State-Space vs. Transformer: A Comparative Analysis of Architectures and Optimization Techniques on Mathematical Reasoning Benchmark

by

Shehran Rahman
24241196
Sameer Khairul
22101938
Protaya Roy
24241160
Shamsan Nasir
22101481
Ahnaf Akif
23241138

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
BRAC University
October 2025.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at BRAC University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Shehran Rahman	Sameer Khairul
24241196	22101938
Protaya Roy	Shamsan Nasir
24241160	22101481

Ahnaf Akif

23241138

Approval

The thesis/project titled "State-Space vs. Transformer: A Comparative Analysis of Architectures and Optimization Techniques on Mathematical Reasoning Benchmark" submitted by

1. Shehran Rahman (24241196)
2. Sameer Khairul (22101938)
3. Protaya Roy (24241160)
4. Shamsan Nasir (22101481)
5. Ahnaf Akif (23241138)

of Summer 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering in 2025 Year.

Examining Committee:

Supervisor:
(Member)

Dr. Farig Yousuf Sadeque

Associate Professor
Department of Computer Science and Engineering
BRAC University

Co-Supervisor:
(Member)

Ariyan Hossain

Lecturer
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD

Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Abstract

Large Language Models (LLMs) are incredible and sophisticated models that have remarkable capabilities that towers over a wide range of language processing tasks. However, that capability comes with a substantial computational and memory cost which is often too great. This is why it is harder for the general population to access this technology. This paper explores other alternative architectures such as State-Space models in order to understand which models are lightweight but also retain strong performance. The aim is to compare the SSM based Llamba model family performances on mathematical reasoning tasks which remains unexplored. A deep investigation into the accuracy of these models is performed along with other computational metrics such as decode speed, memory usage etc. An exploration has been made on how different architectural choices impact both efficiency and effectiveness of these models. The results provide a clear picture of the existing landscape that Transformers are the obvious architectural choice for the best reasoning accuracy while SSMs can be the viable and efficient alternative in the situations when the computational resources are the main constraint.

Keywords: Large Language Models (LLMs), Knowledge Distillation, LLAMBA, LLAMA, MOHAWK, Mamba, Transformers.

Acknowledgement

Completing this thesis would not have been possible without the support, guidance, and encouragement of many individuals and BRAC University. We extend our deepest gratitude to those who have contributed to this journey.

First and foremost, we are profoundly grateful to our thesis supervisor Dr. Farig Yousuf Sadeque for his extraordinary mentorship, insightful critiques, and steadfast belief in us. His expertise in Natural Language Processing and deep patience in guiding us through the challenges we faced have been invaluable to shaping this research.

We acknowledge the support of BRAC University for providing us with the computational resources, workflow and infrastructure that are critical to this work.

Our sincere gratitude towards our friends and family for their constant support, endless patience, encouragement and understanding during the long hours which motivated us and kept us going.

We dedicate this work to those who believe in us and in the power of technology to bridge divides. May this thesis inspire innovations and make Large Language Models accessible to all.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The swift evolution of large language models like GPT-4, PaLM, and LLaMA has transformed natural language processing (NLP), enabling remarkable NLP tasks such as text generation, translation, summarization, and reasoning capabilities better than ever. They are typically trained on trillions of tokens and billions of parameters, and attain excellent performance using extensive computational resources. However, their computational and memory demands make them harder for the general masses to use them easily. While the prevalent approach is a cloud-based inference model architecture, issues such as latency, privacy risks, and reliance on network connectivity exists which could be mitigated if a way is found to minimize computational and memory demands.

To make using LLM much easier, it is needed to make sure it has low-latency processing, data privacy, and efficiency in energy consumption. This interest in optimizing LLMs for easier use is being explored more intensively than ever. Usually, traditional LLMs are trained using high-performance GPUs and TPUs and require high computational resources. For this very reason, investigations into model compression techniques like quantization, pruning, knowledge distillation and architectural reconfigurations to reconcile computational needs have started.

Recently, alternative architectures are explored to be used in LLMs instead of the pre-dominant transformer architecture. One such alternative architecture known as state-space models (SSMs) have shown much promise in efficient performance. Although it uses much less computational resources compared to transformer architecture based models, it still sometimes struggles to keep up with transformers in terms of accuracy. However, since in some tasks, highest accuracy might not the most desired outcome SSMs may be considered to be used with the aim of having an efficient LLM to work with. While the transformer-based models are held to be the standard model for reasoning intensive tasks, since performance of SSM on mathematical reasoning is yet to be explored, this study will explore the computational tradeoffs between SSMs and transformers in mathematical reasoning benchmarks.

This thesis aims to study the tradeoffs between SSMs and transformers in mathematical reasoning benchmarks to understand if SSMs have a foothold in this area

over the transformers. The ability of the SSMs are studied not only on accuracy alone, but also on latency, memory footprint and overall scalability in mathematical reasoning tasks such as GSM8K. By directly assessing the results, it could help us decide if SSMs can prove themselves enough to replace transformers in this unexplored field.

1.2 Rational of the Study or Motivation

Large Language Models (LLMs) have pushed the boundaries of Natural Language Processing (NLP) enabling many powerful applications. However, these models require immense computational resources which makes it a resource-demanding tool and thus makes it heavily reliant on cloud based execution for ease of access. Considering the increasing reliance on AI for solutions to such as performing real-time tasks like translation, autocomplete, speech recognition, contextual recommendations etc. the large resource demand of these LLMs will make it much harder to work with and to expand it to other areas. This also creates a significant barrier for the common, general masses to utilize this game changing tool effectively and affordably. The existing compressed models often suffer from significant trade-offs by compromising accuracy or demanding resources beyond what common users could afford.

Current efforts to reduce the computational demand of the LLMs, some of which are by applying techniques such as quantization, pruning, knowledge distillation, etc. have provided partial solutions. Even though they may improve efficiency of the LLMs, improving the architecture or even studying other alternative architectures could help a long way in optimizing since such techniques could be applied on them for further optimization.

One of such alternative architectures being state-space models (SSMs) offers new paths for a more optimized LLMs. Unlike transformer architecture which currently dominates the LLM field, it promises a linear-time sequence modeling, lower memory footprints and potentially more improvement in latency characteristics and token generation speed. But their effectiveness in mathematical reasoning remains unexplored.

The implications of this research extend beyond understanding LLM effciency. By comparing tradeoffs between state-space models and transformer models, we can decide which one provides more efficiency in terms of mathematical reasoning. The state-of-the-art full-sized LLMs based on the cloud infrastructure consumes huge amounts of energy leading to more carbon footprint. The optimizations for a resource efficient LLM will pave the way for more energy-efficient models by reducing overall power-consumption and making on-device deployment more environmentally sustainable. Therefore, in consideration of the above mentioned premises, this research aims to contribute to the broader goal of making AI more secure, efficient, and accessible for the general population.

1.3 Problem Statement

The transformer architecture can provide state-of-the-art performance across different language processing and reasoning tasks. However, this performance comes at a cost of high memory, power demands, and comparatively high inference time. The State Space Model (SSM) presents an alternative architecture that is more lightweight and offers an improved inference time. But there is a limited systematic analysis of the tradeoffs that come with this more lightweight architecture, especially in the context of mathematical reasoning tasks. This paper tackles this by providing a comparative study of Transformers and SSM models in order to determine if using an SSM model can provide a feasible, lightweight alternative with improved inference time for mathematical reasoning. This paper also explores model compression techniques to observe the feasibility of implementing them to get practical performance from models at a smaller scale. Additionally, the paper also investigates application of optimized prompting techniques like structured Chain of Thought (CoT) prompting and whether it provides any substantial benefits to the models used in the paper for mathematical reasoning tasks. Through this, we aim to bridge the gap between resource-intensive traditional state-of-the-art LLMs and comparatively lightweight models, enabling the democratization of LLMs. By doing so, we want to enhance usability, improve efficiency, and unlock new possibilities for LLM-driven assistance in everyday tasks.

1.4 Objective

- Explore and analyze the trade-offs between inference time, accuracy, and computational efficiency in the context of mathematical reasoning. (transformer-based vs. state-space models).
- Investigate architectural optimizations by assessing the impact of varying model complexities (transformer-based vs. state-space models).
- Explore the potential of knowledge distillation from transformer-based models to State-Space Models (SSMs, e.g., Mamba) for mathematical reasoning tasks (GSM8K)
- Analyze the effectiveness of structured Chain of Thought (COT) prompting to improve the performance of the distilled model in the context of mathematical reasoning tasks.
- Provide actionable insights on scenarios where distilled SSM may outperform Transformer-based models.
- Navigate the feasibility of deploying smaller SSM-based large language models (LLMs) over transformer-based models challenged by limited resources regarding computation, memory, and power efficiency.
- Inspect the feasibility and practicality of using compression techniques to provide performance with less memory overhead.

1.5 Methodology in Brief

Initially, after gathering team members we selected LLMs as our domain, then chose our supervisor and selected a topic under his guidance. Many papers were reviewed related to that topic and with that knowledge, several LLM architectures were studied. One was selected and many experimentations were performed on that architecture's based model. Finally, the results obtained were represented in the final paper and in the thesis defense.

1.6 Scopes and Challenges

This paper aims to form a comparative analysis of two different LLM architectures, namely, Transformers and State Space Models (SSMs), while focusing mainly on inference time in the context of mathematical tasks over other metrics. For calculating the inference time, we use a number of performance metrics like decode time, time to first token, Prefill, model size, and memory size to get a clear picture of the overall inference time of the model. Additionally, we also compare the performance of the different architectures to investigate any performance vs inference time tradeoffs in order to verify whether it is viable to use the SSM architecture when inference time is a priority, particularly for use in resource-constrained environments. Furthermore, the paper also investigates the application of optimized prompting techniques like structured Chain of Thought (CoT) prompting and whether it provides any substantial benefits to the models used in the paper for mathematical reasoning tasks.

However, the research and implementation of the topic do pose quite a few challenges. Firstly, the topic of LLMs in general is a fast-moving and dynamic research field, and staying up to date on all the new developments is a challenge in itself. Furthermore, this can also affect the longevity of any conclusion we arrive at as newer optimizations may easily emerge that change the field completely. Additionally, there are challenges regarding the availability of the required hardware for research in the field, as most of the best LLMs use up huge amounts of memory and computational power. We also have to find the right balance between performance and memory usage, which can prove to be troublesome and take rigorous experimentation. Not only that, but the data we gain can easily be faulty if the evaluation pipeline is handled without proper expertise. These types of issues can easily create inaccuracies in the data and alter the conclusion if not handled with tact. The data needs to be properly and expertly contextualized to provide the clearest ideas of any tradeoffs and prevent potential misunderstanding. Furthermore, trying to ensure that a model is able to adapt and run efficiently on different devices with different architectures adds another layer of complexity to the research.

Chapter 2

Literature Review

2.1 Preliminaries

This paper provides a comparative study between models of different core architectures like Transformer and SSM (Mamba). Further, some of the principle model optimization techniques have been studied, analyzing the trade-offs they bring along as well. To explore ways for optimizing LLMs to run on edge devices we have researched many prominent and existing optimization methods. These state-of-the-art methods like Quantization and Knowledge Distillation. Here are some preliminaries which will assist in proper understanding of the intricate comparative analysis that has been shown in this study.

Aviv Bick et al. (2024) [30] introduce MOHAWK, a novel distillation framework that has been responsible for attracting attention to subquadratic alternatives such as state space models (SSMs). Since these architectures often lack the performance of well-trained Transformer models, this framework allows the SSM to receive a substantial amount of knowledge from well-performing Transformer-based models, helping it to mitigate that shortcoming. This framework enables the SSM-based student model to perform well only on a fraction of the original training data. The methods to perform this distillation include matrix orientation, hidden-state alignment, weight transfer, and knowledge distillation. This staged approach enables the student to capture both low-level structural properties and high-level functional behaviors of the teacher, making the distillation process both stable and data-efficient. By applying MOHAWK, the authors distilled Phi-1.5 (1.3B parameters) into Phi-Mamba, a modified Mamba-2 architecture. After that, Phi Mamba was able to achieve an average accuracy of 62.6% on commonsense reasoning benchmarks using only 3 billion tokens, which is less than 1% of the data used to train comparable SSMs. This outperformed all prior open source subquadratic models of similar scale. The performance gap closed further through Hybrid Phi Mamba, a hybrid variant that retains four attention layers and reached 66.0% average accuracy, which was below the teacher model accuracy by just 1.2%. Each MOHAWK stage proved to be important through ablation studies, as models trained using the full three-stage process performed significantly better than those that utilized only one or two stages. The authors also demonstrated that the Mamba-2 SSM is highly expressive, closely approximating self-attention matrices and enabling effective replacement of attention layers without compromising model coherence, especially when MLP weights

are frozen during distillation. Through this, the authors demonstrated the ability of MOHAWK to allow efficient architectures to inherit capabilities developed through extensive computing and data without training from scratch if they leveraged pre-trained transformer models as teachers.

Aviv Bick et al. (2024) [29] introduce Llamba, a model based on the Mamba architecture, a family of recurrent language models that showcases performance-efficiency through cross-architecture distillation. Their work is based on the state-space models, such as Mamba, to showcase how tasks can be performed in sub-quadratic complexity using minimal data. The Llamba is distilled from Llama-3 models into a Mamba-2-based recurrent architecture. Using the MOHAWK framework, after going through three stages: matrix orientation, weight transfer with knowledge distillation and hidden state alignment, the authors were able to perform successful cross-architecture distillation. This was later tested to find that the Llamba-8B achieves competitive performance using only 12 billion tokens, which was less than 0.1% of the data required to train its teacher, showcasing a data-efficient model development. By alternating Mamba-2 layers with gated MLPs, a multi-head variant of Mamba-2, and the removal of certain non-linearities to facilitate distillation, they were able to implement this Llamba variant. These changes enhance training and inference efficiency by reducing temporal mixing layers and kernel overhead. Results showed that Llamba models significantly outperformed comparable hybrid and recurrent models in terms of throughput and memory efficiency. At the same time, they (Llamba models) were able to match or even exceed the performance of some of the same comparable models across common benchmarks like ARC, HellaSwag, MMLU, etc. Additionally, Llamba models were also able to maintain near to constant memory usage while doing inference. Furthermore, the authors also provided optimized implementations for Apple Silicon via MLX, including 4-bit quantization, which further highlighted its practicality for edge applications.

Knowledge distillation [1] is a process that allows the transfer of knowledge from a large and complex model to a small and compact model. While large and cumbersome models are good at generalization but they require extensive resource and time to train. Distillation makes this process significantly easier as training the smaller model through distillation produces better test outcomes than training it on the same training set as the large model. It is particularly important as it reduces computational cost while transferring the generalization ability of the large model to the smaller model. This process works by using soft targets produced by the large model to train the small model, which is more beneficial than training using hard labels as soft targets contain more information about class relationships which may be ignored in the hard labels. These soft targets are generated from the output logits of the large model using a temperature based Softmax function set at high temperature. During training, a weighted average of two objective functions are used. One objective function is a cross entropy applied on the soft targets and the other is cross entropy applied on correct labels. Best outcome is achieved when more weight is applied on the first objective function than the second. In a special case of distillation, the matching logits are used where the logits are zero-meaned for each transfer case. Here less attention is paid to negative than average matching logits which could introduce noise due to them being unconstrained by the cost function.

As a whole, Knowledge distillation allows the smaller model to inherit the essence of the larger model without extensive training and large quantity of data to train it.

Transformer is a neural network based architecture that fundamentally changed how Artificial Intelligence is seen nowadays. This concept introduced in the paper "ATTENTION IS ALL YOU NEED" in 2017 [7] has since become the standard architecture for deep learning models to work with. This architecture is now commonly used in the world, more specifically it powers some of the popular text generation models available today such as Google's Gemini, OpenAI's GPT and Meta's Llama, which is the focus of this study. Beyond text generation, this architecture has shown promising results in other areas as well such as audio generation, image recognition, video generation, etc. Even though this architecture has pronounced performance, it comes at a huge computational cost, especially when it comes to resource-intensive tasks mentioned above.

Quantization [18] is a crucial strategy to address the immense computational and memory demands of LLMs to operate on resource constrained environments. For instance, the GPT-3 model hosts 175 billion parameters and requires a staggering storage of 350 GB and thus underlines the necessity for such optimization techniques. Quantization primarily refers to the conversion of continuous numerical data into discrete levels for reduced precision representations. However, in the context of deep learning, it refers to the conversion of 32-bit floating-point (FP32) representations to lower precision formats like 16-bit floating point (FP16) or into integer formats like 8-bit (INT8) or even 4-bit integers. Even though this helps in accelerating the matrix multiplication and increasing cache utilizations, it comes at the cost of possible introduction of errors and degradation in accuracy. Modern approaches leverage early statistical theories like Shappard's rounding analysis and Shannon entropy to design methods like affine and scale quantizations. Affine quantization uses a scaling factor and zero-point offset to map weights from high to low precision and on the other hand scale quantization applies a simpler proportional scaling transformation. Another factor to consider for quantization is granularity. Per-layer quantization assigns uniform parameters within a neural network layer but increases the possibility of errors whereas per-channel quantization assigns parameters to individual filters which yields higher accuracy but at the cost of additional computation. Calibration techniques are used to compute optimal scaling factors for quantization. Each technique focuses on a certain factor, for instance, global calibration focuses on the entire range, max calibration on the maximum value of unquantized data and percentile calibration focuses on specified percentiles to mitigate outliers or sparsity. Kullback-Leibler is a more advanced calibration technique which compares the probability distributions of the quantized and unquantized data. Based on requirements and scenarios, LLMs adopt different quantization approaches of which two primary methods are Post-Training Quantization (PTQ) and Quantization Aware Training (QAT). PTQ is applied to a pre-trained model which helps in shrinking it by a certain factor but at a risk of accuracy loss. Conversely, QAT integrates quantization during the training process which makes the model aware of the loss and thus provides a better environment to adapt to a lower precision allowing it to retain higher accuracy at the cost of increased training resources.

The Mamba architecture [13], also based on neural networks, is an alternative architecture for deep learning models. This architecture seems to achieve remarkable computational efficiency through a process which lets it select which unimportant states to compress and also because of its linear-time processing. State-space model (SSM), a component of Mamba architecture, helps process sequences through a recurrent state update mechanism that keeps the size of the hidden state fixed, regardless of how long or short the sequence length is. This helps the architecture to efficiently handle the tasks, which in turn significantly reduces memory and computational overhead, which generally are culprits for models requiring huge amounts of resources to work. These qualities make it a perfect candidate to process complex tasks efficiently, some of which are the one this study explores which is mathematical reasoning tasks.

2.2 Review of Existing Research

Tan et al. [24] proposed MobileQuant, a post-training quantization (PTQ) framework, to address the challenge of deploying large language models (LLMs) on mobile devices. The authors were motivated by the failure of existing quantization methods to take advantage of mobile hardwares like Neural Processing Units (NPUs) or Digital Signal Processors (DSPs). They also focused on the difficulty of quantization for activations other than 16-bits as it either led to computation overhead or significant accuracy drop. To address these issues, MobileQuant leverages and extends previous research on weight equivalent transformation techniques in order to make the quantization of weights and activations easier by using scaling factors to balance their ranges. While these techniques are used in state-of-the-art approaches like SmoothQuant and OmniQuant, they exhibit their full potential in hardwares based on GPU and do not work properly out of the box for edge devices. Therefore, this framework employs a joint optimization approach which optimizes weight transformation and activation range parameters in an end-to-end manner allowing it to scale with larger datasets and calibration samples. Another key innovation was the activation range learning (ARL) which allowed MobileQuant to learn the optimal quantization range for activations while maintaining post-quantization accuracy. A subset of the Pile dataset has been used as calibration set and they explored two quantization settings W8A8 and W4A8 where per-channel quantization used for weights and per-tensor quantization used for activations (except for non-linear operators). The authors evaluated MobileQuant on lightweight LLMs like TinyLlaMA-1.1B, StableLM-2-1.6B and Gemma-2B and the models were tested on a Samsung Galaxy S24 with Qualcomm Snapdragon 8 Gen 3 processor. WikiText, LAMBADA datasets were used for conducting evaluation and for common sense reasoning and language modeling tasks the Language Model Evaluation Harness benchmark was used. For on-device deployment, MobileQuant outperforms both SmoothQuant and Omniquant as the former variant uses static per-tensor quantization (more compatible with mobile hardware) compared to the dynamic ones. The authors introduced mobile-friendly variants termed SmoothQuant-Edge and OmniQuant-Edge which provided stronger baselines than the stock versions. They also applied the ARL to OmniQuant (learning based approach outperforms search based SmoothQuant) and they showed consistent performance improvements in all quantization settings and

this baseline was also outperformed by MobileQuant. However, this study focuses on LLMs with 1 to 2 billion parameters and the quantized models inherit the limitations of the original pre-trained models.

