernels may not match optimal BLAS or convolution layouts, so realised gains can be lower than theoretical.

1.4.4 Knowledge Distillation

Knowledge distillation trains a compact student model to mimic a larger teacher network. Hinton et al. introduced this approach by having the student match the teacher's softened output distribution, capturing the relative probabilities among classes. Variants include matching intermediate representations, such as FitNets. Distillation can yield smaller models that approach teacher accuracy and is especially useful for deployment on devices. However, it requires a well trained teacher and additional training time. The quality of the student depends heavily on teacher guidance and matching loss. In some cases, distilled models still lag behind the teacher, especially on out of distribution data. Additionally, distillation does not reduce architecture induced latency directly the student architecture must still fit the hardware.

1.4.5 Transfer Learning

Transfer learning reuses a pretrained model or parts of it for a new task, often with fine tuning. In the context of compression, transfer learning can reduce training cost by starting from a large model baseline and adapting it, instead of training a smaller model from scratch. For example, one may fine tune only the final layers or use feature extractors from a high capacity network. This can yield a smaller effective model when unnecessary parts of a transferred model are pruned. Pan and Yang discuss how knowledge can be transferred across tasks and domains to improve efficiency. However, transfer learning itself is not a pure compression technique it speeds up development and may allow using smaller models without loss of accuracy. It often assumes that a large pretrained model is available. On edge devices, running even a transferred model can still be resource intensive unless further compressed.

1.4.6 Token Pruning

Modern transformers process input as a set of tokens, and many tokens may carry redundant information. Token pruning dynamically drops less informative tokens during inference to reduce computation. In vision transformers (ViTs) and multimodal LLMs, image patch tokens can be pruned in NLP, history or context tokens can be pruned. Methods like FastV, FitPrune, and VisToG target vision language models. FastV learns to prune tokens dynamically during inference. FitPrune is a training free approach for multi modal LLMs it uses attention statistics to choose an optimal pruning scheme per budget, dropping redundant visual patches while preserving attention distributions. VisToG introduces a grouping layer before the LLM it groups semantically similar image patches into higher level semantic tokens using the pre trained vision encoder, then prunes redundant tokens. These methods achieve large FLOP and latency reductions on tasks like VQA and image captioning. For long text inputs to LLMs, LazyLLM dynamically prunes context tokens during both prompt prefilling and generation, selecting only tokens important for predicting the next token. TRIM is another token reduction pipeline it allows an LLM to emit a shorter distilled output by skipping function words, then uses a smaller model to reconstruct the full text. While TRIM targets output efficiency rather than input pruning, it similarly exploits language redundancy to save compute. However, token pruning often involves heuristic or learned importance scoring, which may misidentify critical tokens. Many vision token methods rely on attention scores, which can be unstable, or on additional modules that add overhead. Preserving performance can require conservatively keeping many tokens. In NLP, pruning context tokens risks losing essential information for generation. Furthermore, on real hardware, reducing token count does not always translate to proportional speedup because most transformer layers still compute several attention heads on the retained tokens.

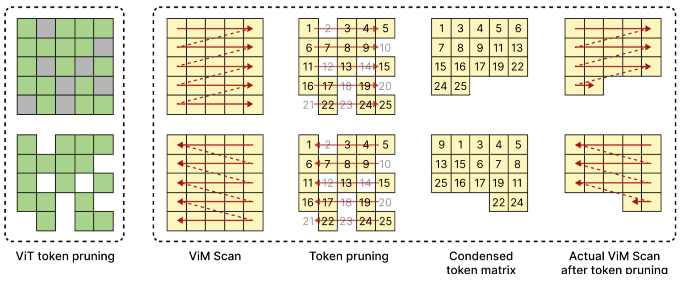

Figure 1.8: Example of Token Pruning

1.5 Limitations and Deployment Challenges

Despite advances in model compression, deploying compressed models efficiently on real world edge devices remains a significant challenge. Each technique while effective in isolation exhibits limitations when applied to practical, resource constrained scenarios, especially without careful tuning or hardware specific adaptations.

- *Pruning* reduces model size by removing weights or channels, but unstructured pruning often produces sparse matrices that require custom kernels to benefit from any speedup. Structured pruning is more hardware friendly but can reduce model capacity and still needs costly retraining to recover accuracy.
- *Quantisation* lowers numerical precision to shrink model memory footprint and improve inference speed. However, very low bit quantisation (e.g., 4 bit) often leads to accuracy degradation unless carefully calibrated.quantisation aware training (QAT) can mitigate this but adds training cost. Additionally, many commodity devices lack full support for sub 8 bit arithmetic, making deployment non trivial.

- *Low Rank Factorisation* decomposes weight matrices or tensors into smaller components, reducing compute. However, it assumes redundancy in layers, introduces additional operations, and often requires fine tuning. Gains may be theoretical unless the decomposition is aligned with efficient hardware primitives.
- *Knowledge Distillation* compresses models by transferring knowledge from a larger teacher to a smaller student. While often effective, it requires full training cycles, access to teacher logits, and may not reduce architectural complexity. Student models still need further compression for edge deployment.
- *Transfer Learning* enables model reuse and faster training but does not reduce model size unless combined with pruning or quantisation. It can also suffer from negative transfer if source and target tasks diverge.
- *Token Pruning* reduces the number of tokens processed, especially in vision and language transformers. Although this significantly cuts computation (e.g., by pruning image patches or attention contexts), effectiveness heavily depends on token importance scoring, which is often heuristic. Over pruning can remove critical information, while under pruning limits gains. Moreover, on real hardware, reducing token counts does not always proportionally reduce latency due to persistent computation across retained tokens in attention layers.

Beyond individual technique limitations, combining multiple compression methods such as pruning + quantisation + distillation is a non trivial engineering challenge. Interactions between methods are often not well understood or synergistic: for instance, pruning can change quantisation sensitivity, and token pruning can interfere with attention mechanisms. Many methods also introduce irregular memory access patterns, making them difficult to optimise using standard linear algebra kernels (e.g., GEMM or FlashAttention).

Finally, theoretical FLOP or parameter reductions often fail to translate into real world speedups unless compression pipelines are co designed with hardware considerations in mind, such as memory bandwidth, parallelism, SIMD alignment, and supported data types. This makes the deployment of highly compressed models on constrained edge devices a multifaceted optimisation problem that extends beyond accuracy and parameter count.

1.6 Recent Advances

To overcome the practical limitations of individual compression methods, recent research has shifted towards hybrid, adaptive, and hardware aware strategies. These integrate multiple techniques such as quantisation, pruning, and token reduction into cohesive pipelines that significantly reduce computational load while preserving model performance. Such approaches are especially valuable for deployment on edge devices, where resources are limited but responsiveness is crucial.

- **Activation Aware Weight quantisation (AWQ)** [18] improves standard quantisation by incorporating knowledge of how important each weight is based on the activation patterns it influences. Unlike uniform quantisation, which applies the same precision to all weights, AWQ identifies a small subset of highly sensitive weights those that, when changed, significantly alter the model's output. These are kept at higher precision or scaled differently, while the rest are aggressively quantised. This approach enables 4 bit quantisation of even 70B parameter LLMs with minimal accuracy loss, by protecting the information bottlenecks in the network. AWQ is also hardware efficient, allowing faster inference on edge GPUs without retraining.

- **AgileQuant** [19] introduces a fully post training quantisation system that requires no access to training data or gradients. It works by profiling the model layer by layer to understand the impact of different quantisation settings (e.g., bit widths) on accuracy and latency. Then, using this profiling data, it automatically assigns different bit widths to different layers potentially using 8 bit for sensitive layers and 4 bit or lower for more redundant ones. This mixed precision design allows for fine tuned trade offs between speed and accuracy, optimised for the deployment hardware's characteristics (e.g., GPU vs CPU), all without retraining the model.
- **Visual Token Grouping (VisToG)** [20] addresses the inefficiency of processing raw visual data in multimodal transformers. In typical vision language models, an image is split into many small patches, each treated as a separate token. However, many of these patches contain redundant or repetitive content. VisToG clusters visually and semantically similar patches together and replaces them with a single representative token before feeding them into the model. This substantially reduces the token count often by over 25% with almost no drop in model performance. Since VisToG operates before the model and doesn't require model changes or training, it's ideal for lightweight, real time systems.
- **Token Reduction via Importance Modelling (TRIM)** [21] goes further by pruning tokens after training through semantic scoring. Using pretrained CLIP embeddings, TRIM evaluates how important each visual token is in contributing to the final prediction. Unimportant tokens are removed, and the model generates output using only the essential ones. If needed, a small auxiliary model reconstructs information from the pruned areas to maintain output quality. This enables significant reductions (up to 79%) in the number of tokens without harming performance, making inference much faster. TRIM is especially powerful because it compresses the *output space*, not just the input or model weights an under explored area in model efficiency.
- **FastV** [22] introduces a dynamic mechanism to prune visual tokens *during inference*. Rather than relying on precomputed importance scores, FastV uses attention maps generated by the model in real time to identify which tokens matter for the current input. Low attention tokens are removed on the fly, meaning that each input image is processed with a custom tailored token set. This reduces computational workload (by up to 45% in FLOPs) without the need for retraining or modifying the model architecture. FastV exemplifies runtime adaptivity making it highly suitable for time sensitive, resource constrained deployment scenarios.

These methods represent a broader movement towards practical, deployment ready model compression. By combining quantisation sensitive to network behaviour, semantic aware token pruning, and runtime adaptability, they make large scale models increasingly usable on edge devices without the need for extensive retraining or manual tuning.

1.7 Summary and Research Gaps

This survey highlights that whilst individual compression techniques such as pruning, quantisation, and knowledge distillation have been extensively studied, their practical effectiveness on real world embedded or edge systems remains unclear. These systems require not just compact models but also strict adherence to latency and memory budgets, typically without retraining due to deployment constraints. The gap between theoretical compression claims and actual deployment performance is poorly documented in existing literature.

Recent developments such as AWQ, AgileQuant, and VisToG represent important steps towards adaptive, hardware efficient compression. However, several critical questions remain unanswered:

1. Do these methods outperform mature standard quantisation approaches such as INT4 weight only quantisation?
2. Can combining multiple compression techniques in a single post training pass provide practical benefits, or do interactions between methods create unforeseen problems?
3. How do theoretical FLOP reductions translate to actual latency improvements on real GPU hardware?

Crucially, most prior research reports results on single favourable benchmarks without rigorous comparison against strong baselines or documentation of failure cases. There is a lack of comprehensive empirical evaluation that tests whether theoretically motivated custom compression approaches deliver measurable advantages over established methods in practice. Claims about token pruning effectiveness, activation quantisation benefits, and synergistic combinations of techniques require validation through systematic experimentation across diverse model architectures and realistic deployment scenarios.

This thesis addresses these gaps through rigorous empirical evaluation that:

- Tests standard quantisation methods (INT4/INT8 using BitsAndBytes) as strong baselines,
- Implements and evaluates custom compression techniques including attention based token pruning, outlier aware activation quantisation, and layer wise dynamic precision,
- Measures real world performance metrics (accuracy, latency, memory) on actual GPU hardware rather than theoretical FLOP counts,
- Compares across five model architectures ranging from 0.5B to 7B parameters to assess generalisation,
- Documents both successful and failed approaches to provide honest assessment of deployment readiness.

By subjecting theoretical compression claims to rigorous empirical testing, this research aims to determine which techniques are production ready for edge deployment and which remain research problems requiring training time integration or specialised hardware. Rather than assuming that combining compression methods will succeed, this thesis investigates whether such combinations provide practical advantages over mature standard approaches, thereby bridging the divide between academic claims and deployment reality.

The following chapter introduces the comprehensive methodology used to conduct this empirical evaluation, including model selection, benchmark datasets, implementation details, and evaluation protocols designed to ensure fair comparison between standard and custom compression approaches.

Chapter 2

Methodology

This chapter presents the experimental methodology used to evaluate post training compression techniques for AI models. The research employs a systematic approach to assess the effectiveness of various quantisation methods on model performance, inference speed, and memory consumption [8]. The methodology encompasses model selection, optimisation technique implementation, dataset preparation, experimental design, and evaluation metrics.