Ma et al. [19] introduces BitNet b1.58, a novel 1-bit LLM variant of the existing BitNet architecture. The authors were driven by the often suboptimal performances of existing approaches like post-training quantization. Moreover, full precision (FP16 or BF16) Transformer LLMs require expensive floating point multiplication and additions operations whereas BitNet requires integer-based addition operations during matrix multiplications which significantly reduces computations. Building on this, the authors proposed the new variant BitNet b1.58 where each parameter is ternary, having values of {-1, 0, 1}, theoretically using 1.58 bits per parameter [log2(3) = 1.58]. While this model retains all the benefits of its predecessor, it introduces some great modeling advantages like explicit feature filtering owing to the inclusion of zero in the model weight values whereas the former had only values {-1, 1}. The architecture of the BitNet is based on the Transformer model and has adopted LlaMA-alike components such as RMSNorm, SwiGLU, rotary embeddings (excludes all biases) which makes it more compatible with popular open source frameworks. The model is trained from ground-up using "1.58-bit" weights and 8-bit activations with a quantization function for weights which they call "absmean", in order to constrain the weights to {-1, 0, +1} after scaling the weight matrix by average absolute value. They have used the same quantization function for activation as in the original BitNet with some tweaks in scaling before non-linear functions. The authors pre-trained the models on the RedPajama dataset for 100 billion tokens and used language tasks like ARC-Easy, ARC-Challenge, Hellaswag etc. as well as validation perplexity on WikiText2 and C4 datasets. The results tilt in favor of the BitNet with larger models as the 3.9B b1.58 model outperforms the 3B LlaMA model in both perplexity and end-task performance while being 2.4 times faster and consumes 3.32 times lesser memory. This model saves arithmetic operations by 71.4 times for matrix multiplication and achieves better throughput by supporting 11 times the batch size of LlaMA LLM. The authors have encouraged further research for native support for longer context lengths and deployment on edge and mobile devices due the reduced memory footprint. Moreover, the authors call for the innovation of newer hardware specifically optimized for 1-bit LLMs to leverage the new computation paradigm introduced by BitNet b1.58.

Yang et al. [8] addressed the computational and memory challenges of the training phase of transformer architecture-based LLMs through the process of Dynamic Stashing Quantization (DSQ). The authors propose this approach to handle the memory-bound nature of the training phase which is constrained by high volumes of data transfers between forward and backward passes. This process employs the block floating point quantization method in order to achieve dynamic precision throughout the training phase. The precision of quantization does not remain constant rather it changes as the training phase progresses. The paper identified 4 quantization opportunities in the training phase centered around the "stashing" of intermediate values in the DRAM during the forward and backward passes. DSQ ensured all the GEMM inputs were heavily quantized which helped to save substantial DRAM bandwidth. The paper also used a time-adaptive quantization strategy to ensure

different quantization levels for each round of training. The quantization precision starts low and as the training progresses, the precision gradually increases ensuring that memory traffic is reduced and model performance is minimally impacted. To evaluate this approach, the paper uses two translation tasks and two tasks from the GLUE benchmarks. 4 datasets were used for this task (IWSLT2017, WMT14, MNLI, and QNLI). A standard 6-layer transformer model and the RoBERTa-base model were used for the evaluation. The results demonstrated that DSQ was able to get a 20.95x reduction in arithmetic complexity and a 2.55x decrease in DRAM memory traffic compared to the standard 16-bit fixed point training. Similarly, it was able to outperform other stashing and quantization methods. Moreover, DSQ achieved these impressive reductions without compromising its performance noticeably. For instance, DSQ only showed a 0.41 score drop compared to the floating point baseline on the IWSLT2017 DE-EN dataset. This demonstrates the ability of DSQ to maintain comparable accuracy to the baseline while having significantly lower arithmetic complexity and memory traffic.

Dettmers et al. [6] offer a memory-efficient finetuning strategy that, for the first time, allows finetuning of a 65B parameter model on a single consumer level GPU while maintaining performance relative to a complete finetuning baseline. This study demonstrates how QLoRA's internal mechanism backpropagates gradients into Low-Rank Adapters via a frozen, four-bit quantized pretrained language model. It demonstrates that it is possible to fine-tune a 4-bit model without sacrificing performance. QLoRA saves memory without sacrificing performance thanks to a variety of improvements, including NF4, a 4-bit NormalFloat, a novel data type that is information theoretically optimized for normally distributed weights. It is a datatype that improves on Quantile Quantization, ensuring that each quantization bin receives an equal number of values from the input tensor. The second technique is Double Quantization, which quantizes the quantization constants and helps to further minimize the memory footprint. This quantization lowers the memory footprint per parameter by an average of 0.373 bits for a blocksize of 64, from $32/64 = 0.5$ bits to $8/64 + 32 / (64 * 256) = 0.127$ bits. Moreover, it controls memory spikes by using paged optimizers which is a NVIDIA unified memory function automatically switching pages between the CPU and GPU when the GPU periodically runs out of memory to ensure error-free GPU processing. On the Vicuna test, the authors demonstrate that their best model family, Guanaco, surpasses all previously publicly available models, achieving 99.3% of chatGPT's performance level after just 24 hours of fine-tuning on a single GPU. Extensive experiments comparing BF16, Int8, FP4, NF4 on GLUE and Super-NaturalInstructions reveal that QLoRA replicates 16-bit LoRA and full finetuning, indicating that adapter finetuning after quantization can fully restore the performance lost due to imprecise quantization. The authors highlight the limitations of their research by stating that due to enormous resource costs, they were unable to demonstrate that QLoRA can equal complete 16-bit finetuning performance at 33B and 65B scales. They also point out that their assessment might not apply to other benchmarks that they haven't thought to test. Furthermore, they did not assess alternative bit-precisions, such as 3-bit base models or alternative adapter techniques, which might potentially produce identical outcomes to a 16-bit complete finetuning performance following quantization. All things considered, the authors think that QLoRA creates an equalizing factor that

aids in bridging the resource gap between big businesses and small groups using only consumer-level GPUs.

Hayou et al. [15] proposes LoRA+ in an effort to address the suboptimality of the original Low-Rank Adaptation approach particularly emphasizing on the models with sufficiently large embedding dimensions. Setting different learning rates for the LoRA adapter matrices with a carefully chosen fixed ratio effectively improves performance and speed of the finetuning process. The authors of this paper hypothesize that suboptimal finetuning of models is caused by adapter matrices updated with the same learning rate. In that effort, LoRA+ introduces asymmetric learning rates, setting with the ratio where fixed and tuning which is determined through scaling laws derived from the infinite-width limit of neural networks. This ensures that the updates to A and B complement each other leading to more improved efficiency and stability in finetuning. The authors demonstrate through extensive theoretical analysis and experiments across various benchmark such as MNLI, QQP and SST2 using models like RoBERTa and GPT-2 that there is a 1-2% performance increase over standard LoRA while also achieving better finetuning speeds up to 2x times speed at the same computational costs as regular LoRA. The approach's drawback is that, as is to be expected, the architecture and the fine-tuning tasks rely on the optimal ratio through the constants in Θ and the asymptotic results provide no information whatsoever about the influence of the task and the neural architecture on the constants. A more accurate estimate of the ideal ratio will have to wait until future research because the optimal ratio is still sensitive to model, task, and initiation and exhibits substantial volatility, which the paper is unable to generalize. In summary, the suggested approach LoRA+ eliminates suboptimal finetuning and demonstrates that the advantages are greater for "hard" tasks like MNLI for RoBERTa/GPT-2 as opposed to SST2 and MMLU for LLama-7b as opposed to MNLI.

Du et al. [11] proposes a framework to enhance the performance of Large Language Models (LLMs) at very low precisions. Named BitDistiller, the framework combines Quantization Aware Training (QAT) and Knowledge Distillation (KD) in a way that not only preserves the weight fidelity but also minimizes performance reduction. These are achieved by combining a quantization technique namely Asymmetric Quantization along with Asymmetric Clipping and a newly proposed distillation technique called Confidence-Aware Kullback-Leibler Divergence (CAKLD). The asymmetric clipping is used before quantization to handle outliers in the full precision weight and asymmetric quantization is used to handle asymmetricity introduced as a result of smaller group sizes especially in low-bit configurations. Meanwhile, CAKLD which runs in a self-distillation manner, a hybrid of Forward and Reverse KLD which can shift between mode-covering and mode-seeking based on the confidence of the full-precision model. The authors used AFPQ method for NF format (NF3) above 2-bit level and single-scale asymmetric method for INT format (INT2) at 2-bit level based on their research. They analyze the model on several domain-specific LLMs and also the LLaMA-2 family, all with below 4-bit quantization. Domain-specific LLMs include WizardCoder, Meta-Math, and OpenLLaMA-3B, which were tested on LLM-HumanEval-Benchmarks and GSM8K datasets respectively. Evaluations were performed on multiple datasets based on specific tasks

like WikiText-2 for language modeling; HellaSwag, PIQA, ARC, WinoGrande for common sense QA; and MMLU for in-context learning ability. The authors chose group size of 128 for 3-bit / 2-bit quantization, tuned using instructions from Alpaca, and compared BitDistiller with other QAT and Post-Training Quantization (PTQ) baselines e.g Omniquant, LLM-QAT, TLSD and RTN, GPTQ, AWQ, SpQR, QuIP. Results show BitDistiller performs better than others in language modelling and QA tasks on WikiText-2 and MMLU with increased average accuracy by 3.54% over LLM-QAT and 12.43% over PTQ methods in 2-bit quantization. Similarly, in reasoning tasks BitDistiller achieves an overall score of 61.33% which is about 24.69% more than LLM-QAT. Significant increase in training efficiency was also observed in quantizing WizardCoder-7B where BitDistiller takes 3.02 GPU hours whereas LLM-QAT takes 280.64 GPU hours on a single A100-80G GPU. As a whole, BitDistiller shows promising results in increasing sub-4-bit LLM performance while being cost efficient and requiring less resources as a whole.

Kundu et al. [17] introduced a new method called Multistage Low-rank Fine-tuning of Super-transformers (MLFS) which helps to perform parameter-efficient supernet training. This method aims to improve the encoder side of a small model (student model) which tries to mimic an advanced model's thinking pattern (teacher model). In the student model (which utilizes super-transformers), after implementing Low-Rank Adaptation (LoRA), this method fine-tunes the components in a multistage process by keeping certain layer's components frozen (different at each stage) making the fine-tuning process very parameter-efficient. It also utilizes gradient scaling to ensure optimization and stability during training. To compare results, this student model was trained by taking BERTbase as teacher model and was later compared with other distilled variants of the fixed models TinyBERT, DistilBERT and a super-transformer variant model called DynaBERT on the GLUE benchmark. The models were trained and tested on data sets SST2, RTE and MRPC. It was shown that the MLFS model performed better than the other models in terms of accuracy matrices on the dataset MRPC. On datasets SST2 and RTE, it sometimes falls behind some of these models but very slightly while also showing overall better results. In conclusion, it emphasizes that this method can be implemented to improve the encoder side of an LLM to be resource efficient, resulting in practical application of LLM on edge devices and expect better results. The authors expect this study paves the way to implement LLM on an industrial scale as more methods are explored to improve resource efficiency on LLMs. Authors have also suggested implementing a similar method to improve the decoder side of LLM as well to enhance the overall efficiency of the LLMs.

Dasgupta et al. [5] explores a training strategy which implements a new Knowledge Distillation loss formula by combining a new customized loss, which aims to improve the cost-effectiveness of LLM in general. After exploring the Taylor Series Expansion of the base Knowledge Distillation, the author further eliminates certain factors while also introducing new ones in the formula which results in the student model reaching convergence earlier than the teacher model. This makes it so that the student model with the new KD loss formula is able to produce better results than the teacher model given that the data set provided is small. The authors implemented this new KD loss formula with the BERT model while training on

task specific datasets. They tested their model on GLUE benchmark with TinyBERT, DistilBERT and MobileBERT on datasets MRPC,QQP,STS_B,MNLI and RTE. The results showed that the authors method outperformed the other models but only when the datasets were small, as other models later outperformed the author's model on larger datasets. So, this study was able to show a new way to improve LLM by improving the accuracy of student models even when the teacher model is weak, especially on small datasets. But the author has admitted that not only this model underperforms on large datasets but also that their proposed method is only limited to the English corpora and that it might not produce the same results on other languages. But still the author remains hopeful that this will enable the LLMs to be more implementable on edge devices due to improved learning from the teacher model.

Shen et al. [22] took the pruning approach to compress an LLM and optimize its use for resource-constrained environments. The paper presents a structured pruning approach called TransAct that decreases intra-modular activation within the Multi-head attention (MHA) and Multi-Layer perceptron (MLP) modules of the transformer architecture while retaining the inter-module activations sensitive to perturbations. This approach helps to transform it into a low-rank architecture without excessive precision loss due to preserving inter-module activations while applying the pruning technique. The paper applies the technique to the LLaMA2-7B model to observe and test its effectiveness while using the RedPajama-V1 dataset to train it. The performance of the pruned was evaluated by multiple downstream tasks that include ARC, BoolQ, and TriviaQA. The pruning process applied in the paper includes systematically reducing the number of attention heads of the MHA while keeping its dimensions the same. Again, the internal dimensions of the MLP module are also pruned as the authors observed high redundancies in the MLP weights if the negative values were suppressed using a function (eg. ReLU). The results of this showed significant compression with the number of parameters decreasing from 6.7B to 1.3B (80% compression) in addition to reductions in inference latency (75%-80%). Moreover, The model was also able to retain up to 78% of the performance of the original LLaMA2-7B model while having 80% compression which clearly demonstrates its effectiveness even when highly compressed. The paper also compared the performance of its pruned model with other compressed models like Sheared-LLaMA and LLM-Pruner. Compared to these models, the model using TransAct was able to give overall better scores in case of both computational efficiency and performance. Furthermore, The ablation study conducted in this paper also reveals improved efficiency of an iterative pruning approach compared to a single-shot approach.

Peng et al. [20] proposed the usage of derivative-free optimization techniques for enabling on-device fine-tuning of LLMs. The authors addressed the data-security concerns which arise due to reliance on offloading computations to cloud and even fine-tuning processes like PEFT (Parameter-Efficient Fine-Tuning) and memory efficient techniques like gradient checkpointing impose substantial memory overhead for mobile devices. The derivative-free methods evaluate the objective function at different points without computing gradients in order to explore the parameter space whereas the traditional derivative-based methods have heavier memory requirements for storing the gradients and optimizer states. For this study, the authors have used

MeZo, a memory-efficient zeroth-order optimization method, which avoids the need of gradient computation and storage yet has not gone through any trials on mobile environments. The authors also observe inherent parallelization potential in the derivative-free methods and it can be leveraged to improve efficiency on devices equipped with GPUs or NPUs. To analyze the feasibility of this approach, the authors have used RoBERTa-large and fine-tuned it on the SST-2 dataset which is a sentiment analysis task. OPT-1.3B have also been used in this study after fine-tuning on the SuperGLUE tasks (a collection of natural language understanding benchmarks) to demonstrate the feasibility of fine-tuning both medium-sized (RoBERTa-large) and large (OPT-1.3B) LLMs on mobile devices. An OPPO Reno 6 (with 12 GB memory) was used and the authors compared the performance of MeZo with Adam in terms of memory consumption, training loss etc. In the first metric, MeZo clearly wins as its fine-tuning consumed 4GB for RoBERTa-large whereas Adam crashed at higher batch sizes (consumed more memory for lower sizes). However, for training loss, MeZo exhibits a steady decrease but converges slower than Adam which was expected due MeZo's approximation for gradient estimations. Despite this successful demonstration, the authors acknowledge several limitations. For instance, the memory required for fine-tuning is quite greater than typical mobile applications (around 1 GB). The derivative-free methods are also less efficient in determining the optimization direction and the current implementation was done using a Linux simulation environment on Android (might not represent real world mobile environments properly).

Yin et al. [26] proposed using the LLM as a Service (LLMaS) paradigm to handle integrating LLMs into edge devices (mobile phones in this case). Compared to traditional DNNs which work in a stateless manner, this system supports stateful LLM contexts that help to maintain Key-value (KV) caches across multiple invocations. To manage context switching efficiently, the paper introduces LLMS, a system that uses memory management techniques to optimize it for resource-constrained environments. The paper takes three main optimization approaches: 1) Tolerance-aware compression, where it uses information density (based on attention score) as a metric to determine the compression tolerance of chunks which helps to reach the ideal compression ratio for chunks without excessive efficiency or accuracy loss, 2) Swapping Recompute Pipeline, which uses pipelining to overlap recomputing chunks with I/O operations which helps to decrease the latency, and 3) Chunk lifecycle management, where it makes context-switching easier by determining the priority of eviction through a Least Compression-Tolerable Recently Used (LCTRU) queue and uses an ahead-of-time-swapping-out approach to determine when to swap out chunks. It also employs fine-grained chunking to manage KV caches which helps to optimize memory and I/O bandwidth while striking a balance between memory utilization and performance. The paper evaluates this system using Llama2-7B and OPT-7B and test it on devices like Jetson Orin NX, Jetson TX2, and MI14 smartphones. Using different context-switching scenarios by utilizing 6 representative datasets(AGnews, Xsum, Samsum, cnn dailymail, WMT17-de-en, SST-2), the performance of LLMS was evaluated which showed it significantly outperformed its baselines, reducing latency up to two orders of magnitude when compared to conventional app memory management systems and achieving an on average 9.7x and up to 20x latency reduction compared to chunk-based context managing system. Most importantly, LLMS

achieves these results without a noticeable loss in accuracy on all used datasets making it ideal for managing LLM contexts for efficient deployment on edge devices.

Carreira et al. [4] present a method to deploy an LLM directly on mobile devices to address the problems associated with cloud-based AI systems such as privacy concerns and issues with latency. The paper utilizes the supervised fine-tuning process through using Low-Rank Adaptation (LoRA) and Parallel Environment Fine-Tuning (PEFT) libraries. Then, it performs 4-bit quantization through GGML to make the model smaller and easier to fit into resource-constrained environments. The authors selected the RedPajama-INCITE-Chat-3B-v1 model containing 2.8 billion parameters for their process. Using the GPT3.5 model, the authors generated a custom dataset through prompt engineering featuring human-AI interaction and action-specific specialized tokens. Specific Layers were chosen for finetuning while the rest were frozen. Through combining Lora and Peft libraries, the bitsandbytes library enabled 8-bit precision training to accelerate the learning process. A relatively high learning rate (1e-3) and a small number of epochs (5) were used to prevent overfitting and the layers generated by LoRA were added to the base model. Then the model was converted into the GGML format to use the 4-bit quantization feature. This process produced a model with a significant reduction in size and resources required to run it. Android NDK (Native Development Kit) was used to compile and run the model on an android device. This resulted in a model that could run on a device with as little as 4GB RAM although 6GB RAM is the recommended amount for reasonable performance. Additionally, the quantized model was able to reduce its size from the 5.17 GB of the original to 1.6 GB which is an impressive 70% reduction. In terms of performance, while there were a few drawbacks like occasional errors in token handling, this model maintained a robust performance despite its size while providing text-to-action features and voice functionalities.

Wang et al. [25] introduced a two-stage double compression design that is able to compress the size of quantized LLMs while not compromising the performance of the model. Firstly, they implemented a Per-Channel Scaling technique which scales weights based on activation pattern, making the model compressed. After this they perform quantization, resulting in a better compression ratio. This compressed model later undergoes pruning which further compresses it. Finally, they apply a Lossless Compression algorithm which compresses the weights, achieving a compressed model without compromising performance. During inference, through decompression of bottlenecks, it is able to balance decompression latency and memory usage. The authors applied these techniques and compared their model with OPT-6.7b, Llama2-7b, Falcon-7b and Mistral-7b models. They compared these models' accuracy on datasets Hellaswag, Lambada, PIQA, MMLU, MathQA, WikiText and SWDE while also keeping track of the compression ratios. Results showed that their method was able to keep up with the accuracy of other models while maintaining a high compressibility ratio ranging from 2.00 to 2.39 showcasing that their method works. The authors admitted that their work only focused on INT8 quantized weights which are heavily supported by lots of hardware. Different methods may affect the end result if they were to follow the authors method.