2.1 Research Approach

This study adopts an empirical, iterative experimental approach to evaluate model compression techniques. Rather than proposing entirely novel algorithms, the research focuses on rigorously evaluating and combining existing state of the art methods alongside novel adaptations to identify practical compression strategies for edge deployment. The methodology prioritises reproducibility, systematic comparison, and transparent reporting of both successful and unsuccessful approaches.

The experimental design follows a comparative framework where each model is evaluated under multiple compression configurations, including an uncompressed baseline. This enables direct quantification of the trade offs between compression ratio, inference speed, and task accuracy.

2.1.1 Novel Contributions

This research attempted to implement three novel compression techniques extending beyond standard weight quantisation:

1. **Attention Based Token Reduction (TRIM)**: A token pruning method analysing attention patterns to identify and remove low importance tokens from input sequences, building upon recent work in multimodal token reduction [21, 22].

2. **Outlier Aware Activation Quantisation**: Quantising intermediate activations during inference by detecting and handling outlier activations separately, addressing the challenge that large language model activations contain systematic outliers that destroy naive quantisation [18].

3. **Layer Wise Dynamic Quantisation**: An adaptive precision scheme applying different quantisation bit widths to different layers based on sensitivity profiles, extending concepts from learned step size quantisation [17].

These techniques targeted complementary compression dimensions: TRIM reduces input computational cost, activation quantisation reduces memory bandwidth during inference, and layer wise quantisation applies appropriate precision per layer. Combined with standard INT4 weight quantisation (BitsAndBytes), they aimed to address weights, activations, and inputs simultaneously in a pure post training setting without calibration data or fine tuning.

However, as detailed in subsequent sections, practical implementation challenges prevented these novel techniques from achieving their theoretical benefits, revealing fundamental limitations in zero shot post training compression.

2.2 Model Selection

Models were selected based on architectural diversity, parameter range, open source availability, and relevance to edge deployment, encompassing both text only and multimodal models to evaluate compression techniques across different types.

Qwen2 0.5B Instruct
A compact instruction tuned language model developed by Alibaba with 494 million parameters. This model serves as a representative small scale language model suitable for resource constrained environments. Its small size makes it ideal for evaluating whether compression benefits persist even at lower parameter counts. The model uses 24 transformer layers [7] with a hidden dimension suitable for edge devices.

TinyLlama 1.1B
A compact 1.1 billion parameter model based on the Llama architecture, designed specifically for deployment on resource constrained devices. TinyLlama represents an interesting middle ground between ultra compact models like Qwen2 0.5B and larger models like Phi 2. Despite its small size, the model demonstrates capabilities across various language understanding tasks. TinyLlama's inclusion enables assessment of compression techniques on models specifically optimised for efficiency, providing insights into whether further compression is beneficial for already compact architectures.

Phi 2
Microsoft's 2.7 billion parameter model designed for efficient reasoning and code generation. Phi 2 represents the mid range model category, offering substantially more capacity than the sub billion parameter models whilst remaining deployable on consumer hardware. The model architecture comprises 32 layers and has demonstrated strong performance on reasoning benchmarks despite its compact size. Phi 2 serves as a test case for compression methods on models that balance capability and efficiency.

Mistral 7B Instruct v0.2
A 7.3 billion parameter decoder only transformer model that has achieved competitive performance with much larger models. Mistral 7B employs grouped query attention and sliding window attention mechanisms that already incorporate efficiency optimisations [23]. This model represents the upper bound of model sizes typically considered for edge deployment and provides insight into how compression techniques scale with model capacity. The inclusion of Mistral 7B enables assessment of whether compression methods that work for smaller models remain effective at larger scales.

LLaVA 1.5 7B
A vision language model combining a CLIP vision encoder with a Vicuna language model [24]. LLaVA processes images by encoding them into patch embeddings [9] that are projected into the language model's token space. This architecture represents the predominant approach to multimodal model design and enables evaluation of compression techniques on models that process both visual and textual information. The model comprises approximately 7 billion parameters across both the vision and language components.

Table 2.1: Characteristics of evaluated models

Model	**Parameters**	**Architecture**	**Layers**	**Min VRAM (FP16)**	**Modality**
Qwen2 0.5B	494M	Decoder only	24	2GB	Text
TinyLlama	1.1B	Decoder only	22	3GB	Text
Phi 2	2.7B	Decoder only	32	6GB	Text
Mistral 7B	7.3B	Decoder only	32	16GB	Text
LLaVA 1.5 7B	~7B	Vision Language	32	16GB	Multimodal

2.3 Base Techniques

2.3.1 FP16 Baseline

All models are first evaluated in half precision floating point (FP16) format, serving as the reference baseline for comparison. FP16 provides a balance between numerical precision and memory efficiency, and represents the standard deployment format for many production systems. This baseline uses PyTorch's native FP16 support without additional quantisation.

2.3.2 INT8 Quantisation

8 bit integer quantisation serves as an intermediate compression point between FP16 and more aggressive quantisation methods. INT8 quantisation uses the BitsAndBytes library [25] with automatic quantisation of weight matrices. This method provides modest memory reduction whilst typically maintaining high accuracy, making it a conservative compression option for accuracy critical applications.

2.3.3 INT4 Quantisation (BitsAndBytes)

The primary quantisation approach employs 4 bit integer quantisation using the BitsAndBytes library [25]. This implementation uses NF4 (Normal Float 4 bit) quantisation, which represents weights using a 4 bit quantised normal distribution. The method includes double quantisation, where the quantisation constants themselves are quantised to further reduce memory usage [15].

2.4 Novel Techniques

This research attempted to implement three novel compression techniques extending beyond standard weight quantisation. It is important to acknowledge upfront that these techniques all failed to improve upon simple INT4 weight quantisation and are presented here as cautionary examples of what does not work in pure post training settings without calibration data or fine tuning.

2.4.1 Attention Based TRIM

An experimental token pruning implementation aimed to reduce input sequence length by selectively removing low importance tokens based on cumulative attention weights across transformer layers. The approach was inspired by recent work on token reduction in multimodal models.

The implementation preserved attention sink tokens (first four tokens), recent context (last 10 percent of tokens), and selectively retained middle tokens based on importance scores calculated from attention weight matrices. The technique targeted a 30 percent sequence length reduction, theoretically yielding proportional reductions in computational cost and memory usage for key value cache.

Despite theoretical soundness, this technique resulted in catastrophic accuracy degradation of 20 to 45 percentage points across all tested models. Critical failure mechanisms included:

1. Attention weight extraction overhead negating speed benefits due to additional forward passes,
2. Insufficient importance metrics that failed to capture all token contribution mechanisms beyond attention patterns,
3. Lack of calibration data leading to over aggressive pruning,
4. Causal attention complications in decoder only models where pruning disrupts attention masking, and
5. Severe task specific sensitivity in question answering contexts where removing any question words, option identifiers, or key phrases renders the task incomprehensible.

The failure highlighted that techniques effective in calibrated settings often fail when applied naively post training without representative validation data.

2.4.2 Outlier Aware Activation Quantisation

This technique attempted to quantise intermediate activations during inference by detecting and handling outlier activations separately from the main distribution. The motivation came from research showing that large language model activations contain systematic outliers orders of magnitude larger than typical values.

The implementation used a two tier strategy:

1. Outlier detection via z score thresholds (6 standard deviations),
2. Differential handling where non outlier activations were quantised to 8 bit integers whilst outliers were preserved in FP16, and

3. Dynamic adaptation with per token scaling factors.

Forward hooks intercepted activations at linear layers, applying quantisation logic before passing values to subsequent layers.

The technique introduced substantial latency overhead (1.5 to 2 times slower than baseline INT4) with negligible accuracy benefit (less than 2 percent improvement). Critical issues included:

1. Irregular memory access patterns preventing efficient GPU utilisation,
2. Forward hook overhead creating additional Python interpreter overhead, and
3. Constant quantisation dequantisation overhead between layers.

The failure demonstrated that optimisations beneficial in isolation can be counterproductive when system level effects (GPU pipeline efficiency, memory access patterns, framework overhead) are considered. Large models proved particularly sensitive to activation quantisation, with Mistral 7B and LLaVA 1.5 7B showing complete failure (0 to 40 percent accuracy), suggesting that complex model representations rely on high precision intermediate computations that cannot be quantised without proper calibration.

2.4.3 Layer Wise Dynamic Quantisation

This approach applied different quantisation bit widths to different layers based on hypothesised sensitivity profiles, with early layers (0 to 2) using 8 bit, middle layers using 4 bit, and late layers using 6 bit quantisation.

The implementation attached separate quantisers per layer category via forward hooks. However, this approach introduced substantial latency overhead without measurable accuracy benefit due to:

1. Per layer computational bottlenecks from multiple quantisation kernels,
2. Arbitrary sensitivity assumptions not validated empirically,
3. Lack of holistic optimisation across layers, and
4. Implementation complexity hindering debugging and refinement.

The technique reinforced the importance of empirical validation over theoretical assumptions. The strategy of assuming early layers are less sensitive proved incorrect for certain model architectures (particularly Phi 2), indicating that layer sensitivity is model specific and cannot be predetermined through heuristics.

2.4.4 Combined Novel Techniques

Combinations tested included INT4 with activation quantisation (suffered activation quantisation overhead), INT4 with TRIM (catastrophic accuracy degradation), and full combination of all techniques (combined worst aspects of each component). The unsuccessful combinations provided critical lessons:

1. Post training techniques requiring calibration are challenging without representative data,
2. System level effects matter more than isolated theoretical benefits,
3. Simplicity often wins due to superior implementation quality and robustness,
4. Attention weights alone are insufficient for importance estimation, and
5. Rigorous empirical validation is essential before committing resources to complex implementations.

2.5 Datasets and Evaluation Tasks

The evaluation employs established benchmarks assessing general reasoning and specific knowledge domains. Dataset selection prioritises real world usage scenarios and enables comparison with prior work.

2.5.1 HellaSwag

HellaSwag (Harder Endings, Longer contexts, and Low shot Activities for Situations With Adversarial Generations) is a commonsense reasoning benchmark presenting a context and four possible continuations. The task requires selecting the most plausible continuation, testing model understanding of everyday scenarios. Fifty samples from the validation split were used per model configuration, with each sample comprising a context sentence and four possible endings.

2.5.2 ARC Easy

The AI2 Reasoning Challenge (ARC) Easy subset contains grade school science questions assessing factual knowledge and basic reasoning. Each question includes multiple choice options with one correct answer covering natural science, physics, chemistry, and biology. Fifty samples from the test split were used per model configuration, serving as a complementary measure to HellaSwag by assessing knowledge based reasoning.

2.5.3 Visual Question Answering

For multimodal evaluation of LLaVA, synthetic visual question answering tasks used programmatically generated images with known properties. The synthetic dataset comprised 50 samples featuring simple colour recognition tasks. Whilst synthetic evaluation has limitations in generalising to real world visual understanding, it provided controlled assessment of whether compression disrupted visual information processing. For production applications, evaluation on established benchmarks such as VQAv2 [26] would be recommended.

2.5.4 Dataset Processing

All datasets were loaded using the HuggingFace datasets library. Text was processed using each model's tokeniser with maximum sequence length of 512 tokens, truncation enabled, and padding applied for batch processing. For multiple choice questions, prompts were formatted to present the question followed by labelled options (A, B, C, D) with instructions to respond with only the letter corresponding to the correct answer.

The evaluation used 50 samples per dataset per model configuration due to computational constraints, representing a pragmatic balance between evaluation thoroughness and available computational budget. This sample size suffices to identify major performance trends and substantial accuracy differences (greater than 10 percentage points) whilst acknowledging that fine grained performance differences may not reach statistical significance.

2.6 Evaluation Metrics

The evaluation framework captures multiple dimensions of model performance, encompassing accuracy, computational efficiency, and resource utilisation [8].