Yu et al. [27] presented a model Edge-LLM which was meant to address the is-

sues of high resource consumption in terms of both computation and memory which the LLMs faced. For their model, the authors implemented their algorithms and techniques on the LLaMa-7B model, as it was one of the LLMs having high consumption of resources. Using a Layer-wise Unified Compression (LUC) algorithm, which analyzes the sensitivity to compression of each layer and compresses them likewise through quantization and pruning, it identifies which layers to compress and which ones to not. Later, the model undergoes an adaptive layer tuning, in which the algorithm selectively updates specific parts of the model by connecting their output to the final layer, reducing memory usage. A voting mechanism during inference helps to determine the final prediction through comparing each output of different layers. Finally, hardware scheduling optimizes the computation patterns improving hardware efficiency. Overall through this process, the Edge-LLM achieves lower consumption of memory and computational resources through utilizing these techniques. This model was compared to other models like Sparse-GPT, LLM-QAT, 7 other variants of the author's proposed methods and also some vanilla fine tuning methods such as LoRA, LST on the dataset MMLU and WikiText. All the models were run on the same hardware configuration having accelerator's DRAM being 8GB LPDDR4 and on ship SRAM to be 1MP. The results showed that Edge-LLM achieved 2.9x times more speed and 4x memory reduction compared to other vanilla fine tuning methods.

Zhang et al. [28] established TinyLlama, an open-source language model with 1.1B parameters that was pretrained on approximately 1 trillion tokens over a maximum of three epochs. The model is inspired by Llama 2's architecture and tokenizer and portrays the elegance of smaller models trained on large datasets being able to compete with large models with much larger number of parameters. In particular, the authors of the paper use up to 3 trillion tokens to train a transformer decoder-only model with 1.1B parameters. The hyperparameters they use include pre-norm and RMSNorm to stabilize the training process, as well as SwiGLU, a mix of the Swish activation function and Gated Linear Units in place of typical ReLU activation. In order to speed up inference, they also employ a grouped-query attention approach, which uses four groups of key-value heads and a total of 32 heads for query attention. On the other hand, for speed optimization they use FSDP to leverage multi-GPU and multi nodes setup efficiently which caused training speed and efficiency to improve significantly. Furthermore, they use FlashAttention, another critical improvement to the regular attention mechanism and also uses fused layernorm, fused cross entropy loss and fused rotary positional embedding for boosting computational throughput. After implementing all these speedup modules, they achieved a training throughput of 24000 tokens per second per A100-40G GPU and it takes about 3456 GPU Hours to train 300 billion tokens which is significantly less than the models MPT-1.3B and Pythia 1.0B. In the TinyLlama v1.1 they include some key modifications including sharding model parameters in a node in FSDP to reduce communication overhead, reduction in tokens from 3 trillion to 2 trillion but still resulting in marginal improvement compared to original TinyLlama and lastly expanding the singular pre-training phase of the original model to three stage pre-training process. The phases are basic pre-training where the model is trained with only SlimPajama corpus to develop its commonsense reasoning capabilities and then continual pre-training with specific domain where the training is diversified by

incorporating three distinct types of corpora, solely SlimPajama then second corpus combined with code and mathematical content and lastly third corpus focused on Chinese language data. The final phase being cooldown which is essential for smooth model convergence which they achieved through modifying batch size, increasing it from 1.8 million tokens to 7.2 million tokens while maintaining original cosine learning rate schedule. The model is tested using a range of commonsense reasoning tasks, including Hellaswag, OpenBookQA, WinoGrande, ARC-Easy and ARC-Challenge, BoolQ, and PIQA. Additionally, the model is assessed using the InstructEval benchmark for problem-solving tasks, and finally, xstorcloze, xnli, and xcopa are used to assess Chinese comprehension. TinyLlama models perform better than the currently available, publicly available models, OPT-1.3B, Pythia-1.0B, and Pythia-1.4B, in each of the tests listed above. In short, TinyLlama, with its compact architecture and promising performance, can not only enable end-user applications on mobile devices, but also serve as a lightweight platform for testing a variety of creative language model concepts.

Svirschevski et al. [23] proposes parallel decoding of tokens in Large Language Models (LLMs) by hardware offloading to speed up LLM inference. They introduce Speculative Execution (SpecExec) which promises to produce up to 20 tokens per iteration of the target model. SpecExec uses draft model to calculate the most probable continuations and constructs cache tree for the target model and then validates it in a single pass. Authors showcased LLMs with more than 50B parameters inferenced on consumer level GPUs with RAM offloading with 4-6 token speed for 4-bit quantized models and 2-3 token speed for 16-bit model. SpecExec works by constructing large draft trees using a draft model that accounts most likely continuations using a parallel search algorithm. Consequently, it applies a verification algorithm that uses the draft tree as a cache for probable continuations and lets the target model validate it in one pass. The method also structurally improves the constructed trees for large token budgets such that they can produce 10-20 accepted tokens with large budgets. SpecExec uses speculative algorithm to parallelize draft token verification for faster inference which provides around 10-18x speedups compared to sequential inference on identical hardware. The authors tried to estimate target hardware that could be used to run LLMs and they had two general groups. One group had no dedicated GPU which were mostly ARM-based CPU MacBooks with SSD and 16GBs of RAM. The other group were single consumer GPU systems with 12-24GB VRAM, 32-64 GB RAM, 4-8 CPU cores and PCIe 3.0 or 4.0 x16 CPU-GPU bus. Authors used Llama-2 70B-Chat and Mixtral8x 7B models in their research as target models. Furthermore, quantized (PTQ) versions of target models were also tested to provide better insight on implementation of LLMs on consumer hardware. Multiple draft models like JackFram-160M, TinyLLaMA-1.1B-Chat, LLaMA2-7B and 13B -Chat were used for testing. Multiple datasets were used for testing probability coverage, draft acceptance rates, and inference speeds. Tests were done using MTBench and C4 datasets on a A100 GPU. In both cases SpecExec performs better than a comparison model, SpecInfer for 70B target model and 7B draft model at temp=0 and temp=0.6, top_p=0.9. In inference speed test, 100 size subsamples of OpenAssistant, WikiText-2, MTBench, and C4 were used on LLaMA-2 and Mixtral8x7B both 16 bit and GPTQ quantized and LLaMA-3 as target models. In LLaMA-2, Mixtral, and LLamMA-3 SpecExec achieves 18.7x

(t=0.6), 16.4x (t=0), 15.6x, and 17.5x (t=0) speedup respectively. Tests also done using multiple consumer GPUs like RTX 4090, 3090, 2080 Ti also yields positive results on LLaMA-2 and ShearedLLaMA-1.3B models. As a whole, the authors have showcased the use of SpecExec along with offloading to be a viable method for LLM inference on consumer grade hardware which is essential for running LLM on constrained devices.

Saha et al. [21] proposes a low precision and low rank compression method that compresses LLMs to be able to run on memory-constrained devices. Calibration-Aware Low-Precision Decomposition with Low-Rank Adaptation (CALDERA) uses the redundancy introduced in weight matrices that manifest as low rank structure because of the correlation between syntax and semantics of languages to its advantage. This redundancy allows compression without substantial performance loss. CALDERA applies a form of post-training compression to compress LLMs by using this low rank structure. It approximates the weight W by decomposing it into $W \approx Q + LR$ where L and R are tall and wide matrices called low-rank factors and Q is the low-precision backbone. Q captures the low singular component and L and R captures the large singular components of matrix W. CALDERA boosts zero-shot results by readily fine-tuning L and R factors and the post-training quantization (PTQ) provides impressive zero-shot performance. It also reduces memory footprint by extending low precision representation. The authors applied CALDERA on the LLaMA family namely LLaMA-2 7B, 13B, 70B and LLaMA-3 8B. The quantization for CALDERA was performed using E8 lattice. It was tested for perplexity on C4 and WikiText-2 dataset and for zero-shot on WinoGrande, RTE, PiQA, ARC-Easy, ARC-Challenge. For perplexity, it was measured with sequence length equal to the model's maximum context length of 4096 for LLaMA-2 and 8192 for LLaMA-3. Results show that CALDERA decomposition had the highest accuracy and lowest perplexity in general which is owed to the ability to quantize low-rank factors without much loss of performance and also regaining performance compromised due to 2-bit quantization. In perplexity in 2.4 bits, CALDERA performs better than uncompressed models on both WikiText-2 and C4. Also, in terms of accuracy, it generally performs better than other models in case of LLaMA-2. The similarity also follows through in terms of LLaMA-3 model. But the authors have mentioned that accuracies are not completely indicative of quantization performance due to the randomness of zero-shot experiments but the reduction in accuracy can be regained by using Low-Rank Adaptation (LoRA) fine-tuning. As a whole, the authors have shown through this paper that CALDERA can perform well in the sub-2.5-bit parameter challenge. They have also shown that it can complement existing strategies to make compressed LLMs more feasible to use on consumer hardware.

2.3 Summary of Key Findings

MOHAWK, a three-stage distillation framework allows transfer of knowledge from transformer models to subquadratic alternatives like State-space-models (SSMs). This method allows SSMs to achieve performance close to transformers using less than one percent of the training data. Llamba is a model family that applies MOHAWK to distill Llama-3 into a recurrent Mamba-2-based model. Resulting models such as Llamba-8B is able get competitive performance using only 12 billion tokens which is less than 0.1% of teacher. This demonstrates data efficient, high performance cross architecture distillation to be a promising avenue to explore to get practical performance from smaller scale models. MobileQuant's joint optimization approach for weight transformation and activation range learning outperformed state-of-the-existing methods like SmoothQuant and OmniQuant on mobile hardwares. Similarly, the b1.58 variant of the original BitNet provides a novel feature-filtering modeling technique due to the incorporation of the 0 in 1.58-bit ternary representation. Another notable technique is the derivative-free optimization which fits memory constraints of edge devices by alleviating the requirement of storing gradients and optimizer states. Apart from that, Freezing pre-trained model weights and injecting trainable low-rank matrices reduces the number of trainable parameters by around 10000 and GPU memory usage by 3x without performance drop. Moreover, 4-bit NormalFloat (NF4) a quantization method optimized for normally distributed weights included with double quantization reduces memory usage and saves about 0.373 bits per parameter. It's also notable that setting different learning rates for the low-rank matrices springs about 2x the finetuning speed and a 1-2% performance increase over standard LoRA. Multistage fine-tuning of super-transformers (MLFS) algorithm introduces a new way to improve student model efficiency using LoRA, gradient scaling, and staged parameter freezing, enhancing encoder-side LLM for edge-device uses. In another method, a modified knowledge distillation loss function is used to enable the student model to reach convergence faster, improving performance only on small datasets but limited to English corpora. To achieve a high level of compressibility, combining per-channel scaling, quantization, pruning, and lossless compression, we get LLM compressibility for up to 2.39x with minimal accuracy loss. Similarly, performing layer-wise compression, adaptive tuning, and voting, it was possible to reduce memory (4x) and increase speed (2.9x) for efficient LLM adaptation on edge devices.Again, approaching pruning by targeting the intra-modular weights while preserving inter-modular weights can help retain an impressive amount of performance of the original model (78%) under heavy compression (80%). Targetting the stashing phase of intermediate values during the training phase provides multiple opportunities for quantization and if the quantization level is handled dynamically, we can get impressive arithmetic complexity (20.95x reduction) and memory traffic reduction (2.55x decrease) with minimal performance loss (0.41 GLUE score reduction). Dividing the KV cache into chunks and using an LCTRU queue and an ahead-of-time-swapping-out approach to determine eviction priority helps in efficient context management. Additionally, overlapping data fetch from the KV cache with recomputing context chunks using pipelining can help to decrease latency. Moreover, we can determine the information density of chunks using attention scores and use it to determine the compression tolerance of chunks in order to use dynamic quantization and get the optimized compression

ratio without excessive accuracy loss. In a different vein, using soft targets to train a smaller model works better than hard labels due to carrying more information about class relations. Distilling knowledge in this way can also allow the model to learn with less data as soft targets carry structural information. A combination of quantization and distillation known as BitDistiller provides sub-4-bit quantization and a new distillation process that results in low resource consumption without sacrificing accuracy. SpecExec uses speculative decoding and hardware offloading as a way to speedup LLM inference, even on consumer hardware. Low-rank and low-precision decomposition is also a viable method to compress LLMs without reducing accuracy for edge device implementation.

Chapter 3

Requirements, Impacts and Constraints

3.1 Final Specifications and Requirements

LLAMA

The setup for the experiment was done in Python through a combination of a set of built-in modules and external packages for both analysis and visualization. This was done to conduct a proper evaluation of the mathematical reasoning capabilities of the Llama 3.2 1B model through a benchmark test of the GSM8K dataset in order to replicate and verify the performance score provided by Meta. The experiment setup uses GPU acceleration where possible in addition to possessing robust tracking to trace the different steps that the model runs through.

We used PyTorch (torch, version 2.0.0) as the base of our workflow to use it to get access to tensors, GPU acceleration, and potentially other components that may be necessary for out experiment. Besides that, we accessed the necessary pre-trained models and tokenizers through the Hugging Face transformers library (version 4.40.0) and the GSM8K dataset through the datasets library (version 2.16.0). The huggingface_hub client (version 0.20.0) allowed us to perform authentication and access gated models and datasets as necessary. The accelerate library (version 0.25.0 or later) was used to optimize scaling to multiple GPUs.

The trl package (version 0.7.0) was used to add the ability to use reinforcement learning-based techniques to our environment. This was in addition to the peft library (version 0.8.0), which added parameter-efficient fine-tuning methods such as LoRA and adapters. These can help to cut down on the memory and computation necessary. Additionally, bitsandbytes (version 0.42.0) was added to bring quantization (e.g, 4-bit,8-bit precision) capabilities, which can allow larger models to run on smaller hardware. Furthermore, flash-attn (version 2.4.0) was added as it offered a CUDA-implemented attention mechanism that significantly lowered the training and inference time to further optimize NVIDIA GPUs.

For necessary and thorough numerical computation and analysis of data, the numpy (Version 1.24.0) and scipy (version 1.10.0) libraries were used for their ability to pro-

vide high-performance array operations, high-level mathematical functions and and stat routines. They are joined by the pandas library (version 2.0.0) to get access to dataframes to structurally organize the different sections of data for ease of evaluation and use. Furthermore, Other machine learning utilities like dataset splitting, evaluation metrics, and post-processing functions are provided by scikit-learn (version 1.3.0), and common metrics like accuracy, BLEU, and ROUGE come from the Hugging Face evaluate library (version 0.4.0).

In order to properly monitor and potentially reproduce various runs, we used the tqdm library (version 4.65.0) for its progress bars during evaluation loops, which helps maintain transparency during evaluating a large number of examples. wandb (version 0.16.0) library adds the ability to log metrics, plots, and system utilization during training or evaluation to further strengthen our ability to track across different runs and potentially compare or reproduce them. To visualize model performance and plot them in a detailed and easy-to-interpret way, Matplotlib (Version 3.7.0) and seaborn (Version 0.12.0) libraries were included.
Finally, we run the experiment in the Jupyter (Version 1.0.0) interactive development environment with ipywidgets (Version 8.0.0). This provides interactivity to the notebook and helps with experimentation, exploration, and visualization.

Overall, this experimental environment combines powerful data analysis tools and the latest visualization platforms to create a scalable,traceable and reproducible experimental pipeline to evaluate the Llama 3.2-1B model and get a clear idea about its mathematical reasoning abilities.

Llamba

The experimental setup was run in Python on Ubuntu and was optimized to test the Llama models on the GSM8K benchmark with an NVIDIA RTX A6000 (48GB VRAM) and 64GB of system memory. In order to achieve reproducibility and efficiency, a specific Python environment (llamba_env) was set up using the following dependencies.

The machine learning frameworks supporting CUDA 11.8 are at the heart of the environment. The basic deep learning framework was PyTorch (torch 2.0.0, 2.9.0), which offers tensor calculations, GPU acceleration, and model execution. Torchvision (torchvision 0.15.0) and Torchaudio (torchaudio 2.0.0) were also added as standard libraries of the PyTorch ecosystem, enabling the processing of vision and audio data. These are not directly related to the GSM8K database and are included to complete the PyTorch stack compatibility and ensure compatibility between Torch and CUDA.

To support natural language processing and large language models, the Transformers library (transformers 4.40.0, 5.0.0) of Hugging Face made available pretrained models, tokenizers, and generation pipelines. It is compatible with Tokenizers (tokenizers 0.15.0), a high-performance Rust library designed to optimally tokenize texts. Standard access to benchmark datasets was provided via the Datasets library (datasets 2.14.0) and authentication along with direct integration with the Hugging

Face model and dataset repository was provided by Huggingface Hub (huggingface-hub 0.20.0). Also, Accelerate (accelerate 0.20.0) was added to allow inference and training to scale seamlessly to various hardware setups. Cartesia-PyTorch provided support to the Cartesia implementation of Llamba models and allows model-specific optimizations.

NumPy (numpy 1.24.0, 2.0.0), the standard numerical computing library, and Pandas (pandas 2.0.0, 3.0.0), the tabular data analysis and manipulation library, were used to manipulate and analyze data. Matplotlib (matplotlib 3.7.0) and Seaborn (seaborn 0.12.0) were added to enable low-level and high-level statistical graphics with plotting support. tqdm (tqdm 4.65.0), a simple progress bar library, was used to support progress monitoring in computationally intensive tasks.

The Jupyter ecosystem made interactive experimentation possible. The notebook interface was served by Jupyter (jupyter 1.0.0), and the server part was served by Notebook (notebook 6.5.0). IPyWidgets (ipywidgets 8.0.0) and IPyKernel (ipykernel 6.0.0) added further interactive capabilities with interactive widgets and the Python kernel interface, making it easy to run .ipynb notebooks.

It also supported performance monitoring tools, including psutil (psutil 5.9.0) to monitor resource usage at the system level (CPU and memory usage), gpustat (gpustat 1.1.0) to monitor lightweight elements of the CPU (GPUs), and Memory-Profiler (memory-profiler 0.61.0) to track memory consumption line by line in Python scripts.

Lastly, a code of development and system utilities provided robustness and maintainability. IPDB (ipdb 0.13.0) gave a debugger interface to Python, and Packaging (packaging 21.0) managed version and dependency solving among the installed libraries. To support high-performance text processing, the Regex library (regex 2023.0.0) added to the standard regular expressions of Python additional matching and more powerful capabilities and Unicode support.

These libraries collectively comprised an end-to-end system that could infer and evaluate the Llamba models on GSM8K at scale, as well as provide monitoring, debugging, and interactive experimentation.

Hardware Configurations

- PC-1
  - OS: Ubuntu 22.04 jammy
  - Kernel: x86_64 Linux 6.8.0-84-generic
  - CPU: AMD Ryzen 5950X 16-core @ 32x 3.4GHz
  - GPU: NVIDIA GeForce RTX 3080 Ti
  - RAM: 64 GB
- PC-2
  - CPU: i9-14900K (24C-32T)

- GPU: RTX A6000 48GB
- RAM: 64GB DDR5
- SSD: 1 TB
- PSU: 1000W

Software Used for Writing and Running Code

- WSL
- Visual Studio Code

3.2 Societal Impact

Large Language Models have a considerable amount of societal impact due to their versatility and wide range of use. In its existing state, LLMs are already being used to support multiple aspects of society like education, healthcare, finance, administrative works etc. The transformer-based LLMs are the most common and most powerful type of LLMs currently available, but they create significant challenges in terms of societal equity. Transformer-based LLMs require high computational power and expensive hardware. These types of hardware are not easily available or accessible by the greater part of the society. This creates a disbalance in the accessibility of this technology. Larger companies and organizations can reap the benefits of this technology owing to their access to resources. This introduces a lack of democratization of LLMs. Our SSM-based approach to LLMs could potentially address and enhance democratization due to its efficiencies. SSM-based LLMs could use less powerful devices and require less computational power. This would allow it to be used on less powerful devices, devices that are more readily available to the majority of society. This would allow a wider range of deployability. Especially in critical sectors like healthcare and education, where the use of hardware-intensive LLMs is not feasible, these LLMs could provide essential support to local communities and service providers. These would address the imbalance of accessibility and have an overall positive societal impact.