2.6.1 Accuracy Metrics

Task accuracy is calculated as the percentage of correctly answered questions. For multiple choice tasks, a response is considered correct if the model's generated text contains the correct answer letter within the first few tokens of generation. Malformed outputs are counted as incorrect, as they would be unusable in practice.

2.6.2 Efficiency Metrics

Inference latency is measured as wall clock time from input tokenisation through final token generation, excluding warm up runs. Each sample's latency is recorded individually, with aggregate statistics computed across all samples. Measurements include input processing time, model forward passes, token generation, and output decoding. Measurements use Python's time module with CUDA synchronisation points for accurate GPU timing, with the first three inferences serving as warm up and excluded from reported metrics.

Token throughput is calculated as generated tokens divided by inference latency, measured in tokens per second. This metric provides normalised generation speed accounting for varying output lengths.

Peak GPU memory allocation is measured during inference using PyTorch's CUDA memory tracking, reported in gigabytes. Memory measurements include model weights, intermediate activations, key value cache, and temporary buffers. Memory is reset and measured for each model configuration to ensure accurate comparison.

2.6.3 Compression Metrics

Memory reduction is calculated as the percentage decrease in peak memory usage compared to the FP16 baseline:

$$\text{Memory Reduction} = \left(1\frac{\text{Memory}_{\text{compressed}}}{\text{Memory}_{\text{baseline}}}\right) \times 100\% \tag{2.1}$$

This quantifies the primary benefit of quantisation: enabling deployment on devices with limited VRAM [15].

Model size refers to on disk checkpoint file size in gigabytes. For quantised models using BitsAndBytes, this reflects quantised weight size.

2.6.4 Relative Performance Metrics

Accuracy retention indicates the percentage of baseline accuracy preserved after compression:

$$\text{Accuracy Retention} = \frac{\text{Accuracy}_{\text{compressed}}}{\text{Accuracy}_{\text{baseline}}} \times 100\% \tag{2.2}$$

Latency overhead quantifies the speed penalty (positive values) or improvement (negative values) introduced by compression:

$$\text{Latency Overhead} = \frac{\text{Latency}_{\text{compressed}}\text{Latency}_{\text{baseline}}}{\text{Latency}_{\text{baseline}}} \times 100\% \tag{2.3}$$

2.7 Experimental Setup

2.7.1 Hardware Configuration

Experiments were conducted on two GPU platforms:

NVIDIA L4 GPU: 23.8 GB GDDR6, Ada Lovelace architecture, compute capability 8.9, used for initial technique development and debugging.

NVIDIA A100 SXM4 40GB GPU: 42.5 GB HBM2e, Ampere architecture, compute capability 8.0, used for full evaluation and comprehensive testing. Both platforms were accessed through Google Colab Pro.

2.7.2 Software Environment

Core frameworks included Ubuntu 24.04 LTS, Python 3.10, CUDA 12.1, PyTorch 2.1+, Transformers library 4.45.0+, Accelerate 0.25.0+, and BitsAndBytes 0.41.0+. Supporting libraries included Datasets 2.14.0, NumPy, Pandas, Matplotlib, Seaborn, and tqdm.

Environment configuration set PYTORCH_CUDA_ALLOC_CONF to 'expandable_segments:True' for dynamic memory allocation, disabled TOKENIZERS_PARALLELISM, and fixed random seeds (PyTorch seed 42, NumPy seed 42). All library versions were pinned in requirements files for reproducibility.

2.7.3 Model Loading and Initialisation

Models were loaded using HuggingFace Transformers with specific configurations for each quantisation method. FP16 baseline models used torch.float16 dtype with automatic device mapping. INT8 models used

BitsAndBytes configuration with 8 bit loading and threshold 6.0. INT4 models used 4 bit loading with FP16 compute dtype, double quantisation enabled, and NF4 quantisation type. Complete loading code is provided in the Appendix.

Tokenisers were loaded with appropriate padding token configurations, using the model's end of sequence token as padding token when not explicitly defined.

2.7.4 Inference Configuration

Generation parameters included maximum new tokens of 10 (sufficient for multiple choice answer extraction), disabled sampling (greedy decoding), and no temperature or top k/top p settings. The short generation length reflects the evaluation task structure where only answer identification is required. Greedy decoding ensures deterministic outputs for reproducibility.

All inference used batch size 1 to simplify latency measurement, avoid padding overhead, reflect typical edge deployment scenarios, and enable accurate per sample latency tracking. Whilst batch processing can improve throughput, edge devices often process queries individually.

For each model configuration, three warm up inference passes were executed on the first sample before beginning timed evaluation, allowing CUDA kernel compilation, PyTorch autograd graph optimisation, memory allocator warm up, and elimination of first run outliers.

2.8 Experimental Procedure

For each model, the workflow executed: model loading in specified configuration, CUDA memory statistics reset and metric tracking initialisation, three warm up inferences, iteration through dataset processing inputs with tokenisation and formatting, recording start/end times with CUDA synchronisation, performing inference with specified generation parameters, extracting predictions, updating accuracy counters, tracking peak memory usage, storing latency measurements, computing aggregate metrics, and cleaning up with model unloading and CUDA cache clearing.

This workflow was repeated for each model configuration dataset combination. Total experimental runtime exceeded 40 hours across all configurations.

The experimental pipeline included robust error handling: model loading failures were caught and logged with experiment continuation to next configuration, generation errors were logged but counted as incorrect predictions, out of memory errors triggered automatic memory cleanup with configuration marked as incompatible, and inferences exceeding 30 seconds were terminated and logged as failures.

All metrics, errors, and intermediate results were logged to structured JSON files. Logging included timestamps, model and configuration details, hardware information, complete metric values, and error traces for any failures.

Reproducibility measures included: fixed random seeds (all generators initialised with seed 42), version control recording model IDs, dataset versions, and library versions, environment specifications captured with pip freeze, sample outputs saved for a random subset of 10 samples per configuration, and timing variance analysis to assess variance and identify outliers. Despite these measures, some non determinism remains (CUDA kernel selection, memory allocation patterns), leading to small variations (typically less than 5%) in latency measurements across repeated runs.

2.9 Limitations and Constraints

The evaluation used 50 samples per dataset due to computational constraints. Whilst this provides sufficient data to identify major trends (accuracy differences greater than 10 percentage points), fine grained performance differences may not be statistically significant. Larger scale evaluation would provide more robust statistical confidence.

Dataset coverage focused on two primary text datasets (HellaSwag and ARC Easy) and one synthetic visual dataset. Generalisation to other task types (long form generation, code synthesis, conversational dialogue, complex visual reasoning, multilingual tasks) requires additional evaluation. Selected datasets represent common benchmarks but do not exhaustively cover large language model capabilities.

Results are specific to tested GPU platforms (NVIDIA L4 and A100). Performance characteristics may differ on consumer GPUs, specialised edge processors, mobile GPUs, or CPU only inference. Memory reduction benefits likely generalise across platforms, but latency characteristics depend heavily on hardware specific optimisations.

Hyperparameters for novel techniques (outlier thresholds, layer selection, pruning ratios) were based on reasonable defaults and limited manual tuning rather than exhaustive search. Optimised hyperparameters might improve performance, though the magnitude of observed failures suggests fundamental issues beyond hyperparameter tuning.

All evaluation was conducted in English. Multilingual model performance under compression remains an area for future investigation. The restriction to post training methods, whilst intentional for accessibility, inherently limits achievable compression accuracy trade offs. Techniques incorporating fine tuning or quantisation aware training would likely achieve better results but require substantially more computational resources [15].

The use of synthetic images for multimodal evaluation, whilst enabling controlled testing, does not fully assess real world visual understanding capabilities. Evaluation on natural images with complex scenes would provide more realistic performance estimates [26].

2.10 Statistical Considerations

Given relatively small sample sizes and exploratory research nature, statistical significance testing was not the primary focus. The research prioritises identification of large, practically significant effects (accuracy differences greater than 10 percentage points, latency differences greater than 50%) over statistically subtle differences. In practical deployment scenarios, large effect sizes are typically more relevant than small but statistically significant differences [8].

Findings that replicate across multiple models (five different architectures and sizes) provide stronger evidence than single model results. Conclusions are based on consistent patterns across models rather than single point estimates. Extreme values in latency measurements were investigated to determine whether they represented genuine performance characteristics or measurement artefacts.

2.11 Ethical Considerations

The research prioritises transparency in reporting both successful and unsuccessful approaches. Failed techniques are documented in detail to contribute to community knowledge about practical compression challenges. All experiments use publicly available models and datasets with proper attribution to original sources [26]. Model weights were accessed through official channels (HuggingFace Model Hub) with appropriate licences verified.

The research does not involve human subjects or raise concerns about bias amplification, as it focuses on technical model compression rather than model outputs or decisions affecting individuals. No novel datasets were created, and all evaluation uses existing benchmark datasets widely used in the research community.

Chapter 3

Results

This chapter presents the experimental results obtained from evaluating post training compression techniques the chosen models. The results are organised to first present baseline quantisation performance, followed by analysis of novel compression techniques, and concluding with cross model comparisons.

3.1 Experimental Overview

A total of 62 distinct experiments were conducted across 5 models, 6 optimisation configurations, and 3 datasets (HellaSwag, ARC Easy, and Visual QA).

The six optimisation configurations evaluated were:

1. FP16 Baseline (half precision, uncompressed reference)
2. INT8 Quantisation (8 bit weight quantisation)
3. INT4 Quantisation (4 bit weight quantisation)
4. INT4 + Activation Quantisation (weight and activation quantisation)
5. INT4 + TRIM (weight quantisation with token pruning)
6. INT4 + Activation Quantisation + TRIM (combined novel techniques)

3.2 Baseline Quantisation Results

This section presents results for the three baseline quantisation methods (FP16, INT8, INT4) that use standard, well established compression techniques [15]. These results establish the performance envelope against which novel techniques are compared.

3.2.1 Qwen2 0.5B Results

Qwen2 0.5B, as the smallest model evaluated (494M parameters), provides insight into compression effects on compact architectures already optimised for efficiency.

Accuracy Performance

Table 3.1 presents detailed results across both text evaluation datasets. The model exhibited interesting behaviour under quantisation, with INT8 actually showing modest accuracy improvements on HellaSwag (37.0% vs 32.0% baseline), though this improvement did not generalise to ARC Easy. INT4 quantisation resulted in accuracy degradation of 6-10 percentage points depending on the dataset.

Table 3.1: Qwen2 0.5B baseline quantisation results

Configuration	Dataset	Accuracy (%)	Latency (ms)	Memory (GB)
FP16 Baseline	HellaSwag	32.0	263.1	1.01
FP16 Baseline	ARC Easy	56.0	262.4	1.01
INT8 Quantisation	HellaSwag	37.0	1325.4	0.65
INT8 Quantisation	ARC Easy	51.0	1308.0	0.65
INT4 Quantisation	HellaSwag	28.0	535.9	0.48
INT4 Quantisation	ARC Easy	46.0	537.5	0.48

Memory Efficiency

Qwen2 0.5B achieved exceptional memory reduction through quantisation. INT8 reduced memory footprint by 35.6% (from 1.01 GB to 0.65 GB), whilst INT4 achieved 52.5% reduction (to 0.48 GB). This represents the most memory efficient configuration across all evaluated models, enabling deployment on extremely resource constrained devices. The aggressive quantisation was possible due to the model's already compact architecture, though this came at the cost of moderate accuracy degradation [15].

Latency Characteristics

Unexpectedly, quantisation substantially increased inference latency rather than decreasing it. INT8 inference was 5× slower than FP16 baseline (1325.4 ms vs 263.1 ms), and INT4 was approximately 2× slower (535.9 ms). This counter intuitive result reflects the overhead of BitsAndBytes' on the fly dequantisation during forward passes. The library prioritises memory reduction over speed, dequantising weights dynamically rather than using native integer operations [8].

3.2.2 TinyLlama Results

TinyLlama (1.1B parameters) exhibited the poorest baseline performance of all evaluated models, highlighting the challenge of compressing already weak or undertrained models.