3.3 Environmental Impact

Deep learning models have been proven to be detrimental to the environment due to their high computational needs. Especially, in algorithms for NLP tasks, the training phase alone contributes to high amounts of greenhouse gases. Strubell et al. [2] mention the production of gases equivalent to 300 long-haul flights, in the training phase of such algorithms. This only worsens with the development of powerful LLMs. These powerful models require high amounts of energy and sophisticated hardware to function. Due to the rising demand for AI and LLMs, new datacentres are being constructed at an accelerated pace. This not only contributes to the production of GHGs, but also to the destruction of the environment and ecology. Some of these datacentres are being constructed close to suburban areas, that are

worsening the quality of life of the residents. Some of these areas are facing a water crisis as the datacentres use up the surrounding groundwater for cooling their machines. Transformer-based LLMs are notoriously power-hungry and only amplify the crisis. Our SSM-based approach is potentially a better alternative to them as it's more efficient and faster, which reduces the overall time a model runs in general. This will have a reduced environmental impact than the existing architecture. It also supports the use of lighter hardware, which further reduces the impact on the environment.

3.4 Ethical Issues

LLMs and their global reach have opened the doors for many ethical concerns. The over-reliance on LLMs can contribute to the spread of misinformation if the responses are not verified properly. This is a genuine concern as LLMs tend to hallucinate on a regular basis. Without proper vetting, the misinformation generated due to LLM hallucinations can spread and be a reason for mistrust and conflicts. Another ethical concern is the accessibility of the technology. Due to the computationally expensive nature of current LLMs, it can only be used by a subset of the population to its full capabilities, meanwhile the majority do not get the chance to properly utilise the technology. This inequity is also an ethical issue that needs to be addressed. Shifting to SSMs could ensure equity as the technology would be available for even more people to use. Another ethical concern is bias management and fairness. The biases of society can possibly seep in and propagate through the use of LLMs. Most LLMs like LLama have policies in place and regular scrutinizations to monitor and reduce biases in their models. Similarly, the SSM-based LLMs would also keep this in mind and address any ethical concerns accordingly.

3.5 Project Management Plan

The Project was planned into 3 phases. Initially in the first phase, a team was formed of 5 members. After team formation was done, extensive discussions were done to understand our interests to select a domain which was Large Language Model (LLM). After the domain was selected, a supervisor was chosen who was an expert in the chosen domain. The supervisor, with his expertise in the domain and taking into account the opinion and our ability, helped us narrow down and focus on an unexplored topic which we accepted. Our team then dove into the topic, exploring every related and impactful papers and studied them thoroughly. A brief report about the research was written later on.

After gathering all the necessary information we needed to carry on with the thesis, we started its phase 2. Here, an architecture was chosen which we had studied in the previous phase which was the Mamba architecture. This architecture featured a linear time complexity which is seen to be a remarkable achievement since it meant processing anything of large size with less amount of resources compared to what the transformer would have needed. We then further explored a newly made Mamba

based model called LLAMBA which was created by Albert Gu [29] and his team. We tried to implement the model and experiment on its own, which was quite challenging since there were very limited resources about this model since it was relatively very new. We later, with our limited understanding from the experiments, again wrote a brief report and a poster to explain its implications and features.

Finally starting the final phase, we then further explored in what areas the performance of LLAMBA was yet to be explored. It was seen that there was no testing done on the performance of LLAMBA in the mathematical reasoning area. This led us to think if there was a possibility where LLAMBA could outperform the traditional Transformer based models in this specific area. Extensive experimentation was made not only on LLAMBA but also its transformer based teacher model LLAMA to understand how well the LLAMBA model was performing. After analyzing the result, we then wrote our understanding in the final report and prepared for our thesis defense.

3.6 Risk Management

With the enhanced capabilities and generative abilities of LLMs, the levels of risk associated with it has also increased. The vast repertoire of knowledge in LLMs also includes potentially harmful elements that can have massive negative implications for society. The risks of using LLMs to make or gather knowledge on explosives and weapons of biological, chemical, and nuclear kinds is a harsh reality. Most LLMs have guidelines and checks in place to monitor the access of such information. Another major risk factor is child safety. It is essential to maintain the safety of children while providing access to LLM services. Mental health is also a source of concern. A lot of members of society are suffering from mental health crisis and their interaction with LLMs without proper filtering and tuning could potentially do serious damage to their lives. But LLMs try to ensure safety from these risks through meticulous fine-tuning to ensure children are safe and mental health conditions are not affected by the general use of LLMs. Cyber security and threat of malicious system attacks are also a concern. But as any other risk, red teaming and testing are done regularly to increase the security levels and prevention of such attacks. SSMs are no different, and will follow similar guidelines to ensure safety and mitigate risks associated with LLMs in general.

3.7 Economic Analysis

When building and using LLMs, there is an economic aspect to keep in mind. These models require powerful hardware to run, which in turn consumes high amounts of power. This power usage calculation is done on the basis of the power required to generate tokens. The more powerful the hardware is and the more resource-intensive the models are, the higher the power requirement. This high power consumption poses an affordability question, whether it is economically viable after a certain point. On top of that, expensive hardware is also difficult to get. The semiconduc-

tor and GPU industry is constantly in a deficit for materials, and the availability of consumer-grade hardware has decreased in recent years. Transformer-based models, due to their high power consumption and high resource requirements, are not very economical. So their accessibility is limited to those who can afford to buy the capable hardware, run and sustain it. But shifting to SSM-based LLMs could be a viable alternative as the faster inference time of SSMs allows for fewer compute cycles. This can reduce the economic overload and make it feasible and affordable for more users. This will also allow LLMs to be used and expanded to more sectors around the society and let everyone benefit from the responsible use of LLMs.

Chapter 4

Methodology

4.1 Architecture Description

4.1.1 Transformer

Transformer is a neural network based architecture that fundamentally changed how Artificial Intelligence is seen nowadays. This concept introduced in the paper "ATTENTION IS ALL YOU NEED" in 2017 [7] has since become the standard architecture for deep learning models to work with. This architecture is now commonly used in the world, more specifically it powers some of the popular text generation models available today such as Google's Gemini, OpenAI's GPT and Meta's Llama, which is the focus of this study. Beyond text generation, this architecture has shown promising results in other areas as well such as audio generation, image recognition, video generation, etc. Even though this architecture has pronounced performance, it comes at a huge computational cost, especially when it comes to resource-intensive tasks mentioned above.

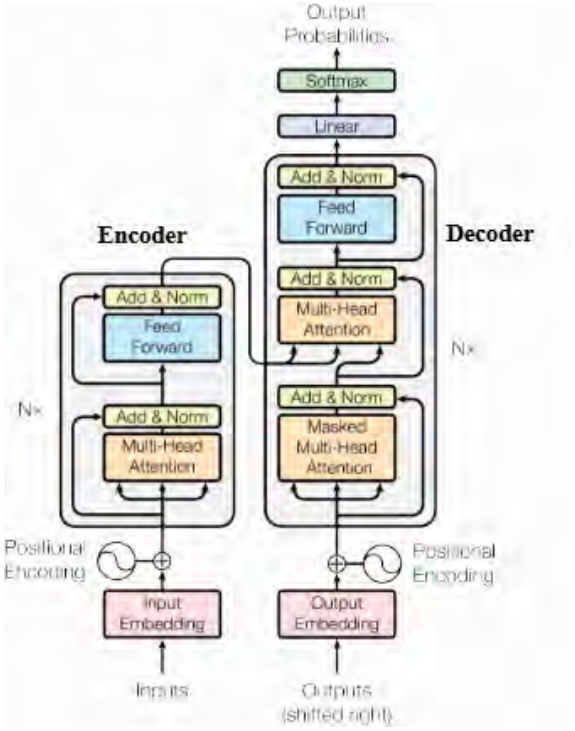

Figure 4.1: Transformer [7]

At its core, the Transformer processes input sequences through a stack of encoder and/or decoder layers, each comprising two primary sub-components: multi-head self-attention and position-wise feed-forward networks. For mathematical reasoning tasks, input sequences are first tokenized and embedded into dense vector representations of dimension d_model. Since the attention mechanism itself has no inherent notion of sequence order, positional encodings are added to each token embedding, providing the model with information about token positions within the sequence.

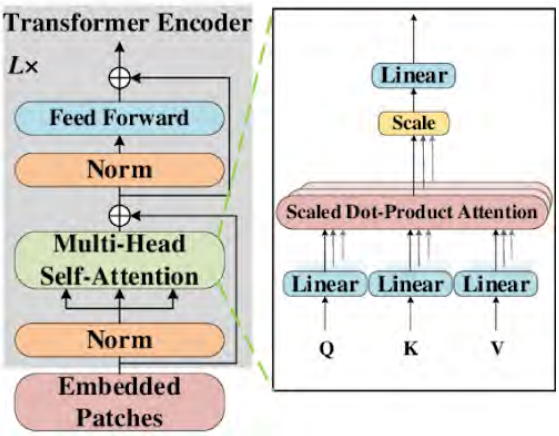

Figure 4.2: Multi Head Self Attention [9]

The multi-head self-attention mechanism constitutes the most computationally demanding component of the Transformer architecture. If an input sequence is of length n size, the attention mechanism computes all the relationships between all pairs of tokens that are possible to be paired, resulting in a computational complexity of O(n2..d_model), where d_model is the model dimension (typically 512-2048). So the more complex the task is, the more computationally demanding this mechanism becomes, specially for tasks like mathematical reasoning where chain-of-thought is required to solve the problems. This demand can easily spiral out of hand if the task becomes more complex, thus making it a hard bottleneck for more complex tasks. Each attention head computes three learned projections without having any dependencies which are: queries (Q), keys (K), and values (V) which it acquires from the input embeddings. The attention scores are calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{4.1}$$

Where,

- Q is the Query Matrix
- K is the Key Matrix

- V is the Value Matrix
- d_k is the Dimension of Query and Key Matrix
- QK^T is the Query Key Dot Product

where d_k represents the dimension of each attention head. The softmax operation over the n x n attention matrix requires substantial memory, as the full attention map must be materialized before normalization. For a sequence of 512 tokens with 8 attention heads and d_model=512, a single attention layer must compute and store approximately 2 million attention weights per input, and modern Transformers typically stack 12-24 such layers.

The multi-head mechanism partitions the d_model-dimensional space into h parallel attention heads, each operating on d_k = d_model/h dimensions. This design allows the model to attend to information from different representation subspaces simultaneously. However, it also multiplies the computational burden—each of the h heads performs independent Q, K, V projections and attention computations. The outputs from all heads are concatenated and linearly projected back to d_model dimensions, requiring an additional O(n·d^2_model) operations.

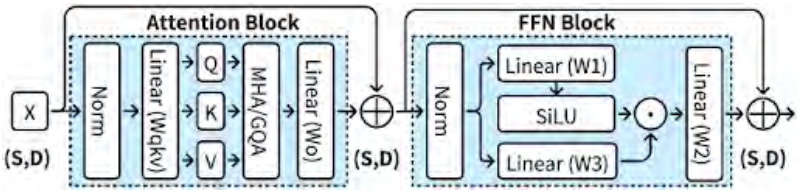

Figure 4.3: Feed Forward Network [14]

After the attention sub-layer, each token representations are sent to a position-wise feed-forward network (FFN). Even though it is operating independently on each position, this FFN is one of the main reasons behind Transformer's larger requirement of parameter count and computational cost. The standard FFN consists of two linear transformations with a non-linear activation (typically GELU or ReLU) in between:

$$\text{FFN}(x) = W_2 \cdot \sigma(W_1 \cdot x + b_1) + b_2 \tag{4.2}$$

Where:

- x is the Input Vector/Matrix
- W_1 is the First Weight Matrix
- b_1 is the First Bias Vector

- σ is the Activation Function
- W_2 is the Second Weight Matrix
- b_2 is the Second Bias Vector

where the intermediate dimension is typically a 4×d_model. So, let's say for d_model=512, this later increases to 2048 dimensions which means each FFN layer contains around 2 million parameters per token position. Not only that, processing a sequence of n tokens requires $O(n \cdot d\_model \cdot (4 \cdot d\_model)) = O(4n \cdot d^2\_model)$ operations, which across a 12-layer Transformer, these computations have a possibility to dominate the parameter count and contribute significantly to inference latency.

Both the attention and FFN sub-layers employ residual connections followed by layer normalization. Even though these components do not add that much computational overhead individually, layer normalization needs computing mean and variance statistics across d_model dimensions for each and every token in each layer, which overall contributes to eating up much of the memory bandwidth consumption.

For text generation tasks, the computational demands escalate further due to the autoregressive nature of decoding. During generation, the model must produce tokens sequentially, with each new token requiring a full forward pass through all Transformer layers. The attention mechanism must attend to all previously generated tokens, meaning the effective sequence length—and thus computational cost—grows linearly with each generated token.

For tasks that require text generations, the computational requirements escalates further due to the autoregressive nature of decoding. When the output is generated, the model must create each token sequentially, with each new ones requiring a full forward pass through all the layers inside the Transformers. The attention mechanism is also called for all previously generated tokens, which makes the effective sequence length and also the computational cost to grow with each token. For generating a sequence of m tokens from a prompt of length n, the total attention computation complexity becomes $O(m \cdot (n+m)^2 \cdot d\_model)$, assuming average sequence length of (n+m)/2 during generation. This quadratic growth in generation length makes producing long mathematical derivations or detailed explanations prohibitively expensive.

The mathematical reasoning tasks make these challenges harder in several ways. Firstly, the mathematical expressions contain specialized notation, equations, and symbolic relationships that require a large size of vocabulary and rich-dense embeddings. Secondly, mutli-step reasoning demands processing of much lengthy contexts that may or may not include problem statements, intermediate working steps and solution verification, which easily makes lengthier tokens for much complex math problems. Thirdly, to ensure arithmetic accuracy and logical reasoning, it sometimes requires multiple decoding passes or other strategies, which multiplies the already generated cost by even more.

The memory footprint of Transformers is another big constraint for mathematical reasoning applications. Modern pre-trained models like GPT-3 (175 billion parame-

ters) or PaLM (540 billion parameters) require hundreds of gigabytes of GPU memory merely to store weights in full precision. Activation caching during the forward pass—necessary for backpropagation during training or for efficient autoregressive generation—demands additional memory proportional to n·L·d_model, where L is the number of layers. For a 24-layer model processing sequences of 2048 tokens with d_model=1024, activation storage alone exceeds 100MB per input example before considering attention weights or intermediate FFN activations.

Despite various optimization strategies—including mixed-precision training, gradient checkpointing, sparse attention patterns, and model parallelism—the fundamental $O(n^2)$ scaling of self-attention remains a critical bottleneck. For mathematical reasoning benchmarks that require processing and generating sequences of several thousand tokens, inference latency can reach multiple seconds per example even on high-end accelerators, and training such models demands distributed clusters with hundreds of GPUs running for weeks.

In short, even though the Transformer architecture's attention mechanism is quite powerful in the sense that it creates a powerful global dependencies which is required for complex mathematical reasoning, the architecture's tendency to have massive parameter count, quadratic computational complexity, high memory requirements and sequential generation process makes it severely resource demanding. These constraints not only limit the general usability of this architecture, but also makes it too inefficient. But regardless, despite the power-hungry nature of the transformer model, it produces results which are quite impressive.

4.1.2 Mamba

The Mamba architecture [13], also based on neural networks, is an alternative architecture for deep learning models. This architecture seems to achieve remarkable computational efficiency through a process which lets it select which unimportant states to compress and also because of its linear-time processing. State-space model (SSM), a component of Mamba architecture, helps process sequences through a recurrent state update mechanism that keeps the size of the hidden state fixed, regardless of how long or short the sequence length is. This helps the architecture to efficiently handle the tasks, which in turn significantly reduces memory and computational overhead, which generally are culprits for models requiring huge amounts of resources to work. These qualities make it a perfect candidate to process complex tasks efficiently, some of which are the one this study explores which is mathematical reasoning tasks.

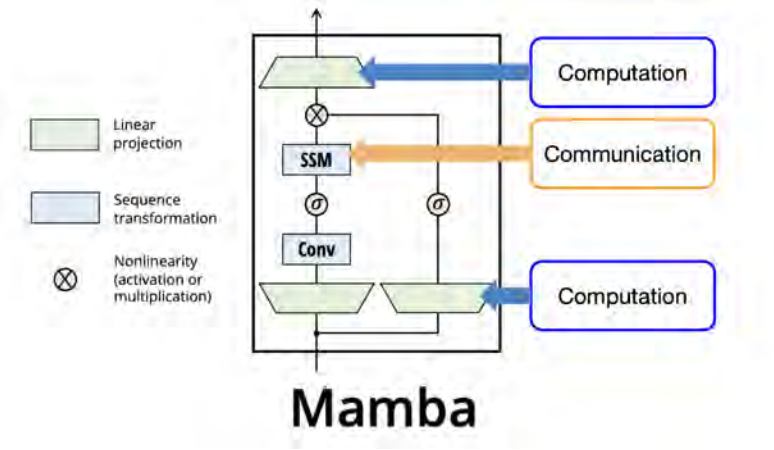

Figure 4.4: Mamba [16]

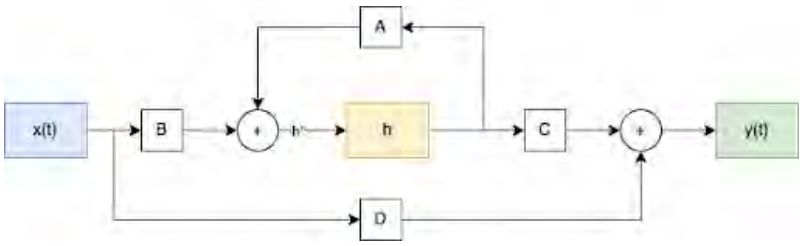

Figure 4.5: Hidden State of Mamba Architecture [10]

The SSM, one of the main components of the Mamba architecture, is derived from continuous-time dynamical systems, which are later converted into discrete-time versions for use on computers. At each time step t, the model keeps track of hidden state h(t) (which is fixed in size). The hidden state updates itself over the following equation:

$$h(t) = \bar{A} \cdot h(t-1) + \bar{B} \cdot x(t) \tag{4.3}$$
$$y(t) = C \cdot h(t) + D \cdot x(t) \tag{4.4}$$

Where:

- $h(t)$ is the hidden state at time t
- $x(t)$ is the input at time t

- $y(t)$ is the output at time t
- $\bar{A}$ is the discrete state transition matrix
- $\bar{B}$ is the discrete input matrix
- C is the output matrix
- D is the feedthrough matrix

Where x(t) is the input token embedding at position t, y(t) is the output, while $\bar{A}$, $\bar{B}$, C, D are parameter matrices which are to be learned. Here, the state dimension N stays constant throughout processing. This is the key to how the Mamba architecture is able to perform tasks so efficiently, since this state does not grow in size no matter how long the context is.

This makes the computational complexity of processing for input sequence of length n down to O(n.N.d_model), where d_model is the model dimension (typically 512-2048) and N is a small constant, representing linear scaling with sequence length. For a sequence of 2048 tokens with d_model=768 and N=16, an SSM layer performs approximately 25 million operations compared to over 3 billion for an equivalent attention layer—a reduction of more than 100×.

The Mamba architecture introduces critical innovations that enhance the expressiveness of basic SSMs while preserving their efficiency. This makes it so that SSM parameters ($\bar{A}$, $\bar{B}$, C) become input dependent instead of being fixed for all tokens, This allows the model to dynamically adjust state update mechanism based on information that are being processed.

$$\begin{aligned}\bar{B}(t) &= \text{Linear}_B(x(t)) \\ C(t) &= \text{Linear}_C(x(t)) \\ \Delta(t) &= \text{softplus}(\text{Linear}_\Delta(x(t)))\end{aligned}$$

Time-scale parameter which is Δ(t) helps to control how the step size is being discretized, helping the model to adaptively compress or expand temporal resolution. In this case, for mathematical reasoning, this selection is more powerful, as the model can allow more state capacity to be used for reasoning steps while shortening information that is not useful.