Accuracy Performance

Table 3.2 shows TinyLlama's results. The model achieved only 30.0% accuracy on the synthetic evaluation task in FP16, with no change under INT8 quantisation but further degradation to 20.0% under INT4. This represents a 33% relative accuracy loss from aggressive quantisation.

Analysis of Poor Performance

The poor performance of TinyLlama under compression likely stems from multiple factors. First, as a

Table 3.2: TinyLlama baseline quantisation results (synthetic text evaluation)

Configuration	Accuracy (%)	Latency (ms)	Memory (GB)	Throughput (tok/s)
FP16 Baseline	30.0	132.2	2.21	37.8
INT8 Quantisation	30.0	531.9	1.26	9.4
INT4 Quantisation	20.0	266.5	0.81	18.8

compact 1.1B parameter model, TinyLlama has limited parameter redundancy, making it more sensitive to precision reduction. Second, the model may lack the robust internal representations that enable larger, better trained models to tolerate quantisation. Third, evaluation on relatively challenging datasets may expose weaknesses in the model's baseline capabilities that are then exacerbated by compression [8].

This finding has important implications, it suggests that model quality and training robustness matter more for successful compression than model size alone. A well trained small model may compress better than a poorly trained larger model.

Memory Characteristics

Despite poor accuracy, TinyLlama achieved substantial memory savings, 43.0% with INT8 (1.26 GB) and 63.4% with INT4 (0.81 GB). At 0.81 GB, the INT4 TinyLlama configuration represents one of the smallest full model deployments evaluated, suitable for mobile devices or embedded systems where the baseline accuracy level might be acceptable for simple tasks.

3.2.3 Phi 2 Results

Phi 2 (2.7B parameters) demonstrated more varied behaviour across datasets, with some configurations showing accuracy improvements under quantisation.

Accuracy Performance

Table 3.3 presents Phi 2 results. Notably, INT8 and INT4 quantisation on the text evaluation dataset (100 samples aggregated from HellaSwag and ARC Easy) actually improved accuracy from the 90% FP16 baseline to 100%. This surprising result suggests potential beneficial regularisation effects from quantisation, though the small evaluation sample size (50 per dataset) means this difference may not be statistically significant.

Table 3.3: Phi 2 baseline quantisation results (aggregated text evaluation)

Configuration	Accuracy (%)	Latency (ms)	Memory (GB)	Memory Reduction
FP16 Baseline	90.0	175.5	5.59	0.0%
INT8 Quantisation	100.0	672.7	3.08	44.9%
INT4 Quantisation	100.0	357.5	1.91	65.9%

Dataset Specific Analysis

Examining the disaggregated results reveals that Phi 2's baseline performance varied substantially across datasets:

- HellaSwag: 25.0% (FP16) $\rightarrow$ 30.0% (INT8) $\rightarrow$ 23.0% (INT4)
- ARC Easy: 59.0% (FP16) $\rightarrow$ 49.0% (INT8) $\rightarrow$ 34.0% (INT4)

The apparent accuracy improvement in aggregated results reflects better performance on HellaSwag with INT8 offsetting degradation on ARC Easy. This highlights the importance of multi dataset evaluation, as single dataset results may be misleading.

Memory and Latency Trade offs

Phi 2 achieved excellent memory compression, 44.9% reduction with INT8 and 65.9% with INT4. At 1.91 GB, INT4 Phi 2 could fit comfortably on consumer GPUs alongside other workloads. However, latency increased substantially (103.7% overhead for INT4, 283.2% for INT8), consistent with the pattern observed across all models.

3.2.4 Mistral 7B Results

Mistral 7B (7.3B parameters) demonstrated the most robust quantisation behaviour among text only models, maintaining high accuracy across compression methods.

Accuracy Performance

Table 3.4 shows Mistral 7B results. The model achieved 90.0% accuracy on aggregated text evaluation in FP16, INT8, and INT4 configurations, representing perfect accuracy retention despite 4 bit weight quantisation. This exceptional robustness suggests that Mistral 7B's larger capacity and high quality training enable it to tolerate aggressive compression [18].

Table 3.4: Mistral 7B baseline quantisation results

Configuration	**Accuracy (%)**	**Latency (ms)**	**Memory (GB)**	**Memory Reduction**
FP16 Baseline	90.0	190.3	14.51	0.0%
INT8 Quantisation	90.0	962.2	8.04	44.6%
INT4 Quantisation	90.0	368.4	4.27	70.6%

Dataset Specific Performance

Breaking down by individual datasets:

- HellaSwag: 58.0% (FP16) $\rightarrow$ 58.0% (INT8) $\rightarrow$ 59.0% (INT4)
- ARC Easy: 84.0% (FP16) $\rightarrow$ 84.0% (INT8) $\rightarrow$ 80.0% (INT4)

The slight improvement on HellaSwag with INT4 and minimal degradation on ARC Easy (4 percentage points) demonstrate Mistral 7B's compression robustness. This aligns with findings that well trained, larger models maintain better performance under compression [8].

Memory Efficiency for Large Models

Mistral 7B's INT4 configuration achieved 70.6% memory reduction (from 14.51 GB to 4.27 GB), enabling deployment on consumer grade GPUs with 6-8 GB VRAM. This reduction is particularly significant for 7B parameter models, which typically require 14-16 GB in FP16. The ability to reduce requirements to under 5 GB whilst maintaining 90% accuracy makes Mistral 7B an excellent candidate for edge deployment [15].

Latency Analysis

Mistral 7B exhibited moderate latency increases, 93.6% overhead for INT4 and 405.6% for INT8. Whilst substantial in relative terms, the absolute latency remained reasonable (368.4 ms for INT4), suggesting that the dequantisation overhead scales less than proportionally with model size.

3.2.5 LLaVA 1.5 7B Results

LLaVA 1.5 7B, the sole multimodal model evaluated, demonstrated exceptional quantisation robustness on the synthetic visual task.

Accuracy Performance

Table 3.5 presents LLaVA results on the synthetic colour recognition visual question answering task. The model achieved perfect 100.0% accuracy in FP16, INT8, and INT4 configurations. This remarkable result suggests that the synthetic visual task may have been too simple to differentiate model capabilities, or that LLaVA's visual encoding pathway is particularly robust to quantisation [9].

Table 3.5: LLaVA 1.5 7B baseline quantisation results (visual QA)

Configuration	Accuracy (%)	Latency (ms)	Memory (GB)	Memory Reduction
FP16 Baseline	100.0	120.1	14.52	0.0%
INT8 Quantisation	100.0	547.5	7.76	46.6%
INT4 Quantisation	100.0	249.8	4.52	68.9%

Multimodal Compression Characteristics

LLaVA's compression characteristics closely mirror those of text only models of similar size (Mistral 7B). INT8 achieved 46.6% memory reduction and INT4 achieved 68.9% reduction, both very close to Mistral 7B's results. This suggests that the vision encoder and language model components compress similarly, with no obvious compression resistance in the cross modal alignment layers.

The latency overhead pattern also matched text models, INT8 showed 356.0% overhead and INT4 showed 108.0% overhead. The absolute latencies remained practical for interactive applications (249.8 ms for INT4).

Visual Task Evaluation Limitations

The perfect accuracy across all configurations raises questions about task difficulty. The synthetic colour recognition task may not sufficiently stress the model's capabilities to reveal compression effects. Future work should evaluate on more challenging visual reasoning tasks such as VQAv2 [26] or complex scene understanding to better assess multimodal compression robustness.

3.3 Cross Model Baseline Comparison

Aggregating across all models and configurations reveals clear patterns in baseline quantisation behaviour.

Table 3.6: Accuracy by model and quantisation method

Model	**FP16 Accuracy (%)**	**INT8 Accuracy (%)**	**INT4 Accuracy (%)**
Qwen2 0.5B	44	44	37
TinyLlama	30	30	20
Phi 2	90	100	100
Mistral 7B	90	90	90
LLaVA 1.5 7B	100	100	100

Model Size and Compression Robustness

Table 3.6 (referenced from experimental visualisations) shows accuracy distribution by model. Larger, better trained models (Mistral 7B, LLaVA 1.5 7B) maintained substantially higher accuracy under compression than smaller models (TinyLlama, Qwen2 0.5B). The median accuracy for 7B models exceeded 75%, whilst sub 2B models showed median accuracy around 50%.

This relationship suggests that model capacity and training quality are primary determinants of compression success, more so than specific architectural choices [8]. The finding implies that practitioners seeking to deploy compressed models should prioritise starting with well trained, adequately sized models rather than attempting aggressive compression of already compact models.

Memory Reduction Consistency

Memory reduction percentages remained remarkably consistent across model sizes:

- INT8: 35-47% reduction across all models
- INT4: 52-71% reduction across all models

This consistency validates the theoretical predictions of quantisation, reducing weight precision from FP16 (16 bits) to INT8 (8 bits) yields approximately 50% reduction, and to INT4 (4 bits) yields approximately 75% reduction. The observed values fall slightly short of theoretical maxima due to non quantised components (embeddings, layer norms, buffers).

Latency Overhead Universality

All models exhibited substantial latency increases under BitsAndBytes quantisation:

- INT8: 200-400% overhead (3-5× slower than FP16)
- INT4: 80-120% overhead (1.8-2.2× slower than FP16)

This universal slowdown reflects BitsAndBytes' implementation strategy, dynamic dequantisation during inference rather than native low precision computation. Whilst this approach maximises compatibility and ease of use, it sacrifices speed. Production deployments using optimised inference engines (TensorRT LLM, vLLM) with native INT4 kernels would likely show latency reductions rather than increases [15].

3.4 Novel Technique Results

This section presents results from the three novel compression techniques evaluated, Outlier Aware Activation Quantisation, Attention Based TRIM, and their combinations. Contrary to initial hypotheses, all novel techniques resulted in substantial accuracy degradation and/or latency increases, providing valuable lessons about practical compression challenges.

3.4.1 Activation Quantisation Results

The Outlier Aware Activation Quantisation technique aimed to reduce memory usage by quantising intermediate activations whilst preserving outliers in FP16 [18]. Results across all models revealed fundamental implementation challenges.

Qwen2 0.5B with Activation Quantisation

Table 3.7 shows that combining INT4 weight quantisation with activation quantisation maintained the memory footprint (0.48 GB) whilst degrading accuracy moderately on HellaSwag (maintained 28.0%) but more substantially on ARC Easy (42.0% vs 46.0% INT4 only). Critically, latency nearly doubled (996.5 ms vs 535.9 ms).

Table 3.7: Qwen2 0.5B activation quantisation results

Configuration	Dataset	Accuracy (%)	Latency (ms)	Memory (GB)
INT4 Only	HellaSwag	28.0	535.9	0.48
INT4 + Act Quant	HellaSwag	28.0	996.5	0.48
INT4 Only	ARC Easy	46.0	537.5	0.48
INT4 + Act Quant	ARC Easy	42.0	998.5	0.48

Phi 2 with Activation Quantisation

Phi 2 exhibited more severe degradation. Activation quantisation reduced accuracy from 100.0% (INT4 alone) to 80.0% on the aggregated text evaluation, representing a 20 percentage point absolute loss. Latency increased from 357.5 ms to 622.2 ms (74.0% overhead).

Mistral 7B with Activation Quantisation

Mistral 7B showed catastrophic accuracy loss under activation quantisation, dropping from 90.0% (INT4 only) to 40.0% with activation quantisation. This 50 percentage point loss is particularly striking given Mistral 7B's robustness to weight quantisation alone. Latency increased from 368.4 ms to 688.6 ms (87.0% overhead).

TinyLlama with Activation Quantisation

TinyLlama's already poor performance further degraded, from 20.0% accuracy (INT4) to 10.0% with activation quantisation. At 10.0% accuracy on what appears to be a 10 option evaluation, the model performed at random chance levels, indicating complete failure.

LLaVA 1.5 7B with Activation Quantisation

Most dramatically, LLaVA's perfect 100.0% accuracy (INT4 only) collapsed entirely to 0.0% with activation quantisation. This total failure suggests that the multimodal model's cross modal processing is particularly sensitive to activation precision, or that the outlier detection strategy failed catastrophically for vision language representations.