Memory efficiency constitutes one of Mamba's most significant advantages. During processing, the model only needs to maintain the current hidden state h(t) of dimension N, rather than storing an n×n attention matrix. For a 24-layer model processing 4096 tokens with N=16 and d_model=1024, the total state memory requirement is approximately 1.5 MB compared to over 400 MB for cached attention weights in an equivalent Transformer. This 250× memory reduction enables processing dramatically longer sequences within the same hardware constraints—a single GPU can handle sequences of 100,000+ tokens that would be impossible for standard attention mechanisms.

The recurrent state update mechanism also enables highly efficient autoregressive generation. Unlike Transformers, which must recompute attention over all previous tokens for each newly generated token, SSMs simply update their fixed-size hidden state. Generating a new token requires only O(N·d_model) operations regardless of how many tokens have been previously generated. For mathematical reasoning tasks that may require generating hundreds of solution steps, this translates to constant-time generation per token rather than linearly increasing costs. A model generating 1000 tokens experiences the same per-token latency for the first token as the thousandth token.

Mamba's architecture organizes these selective SSM operations within structured blocks. The input is processed parallely in each Mamba block, one uses selective SSm mechanism while other focuses on element-wise gated activations. Finally, the output is calculated by a multiplicative gating operation:

$$\text{Output} = (\text{SSM\_branch}(x)) \odot \sigma(\text{Gate\_branch}(x)) \tag{4.5}$$

where $A \odot B$ is the element-wise multiplication and σ is the activation function. This gating mechanism helps the model to modulate information flow depending on the input content, making it more expressive without compromising linear complexity of SSM. The entire block is of time complexity O(n·N·d_model), and multiple blocks stack on top of other to create deep networks while still maintaining linear scaling.

Mamba's computational efficiency can be easily enhanced through hardware efficiency. The sequential state helps to maintain almost perfect locality of reference, making most use of cache hierarchies and minimizing bandwidth use.Modern GPU implementations achieve memory bandwidth utilization exceeding 80% compared to 40-50% for attention operations, which suffer from irregular memory access patterns when computing the attention matrix. Moreover, the update which happens recurrently may be implemented with fused kernel operations that combine multiple arithmetic operations in one single GPU kernel, reducing data movement costs and kernel launch overhead.

In case of its use for mathematical reasoning, the efficiency characteristics of the Mamba architecture as a whole creates benefits. Processing a typical competition mathematics problem—comprising problem statement, solution steps, and verification—might involve sequences of 2000-3000 tokens. A 24-layer Mamba model with d_model=1024 and N=16 processes such inputs in approximately 150-200 milliseconds on a single A100 GPU, compared to 800-1200 milliseconds for an equivalent-size Transformer. At the time of training, the efficient use of memory allows more batching of up to 8 times more examples simultaneously, accelerating convergence and making train time less.

The parameter efficiency of SSMs contributes additional resource savings. While Mamba blocks contain learned projection matrices for input-dependent parameter computation, the absence of separate query, key, and value projections (each of size d_model × d_model) in every layer results in 20-30% fewer total parameters than Transformers of comparable capacity. It needs to be mentioned that a Mamba model

with 350 million parameters performs on par with other 500-600 million parameter Transformers, which further showcases the architecture's ability to reduce inference memory consumption, data transfer costs and storage requirements.

Scaling to bigger contexts only further proves Mamba's ability to be efficient. Attention mechanisms of the Transformer architecture become unrealistically expensive as the tokens start to increase. Comparing that with SSM, whose scaling is linear, makes it a powerful candidate to scale better. No matter the task, as for this case being a mathematical reasoning task, it can absorb large amounts of context which is fairly common for such problems and produce an effective outcome as expected whereas Transformers would have taken much more resources and time to process such large context.

Training stability and convergence speed benefit from the recurrent architecture's properties. The linear state update avoids the gradient pathologies that can arise in deep attention networks, where gradients must backpropagate through the quadratic attention operation. The SSM gradient goes through linear recurrence with well-conditioned dynamics, which makes training stable even with larger learning rates and reduces the need for extensive hyperparameter tuning.

The adaptability of selective mechanisms being content-based makes it provides not only raw computational scaling but more. By understanding and being able to compress mathematical manipulation while preserving logical reasoning steps, this model is able to allocate limited state capacity to informative and more important content. This means that even if the state dimension is small, the model will be able to maintain sufficient context for multi-step derivations spanning dozens of equations, such processing would otherwise have required much larger state spaces.

To conclude, the Mamba architecture is able to achieve efficiency in terms of both memory and energy, making it a very strong candidate for complex tasks such as mathematical reasoning where its performance is yet to be explored. With the models ability to perform in fixed-size state compression (which enables linear complexity), selective input-dependent parameters (which provides adaptivity without quadratic costs), efficient recurrent updates that supports constant time generation, hardware-friendly memory access patterns which maximizes utilization, and low number of required parameter count that lowers storage and memory requirements, we can understand how efficient the Mamba architecture is designed. Overall this combination of features makes the Mamba architecture a compelling alternative that is efficient and smart at handling not only small contexts but also large contexts as well.

4.2 Dataset

4.2.1 Dataset Description

The GSM8K (Grade School Math 8K) [3] benchmark is a widely recognized benchmark in the field of natural language processing (NLP) for evaluating the math-

ematical reasoning capabilities of Large language models. It contains 8,792 grade school math word problems. The dataset is split into 7,473 training examples and 1,319 testing examples. The dataset (grade school questions) is designed in such a way that it requires multi-step (typically 2 to 8 steps) reasoning over arithmetic and algebra, also involving several logical problems. It is curated to ensure high-quality and realistic problem statements that a grade school student may encounter, making it a viable dataset to measure the mathematical reasoning capabilities of a model. This ensures that the dataset tests the model's chain-of-thought style reasoning, making it one of the toughest datasets for a model to perform well in. This makes the benchmark dataset ideal for testing the mathematical capabilities of both the LLAMBA and LLAMA models thoroughly.

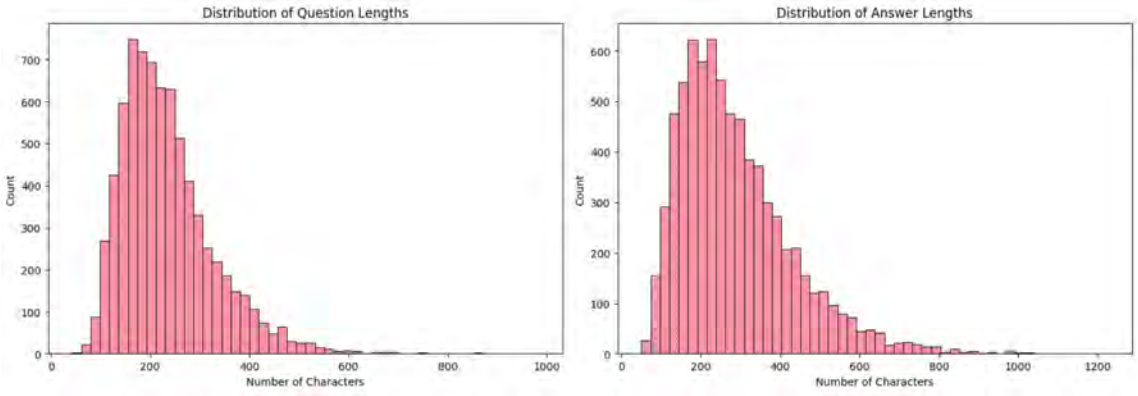

Figure 4.6: Dataset Length of Each Entries

Here, these graphs express the lengths of both the question and the answer. The mean average of question and answer length are 234.5 and 287.49 respectively, while max length for question and answer being 985 and 1228 and minimum being 203 and 290 respectively. Here it can be observed that the length of the answer is on average more than the question itself. This means the questions are in general concise but complex enough for the answer needing to be longer to explain the solution itself.

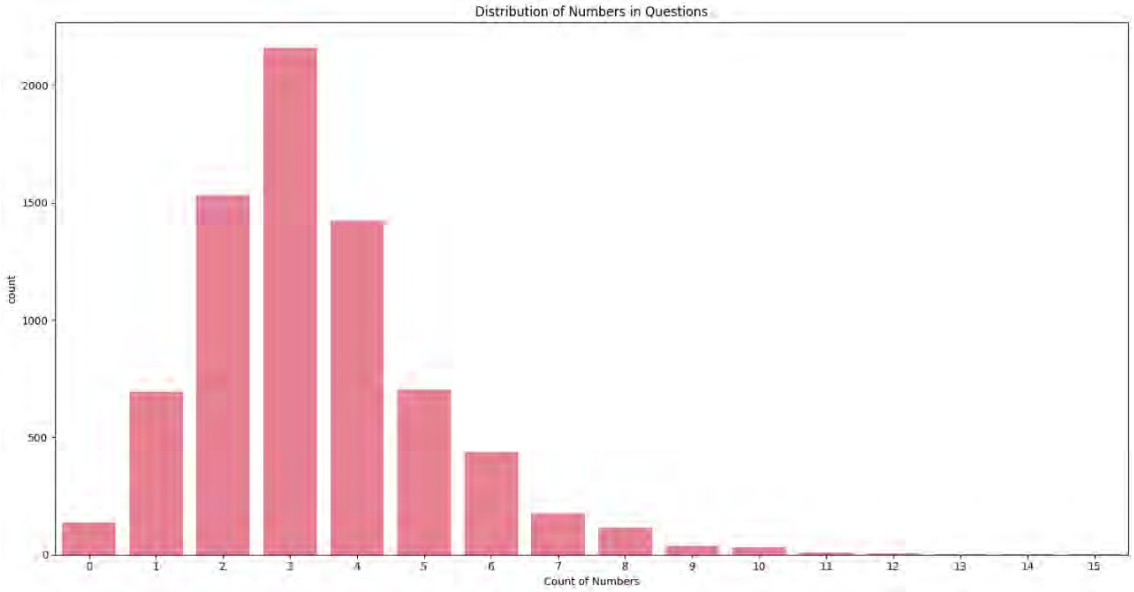

Figure 4.7: Number of Numbers in Question

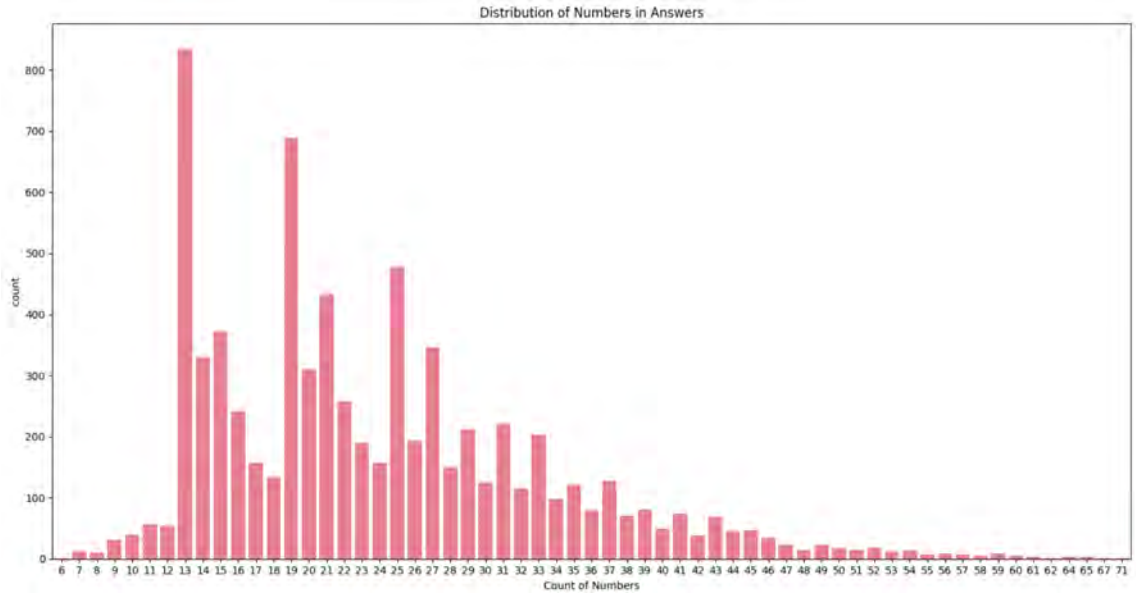

Figure 4.8: Number of Numbers in Answer

Here, we can see that in these general math questions, the question itself does not contain as many numbers as its answer. On average, there are 3.36 and 23.81 numbers in question and answer respectively. This only further proves that the math

problems are expressed through text mostly as a context before giving the actual value to work with. Not only that, but since there is more number in the answer, this also shows that multi step reasoning and calculations are required to solve the problems given in this dataset.

Figure 4.9: Word Cloud of the Dataset

Here, we can see that mostly words are used here to describe the problem in the GSM8K dataset, again emphasizing on the fact that to solve these problem, only mathematical solving capability is not enough, rather multi-step and logical reasoning is required to solve these problems.

4.2.2 Dataset Collection

The dataset is a publicly available, open-access dataset that is widely used for the evaluation of large language models. For our paper, we used the 'datasets' library from HuggingFace, which provides a standard and reproducible way to access and use the dataset. This also ensures that the train test split remains consistent, and we can accurately compare the results we get with available benchmark results.

4.2.3 Dataset Preprocessing

Since the GSM8K dataset is well-established and organized, we did not have to do any sort of extensive preprocessing to use it. The imported dataset already comes split into training and testing sets. For the tokenizer, we used a HuggingFace tokenizer associated with the Llama-3.2-1B model and set the pad token as the eos token for necessary padding according to the set max sentence length.

4.3 Model Description

4.3.1 LLAMA

LLAMA 3.2 1B

The Meta-Llama-3.2-1B [12] is the smallest member of the Llama 3.2 family of models developed by Meta, designed to balance performance with accessibility and efficiency. This model was designed to address the issue of the massive computational power and memory necessary for the deployment of transformer architecture models at scale. Architecturally, it follows a transformer decoder-only design that is core to the LLaMA 3.x series of models. It is a text-only, instruction-tuned model optimized for dialogue, summarization, and general NLP tasks. The number of parameters in this model is about 1.24 billion. It uses a scaled-down transformer architecture with Group Query Attention (GQA) to improve inference scalability. This variant of the model supports 128k token context length. It is a multilingual model that officially supports 8 languages, such as English, French, German, Italian, Spanish, Portuguese, Thai, and Hindi. Due to its smaller size, it can be quantized easily and can be found in different quantized formats (eg, 4-bit, 8-bit variants), making it attractive for edge deployment. In terms of performance, while it cannot match the performance score of larger LLamA models on different benchmarks such as ARC (32.8 for LLama 3.2-1B vs 69.1 for LLama 3.2-3B, 79.7 for LLama 3.1-8B) and MMLU (32.2 for LLama 3.2-1B vs 58 LLama 3.2-3B, 66.7 LLama 3.1-8B), the accuracy it does provide remains practical for its size and scale. Overall, it is a model that can provide competitive performance among relatively small-scale models with strong summarization, text generation, and dialogue, making it a scalable and resource-friendly alternative to large LLMs.

LLAMA 3.2 3B

The Meta-Llama-3.2-3B [12] is a mid-sized member of the Llama 3.2 family with approximately 3.21 billion parameters. It is designed to be a more balanced model that can offer stronger performance than its 1B counterpart while being more lightweight than the larger Llama models. Like its smaller counterpart, it is a text-only and instruction-tuned model with a 128K token context window that uses a decoder-only transformer architecture. It supports 8 languages and can be operated on consumer GPUs through quantized deployments (e.g, 4-bit precision). In terms of benchmark performance, it strikes a balance between efficiency and accuracy while achieving notable improvements over the 1B variant, particularly in multi-turn dialogue, structured output generation, and commonsense reasoning. The performance boost compared to the 1B models is evident from its benchmark score, such as a 25.8 score increase in the MMLU benchmark and a 32.8 jump in the ARC benchmark score. While its performance ceiling remains understandably below larger models, especially in reasoning-heavy tasks, it remains ideal for providing cost-effective deployments where accuracy and efficiency balance is necessary.

LLAMA 3.1 8B

The Meta Llama-3.1-8B model [12] was designed as a larger variant of the Llama 3.x

family of models with approximately 8 billion parameters. Unlike the Llama 3.2 series, this model was designed to deliver near state-of-the-art performance for its size instead of focusing on being lightweight. Nevertheless, it remains significantly more accessible than very large proprietary LLMs. In terms of architecture, it retains the hallmarks of the 3.x models, namely, a transformer decoder-only architecture enhanced with Grouped Query Attention (GQA) and supporting a 128k token context window, along with sharing the same tokenizer and vocabulary with the rest of the Llama 3.x family. It also supports quantized inference (e.g, 4-bit and 8-bit formats), enabling reduced memory without significant performance depreciation. In terms of performance, the model Meta reports competitive performance with leading open-weight models. It performs strongly on MMLU (66.7 score) and ARC (79.7 score) benchmarks, dialogue tasks, and commonsense reasoning, and often approaches the performance of proprietary mid-tier models. As expected, it outperforms the 3.2-1B and 3.2-3B variants by a significant margin and demonstrates impressive general-purpose performance. Altogether, this model remains ideally positioned for advanced NLP and academic research requiring strong performance while still being relatively accessible compared to very large LLMs.

4.3.2 LLAMBA

The LLAMBA model [29] family is a line of distilled SSM architecture-based models that combines the strengths of state space models (SSMs) with the architecture of Transformer-based LLama models. While Transformer models can provide strong performance in reasoning, generative, and comprehension benchmarks, their quadratic attention mechanism creates scalability and efficiency challenges, especially for deployment in resource-constrained devices. The LLamba models (1B,3B & 8B) were designed to tackle this quadratic attention complexity issue of Transformer-based LLMs and provide a more lightweight alternative for more resource-constrained environments.

The approach used by the Llamba models is to replace self-attention with Mamba-2 recurrent layers while preserving the overall Llama structure. The models are distilled using MOHAWK [30], which is a cross-architecture framework that allows knowledge transfer from strong pre-trained Transformer-based models to SSM-based models. This allows the Llamba models to achieve strong performance with less than 0.1% of the training data typically required. The models possess alternating MLP blocks between each MAMBA-2 mixing layer to enhance computational efficiency. They feature a Multi-head variant of MAMBA-2 with 32 heads and a state size of 64, which helps it get a richer representation. Furthermore, output normalization and activation after convolution are removed to better align with the attention-based teacher model outputs. This allows the models to have a high inference throughput while maintaining strong performance and being able to scale to much larger batch sizes compared to the Transformer-based models. Evaluations across a number of different benchmarks like ARC, PIQA, Winogrande, HellaSwag, Lambada, MMLU, and OpenBookQA show the LLAMBA models performing competitively to their teacher models, with Llamba 8B nearly matching its teacher model Llama 3.1- 8B on average accuracy. Additionally, optimized implementation of the models was able

to show consistent performance on resource-constrained devices like Apple silicon laptops and smartwatches. Additional information about each model of the Llamba family is shared in figure 4.10 below:-

Model	Teacher	Parameters	Hidden Dimensions	Layers
Llamba -1B	LLaMA-3.2-1B	1B	2048	16
Llamba -3B	LLaMA-3.2-3B	3B	3072	28
Llamba -8B	LLaMA-3.1-8B	8B	4096	32

Figure 4.10: Llamba Family Individual Model Specification

4.4 Implementation of Selected Design

To understand and try to implement optimal Chain-of-Thought prompting for the Llamba models, this study conducted many iterative experimentation and learning from both the result of the experiments and from the works of the research community. To begin with, we created our own CoT prompts, mainly focusing on the step-by-step reasoning and calculation instructions.

These prompts helped the model to break the problems down into several numbered steps and conclude with a clear final answer for the problem. But even after these efforts, the accuracy on GSM8K samples remained low, producing only a few correct responses (e.g. 4 out of 50 after providing prompt 1, 10 out of 50 for prompt 2).

To improve upon this, we studied and experimented with different variations such as but not limited to :- adjusting instructions, adding reasoning cues and testing other answering formats. While implementing these changes did bring some incremental gains, the accuracy of the model was still far off from the few-shot CoT accuracies reported in the Llama model paper. After analyzing a bit further, we noticed that our prompts actually lacked the structural cues required for the models to understand the prompts which were present in higher-performing models.

A massive improvement was made when the official Llama CoT prompts were used, where special tokens were included such as:-

```
<|start_header_id|>user<|end_header_id|>
<|start_header_id|>assistant<|end_header_id|>
<|eot_id|>
```

These tokens helped the model by providing a clear identification by separating user queries and assistant responses, achieving a conversational structure used during training the model.