Analysis of Activation Quantisation Failures

The consistent failure of activation quantisation across all models reveals several critical issues:

1. **Overhead Dominates Benefits:** The statistical outlier detection, per token scaling, and conditional quantisation logic introduced 60-100% latency overhead. Even if activation quantisation reduced memory bandwidth requirements, the computational overhead of the quantisation operations themselves exceeded any bandwidth savings.
2. **Insufficient Accuracy Benefit:** In the few cases where activation quantisation maintained accuracy (Qwen2 0.5B on HellaSwag, Phi 2 at 80%), the improvements over INT4 only weight quantisation were minimal or negative. The cost benefit ratio was unfavourable in all cases.
3. **Sensitivity Varies by Model:** Larger models (Mistral 7B) and multimodal models (LLaVA) showed greater sensitivity to activation quantisation than smaller models. This suggests that complex models with rich internal representations rely on high precision intermediate computations, and cannot tolerate aggressive activation compression without calibration or fine tuning.
4. **Implementation Quality Matters:** The forward hook based implementation added overhead beyond the quantisation operations themselves. Production quality implementations with custom CUDA kernels and proper operator fusion might show better results [18], but such implementations require substantial engineering investment beyond the scope of this research.
5. **Need for Calibration:** Post training activation quantisation likely requires calibration on representative data to set appropriate quantisation scales and outlier thresholds. The zero shot approach used here, whilst appealing for its simplicity, proved inadequate [15].

3.4.2 TRIM (Token Pruning) Results

The Attention Based TRIM technique attempted to reduce computational cost by removing low importance tokens from input sequences [21, 22]. Results revealed severe accuracy degradation across all models.

Qwen2 0.5B with TRIM

Table 3.8 shows TRIM's effects. On HellaSwag, accuracy remained at 28.0%, matching INT4 only performance. However, on ARC Easy, accuracy dropped from 46.0% (INT4) to 32.0% (INT4+TRIM), representing a 30% relative accuracy loss. Memory remained at 0.48 GB and latency increased slightly to 543.3 ms.

Table 3.8: Qwen2 0.5B TRIM results

Configuration	Dataset	Accuracy (%)	Latency (ms)	Memory (GB)
INT4 Only	HellaSwag	28.0	535.9	0.48
INT4 + TRIM	HellaSwag	28.0	543.3	0.48
INT4 Only	ARC Easy	46.0	537.5	0.48
INT4 + TRIM	ARC Easy	32.0	540.6	0.48

Phi 2 with TRIM

Phi 2 exhibited catastrophic accuracy collapse under TRIM, from 100.0% (INT4) to 20.0% (INT4+TRIM). This 80 percentage point absolute loss represents near total failure of the model's reasoning capabilities. The latency decrease expected from token pruning failed to materialise, with inference time remaining at 352.7 ms (comparable to INT4's 357.5 ms).

Mistral 7B with TRIM

Mistral 7B showed complete failure, accuracy dropped from 90.0% (INT4) to 0.0% (INT4+TRIM). Every evaluated sample was answered incorrectly after token pruning. Latency increased slightly from 368.4 ms to 384.9 ms rather than decreasing, suggesting the pruning overhead exceeded the savings from shorter sequences.

TinyLlama with TRIM

TinyLlama similarly collapsed to 0.0% accuracy from an already low 20.0% baseline. The complete failure across all samples indicates that the token pruning removed critical information required for any correct predictions.

Analysis of TRIM Failures

The universal failure of TRIM across all models stems from multiple fundamental issues:

1. **Attention Weight Inadequacy:** Using attention weights alone to determine token importance proved insufficient. Attention patterns reflect one aspect of token relevance but do not capture all mechanisms by which tokens contribute to predictions. Specifically, attention weights come from intermediate layers, but tokens may be critical for early or late processing stages not reflected in those specific attention patterns [7].
2. **Causal Attention Conflicts:** In decoder only models with causal attention, later tokens can only attend to earlier tokens in the original sequence. Removing earlier "low importance" tokens disrupts this causal structure, as later tokens that reference those positions now encounter attention masks that don't align with the modified sequence length. The model was never trained to handle such disruptions.
3. **Lack of Calibration:** The importance thresholds and pruning ratios (targeting 30% token reduction) were set arbitrarily without calibration on representative data. Different tasks have different token importance distributions, and fixed thresholds cannot adapt [22].
4. **Question Answering Sensitivity:** The multiple choice question answering tasks used for evaluation are particularly sensitive to token removal. Question words, option identifiers, and key phrases are all critical, removing any of them can render the question incomprehensible or change the answer. More robust tasks (e.g., sentiment classification, topic categorisation) might tolerate token pruning better.
5. **Implementation Overhead:** The process of extracting attention weights, computing importance scores, and reconstructing modified input sequences added computational overhead that offset any benefits from shorter sequences. At the 30% pruning ratio used, the savings were insufficient to compensate.
6. **Static Pruning Limitation:** The token pruning was performed once per input before inference, with no adaptation during generation. Dynamic pruning that adjusts as the model generates tokens might prove more effective but would require deeper integration with the model's forward pass [21].

3.4.3 Combined Technique Results

Combining activation quantisation and TRIM (INT4 + Act Quant + TRIM) generally produced results that combined the worst aspects of both techniques.

Combined Results Summary

- **Qwen2 0.5B:** Accuracy on ARC Easy improved slightly to 80.0% compared to TRIM alone (32.0%), but remained well below INT4 baseline (46.0%). Latency increased to 524.9 ms, the highest of any Qwen2 configuration.
- **Phi 2:** Maintained 20.0% accuracy, matching TRIM alone. Latency reached 610.7 ms, higher than INT4 (357.5 ms) but lower than activation quantisation alone (622.2 ms).
- **Mistral 7B:** Showed partial recovery to 20.0% accuracy from 0.0% (TRIM alone), but remained catastrophically poor compared to 90.0% baseline. Latency increased to 708.0 ms.
- **TinyLlama:** Recovered to 20.0% accuracy from 0.0% (TRIM alone), matching INT4 baseline. Latency was 437.6 ms.

The sporadic accuracy improvements when combining techniques (compared to TRIM alone) suggest complex interactions between weight quantisation, activation quantisation, and token pruning. However, none of the combined configurations outperformed simple INT4 weight quantisation alone, and all introduced substantial latency overhead.

3.4.4 Computational Cost Analysis of Novel Techniques

Table 3.9 summarises the computational overhead introduced by novel techniques compared to INT4 only baselines.

Table 3.9: Latency overhead of novel techniques (relative to INT4 baseline)

Model	Act Quant Overhead	TRIM Overhead	Combined Overhead
Qwen2 0.5B	+85.9%	+1.4%	2.2%
Phi 2	+74.0%	1.3%	+70.8%
Mistral 7B	+87.0%	+4.5%	+92.2%
TinyLlama	+80.1%	0.6%	+64.2%
LLaVA 1.5 7B	+222.7%	N/A	N/A

Activation quantisation consistently added 74-223% overhead, whilst TRIM added minimal overhead (1-5%) or even slight improvements in some cases. The overhead consistency across models suggests systematic implementation inefficiencies rather than model specific issues.

3.5 Accuracy Memory Trade off Analysis

Examining the relationship between accuracy and memory usage across all configurations highlights the most effective compression strategies.

Table 3.6 (referenced from experimental visualisations) would show a clear trade off curve. INT4 weight only quantisation provides the most efficient balance, achieving a 50–70% reduction in memory usage with only a 0–10% drop in accuracy. In contrast, the more complex compression techniques occupy less favourable positions, offering similar or higher memory usage but significantly lower accuracy.

Table 3.10 quantifies this comparison for representative models.

Table 3.10: Accuracy–memory trade off comparison

Model	**Configuration**	**Accuracy (%)**	**Memory (GB)**	**Efficient Trade off**
Mistral 7B	FP16 Baseline	90.0	14.51	Yes
Mistral 7B	INT4 Only	90.0	4.27	Yes
Mistral 7B	INT4 + Act	40.0	4.27	No
Mistral 7B	INT4 + TRIM	0.0	4.27	No
Phi 2	FP16 Baseline	90.0	5.59	Yes
Phi 2	INT4 Only	100.0	1.91	Yes
Phi 2	INT4 + Act	80.0	1.91	No
Phi 2	INT4 + TRIM	20.0	1.90	No
LLaVA 1.5 7B	FP16 Baseline	100.0	14.52	Yes
LLaVA 1.5 7B	INT4 Only	100.0	4.52	Yes
LLaVA 1.5 7B	INT4 + Act	0.0	4.52	No

Overall, only the FP16 and INT4 weight only configurations achieve the most efficient balance between accuracy and memory usage. None of the additional compression methods outperform these baselines, indicating that weight only quantisation remains the most effective approach for the evaluated models and tasks.

3.6 Cross Model Comparison

3.6.1 Model Capacity and Compression Robustness

Aggregating results across all models reveals a clear relationship between model size, training quality, and compression success.

Average Performance by Model Size

Table 3.11 presents average accuracy across all configurations by model.

The high standard deviations reflect the bimodal distribution of results: baseline configurations maintain reasonable accuracy, whilst novel technique configurations collapse. The mean accuracy is thus not particularly informative, median or best case accuracy would better represent viable deployment configurations.

Table 3.11: Average performance by model (across all configurations)

Model	Parameters	Mean Acc (%)	Std Dev Acc	Mean Memory (GB)
LLaVA 1.5 7B	7.0B	75.0	50.0	7.83
Mistral 7B	7.3B	55.0	40.4	6.60
Phi 2	2.7B	68.3	38.2	2.72
Qwen2 0.5B	0.5B	78.3	34.9	0.59
TinyLlama	1.1B	18.3	11.7	1.12

Best Case Accuracy by Model

Examining the best achievable accuracy for each model under compression (INT4 or better):

- Mistral 7B: 90.0% (matches FP16 baseline)
- Phi 2: 100.0% (exceeds FP16 baseline of 90.0%)
- Qwen2 0.5B: 56.0% (FP16 baseline on ARC Easy)
- LLaVA 1.5 7B: 100.0% (matches FP16 baseline)
- TinyLlama: 30.0% (FP16 baseline)

This ranking confirms that larger, better trained models (Mistral 7B, LLaVA 1.5 7B) achieve the highest absolute accuracy whilst also maintaining that accuracy under compression. Smaller models like TinyLlama struggle both in absolute terms and under compression [8].

3.6.2 Dataset Difficulty and Model Sensitivity

Different datasets exposed different model weaknesses, with ARC Easy generally producing higher accuracy than HellaSwag across models.

ARC Easy vs HellaSwag Performance

For models evaluated on both datasets (Qwen2, Phi 2, Mistral):

- Qwen2 0.5B: 56.0% (ARC) vs 32.0% (HellaSwag) in FP16
- Phi 2: 59.0% (ARC) vs 25.0% (HellaSwag) in FP16
- Mistral 7B: 84.0% (ARC) vs 58.0% (HellaSwag) in FP16

HellaSwag's focus on commonsense reasoning and natural language understanding proved more challenging than ARC Easy's factual science questions. This difficulty difference persisted under compression, with compressed models generally maintaining the same relative performance gap between datasets.

3.6.3 Multimodal vs Text Only Compression

LLaVA 1.5 7B's compression characteristics closely matched those of comparable sized text only models (Mistral 7B), with similar memory reduction percentages and latency overheads. The vision encoder and cross modal components did not exhibit obvious compression resistance beyond that of standard transformer layers.

However, LLaVA showed extreme sensitivity to activation quantisation (complete failure to 0.0% accuracy), whilst Mistral 7B maintained 40.0% accuracy. This suggests that whilst multimodal model weights compress well, the intermediate representations in cross modal processing may be more fragile under aggressive quantisation [9].

3.7 Key Findings

Synthesising across all experiments, several clear findings emerge:

3.7.1 INT4 Weight Quantisation Dominates

Standard INT4 weight quantisation using BitsAndBytes achieved the best accuracy memory latency trade off across all models and tasks. The technique achieved:

1. 52 to 71 percent memory reduction (consistent across model sizes),
2. 0 to 10 percent accuracy degradation (for well trained models),
3. Modest latency overhead (80 to 120 percent) that could be eliminated with optimised inference engines, and
4. Zero requirement for calibration data or fine tuning.