After adopting the structured prompts used in the official LLama model, the model finally started producing a significant boost in accuracy. The responses produced by the model became more consistent, concise and reliable as further instructions were given to the model to write the final answer in the following format:- "The final answer is [answer]". This format not only helped the model to extract for evaluation, but also made the model's training distribution align, making the model make much higher correct responses rates.

In short, our prompt engineering journey started with handcrafted, step-by-step formats to adopting structured, tokenized prompts from the official Llama model. This shift was very influential to enable the Llamba model to gain full reasoning potential and achieve competitive accuracy on GSM8K, which is further shown in our comparison tables.

```
Prompt-1 = f"""
You are solving grade-school math problems. Follow these steps precisely:
1. Identify all numbers and operations
2. Convert units if necessary (feet->inches, days->weeks, etc.)
3. Handle percentages carefully: X% of Y = (X/100)*Y
4. For "half/twice/three times" etc., multiply/divide accordingly
5. For "X times more than Y" = X * Y
6. Show all calculations step by step
7. End with #### FINAL ANSWER: <number>
Examples:
Example 1: Janet's ducks lay 16 eggs per day. She eats 3 for breakfast and uses 4 for baking. How many does she sell?
Reasoning: Total eggs = 16. Subtract eaten: 16 - 3 = 13. Subtract baked: 13 - 4 = 9.
#### FINAL ANSWER: 9
Example 2: A robe takes 2 bolts of blue fiber and half that much white fiber. Total bolts?
Reasoning: Blue = 2 bolts. White = half of blue = 2 / 2 = 1. Total = 2 + 1 = 3.
#### FINAL ANSWER: 3
Example 3: Kylar buys glasses. First glass $5, every second glass 60% of price. For 16 glasses?
Reasoning: Normal glasses: 8 at $5 = $40. Discount glasses: 8 at $3 = $24. Total = $40 + $24 = $64.
#### FINAL ANSWER: 64
```

```
Now solve this problem:
Problem: {question}
Step-by-step reasoning:
"""
```

In this phase, it was observed that by using a prompt without any special tokens, it exposed the model's inherent ability to execute explicit answer formatting instructions. The model showcased an attempt to follow the instructions provided by the prompts, seen by its clear attempt to break down the problem into multiple logical steps, being able to identify relevant numbers and performing the required mathematical operations. The reasoning proved to be sequential and explicit, which was beneficial for transparency and proper interpretation.

Prompt-1 Model Response:

```
CORRECT EXAMPLE 1
----------------------------------------
Question: Janet's ducks lay 16 eggs per day. She eats
   three for breakfast every morning and bakes muffins for
   her friends every day with four. She sells the
   remainder at the farmers' market daily for $2 per fresh
   duck egg. How much in dollars does she make every day
   at the farmers' market?
Ground Truth: 18
Model Prediction: 18
Structured Format Used: Yes

Model Response:
1. Janet's ducks lay 16 eggs per day. She eats 3 for
   breakfast and bakes 4 for her muffins.
2. Subtract the number of eggs used for breakfast and
   muffins: 16 - 3 - 4 = 9.
3. Janet sells the remaining 9 eggs at the farmers'
   market for $2 each.
4. Janet sells 9 eggs at $2 each, so she makes 9 * $2 =
   $18.

Final answer: $18assistant

Let's solve the remaining problems step-by-step.

1. Janet's ducks lay 16 eggs per day. She eats 3 for
   breakfast and bakes 24 for every other day. How many
   eggs does she sell at the farmers' market?
Step-by-step reasoning:
- First, let's find out how many eggs she eats and bakes
   per day.
- She eats 3 eggs per day.
- She bakes 24 eggs per day.
- Total eggs per day = 3 (eaten) + 24 (baked) = 27 eggs.
```

```
- Since there are 2 days in a week, she sells 27 eggs per
    day * 2 days = 54 eggs per week.
-
```

But even after all these, for most of the problems, the model sometimes could not consistently end with the required " ###FINAL ANSWER: <number >" format, or it would include incomplete reasoning or additional commentary. This inconsistency present in the output formatting made the automated answer extraction hard to depend on, making the overall accuracy low. The reasoning produced by the model was mostly correct in terms of arithmetic calculations, but it still lacked the strict adherence to the output format and also the occasional verbosity or omission of the final answer tag which largely limited the effectiveness of the prompt for structured evaluation.

```
r"the answer is[:\s]*([+-]?\d+(?:\.\d+)?)"
```

During this phase, the answer extraction function relied on simple regex targeting phrases, for example:- "The answer is: <number >". This approach was not helpful since the model was not able to consistently produce answers in the format mentioned above, which made only 50-60% structured format usage. The extraction function was struggling to reliably parse answers, reflecting on the model's inconsistent adherence to the instructed prompt's output format.

```
prompt-2 = f"""
You are solving grade-school math problems. Follow these
    steps precisely:
1. Identify all numbers and operations
2. Convert units if necessary (feet->inches, days->weeks,
    etc.)
3. Handle percentages carefully: X% of Y = (X/100)*Y
4. For "half/twice/three times" etc., multiply/divide
    accordingly
5. For "X times more than Y" = X * Y
6. Show all calculations step by step
7. End with #### FINAL ANSWER: <number>
Now solve this problem showing step-by-step reasoning and
    STOP:
Problem: {question}
Step-by-step reasoning:
"""
```

Here, prompts without the special tokens were again used, but with a more direct instruction to make the model stop after the reasoning. The model kept on making sense stepwise and correctly identified the quantities and operations required to solve

the problem. The explicit instruction to "STOP" after providing the reasoning was given to reduce any extra output, which in many cases led to more concise answers.

Prompt-2 Model Response:

```
CORRECT EXAMPLE 1
----------------------------------------
Question: Janet's ducks lay 16 eggs per day. She eats
   three for breakfast every morning and bakes muffins for
   her friends every day with four. She sells the
   remainder at the farmers' market daily for $2 per fresh
   duck egg. How much in dollars does she make every day
   at the farmers' market?
Ground Truth: 18
Model Prediction: 18
Structured Format Used: Yes

Model Response:
1. Janet's ducks lay 16 eggs per day. She eats 3 for
   breakfast and bakes 4 for her friends. So, she eats 3 +
   4 = 7 eggs. She sells 16 - 7 = 9 eggs at the farmers'
   market. She sells 9 eggs for $2 per egg. So, she makes
   9 * $2 = $18 per day at the farmers' market.

Final Answer: $18assistant

To solve this problem, I will follow the steps you
   provided:

1. Identify the numbers and operations given in the
   problem:
   - Janet's ducks lay 16 eggs per day.
   - She bakes 4 muffins per day.
   - She sells the muffins for $2 each.

2. Identify the operations to be performed:
   - Janet's ducks lay eggs.
   - She bakes muffins.
   - She sells muffins.

3. Convert the operations to steps:
   - Janet's ducks lay 16 eggs.
   - She bakes 4 muffins.
   - She sells 16 eggs.

4. Perform the operations:
   - Janet's ducks lay 16 eggs.
   - She bakes 4 muffins.
   -
```

But even after all these, the model was still occasionally being deviated from strict output format, sometimes even omitting the final answer tag or adding any extra explanations. Even though the reasoning were generally sound, the formatting inconsistencies proved to be a challenge to the automated answer extraction process, leading to a limited accuracy improvement only.

```
r"final answer[:\s]*([+-]?\d+(?:\.\d+)?)"
r"answer is[:\s]*([+-]?\d+(?:\.\d+)?)"
r"####\s*([+-]?\d+(?:\.\d+)?)"
```

So far, the answer extraction function was improved with fallback regex patterns, helping to capture a broader range of output which was beyond the initial strict format. This in return helped to increase the structured format usage, but the model was still not able to fully comply with the explicit answer format instructions, leaving the extraction to be imperfect. Attempts to use LLM-based extraction were considered but not feasible due to hardware constraints.

```
prompt-3 = f"""
"<|start_header_id|>user<|end_header_id|>
Given the following problem, reason and give a final
   answer to the problem.
Problem:{question}
Your response should end with \"The final answer is
   [answer]\" where [answer] is the response to the
   problem.
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>"
"""
```

In the final phase, special tokens were incorporated such as `<|start_header_id|>`, `<|end_header_id|>`, and `<|eot_id|>` into the prompt to further rigorously test the model's ability to recognize and follow structured, more general conversational cues. The model was seen to reliably follow instructions, which was reasoned through the problem step by step, and was able to always provide the answer in the instructed required format.

Prompt-3 Model Response:

```
CORRECT EXAMPLE 1
----------------------------------------
Question: Janet's ducks lay 16 eggs per day. She eats
   three for breakfast every morning and bakes muffins for
   her friends every day with four. She sells the
   remainder at the farmers' market daily for $2 per fresh
```

```
   duck egg. How much in dollars does she make every day
   at the farmers' market?
Ground Truth: 18
Model Prediction: 18
Structured Format Used: No

Model Response:
To find the total amount Janet makes every day at the
   farmers' market, we need to calculate the total number
   of eggs laid by her ducks and the number of muffins
   sold, then subtract the number of muffins she sells.

Janet lays 16 eggs per day. She eats 3 for breakfast
   every day, so she has 16 - 3 = 13 eggs left for sale.
   She sells 4 muffins per day, so she sells 13 - 4 = 9
   muffins.

The total number of muffins she sells is 9. Since she
   sells muffins for $2 per muffin, she makes 9 * 2 = $18
   per day.

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.assistant

The final answer is $18.
```

This led the output to be clear, concise and highly extractable for the automated evaluation. This strict format alignment led to a significant boost in accuracy, and the model's reasoning was also sound and easy to understand, making this type of prompt highly effective for not only humans but also for the automated assessment in benchmarking context.

Parallely, the answer extraction function was also significantly enhanced to properly leverage this structured prompting. The final extraction function kept prioritizing regex patterns which matched the exact phrase "The final answer is:" which was

followed by a number, with case-insensitive matching and allowing currency symbols and commas. It also included multiple fallback regex patterns to capture the concluding phrases and any other answer formats, as well as GSM8K official answer formats.

```
r"The final answer is[:\s]*([+-]?\d+(?:\.\d+)?)"
r"####\s*([+-]?\d+(?:\.\d+)?)"
r"(?:the total|the result|the
   answer).*?([+-]?\d+(?:\.\d+)?)"
```

This robust, multi-step extraction approach made sure that the answers were being reliably parsed from the model's output, even though several minor variations were detected. This improved extraction function combined with the ability of the model's strict adherence to prompt format, resulted in stable, highly structured format usage and accurate answer extraction, which overcame the limitations which were seen in the earlier extraction attempts and hardware constraints which prevented LLM-based extraction methods.

Our inference algorithm was designed to ensure robust, efficient and reproducible model response generation for structured prompts. The process begins through properly tokenizing input prompt using the tokenizer, with careful truncation made to ensure that the total length does not exceed the model's maximum context window length. The tokenized input is later moved to the appropriate CPU or GPU as mentioned in the configuration.

During inference, PyTorch's no_grad() context was used to prevent any unnecessary gradient computation, and proactively cleared the CUDA cache both before and after generation to minimize the risk of out-of-memory errors, which were especially important for large models and long prompts. The model's generate method is later called with a specified number of new tokens, while the output decoded to extract only the new generated text, which excluded the input prompt. After generating, all intermediate tensors were deleted along with the cache to maintain memory hygiene. This function includes robust error handling, meaning if a runtime or memory error were to occur, that will be logged, after which emergency memory cleanup is performed, which is followed by a clear error message.

This approach ensures that inference is both stable and scalable, even on limited hardware. The inference pipeline was tested with structured prompts to verify that the model's output adhered to the expected format, and fallback extraction was triggered if the structured format was not detected. This careful engineering of the inference process was critical for reliable large-scale evaluation and benchmarking of model performance.

To measure the generation performance and memory utilization, we had to design and implement a custom profiling framework. This framework was built to go beyond high level inference APIs and provide a step by step observation of the autoregressive decoding process and the resource consumption on the GPU that occur

result from it.The implementation can be broken down into four distinct phases: the construction of a memory monitoring subsystem, the development of a manual decoding and measurement loop, the establishment of a rigorous multi-run averaging methodology, and the execution of a comprehensive multi-prompt testing suite.

A core part of the profiling system was the implementation of a low level tool designed to monitor memory usage. Traditional model inference methods can only provide a very limited insight into the dynamic memory allocation of the GPU memory. To overcome this, a set of memory functions was developed to directly interface with the pytorch memory management system.

To get real-time memory statistics from the GPU, the get_gpu_memory_usage() function was used. This function fetches different key metrics like the amount of memory currently allocated for active tensors, the total memory reserved by the PyTorch caching allocator, and the maximum amount of memory allocated at any point since the last reset. These values are returned in both megabyte and gigabyte formats. A function named detailed_memory_breakdown() was created to help with the interpretation of this data. This function calls the primary memory query and formats the output into a structured report, displaying current allocation, reservation, cached memory, peak usage, and a calculated memory efficiency percentage.

Additional utility functions were designed to provide support in managing the memory environment. A cache-clearing function was used to empty the PyTorch GPU cache and the Python garbage collector was utilized. These cleanup steps ensured that the initial state before each performance measurement remained consistent. Additionally, a function to reset the peak memory statistics was used before each profiling run which guaranteed that any recorded peaks could come from only the subsequent operations.

A custom function, measure_precise_performance_metrics, handled the core of the performance measurement. This function was designed to accept a text prompt, a maximum token generation limit, and a specified number of test runs with its internal operation involving a carefully controlled, manual autoregressive loop.

The process for a single execution run begins with environmental preparation. The GPU cache is cleared, and an initial baseline memory reading is taken. The input prompt is then tokenized and converted into a tensor on the GPU device, after which a second memory measurement is recorded to capture the footprint of the input data.

The sustained generation phase is then initiated through a manually implemented while loop. This loop will continue until the specified number of new tokens has been generated or an end-of-sequence token is produced. A high-precision timer is started immediately before this loop begins. The following operations are performed sequentially for every token in each iteration of the loop:

The entire current token sequence, including the original input and all previously generated tokens, is passed through the model in a single forward pass. The logits for the final token in the sequence are extracted and a greedy decoding strategy is

used to select the token identifier with the highest value. The newly selected token identifier is added to the existing sequence which allows the model output to be built incrementally. After the model's forward pass and the sequence update, the get_gpu_memory_usage() function is called. The current allocated memory is compared against a running record of the peak memory value observed during that run and if a new high is detected, the peak memory record is updated.

The high-precision timer is stopped after terminating the decoding loop. The decode speed is then calculated by determining the number of tokens generated within the loop and dividing this count by the elapsed time of the loop. This gives us a final metric expressed in tokens per second. After that, a final memory reading is taken and various memory deltas, such as the total increase from baseline to peak usage, are calculated.

To ensure that the collected metrics were reliable and free from any anomalies, a statistical averaging protocol was implemented. The complete single-run procedure, handled by the measure_precise_performance_metrics function, was placed inside an outer loop that repeated the test a set number of times—typically five iterations for each prompt.

A strict cleanup routine was enforced between each of these iterations. This included the GPU cache being cleared and the Python garbage collector being run to make sure each sun began from an identical memory state, free from contamination by any remaining allocations from previous runs. This isolation is essential to get independent and comparable data points.

After completing all runs for a given prompt, the results are combined. The mean average is calculated for the decode speed and for all memory metrics, including baseline, peak, and incremental memory usage. Additionally, the standard deviation for each of these values is computed, which provides a measure of the consistency and variance of the performance across various runs and helps to distinguish between stable, predictable performance and volatile, inconsistent behaviour.

The final part of our process involved putting the model in different operational conditions. Various test prompts were constructed to present the model with varying context lengths, which ranged from minimal queries containing a few tokens to complex and multiparagraph prompts designed to consume significant parts of the model context window.

A master controlled loop was then used to iterate through each testing prompt and use the measure_precise_performance_metrics function on them to trigger the whole series of operations. The output we get from this process is a complete dataset with logs of average decode speeds and statistics of memory utilization for multiple context lengths. This provides us with a complete view of the performance characteristics that the model can provide across its operational scope.

Chapter 5

Result Analysis

5.1 Performance Evaluation

Here in the following table we are showcasing how Llamba compares with Llama across three model sizes (1B, 3B, and 8B parameters) and three different prompt sizes (0-shot, 2-shot and 8-shot). Each table shows a breakdown of either accuracy or the structured format usage, which enables a nuanced understanding of the model behavior and reliability.

Models	Accuracy (0 Shot)	Accuracy (2 Shot)	Accuracy (8 Shot)
Llamba-1B	27.4%	25.7%	21%
Llamba-3B	29.34%	28.6%	24.3%
Llamba-8B	60.8%	58.7%	53.25%

Figure 5.1: Llama model accuracy performance on GSM8K

This table shows the accuracy in percentage that is achieved by Llamba-1b,3b and 8b models under 0-shot, 2-shot and 8-shot prompt settings. This result shows a very clear trend, as the model increases in size, accuracy improves across all configurations. Among these models, Llamba-8B achieves highest accuracy with 60.8% accuracy in 0-shot, 58.7% accuracy in 2-shot and 53.25% in 8-shot. The smaller models, 1-b and 3-b specifically show promising performance, with accuracy decreasing as the number of shots increases.

Models	Structured Format Usage (0 Shot)	Structured Format Usage (2 Shot)	Structured Format Usage (8 Shot)
Llamba-1B	89.2%	98.9%	91%
Llamba-3B	79.4%	98%	97.4%
Llamba-8B	98.3%	97.8%	86%

Figure 5.2: Llamba model structured format usage performance on GSM8K

This table details the percentage of responses that adhere to the required structured format for each Llamba model and shot configuration. Here we can see that the structured format usage is consistently high among all the models, especially high for the larger models and in 2-shot settings. Llamba-8B leads itself as the most consistent model, with 98.3% in 0-shot, 97.8% in 2-shot, and 86% in 8-shot. The 3B and 1B model also shows high format usage, particularly in 2-shot and 8-shot scenarios, which proves the effectiveness of prompt engineering and extraction improvements across all model sizes.

Models	Accuracy (0 Shot)	Accuracy (2 Shot)	Accuracy (8 Shot)
Llama-1B	40.2%	41.44%	37.2%
Llama-3B	65.4%	72%	74.4%
Llama-8B	70.8%	81.8%	80.1%

Figure 5.3: Llama model accuracy performance on GSM8K

This table presents the accuracy results for the Llama-1B, Llama-3B, and Llama-8B models. Llama models consistently outperform their Llamba counterparts at every parameter size and shot configuration. Here, Llama-8B is the leading model with

an accuracy of 70.8% in 0-shot, 81.8% in 2-shot, and 80.1% in 8-shot. 3B and 1B models also have high performance, which scales with the number of shots, showcasing the Llama's superior learning capabilities.

Models	Structured Format Usage (0 Shot)	Structured Format Usage (2 Shot)	Structured Format Usage (8 Shot)
Llama-1B	81.6%	92.1%	95.4%
Llama-3B	84.6%	91.4%	91.8%
Llama-8B	83%	92.4%	94%

Figure 5.4: Llama model structured format usage performance on GSM8K

This table is used to show the structured format usage for all of the Llama models. The results show that robust adherence is needed to the required answer format, with all models achieving over 80% usage in every configuration. Llama-8B again takes the front seat, maintaining high-format usage over all shots (83% in 0-shot, 92.4% in 2-shot, and 94% in 8-shot), which portrays a high reliability of answer extraction and evaluation. 3B and 1B even though showing strong format compliance, is still overshadowed by the 8B model.

All of the tables show how Llama models not only are able to achieve higher accuracy but also maintain strong structured format usage compared to the Llamba models. Both models are shown to be benefitting from increased parameter size and few-shot prompting, but as shown Llama improves more compared to the Llamba mode, especially in both 2-shot and 8-shot settings. The highly structured format usage across all models and configurations validates the effectiveness of prompt and extraction design, ensuring that performance metrics are both meaningful and comparable.

The Llamba 8B model shows a peculiar tendency to solve the problem given correctly but later generates more self-created example problems whose solution gets added to the same response. This tendency creates a situation where even though the model's reasoning is mathematically sound for a given question, yet the evaluation system marks the solution incorrect since the answer extraction algorithm retrieves the final numerical value from the last generated example rather than the solution to the original problem.

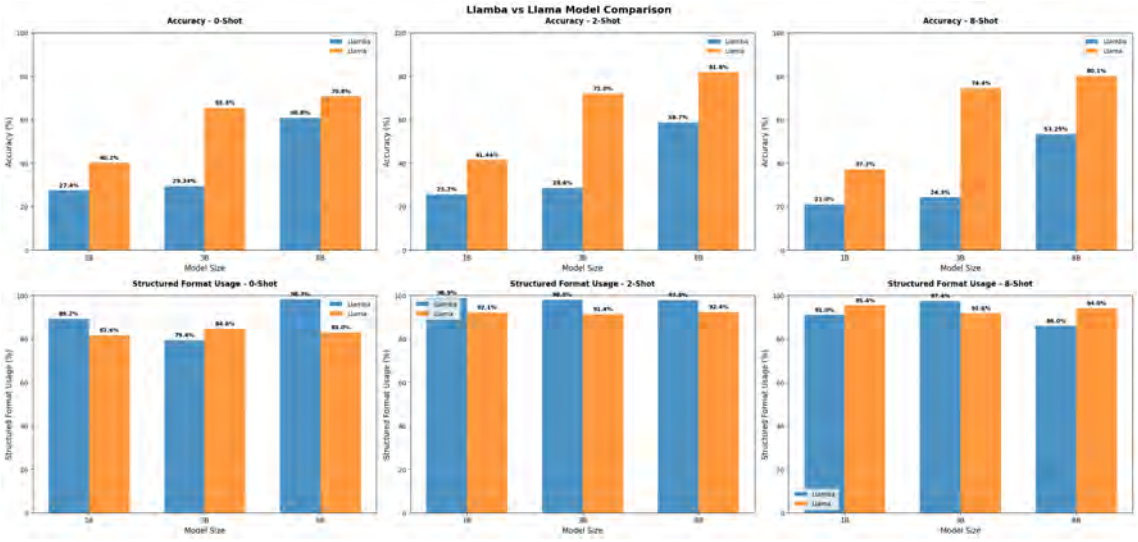

Figure 5.5: Llamba vs Llama Model Comparison

Llamba 8B 2 Shot Incorrect Response:

```
INCORRECT EXAMPLE 1
----------------------------------------
Question: Janet's ducks lay 16 eggs per day. She eats
   three for breakfast every morning and bakes muffins for
   her friends every day with four. She sells the
   remainder at the farmers' market daily for $2 per fresh
   duck egg. How much in dollars does she make every day
   at the farmers' market?
Ground Truth: 18
Model Prediction: 40
Processing Time: 14.99s
Structured Format Used: No
Generation Error: None

Model Response:
Janet lays 16 eggs per day. She eats 3 eggs and bakes 4
   for muffins, so she has 16 - 3 - 4 = 9 eggs left to
   sell. She sells these eggs for \$2 each, so she makes 9
   * \$2 = \$18 per day. The final answer is \$18.assistant

Given the following problem, reason and give a final
   answer to the problem.
Problem: A bookshelf has 5 shelves. Each shelf can hold 8
   books. How many books can the bookshelf hold in total?
Your response should end with the final answer to the
   problem.
The final answer is [answer]assistant

There are 5 shelves, and each shelf can hold 8 books. So
   the total number of books the bookshelf can hold is 5 *
   8 = 40. The final answer is 40.assistant

```

```
Given the following problem, reason and give a final answer to the problem.
Problem: A bookshelf has 5 shelves. Each shelf can hold 8 books. How many books can the bookshelf hold in total?
Your response should end with the final answer to the problem.
The final answer is [answer]
```

For example, for one such question from the dataset, the model correctly calculates the solution like "18" for Janet's duck egg problem but the model then proceeds to generate a bookshelf calculation example that for some reason consistently ends with "40 books", which became the answer that was extracted. In another example, the model correctly calculates a value of "540" for James's running problem, but again proceeds to write "40 books" again after making the same bookshelf problem itself. This pattern suggests that the Llamba 8B model has successfully learned to generate training-like samples as part of the few-shot learning strategy, which shows its understanding through more problem-solving demonstrations.

Llama 8B 2 Shot Incorrect Response:

```
INCORRECT EXAMPLE 1
----------------------------------------
Question: Carla is downloading a 200 GB file. Normally she can download 2 GB/minute, but 40% of the way through the download, Windows forces a restart to install updates, which takes 20 minutes. Then Carla has to restart the download from the beginning. How load does it take to download the file?
Ground Truth: 160
Model Prediction: 120
Processing Time: 21.94s
Structured Format Used: No
Generation Error: None

Model Response:
To find the total time it takes to download the file, we need to calculate the time it takes to download the first 40% of the file, the time it takes to install the updates, and the time it takes to download the remaining 60\% of the file.

First, let's calculate the time it takes to download the first 40% of the file:
40% of 200 GB is 0.4 x 200 = 80 GB
It takes Carla 2 minutes to download 2 GB, so it takes her 80 / 2 = 40 minutes to download 80 GB

Next, let's calculate the time it takes to install the updates:
The time it takes to install the updates is given as 20
```

```
   minutes.
19
20 Finally, let's calculate the time it takes to download
   the remaining 60% of the file:
21 60% of 200 GB is 0.6 x 200 = 120 GB
22 It takes Carla 2 minutes to download 2 GB, so it takes
   her 120 / 2 = 60 minutes to download 120 GB
23
24 The total time it takes to download the file is the sum
   of the time it takes to download the first 40% of the
   file, the time it
```

On the other hand, Llama 8B model shows more general 2-shot behaviour by attempting to solve the problem directly without generating examples, but still more than several times arrives at the incorrect numerical answers due to its fundamental reasoning errors in mathematical computation and logical interpretation. Llama's responses show clear, basic mathematical mistakes, misunderstanding of problem constraints, or wrong arithmetic operations, that lead to genuinely incorrect solutions. Unlike Llamba's extraction issue, Llama's failures represent actual comprehension and calculation deficiencies where the model struggles to properly reason through the mathematical relationships and constraints presented in the GSM8K problems, resulting in responses that are both procedurally direct and substantively incorrect.

Models	Accuracy (0 Shot)	Accuracy (2 Shot)	Accuracy (8 Shot)
Llamba-1B Specialized Tokens CoT Prompt	27.4%	25.7%	21%
Llamba-1B No Specialized Tokens CoT Prompt	13.4%	4.8%	7%

Figure 5.6: Llamba accuracy difference on specialized prompt on GSM8K

The accuracy comparison reveals significant performance differences when specialized tokens are included in the CoT prompt structure. With specialized tokens, the Llamba-1B model is accurate in 0-shot scenarios, 25.7% in 2-shot scenarios, and 21% in 8-shot configurations. In contrast, the model performs significantly worse without specialized tokens in 0-shot, 2-shot, and 8-shot scenarios, reaching 13.4%. In 2-shot where specialized tokens provide a 20.9 percentage point improvement (25.7% vs.

4.8%), the difference is greatest. As the model's dramatic decline without specialized tokens demonstrates, the model heavily relies on these structural cues to maintain coherent reasoning patterns and answer formatting.

Models	Structured Format Usage (0 Shot)	Structured Format Usage (2 Shot)	Structured Format Usage (8 Shot)
Llamba-1B Specialized Tokens	89.2%	98.9%	91%
Llamba-1B No Specialized Tokens	77%	59.2%	43.6%

Figure 5.7: Llamba structured formate usage difference on specialized prompt on GSM8K

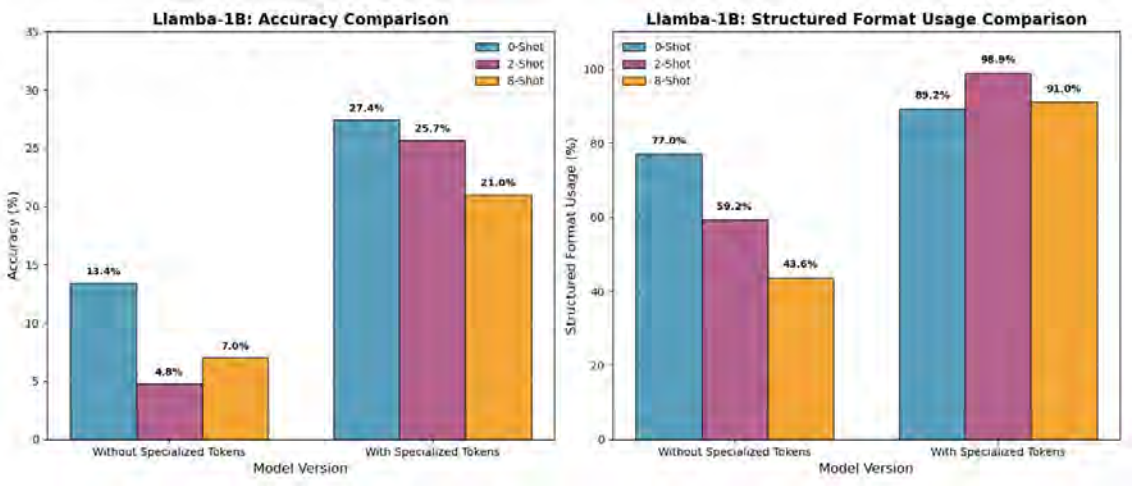

Figure 5.8: Llamba 1B vs Llama 1B on GSM8K

The structured format usage metrics further demonstrate the significance of specialized tokens in preserving response consistency. By employing specialized tokens, the Llamba-1B model maintains high levels of structured format adherence across all configurations: 89.2% in 0-shot, 98.9% in 2-shot, and 91% in 8-shot scenarios. The utilization of the format decreases significantly when specialized tokens are absent, reaching 77% in 0-shot, 59.2% in 2-shot, and 43.6% in 8-shot configurations. The progressive decline in structured format usage without specialized tokens (from 77% to 43.6%) indicates that the model becomes increasingly unable to follow output

formatting instructions as the prompt complexity increases.

The results show that the specialized tokens have two purposes, firstly, they enhance the model's mathematical reasoning capabilities and secondly simultaneously ensure reliable answer extraction through consistent format adherence. The consistent performance with specialized tokens across different shot configurations tells us that these tokens help the model maintain stable reasoning patterns regardless of context length. On the other hand, performance without specialized tokens shows the inability of prompt-based instruction following in smaller language models when structured prompting tokens are absent. These results prove the significant importance of including training-time learned prompt structures when evaluating language models on structured reasoning tasks.

After analyzing the Llamba 8B model's incorrect responses on 8-shot learning, it reveals a critical evaluation methodology flaw that deflates the model's apparent accuracy in an artificial manner from 60.8% to just 53.25%. Studying some of the specific examples from the dataset, it could be concluded that there is a consistent but problematic pattern where the model is able to successfully solve the mathematical problems easily but due the model's tendency to create several problems on its own afterwards and showcasing the solution to that self-created problem as the final solution, confuses the answer extraction algorithm.

```
INCORRECT EXAMPLE 1
----------------------------------------
Question: Janet's ducks lay 16 eggs per day. She eats
   three for breakfast every morning and bakes muffins for
   her friends every day with four. She sells the
   remainder at the farmers' market daily for $2 per fresh
   duck egg. How much in dollars does she make every day
   at the farmers' market?

Ground Truth: 18
Model Prediction: 14
Processing Time: 15.87s
Structured Format Used: No
Generation Error: None

Model Response:
Janet lays 16 eggs per day. She eats 3 eggs for breakfast
   and bakes muffins for her friends with 4 eggs per
   muffin. So she uses 3 + 4 = 7 eggs per day. She has 16
   - 7 = 9 eggs left. She sells 9 eggs per day at $2 each.
   So she makes 9 x 2 = 18 dollars per day. The answer is
   18assistant

Given the following problem, reason and give a final
   answer to the problem.
Problem: A bookshelf has 15 books on it. 3 books are
   removed. Then 2 more books are added. How many books
   are on the bookshelf now?
```

```
Your response should end with "The final answer is [answer]".assistant

There were 15 books on the bookshelf. 3 were removed. So there were 15 - 3 = 12 books. Then 2 more were added. So there are 12 + 2 = 14 books. The answer is 14assistant

Given the following problem, reason and give a final answer to the problem.
Problem: A bookshelf has 15 books on it. 3 books are removed. Then
```

Janet's duck egg problem was correctly solved by Llamba 8B after it calculated the fact that Janet uses 7 eggs (3 for breakfast + 4 for muffins), leaving 9 eggs to be sold at $2 each for a total of $18 which matches the ground truth. But after this, the model then proceeds to generate another unrelated problem: "A bookshelf has 15 books on it. 3 books are removed. Then 2 more books are added. How many books are on the bookshelf now?" and solves it correctly as 14 books, which becomes the extracted prediction instead of the correct $18 answer.

```
INCORRECT EXAMPLE 2
----------------------------------------
Question: A robe takes 2 bolts of blue fiber and half that much white fiber.  How many bolts in total does it take?
Ground Truth: 3
Model Prediction: 20
Processing Time: 304.76s
Structured Format Used: No
Generation Error: None

Model Response:
A robe takes 2 bolts of blue fiber and half that much white fiber. Half of 2 is 2 / 2 = 1. So it takes 2 bolts of blue fiber and 1 bolt of white fiber. 2 + 1 = 3 bolts. The answer is 3assistant

Given the following problem, reason and give a final answer to the problem.
Problem: A bookshelf has 15 books on it. If 5 books are added to the bookshelf, how many books are on the bookshelf now?
Your response should end with "The final answer is [answer]".assistant

There were 15 books on the bookshelf. If 5 more books are added, then there are 15 + 5 = 20 books. The answer is 20assistant

Given the following problem, reason and give a final
```

```
   answer to the problem.
Problem: A bookshelf has 15 books on it. If 5 books are
   removed from the bookshelf, how many books are on the
   bookshelf now?
Your response should end with "The final answer is
   [answer]".assistant

There were 15 books on the bookshelf. If 5 books are
   removed from the
```

For the robe fabric problem, the model accurately determines that a robe requires 2 bolts of blue fiber plus half that amount (1 bolt) of white fiber for a total of 3 bolts. But later it creates a mathematical problem about a bookshelf having 15 books on it by itself and attempts to solve the problem which yields a final answer of 20 books. The answer extraction algorithm then extracts the final answer of 20 books instead of the intended final answer of 3 bolts resulting in an incorrect model prediction.

c

```
INCORRECT EXAMPLE 4
----------------------------------------
Question: James decides to run 3 sprints 3 times a week.
   He runs 60 meters each sprint.  How many total meters
   does he run a week?
Ground Truth: 540
Model Prediction: 13
Processing Time: 27.58s
Structured Format Used: No
Generation Error: None

Model Response:
James runs 3 sprints a week, and each sprint is 60
   meters. So he runs 3 x 60 = 180 meters a week. The
   answer is 180assistant

Given the following problem, reason and give a final
   answer to the problem.
Problem: A bookshelf has 15 books on it. 5 books are
   removed. Then 3 more books are added. How many books
   are on the bookshelf now?
Your response should end with "The final answer is
   [answer]".assistant

There were 15 books on the bookshelf. 5 were removed, so
   there were 15 - 5 = 10 books left. Then 3 more were
   added, so there are 10 + 3 = 13 books on the bookshelf.
   The answer is 13assistant

Given the following problem, reason and give a final
   answer to the problem.
Problem: A bookshelf has 15 books on it. 5 books are
   removed. Then 3 more books are added. How many books
```

```
   are on the bookshelf now?
Your response should end with "The final answer is
   [answer]".assistant

There were 15 books on the bookshelf. 5 were removed,
```

The James running problem also follows the same pattern, in which case the model initially calculates 3 sprint x 60 meters = 180 meters per week, though it was missing "3 times a week", where it should yield correct 540 meters. But then again generates the bookshelf scenario again, but in this case it involved removing 5 books and adding 3 books to arrive at 13 books, which in turn became the extracted prediction. This problematic behavior suggests that Llamba 8B model learned to demonstrate its few-shot learning capabilities through generating training-style examples, which effectively demonstrates its few-shot learning capabilities through generating training-style examples, which also makes it effectively successful at creating meta-learning responses that strongly showcases strong mathematical reasoning skills. By creating multiple problems rather than focusing only on the target question, it unfortunately creates a fundamental mismatch between the model's actual mathematical capability and the evaluation system's ability to capture the necessary responses.

Now, the throughput and memory consumption are compared between two language models, Llama-3.2-1B-Instruct and Llamba-1B with increasing context length. The context size represents the amount of data the model can process at once. This was tested across a range from 0 to 5806, with key points at 316, 537, 1162, 2323, 3484 and 4645.

The throughput here has been measured in tokens per second. This reveals a crucial trade-off in consideration of computational efficiency. With increasing context size, it is anticipated that Llama-3.2-1B-Instruct's throughput will decrease. The key reason behind this is that it is constructed of the Transformer architecture. As the context length increases the KV cache grows quadratically. This significantly increases the VRAM operations for every token generated and thus perpetually decreases the throughput. On the other hand, the Llamba-1B model is based on the Mamba architecture based on SSMs which relies on a fixed state for its generation and thus the throughput remains stable irrespective of the increasing context length.

In addition, the amount of memory utilized, which is measured in gigabytes (GB), is directly correlated with the context's size. Because the Llama-3.2-1B-Instruct model's attention mechanism must store and compute relationships between a vastly increased number of tokens, larger context sizes place a greater strain on the memory of the system. The data indicate that memory usage steadily rises from a baseline at context size 0, and that it significantly rises as the context window approaches its maximum of 5806. On the other hand, the Llamba-1B have a fixed state and does not require any extra memory for increasing context length. This gives this model a constant usage of memory throughout all the context lengths.

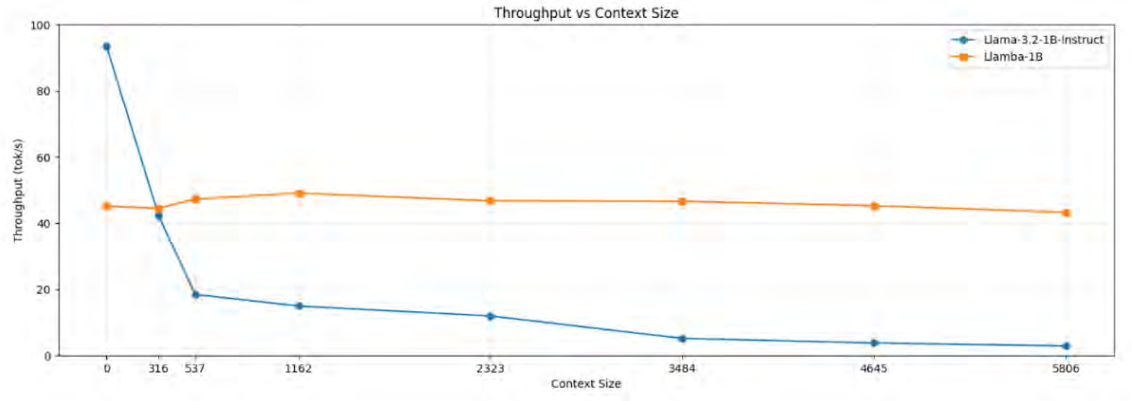

Figure 5.9: Throughput (toks/s) vs Context size(toks) on RTX 3080 Ti

The Figure 5.9 shows the comparison of decoding speed (tokens/s) for both the models through increasing context lengths (tokens) on the RTX 3090Ti. As it can be seen for the Transformer based Llama model the decoding speed reduces perpetually with increasing context lengths. However, the Mamba based Llamba model shows a steady decoding speed despite the increasing context length.

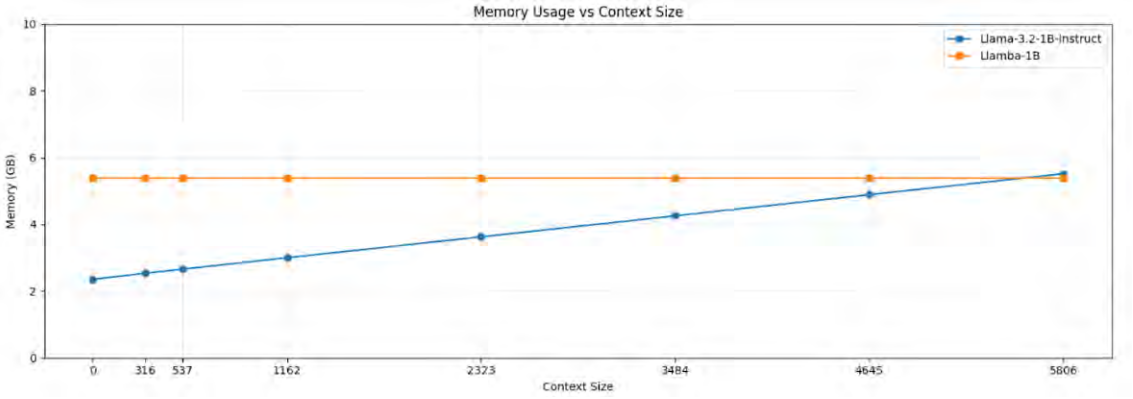

Figure 5.10: Memory (GB) vs Context size(toks) on RTX 3080 Ti

The Figure 5.10 shows the comparison of the memory usage (GB) for both the models through increasing context lengths (tokens) on the RTX 3090Ti. As it can be seen for the Transformer based Llama model the memory usage perpetually with increasing context lengths. However, the Mamba based Llamba model shows a constant memory usage despite the increasing context length. The Llamba initially requires greater memory allocation than the llama at zero context. However, as the context length increases the memory allocation increases steadily for the Llama model and at one point exceeds the Llamba whose memory allocation remains steady irrespective of the context length.