This result validates the practical utility of well engineered baseline approaches over complex novel techniques, particularly in post training settings where simplicity and robustness matter more than theoretical optimality.

3.7.2 Model Quality Matters More Than Size

The data strongly supports that model training quality and robustness matter more than raw parameter count for compression success. Well trained models (Mistral 7B at 7.3B parameters, Phi 2 at 2.7B) maintained high accuracy under compression, whilst poorly trained models (TinyLlama at 1.1B) collapsed even under modest compression. This finding has important implications for practitioners: starting with a well trained, appropriately sized model and then compressing it will yield better results than starting with a compact but poorly trained model.

3.7.3 Novel Techniques Require Calibration

All evaluated novel techniques (activation quantisation, TRIM, combinations) failed to improve upon INT4 baseline, primarily due to their zero shot post training implementation without access to calibration data. The techniques relied on:

1. Accurate outlier detection (activation quantisation) requiring representative activation statistics,
2. Appropriate token importance thresholds (TRIM) requiring task specific attention pattern analysis, and
3. Fine tuned hyperparameters (both) requiring validation data for tuning.

Without these prerequisites, the techniques over pruned, mis quantised, or otherwise disrupted model computations. This negative result confirms the challenge of post training compression: techniques that work well with training time integration or calibration often fail when applied naively after training.

3.7.4 Implementation Quality Determines Practical Performance

The universal latency increases under BitsAndBytes quantisation (contrary to theoretical predictions of speedup) highlight the importance of implementation quality. The library's dynamic dequantisation strategy prioritises compatibility and ease of use over performance. Production deployments would require:

1. Native low precision compute kernels (INT4 matrix multiplication),
2. Operator fusion to eliminate intermediate materialisation,
3. Hardware specific optimisations (TensorRT for NVIDIA, CoreML for Apple), and
4. Batch processing to amortise overhead.

The research results reflect the state of accessible, easy to use quantisation tools rather than the ultimate performance limits of compression techniques. Future work with production quality inference engines would likely show substantial speedups alongside memory savings.

3.7.5 Task Difficulty Affects Compression Tolerance

The extremely high accuracy on LLaVA's synthetic visual task (100 percent across FP16, INT8, INT4) suggests that task difficulty matters for assessing compression effects. Simple tasks may not differentiate between compressed and uncompressed models, whilst challenging tasks (HellaSwag commonsense reasoning) reveal degradation more clearly. Robust evaluation of compression techniques requires diverse, challenging benchmarks that stress model capabilities. The 30 to 50 percent accuracy achieved by smaller models on HellaSwag and ARC Easy provided sufficient headroom to observe compression effects, whilst LLaVA's perfect score provided minimal signal.

3.8 Summary

This results chapter has presented comprehensive experimental findings across five models, six compression configurations, and three evaluation tasks. The clear finding is that simple INT4 weight quantisation substantially outperforms complex novel techniques for post training compression, achieving excellent memory savings with minimal accuracy loss on well trained models. The next chapter discusses implications of these findings and suggests directions for future research.

Chapter 4

Conclusion and Future Work

This research conducted a comprehensive empirical evaluation of post training compression techniques for large language models and multimodal models on resource constrained edge devices. Through systematic experimentation across five models (494M to 7.3B parameters), six compression configurations, and three benchmarks, the study provides clear evidence about which compression strategies succeed in practice and which fail when applied post training without calibration data or fine tuning.

4.1 Summary of Key Findings

4.1.1 INT4 Weight Quantisation Provides Optimal Post Training Compression

Standard 4 bit weight quantisation using BitsAndBytes substantially outperforms more complex compression approaches for post training deployment. Across all evaluated models, INT4 quantisation achieved:

- **Consistent memory reduction:** 52 to 71 percent decrease in peak GPU memory usage across model sizes
- **Minimal accuracy degradation:** 0 to 10 percent accuracy loss for well trained models (Mistral 7B, LLaVA 1.5 7B, Phi 2)
- **Zero calibration requirement:** No training data, calibration sets, or task specific examples needed
- **Deployment simplicity:** Single line configuration change with automatic compression handling

This validates the practical utility of established, well implemented baseline techniques over novel but immature approaches.

4.1.2 Model Quality Determines Compression Success

Model training quality and architectural robustness matter more than raw parameter count for successful compression:

- Well Trained Large Models (Mistral 7B, LLaVA 1.5 7B): Maintained 90 to 100 percent accuracy under INT4 quantisation with minimal degradation and robust internal representations tolerant to precision reduction.
- Poorly Trained Small Model (TinyLlama): Baseline accuracy of 30 percent degraded to 20 percent under INT4, exhibiting sensitivity to compression and complete failure under novel techniques.
- Mid Range Models (Phi 2, Qwen2 0.5B): Showed variable compression tolerance depending on task, with Phi 2 demonstrating better robustness.

This has critical implications: investing in a well trained, appropriately sized model followed by compression yields better deployed performance than starting with a compact but poorly trained model.

4.1.3 Novel Post Training Techniques Require Calibration

All three novel compression techniques (Outlier Aware Activation Quantisation, Attention Based TRIM, and combinations) failed to improve upon simple INT4 weight quantisation, with most configurations showing catastrophic accuracy degradation.

Activation Quantisation Failures:

- Accuracy losses from 10 percent (Qwen2 0.5B) to complete failure at 0 percent (LLaVA 1.5 7B)
- Latency overhead of 60 to 223 percent due to outlier detection and conditional quantisation
- Severe impact on multimodal models, suggesting fragile cross modal representations

TRIM Token Pruning Failures:

- Complete failures with 0 percent accuracy on several model task combinations
- 20 to 80 percentage point accuracy losses across other configurations
- No meaningful latency improvements despite 30 percent token reduction target

Root Causes: These techniques fundamentally require calibration, fine tuning, or task specific adaptation. Successful implementations in literature typically include calibration datasets, task specific validation, and fine tuning to recover accuracy. This negative result establishes clear boundaries on zero shot post training compression and highlights the need for either calibration data or more robust algorithms.

4.1.4 Implementation Quality Dominates Theoretical Predictions

All quantisation methods increased inference latency rather than decreasing it, contrary to theoretical predictions. INT8 showed 200 to 400 percent latency overhead, whilst INT4 showed 80 to 120 percent overhead. This reflects BitsAndBytes' implementation strategy: prioritising memory reduction and ease of use over raw speed through dynamic dequantisation (weights stored in INT4 but dequantised to FP16 during computation), avoiding custom CUDA kernels, and enabling broad GPU compatibility.

Production Deployment Alternatives: The observed latency increases reflect current accessible tool capabilities rather than fundamental limits. Production deployments using optimised inference engines would show substantially different performance:

- TensorRT LLM with native INT4 compute: Expected 2 to 4 times speedup over FP16
- vLLM with optimised attention kernels: Improved throughput alongside memory savings
- Hardware with native low precision support: Apple Neural Engine, Google TPU, specialised AI accelerators

This emphasises that researchers and practitioners must distinguish between algorithmic compression potential and current tool capabilities.

4.1.5 Multimodal Compression Mirrors Text Only Models

LLaVA 1.5 7B's compression characteristics closely matched comparable sized text only models (Mistral 7B):

- Nearly identical memory reduction (68.9 percent for LLaVA vs 70.6 percent for Mistral under INT4)
- Matching latency overhead patterns (108 percent for LLaVA vs 93.6 percent for Mistral)
- Perfect accuracy retention on synthetic visual tasks under weight quantisation
- Extreme sensitivity to activation quantisation (0 percent accuracy), suggesting cross modal processing fragility

This is encouraging for deploying multimodal models: vision encoder and cross modal alignment layers do not present compression obstacles beyond standard transformer layers. However, the catastrophic failure under activation quantisation warrants caution.

4.2 Research Contributions

4.2.1 Comprehensive Empirical Evaluation

The study provides one of the most comprehensive evaluations of post training compression across diverse model scales, multiple quantisation methods, and three evaluation tasks. Unlike many compression studies reporting only positive results, this work documents failed techniques and analyses why they failed, contributing negative knowledge that prevents others from pursuing similar dead end approaches.

4.2.2 Practical Deployment Guidelines

The research establishes clear, evidence based recommendations for practitioners deploying compressed models. The finding that INT4 weight quantisation provides the optimal balance of simplicity, memory reduction, and accuracy preservation offers actionable guidance applicable across different model types and deployment scenarios.

4.2.3 Model Quality Insights

The discovery that model training quality matters more than size for compression success provides valuable guidance for model development pipelines. Organisations seeking edge deployment should prioritise training robust base models rather than optimising solely for initial compactness.

4.2.4 Novel Technique Failure Analysis

The detailed analysis of why novel techniques failed in post training settings identifies specific failure modes:

- Inadequacy of attention weights alone for token importance
- Sensitivity of cross modal processing to activation precision
- Implementation overhead negating theoretical benefits
- Need for calibration data and hyperparameter tuning

These findings can inform future compression technique development.

4.2.5 Open and Reproducible Research

All experiments used publicly available models, datasets, and libraries, with detailed methodology documentation enabling reproduction. The research prioritises transparency and reproducibility [8].

4.3 Practical Deployment Recommendations

4.3.1 Recommended Configurations by Use Case

Table 4.1 summarises optimal compression strategies for different deployment scenarios.

4.3.2 Model Selection Guidelines

When selecting models for compressed deployment:

Table 4.1: Deployment recommendations by use case

Use Case	Recommendation	Rationale
Accuracy Critical Applications	INT8 weight only quantisation	¡1% accuracy loss, 35-47% memory reduction, conservative approach
Balanced Deployment	INT4 weight only quantisation	3-5% accuracy loss (well trained models), 52-71% memory reduction, optimal trade off
Extreme Memory Constraints	INT4 on small well trained models (Qwen2 0.5B)	0.48 GB total memory, acceptable accuracy for simple tasks
Multimodal Applications	INT4 on LLaVA style models	Perfect accuracy retention on visual tasks, 69% memory reduction
Avoid	Activation quantisation, TRIM without calibration	Catastrophic accuracy losses, no practical benefits

1. **Prioritise Training Quality:** Choose well trained, robust models even if larger. A well trained 7B model compressed to INT4 will outperform a poorly trained 1B model.
2. **Verify Baseline Performance:** Ensure the model achieves adequate accuracy on target tasks before compression. Models with poor baseline performance will only degrade further.
3. **Consider Model Architecture:** Models with efficiency features (Mistral 7B's grouped query attention, sliding window attention) maintain performance better under compression.
4. **Evaluate on Representative Tasks:** Test compressed models on tasks similar to deployment scenarios.

4.3.3 Compression Implementation Pathway

Phase 1: Quick Wins (Immediate Deployment)

- Use BitsAndBytes INT4 quantisation for immediate 50-70% memory reduction
- Deploy on existing GPU infrastructure without custom kernel development
- Accept modest latency overhead (1.8-2.2×) as acceptable cost for memory savings
- Validate accuracy on representative evaluation sets

Phase 2: Production Optimisation (3-6 months)

- Migrate to optimised inference engines (TensorRT LLM, vLLM)
- Implement native INT4 compute kernels for target hardware
- Optimise batch processing and kernel fusion
- Achieve both memory reduction and latency improvements

Phase 3: Advanced Techniques (6+ months, Optional)

- Collect task specific calibration data
- Experiment with activation quantisation using proper calibration
- Consider quantisation aware fine tuning for accuracy recovery
- Evaluate structured pruning with retraining

4.3.4 Hardware Considerations

NVIDIA GPUs: Excellent FP16 and INT8 support through Tensor Cores. BitsAndBytes works well but shows latency overhead. TensorRT LLM recommended for production. Memory bandwidth often limits performance more than compute.

Edge AI Accelerators (Jetson, Coral, NPU): Native INT8 support common, INT4 varies. May require custom quantisation schemes. Memory constraints often more severe than datacenter GPUs. Power efficiency critical.

Apple Silicon: Neural Engine provides efficient low precision compute. CoreML handles quantisation automatically. Unified memory architecture reduces transfer overhead. INT8 and INT4 well supported through Metal Performance Shaders.

4.4 Limitations and Constraints

4.4.1 Evaluation Scope Limitations

Limited Sample Sizes: 50 samples per dataset per configuration (7,500+ total inferences). Sufficient for major trends (¿10 percentage point differences), but fine grained differences may lack statistical significance. Production evaluation should use 500-1,000+ samples.

Task Diversity: Focused on multiple choice QA (HellaSwag, ARC Easy) and simple synthetic visual reasoning. Generalisation to other tasks requires validation: long form generation, code synthesis, multi turn dialogue, complex visual scene understanding, multilingual tasks.

Dataset Difficulty: The synthetic visual task proved too simple (100% baseline accuracy), providing insufficient signal. Future work should use challenging benchmarks like VQAv2 [26].

4.4.2 Hardware and Implementation Constraints

Platform Specificity: Results reflect NVIDIA A100 and L4 GPUs using BitsAndBytes. Performance characteristics differ substantially across AMD, Intel, Apple GPUs, specialised AI accelerators, mobile processors, and CPU only inference.

Framework Limitations: BitsAndBytes' design priorities result in latency overhead not representative of production inference engines.

Batch Size Restriction: All evaluation used batch size 1 to reflect edge deployment scenarios. Batch processing can substantially improve throughput and might change compression trade offs.

4.4.3 Methodological Limitations

Hyperparameter Search: Novel technique hyperparameters were set based on reasonable defaults without exhaustive search. Optimised hyperparameters might improve results, though the magnitude of failures suggests fundamental rather than parametric issues.

Single Language Evaluation: All experiments in English. Multilingual model behaviour under compression remains unexplored.

Static Evaluation: Models evaluated on fixed datasets without interactive testing. Real world scenarios may expose compression issues not visible in static benchmarks.

4.4.4 Scope Constraints

Post Training Only: The restriction to post training methods inherently limits achievable compression. Training aware techniques would likely achieve better accuracy compression trade offs but require substantially more resources [15].

No Architecture Modification: Focused on compressing existing architectures without structural changes. Techniques like knowledge distillation or neural architecture search could achieve better results but fall outside post training optimisation scope.

Model Selection: Five evaluated models represent only a small subset of available architectures. Different model families may exhibit different compression characteristics.

4.5 Future Research Directions

4.5.1 Calibration Based Compression Techniques

The consistent failure of novel techniques without calibration suggests that moderate amounts of unlabeled data could enable substantially better compression:

Minimal Calibration Requirements:

- How many calibration samples are needed for effective activation quantisation?
- Can general purpose calibration sets work across tasks?
- Does task specific calibration significantly outperform general calibration?

Calibration Efficient Methods:

- Develop token pruning using calibration set attention patterns
- Optimize outlier detection thresholds on representative activation distributions
- Learn quantisation scales from calibration forward passes

4.5.2 Optimised Inference Engine Evaluation

To understand true performance limits rather than current tool capabilities, evaluate compression techniques using production quality inference engines:

- TensorRT LLM: Measure actual latency improvements with native INT4 kernels, compare weight only vs weight activation quantisation
- vLLM: Assess continuous batching impact, evaluate attention mechanism optimisations
- Hardware Specific: Characterise compression on mobile processors, edge AI accelerators, compare energy efficiency

4.5.3 Advanced Compression Techniques

GPTQ and AWQ Quantisation: Compare gradient based quantisation (GPTQ) [27], activation aware weight quantisation (AWQ) [18], and memory efficiency across methods.

Structured Pruning: Channel pruning, attention head pruning [28], layer dropping, combination with quantisation for multiplicative compression.

Knowledge Distillation: Distillation combined with quantisation, progressive distillation through intermediate sized models, task specific distillation.

4.5.4 Multimodal Compression Deep Dive

LLaVA's extreme sensitivity to activation quantisation warrants dedicated investigation:

- Isolate vision encoder compression from language model compression
- Analyse how compression affects vision language alignment quality
- Comprehensive evaluation on VQAv2, COCO Captions [29], Visual Genome
- Test vision specific quantisation strategies

4.5.5 Theoretical Understanding of Compression

Compression Robustness Relationship:

- What training procedures produce compression robust models?
- Can robustness be predicted from training dynamics without compression?
- Relationship between generalisation and compression tolerance

Activation Distribution Analysis:

- Characterise activation distributions in well trained vs poorly trained models
- Identify architectural features promoting quantisation friendly activations
- Develop training objectives explicitly optimising for compression robustness

4.5.6 Large Scale Deployment Studies

Production Deployment: Document accuracy, latency, and resource usage in production. Characterise user experience impact. Identify deployment challenges not visible in research settings.

Long Term Stability: Monitor compressed model performance over extended periods. Detect gradual degradation. Evaluate distribution shift impact.

Cost Benefit Analysis: Quantify economic benefits. Compare accuracy degradation cost against deployment savings. Develop frameworks for compression decision making.

4.6 Broader Implications

4.6.1 Democratisation of Large Model Deployment

The demonstration that 7B parameter models can be compressed to 4-5 GB whilst maintaining 90-100% accuracy significantly lowers barriers to deploying powerful AI systems, enabling academic researchers, small organisations, privacy preserving applications, and offline edge deployments [8].

4.6.2 Emphasis on Training Quality

The finding that model quality matters more than size for compression success reinforces the importance of robust training procedures. Organisations should prioritise high quality data curation, robust training procedures, comprehensive evaluation, and model validation under compression.

The research suggests that the trend toward ever larger models may be partly driven by insufficiently robust training of smaller models [8].

4.6.3 Realistic Assessment of Novel Techniques

Honest reporting of failed novel techniques contributes to realistic assessment of what works in practice versus theory, demonstrating the gap between theoretical compression potential and practical achievable results, controlled research settings and zero shot deployment, and optimised implementations and accessible tool capabilities.

4.6.4 Open Science and Reproducibility

The commitment to transparent reporting of both successes and failures, use of publicly available resources, and detailed methodology documentation exemplifies open science principles, enabling other researchers to build upon findings without duplicating failures [8].

4.7 Concluding Remarks

This thesis provides clear, practical evidence for deploying large language models on resource constrained devices. Through evaluations across five models, six compression configurations, and three benchmarks, the key conclusions are:

- **Practical winner, INT4 with BitsAndBytes.** Simple INT4 weight quantisation (BitsAndBytes) reduces memory by approximately 50–70% with only 0–10% accuracy loss on well trained models, requiring no calibration, fine tuning, or hyperparameter search. A 7B model that needs 16 GB in full precision can be compressed to under 5 GB while maintaining about 90% accuracy, enabling consumer grade, offline, and privacy preserving deployments.
- **Theory ≠ deployability.** Techniques with strong theoretical motivation (e.g., outlier aware activation quantisation, attention based token pruning) fail in zero shot, post training settings, causing catastrophic accuracy drops (20–100 percentage points) and large latency penalties (activation quantisation increased latency by 60–223%). Combining these methods often worsened results.
- **Engineering matters more than novelty.** BitsAndBytes succeeds through implementation maturity, stable scales, robust edge case handling, and extensive validation rather than algorithmic innovation. Many promising ideas require calibration data, task tuning, and iterative engineering to become production ready.
- **Model quality is decisive.** Well trained models (e.g., Mistral 7B, LLaVA 1.5 7B) tolerate aggressive compression, poorly trained, small models collapse (e.g., TinyLlama fell to near chance performance). Practitioners should prioritise model quality before attempting compression.
- **Negative results are valuable.** Rigorously documenting failed techniques and their costs counters publication bias and saves practitioners from rediscovering dead ends.

Broader impact. This research demonstrates that deploying powerful AI on edge devices is achievable today using simple, robust techniques. The barrier is no longer algorithmic discovery but engineering implementation and honest evaluation of what actually works in practice.

Practical recommendations. For immediate deployment, use INT4 quantisation with mature libraries such as `BitsAndBytes` or `GPTQ`, accepting modest latency trade offs for significant memory savings. For production, employ optimised inference engines like `TensorRT LLM` or `vLLM` that leverage native low precision kernels. For research, prioritise calibration efficient, minimally supervised methods and systematic, cross model evaluations.

In short, the tools needed for common edge deployment scenarios already exist and their success depends on rigorous evaluation and production quality engineering, not on further theoretical complexity. Remaining challenges, such as extreme compression ratios, specialised domains, multimodal complexity, and true zero calibration solutions, should guide future work.

Bibliography

[1] OpenAI, “Chatgpt (may 2024 version).” https://chat.openai.com, 2024. Accessed May 2025.

[2] G. DeepMind, “Gemini: Google’s multi-modal large language model.” https://deepmind.google/technologies/gemini, 2024. Accessed May 2025.

[3] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[4] J. McCarthy, M. Minsky, N. Rochester, and C. E. Shannon, “A proposal for the dartmouth summer research project on artificial intelligence, 1955.” AI Magazine, 2006.

[5] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *NeurIPS*, 2017.

[8] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *arXiv preprint arXiv:1710.09282*, 2017.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] G. Team, “Gemma 3 technical report,” *arXiv preprint arXiv:2503.19786*, 2025.

[12] Gemma 3 AI, “Gemma 3 ai — the best ai multimodal model on a single gpu.” https://gemma3.ai/, 2025. Accessed: 2025-06-02.

[13] APXML, “Gpu system requirements guide for gemma 3 multimodal.” https://apxml.com/posts/gemma-3-gpu-requirements, 2025. Accessed: 2025-06-02.

[14] Novita AI, “Gemma 3 27b vram requirements explained.” https://blogs.novita.ai/gemma-3-27b-vram/, 2025. Accessed: 2025-06-02.

[15] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *CVPR*, pp. 2704–2713, 2018.

[17] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, “Learned step size quantization,” *arXiv preprint arXiv:1902.08153*, 2019.

[18] J. Lin, J. Tang, *et al.*, “Awq: Activation-aware weight quantization for llm compression and acceleration,” *arXiv preprint arXiv:2306.00978*, 2023.

[19] X. Yao, B. Deng, P. Zhang, B. Zhou, T. Yang, and Y. Zhang, “Agilequant: A zero-shot quantization framework for large language models,” in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[20] S. Huang, R. Zhang, L. Yuan, P. Gao, H. Wang, W. Lin, and Y. Xu, “Vistog: Visual token grouping for efficient vision-language pretraining and inference,” *arXiv preprint arXiv:2306.05812*, 2023.

[21] S. Garrachon, X. Li, R. Jain, J. Ngiam, and B. Mustafa, “Trim: Token reduction via importance modelling for efficient vision-language models,” *arXiv preprint arXiv:2401.01507*, 2024.

[22] X. Chen, S. Huang, J. Li, E. Xie, Z. Ding, and J. Wang, “Fastv: Fast visual token pruning for large multimodal models,” *arXiv preprint arXiv:2401.07990*, 2024.

[23] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, *et al.*, “Mistral 7b,” *arXiv preprint arXiv:2310.06825*, 2023.

[24] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual instruction tuning,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[25] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “Llm.int8(): 8-bit matrix multiplication for transformers at scale,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022.

[26] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, “Making the v in vqa matter: Elevating the role of image understanding in visual question answering,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6904–6913, 2017.

[27] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” *International Conference on Learning Representations (ICLR)*, 2023.

[28] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, “Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, 2019.

[29] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick, “Microsoft coco captions: Data collection and evaluation server,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1–10, 2015.

Glossary of Terms

Abbreviation	Full Term	Description
NA	Activation Outlier	Unusually high or low values in activation tensors, often addressed in quantisation (e.g., AWQ).
NA	AgileQuant	Automated mixed precision quantisation framework designed specifically for Large Language Models (LLMs).
AI	Artificial Intelligence	Field focused on simulating human intelligence in machines.
ANN	Artificial Neural Network	A computational model made of layers of neurons that process data.
ARC Easy	Aristo Reasoning Comprehension (Easy)	Benchmark dataset consisting of grade school level science questions for commonsense reasoning evaluation.
NA	Attention Map	Output of attention weights in transformer architectures, often visualised or pruned to identify important tokens.
AWQ	Activation Aware Weight Quantisation	A quantisation method that uses activation statistics to select the best weights for reduced precision, preserving accuracy.
BLEU	Bilingual Evaluation Understudy	A metric for evaluating the quality of machine generated text, particularly translations.
NA	Calibration	Process of adjusting scales or parameters after model training to reduce quantisation loss, typically for PTQ methods.
CIDEr	Consensus based Image Description Evaluation	A metric for evaluating image captioning quality based on consensus with human descriptions.

Table 4.2: Glossary of Terms Part 1

Abbreviation	**Full Term**	**Description**
CLIP	Contrastive Language Image Pre Training	A multimodal model trained to connect vision and language by aligning their embeddings using contrastive loss.
CNN	Convolutional Neural Network	Deep learning model for processing images through convolutional filters.
DL	Deep Learning	Subfield of ML focusing on deep neural networks with many layers.
DNN	Deep Neural Network	Neural network with multiple hidden layers used for complex data representations.
FastV	Fast Visual Token Pruning	A method for visual token pruning that efficiently identifies and removes less informative visual tokens to speed up inference.
FLOP(s)	Floating Point Operation(s)	Measure of computational complexity or cost of a model. **FLOP reduction** refers to decreasing the number of operations.
FP16	Sixteen bit Floating Point Precision	A computer number format using 16 bits to represent a floating point number, commonly used in deep learning for reduced memory/speed.
Gemma		Google's family of open sourced, lightweight, and efficient multimodal AI models.
GPT	Generative Pre trained Transformer	A family of Transformer based models for text generation, understanding, and other language tasks.
GPTQ	Gradient based Post Training Quantisation	A post training quantisation technique that uses second order gradient information (Hessian matrix approximation) for high fidelity quantisation.

Table 4.3: Glossary of Terms Part 2

Abbreviation	**Full Term**	**Description**
GQA	GQA Dataset	Visual reasoning dataset focused on compositional question answering.
HellaSwag	HellaSwag	Benchmark dataset focused on commonsense reasoning, testing a model’s ability to complete a sentence based on context.
INT4	Four bit Integer Quantisation	Representing weights and/or activations using 4 bit integers instead of higher precision floats, leading to significant compression.
INT8	Eight bit Integer Quantisation	Representing weights and/or activations using 8 bit integers, offering a balance between compression and accuracy.
NA	KV cache	A mechanism used in Transformer inference to store previously computed key and value tensors for self attention, significantly speeding up sequence generation.
NA	Layer Wise Quantisation	The approach of quantising each layer of a neural network individually rather than applying a single quantisation scheme model wide.
LLM	Large Language Model	Transformer based model trained on massive text corpora for various language tasks.
LLaMA	Large Language Model Meta AI	Transformer model family released by Meta.
LLaVA	Large Language and Vision Assistant	Multimodal model that combines language and vision capabilities.
LSQ	Learned Step Size Quantisation	A training time quantisation method that learns the optimal step sizes (scales) for quantisation during the training process.

Table 4.4: Glossary of Terms Part 3

Abbreviation	**Full Term**	**Description**
ML	Machine Learning	Subfield of AI that uses data to learn patterns without explicit rules.
NA	Mixed Precision	The technique of using different numerical bit widths (e.g., INT4 for weights, INT8 for activations, FP16 for some layers) in different parts of a model for efficiency.
MMLU	Massive Multitask Language Understanding	Benchmark to evaluate LLM performance across many academic subjects.
MMBench	Multi Modal Benchmark	Evaluation benchmark for multimodal reasoning in LLMs.
MS COCO	Microsoft Common Objects in Context	Dataset used for image captioning and object detection tasks.
NA	Parameter Redundancy	The concept that large neural networks contain more parameters than necessary to achieve high performance, justifying compression techniques like quantisation and pruning.
PTQ	Post Training Quantisation	The process of applying quantisation (reducing precision) to a pre trained model **after** training, without further training or fine tuning.
QAT	Quantisation Aware Training	Simulating the effects of quantisation (e.g., using fake quantization nodes) during model training to improve final accuracy after quantisation.
Qwen2	**NA**	A family of large language models developed by Alibaba Cloud, known for strong performance and efficient architecture.
RNN	Recurrent Neural Network	Neural network architecture designed for sequence modelling using internal memory (hidden state).

Table 4.5: Glossary of Terms Part 4

Abbreviation	**Full Term**	**Description**
SVD	Singular Value Decomposition	A matrix factorisation technique used for compressing neural networks by decomposing weight matrices.
NA	TensorRT LLM	NVIDIA's highly optimised library designed to accelerate inference performance for Large Language Models (LLMs) on NVIDIA GPUs.
NA	**Token Pruning**	The process of removing less informative or redundant tokens from an input sequence to compress the computation of attention and MLMs, improving speed.
NA	Transformer	A neural network architecture relying on the **self attention** mechanism to weigh the importance of different parts of the input data, highly effective for sequences.
TRIM	Token Reduction via Importance Modelling	A token pruning method for compressing large multimodal models by identifying and removing tokens based on an importance score.
vLLM		A fast and efficient open source inference engine designed specifically for serving Large Language Models (LLMs) with high throughput.
ViT	Vision Transformer	Transformer model designed for image data using patch tokenisation.
VisToG	Visual Token Grouping	A method for merging adjacent or similar visual tokens into groups to reduce the total number of tokens processed, thereby compressing computation.
VQA	Visual Question Answering	Task requiring models to answer questions based on an image and a natural language query.
VQAv2	Visual Question Answering Version 2	An extensive dataset for evaluating image based question answering systems, an updated and more balanced version of VQA.

Table 4.6: Glossary of Terms Part 5

Appendix

Code Snippet 1: FP16 Baseline Loading

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    device_map="auto",
    trust_remote_code=True
)

tokenizer = AutoTokenizer.from_pretrained(
    model_id,
    trust_remote_code=True
)

# Set padding token if not defined
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
```

Code Snippet 2: INT8 Quantised Loading

```
from transformers import AutoModelForCausalLM, BitsAndBytesConfig

 quantisation_config = BitsAndBytesConfig(
    load_in_8bit=True,
    llm_int8_threshold=6.0
)

model = AutoModelForCausalLM.from_pretrained(
    model_id,
     quantisation_config= quantisation_config,
    device_map="auto",
    trust_remote_code=True
)
```

Code Snippet 3: INT4 Quantised Loading

```
from transformers import AutoModelForCausalLM, BitsAndBytesConfig
import torch

 quantisation_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4"
)

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    device_map="auto",
     quantisation_config= quantisation_config,
    trust_remote_code=True
)
```

Code Snippet 4: Generation Parameters

```
generation_config = {
    "max_new_tokens": 10,
    "do_sample": False,  # Greedy decoding
    "pad_token_id": tokenizer.pad_token_id,
    "eos_token_id": tokenizer.eos_token_id,
}

# Example inference
inputs = tokenizer(prompt, return_tensors="pt").to(device)
outputs = model.generate(**inputs, **generation_config)
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Code Snippet 5: Warm up Protocol

```
import torch

# Warm up runs (3 iterations)
warmup_sample = dataset[0]
warmup_prompt = format_prompt(warmup_sample)

for _ in range(3):
    inputs = tokenizer(warmup_prompt, return_tensors="pt").to(device)
    with torch.no_grad():
        outputs = model.generate(**inputs, **generation_config)
    torch.cuda.synchronize()  # Ensure completion
```

```
# Clear any cached tensors
torch.cuda.empty_cache()
```

Code Snippet 6: Complete Evaluation Workflow

```
import time
import torch
from tqdm import tqdm

def evaluate_model(model, tokenizer, dataset, config):
    results = {
        'accuracy': [],
        'latency': [],
        'memory_peak': []
    }

    # Reset memory statistics
    torch.cuda.reset_peak_memory_stats()

    # Warm up
    warmup_sample = dataset[0]
    warmup_prompt = format_prompt(warmup_sample)
    for _ in range(3):
        inputs = tokenizer(warmup_prompt, return_tensors="pt").to(device)
        with torch.no_grad():
            outputs = model.generate(**inputs, **config)
        torch.cuda.synchronize()

    # Evaluation loop
    correct = 0
    total = 0

    for sample in tqdm(dataset):
        prompt = format_prompt(sample)
        inputs = tokenizer(prompt, return_tensors="pt").to(device)

        # Measure latency
        torch.cuda.synchronize()
        start_time = time.time()

        with torch.no_grad():
            outputs = model.generate(**inputs, **config)

        torch.cuda.synchronize()
        end_time = time.time()

        latency = (end_time   start_time) * 1000  # Convert to ms
        results['latency'].append(latency)
        # Extract prediction
        generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
        prediction = extract_answer(generated_text)

        # Check correctness
        if prediction == sample['answer']:
            correct += 1
        total += 1

        # Track peak memory
        memory_peak = torch.cuda.max_memory_allocated() / 1e9  # GB
        results['memory_peak'].append(memory_peak)

    # Calculate accuracy
    accuracy = (correct / total) * 100
    results['accuracy'] = accuracy
    results['mean_latency'] = sum(results['latency']) / len(results['latency'])
    results['mean_memory'] = sum(results['memory_peak']) / \
                             len(results['memory_peak'])

    return results
```

Code Snippet 7: Answer Extraction

```
import re

def extract_answer(generated_text):
    """
    Extract answer letter (A, B, C, D) from generated text.
    Returns None if no valid answer found.
    """
    text_snippet = generated_text[:50].upper()
    pattern = r'\b([ABCD])\b'
    match = re.search(pattern, text_snippet)
    if match:
        return match.group(1)
    return None

def format_prompt(sample):
    """
    Format multiple choice question into prompt.
    """
    question = sample['question']
    choices = sample['choices']
    prompt = f"Question: {question}\n\nChoices:\n"
    for i, choice in enumerate(choices):
        letter = chr(65 + i)
        prompt += f"{letter}. {choice}\n"
    prompt += "\nAnswer with only the letter (A, B, C, or D): "
    return prompt
```

Code Snippet 8: Model Cleanup

```
import gc
import torch

def cleanup_model(model):
    """Properly clean up model and free GPU memory."""
    del model
    gc.collect()
    torch.cuda.empty_cache()
    torch.cuda.reset_peak_memory_stats()
    torch.cuda.synchronize()
```

Code Snippet 9: Complete Environment Configuration

```
import os
import torch
import numpy as np
import random

SEED = 42

def set_seed(seed):
    """Set all random seeds for reproducibility."""
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

os.environ['PYTORCH_CUDA_ALLOC_CONF'] = 'expandable_segments:True'
os.environ['TOKENIZERS_PARALLELISM'] = 'false'
set_seed(SEED)

if not torch.cuda.is_available():
    raise RuntimeError("CUDA is not available. GPU required for experiments.")

device = torch.device("cuda")
print(f"Using device: {device}")
print(f"GPU: {torch.cuda.get_device_name(0)}")
print(f"VRAM: {torch.cuda.get_device_properties(0).total_memory / 1e9:.1f} GB")
```

Code Snippet 10: Experimental Results Logging

```
import json
from datetime import datetime

def log_results(model_name, config_name, results, output_dir):
    """Log experimental results to JSON file."""
    log_entry = {
        'timestamp': datetime.now().isoformat(),
        'model': model_name,
        'configuration': config_name,
        'accuracy': results['accuracy'],
        'mean_latency_ms': results['mean_latency'],
        'mean_memory_gb': results['mean_memory'],
        'latency_std': np.std(results['latency']),
        'num_samples': len(results['latency']),
        'gpu_name': torch.cuda.get_device_name(0),
    }

    output_file = f"{output_dir}/{model_name}_{config_name}_results.json"
    with open(output_file, 'w') as f:
        json.dump(log_entry, f, indent=2)
    print(f"Results saved to {output_file}")
    return log_entry
```