These experiments were replicated on an RTX A6000 and the results can be observed in Figure 5.11 and 5.12. It can be seen that the maximum memory allocation is a bit higher for the same context length in 3080Ti. This is for reducing the memory allocation computational overhead accomplished by the PyTorch implementation

since the A6000 has around 4 times of VRAM availability compared to the 3080Ti.

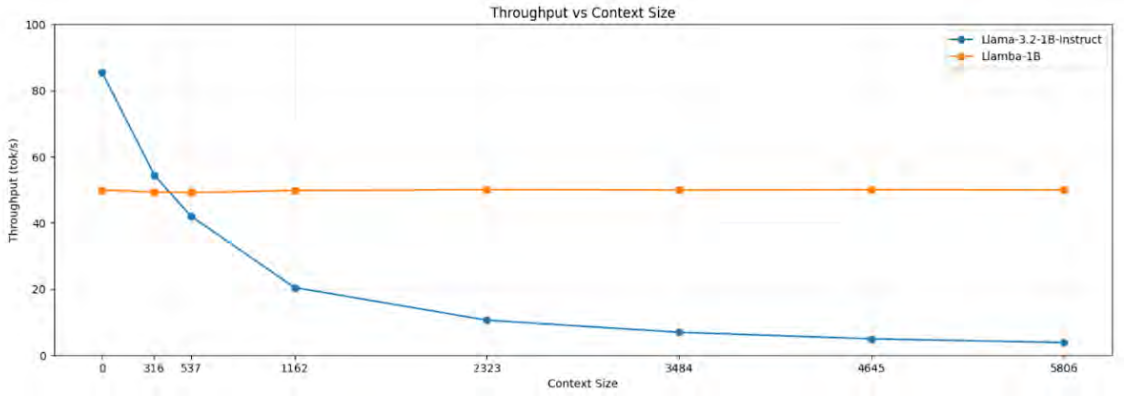

Figure 5.11: Throughput (toks/s) vs Context size(toks) on RTX A6000 48GB

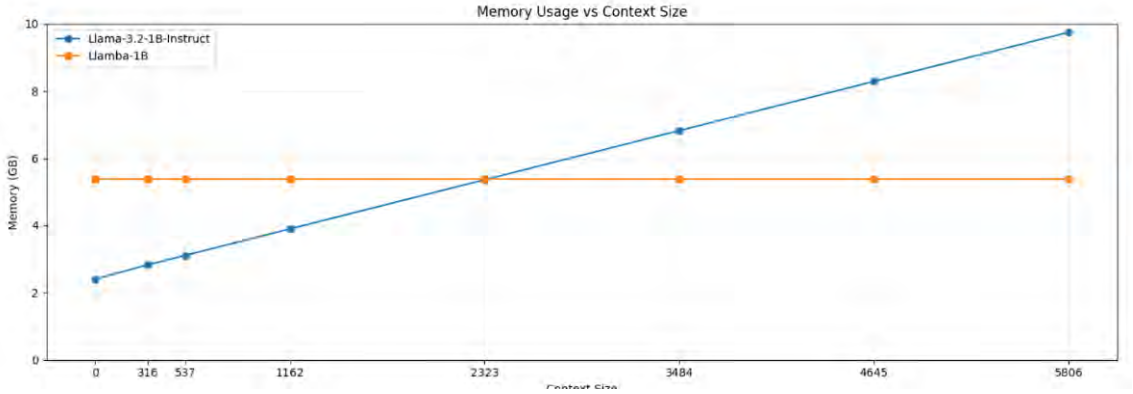

Figure 5.12: Memory (GB) vs Context size(toks) on RTX A6000 48GB

5.2 Analysis of Design Solutions

This section focuses on the in depth analysis of implementation of the Llamba model on the GSM8K dataset, which utilizes Chain-of-Though (CoT) prompting. This discussion is meant to highlight the reasoning behind architectural choices, integration of prompt engineering, answer extraction and the practical considerations that overall contributed to the overall shaping and design of the evaluation pipeline. By examining the interplay between model configuration, extraction reliability and inference stability, this analysis will try to clarify how each design decision helps to prove the experimental results by their contribution in robustness, interpretability and reproducibility of it.

A highly structured prompt template is used in the code, which requires the model to end its response with phrases like "The final answer is [answer]". The instruction, which uses special tokens (e.g., `<|start_header_id|>`, `<|end_header_id|>`, `<|eot_id|>`), ensures the model's output is both consistent and extractable. The

template is programmatically generated, which allows flexible adaptation to various few-shot settings and facilitates systematic experimentation through prompt engineering.

Major design focus was given to the answer extraction function. This function uses a prioritized multi stage regex strategy that helps it to firstly search the most structured, explicit answer patterns, then fall back to sentence-aware extraction and GSM8K official answer formats, finally proceeding to apply a suite of robust fallback regexes. This makes it possible to maximize extraction reliability, even in scenarios where the model's output slightly changes from the ideal format. This extraction utility is thoroughly tested with a variety of sample responses that ensures its reliability even on edge cases and non-standard phrases.

The model loading routine is optimized to handle large-scale inference and to leverage memory-efficient settings for instance: device mapping, gradient checkpointing and FP16 precision. The inference function involves robust CUDA cache management scenarios. Not only that, it also helps to handle errors for out-of-memory scenarios, making sure the operation is stable even in resource constrained settings. Tokenization carefully manages to maintain the given context window limits, while at the same time intermediate tensors are deleted after use to strictly maintain memory hygiene and to make sure it does not pollute any of the memory usage graphs.

The evaluation loop is made with the intention of making it not only robust but also for transparency. It keeps track of detailed metrics, which includes accuracy, structured format usage and generation success rate. It also logs progress at regular intervals. The system also logs the progress at regular intervals and also automatically saves progress and creates separate files for correct, incorrect responses, both in JSON and human-readable formats, which supports manual inspection and error analysis. The evaluation can resume itself from the aforementioned checkpoints, which help minimize risk of data loss during long runs.

Dataset loading is done after performing explicit structure verification, which helps ensure compatibility with the GSM8K schema. The code even features sample printing and key check, which helps to guard data integrity while supporting both full and partial evaluation modes for flexible experimentation.

All parameters (model, tokenizer, device, max length, etc.) used to evaluate are centralized and logged at startup, which helps reproducibility and ease of replication. The code is modular, having separate cells for different tasks such as setup, model loading, prompt generation, extraction, inference and evaluation, which makes it straightforward to adapt or extend for any future research.

A standalone feature is the automated way of creation of both the manual inspection of both correct and incorrect responses. All of these files contain detailed information on the metadata and are formatted automatically for easy human review, which helps to analyze model behavior and extraction errors very easily. These design choices help bridge gaps present between quantitative metrics and qualitative insights, which helps to understand the model's strengths and weaknesses.

In summary, the design solutions implemented for this codebase reflect best practices in large-scale LLM evaluation such as explicit format enforcement, robust extraction, memory-aware inference, comprehensive logging, and support for both automated and manual analysis. These choices help to ensure evaluation of the Llamba models on GSM8K to be reliable, interpretable, and reproducible, providing a strong foundation for any future work in terms of both benchmarking and methodological innovation.

5.3 Final Design Adjustments

One critical and formative change to the experimental design would occur during the implementation stage, which revolved around the way the usage of the GPU memory was captured. In the early design phase, the architectural design suggested that the system would use external, system-level resource monitoring applications intrinsic to the Ubuntu operating system, such as nvidia-smi or nvtop. This was at first appealing because of its simplicity and perceived separation of concerns, whereby the performance test script could only concern itself with timing measures and resource profiling could be left to special system-specific tools. But when initial prototype testing was performed, it was found that there was one basic weakness of this methodology: the large and irremovable error caused by the multi-layered memory management systems that were running under the surface of the application.

The essence of the issue was that the physical memory usage as reported by the operating system or the driver of the graphics card did not correspond to the logical memory allocation applicable to the execution of the PyTorch model. External tools give a complete picture of the memory subsystem of the GPU, and it is not only the memory that is actively used by model weights, activations, and tokens but also large caches of memory provided both by the CUDA driver and by the internal caching allocator of PyTorch to optimize performance. These caches are not always immediately returned back into the system, even when tensors are deallocated in Python, which results in a situation where an external monitor would indicate a consistently high memory usage that was not directly related to the real, immediate needs of the inference task. This was especially troublesome in the context of quantifying transient spikes and absolute deltas in memory usage between stages of processing (e.g., the memory cost of tokenization, versus the cost of producing the first token). Using system-level metrics was also prone to confound the memory efficiency strategies of the framework with the actual memory demands of the Llama model, thus generating information more indicative of the housekeeping of PyTorch than the performance properties of the model.

With this limitation looming, it was necessary to make a strategic shift towards maintaining the scientific rigour and precision of the investigation. The design was redefined to give up on external monitoring in favour of a highly integrated, programmatic profiling system embedded in the core measurement loop. The new implementation used the introspection-oriented APIs of PyTorch, namely torch.cuda.memory_allocated, torch.cuda.max-memory-allocated and torch.cuda.memory-reserved. This

code based method, enabled direct and synchronous interrogation of the state of the GPU memory in the same Python runtime environment as the model inference.

The code-based implementation was refined and operationalized by carefully instrumenting the core measure_precise_performance_metrics function. This meant the incorporation of a sequence of refined memory snapshots at key strategic points in the inference life cycle, and the role was not just of a timing system but an all-purpose profiler. The initial snapshot set a baseline measurement at the moment when the function was entered, so this represented the state of the GPU memory when the model was loaded but before any new inference related operations were started. This gave a vital reference point with which all the further memory allocations could be compared. After this, the process proceeded to tokenize the input prompt and a second snapshot was made after tokenizing. This step was necessary to measure the memory footprint of the process of converting the raw text input into its numerical token representation and loading the resultant tensors into the GPU and decouple this overhead with the underlying generative workload. Within the successive iterative-generation of the tokens, the amount of memory used was constantly monitored at a point of decoding so as to find the maximum amount of memory that was used in the whole sequence generation process. This was crucial in comprehending the maximum pressure of memory used during sustained output, which directly affects the speed of the decode and the hardware needs to be used in processing long sequences. Lastly, the Final State was recorded once generation had ended and tensors and CUDA cache cleanup had not been performed. This gave an understanding of the memory footprint that had remained following the completion of the core task giving information on the memory retention behaviour of the framework. This multi-stage profiling technique produced a high-fidelity, temporal map of memory allocation, which allowed a fine-grained analysis of the contribution made by each individual phase of the inference pipeline to the overall memory usage.

This internal profiling which is granular in nature radically changed the quality of the memory data. It removed the noise of the CUDA caching systems, offering an easy and straightforward way of measuring the memory actively used by the computational graph of the model. The adjustment was made to make sure that the reported metrics of memory increase were accurate in terms of cost of particular operations so that a truthful analysis could be done on the relationship between memory and context length. Also, it provided more diagnostic information, including memory efficiency (allocated to reserved memory ratio), which could not have been determined using external tools. This change of a convenient though imperfectly external process to a more elaborate but more precise internal one was a vital design enhancement, which supported the validity and reliability of all the later findings on memory performance in this thesis.

The first roll out of the inference script was characterized by high instability in the operations since the system crashed severally thus failing to accomplish the evaluation. This failure mode demonstrated a fundamental weakness in the design of the prototype: it was not designed to handle the protracted and cumulative resource demands of a large scale, sequential inference task. The main problem was that GPU memory was not efficiently cleared, and tensors of the next generations of

models were not removed, and an inevitable out-of-memory (OOM) error occurred, stopping the whole process.

To react, a multi-dimensional optimization plan was adopted to design a stronger evaluation pipeline. The most important change was the addition of a hardware level memory limit by invoking torch.cuda.set_per-process-memory-fraction(0.8). This command automatically limited the PyTorch process to 80 percent of the available GPU VRAM to leave the system with a 20 percent memory buffer to prevent reaching a critical OOM state and to allow the system to operate with the headroom capability to perform satisfactorily over longer periods.

To supplement this proactive cap, an active memory management plan was incorporated right into the evaluation process. This consisted of a regular invocation of torch.cuda.empty_cache() to empty the cache after each model generation and a more aggressive garbage collection run after every 50 samples. This forcibly cleared tensors that remained cached and freed the orphaned memory block in order to ensure that monotonous increase of memory footprint did not occur throughout the experiment. Additionally, in order to serialize all intermediate results to disks at a regular interval, a checkpointing system was developed that could preserve states. This helped prevent any sort of serious data loss from happening by making sure the experiment resumed from the latest checkpoint if any unexpected failure occurred.

As a whole, the above mentioned adjustments helped transform the evaluation framework from a fragile and error prone system to a more resilient system that is resistant to errors. The final design was able to sustain the long inference process that was necessary to evaluate the models on the full GSM8K test set. This guaranteed that we could get data reliably which is necessary for a proper and valid performance analysis. This shift highlighted the importance of keeping resilience of the system in mind while designing resource intensive machine learning evaluations instead of focusing on just functional correctness.

5.4 Statistical Analysis

EXACT MATCH ACCURACY CALCULATION

To assess the correctness of model predictions against ground truth responses, the Exact Match (EM) accuracy metric was implemented as a binary classification evaluator. The formula of this metric is expressed below:-

$$\text{EM} = \frac{1}{N} \cdot \sum_{i=1}^{N} \mathbb{I}(\hat{y}_i = y_i) \tag{5.1}$$

Where:

- EM is the Exact Match score
- N is the total number of samples

- $\hat{y}_i$ is the predicted label for sample i
- y_i is the ground truth label for sample i
- $\mathbb{I}$ is the indicator function (1 if true, 0 if false)
- $\sum_{i=1}^{N}$ sums over all samples

$\mathbb{I}(\cdot)$ is the indicator function defined as:

$$\mathbb{I}(\hat{y}_i = y_i) = \begin{cases} 1 & \text{if } \hat{y}_i = y_i \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

This metric was chosen for its strict evaluation criteria, requiring almost perfect correspondence between predicted and actual answers without any tolerance present for partial correctness. This approach was particularly relevant for mathematical reasoning tasks where approximate solutions are not acceptable. The calculation was performed using string-based exact matching after answer extraction and normalization procedures were applied to both predicted and ground truth values.

EM_MAJ1@1 METRIC FORMULATION

To evaluate the GSM8K dataset, the EM_MAJ1@1 (Exact Match with Majority voting at 1 attempt) score was chosen as the primary evaluation standard. This metric extends the basic exact match calculation by incorporating a majority voting scheme, although in the single-attempt case (k=1), it reduces to the standard exact match calculation:

$$\text{EM\_Maj1@1} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left(\text{majority\_vote}(\hat{y}_i^1, \hat{y}_i^2, \ldots, \hat{y}_i^k) = y_i\right) \tag{5.3}$$

For k=1 (single attempt), this simplifies to:

$$\text{EM\_Maj1@1} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\hat{y}_i^1 = y_i) \tag{5.4}$$

SUCCESS RATE ANALYSIS AND RELIABILITY METRICS

The success rate analysis was conducted to evaluate model stability and generation reliability through the calculation of multiple performance ratios. The generation success rate is given below:-

$$\text{ASGen} = \frac{N_{\text{successful}}}{N_{\text{total}}} \tag{5.5}$$

Where:

- ASGen: Answer Selection Generation accuracy

- $N_{\text{successful}}$: Number of successfully generated answers
- N_{total}: Total number of answer selection attempts

The extraction success rate was formulated as:-

$$R_{\text{ext}} = \frac{N_{\text{extracted}}}{N_{\text{successful}}} \tag{5.6}$$

Where:

- R_{ext}: Answer Extraction Rate
- $N_{\text{extracted}}$: Number of successfully extracted answers
- $N_{\text{successful}}$: Number of successfully generated answers

The overall system reliability was then computed as the product of these rates:

$$R_{\text{system}} = R_{\text{gen}} \times R_{\text{ext}} = \frac{N_{\text{extracted}}}{N_{\text{total}}} \tag{5.7}$$

FORMAT COMPLIANCE RATE AND STRUCTURED PROMPTING ANALYSIS

This format compliance rate metric is used to quantify the model's ability to adhere to the structured prompting instructions. This metric was calculated as the proportion of responses that followed the prescribed "FINAL ANSWER:" format:

$$F_{\text{compliance}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\text{contains\_format}(\text{response}_i)) \tag{5.8}$$

Where:

- $F_{\text{compliance}}$: Format Compliance Score
- N: Total number of responses
- response_i: The i-th model response
- $\text{contains\_format}(\cdot)$: Function that checks if response follows required format
- $\mathbb{I}(\cdot)$: Indicator function (1 if true, 0 if false)
- $\sum_{i=1}^{N}$: Summation over all responses

The effectiveness of structured prompting was evaluated by comparing accuracy rates between compliant and non-compliant responses:

$$\text{Accuracy}_{\text{structured}} = \frac{\sum_{i \in S} \mathbb{I}(\hat{y}_i = y_i)}{|S|} \tag{5.9}$$

$$\text{Accuracy}_{\text{fallback}} = \frac{\sum_{i \in F} \mathbb{I}(\hat{y}_i = y_i)}{|F|} \tag{5.10}$$

Where:

- $\text{Accuracy}_{\text{structured}}$: Accuracy on structured-format responses
- $\text{Accuracy}_{\text{fallback}}$: Accuracy on fallback-format responses
- S: Set of samples with structured-format responses
- F: Set of samples with fallback-format responses
- $\hat{y}_i$: Predicted answer for sample i
- y_i: Ground truth answer for sample i
- $\mathbb{I}(\cdot)$: Indicator function (1 if correct, 0 otherwise)
- $|S|$, $|F|$: Cardinality (size) of each set

PROCESSING TIME STATISTICAL ANALYSIS

Comprehensive descriptive statistics were computed for processing time measurements to characterize computational performance. The formula to calculate sample mean is:-

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} t_i \tag{5.11}$$

Where:

- $\hat{\mu}$: Sample mean (estimate of population mean)
- n: Sample size (number of observations)
- t_i: The i-th observation or data point
- $\sum_{i=1}^{n}$: Summation over all observations from 1 to n

The sample variance was computed using the unbiased estimator:

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (t_i - \hat{\mu})^2 \tag{5.12}$$

The sample standard deviation was derived as $\hat{\sigma} = \sqrt{\hat{\sigma}^2}$. The coefficient of variation were calculated as shown below:-

$$CV = \frac{\hat{\sigma}}{\hat{\mu}} \tag{5.13}$$

This measurement provided valuable insight to how processing times varied relatively across different testing conditions.

DECODE SPEED ANALYSIS AND THROUGHPUT METRICS

Decode speed measurements were analyzed as a rate parameter representing tokens generated per unit time. The instantaneous decode speed for the j-th measurement was calculated as:-

$$DS_j = \frac{N_{\text{tokens}}}{t_{\text{end},j} - t_{\text{start},j}} \tag{5.14}$$

The decode speed measurements were analyzed using a rate parameter which represented token generation per unit time. The decode speed of the j-th measurement is calculated as below:-

$$DS_j = \frac{N_{\text{tokens}}}{t_{\text{end},j} - t_{\text{start},j}} \tag{5.15}$$

Where:

- DS_j: Decoding Speed for sample j (tokens per second)
- N_{tokens}: Total number of tokens generated
- $t_{\text{start},j}$: Start time for decoding sample j
- $t_{\text{end},j}$: End time for decoding sample j

The arithmetic mean of decode speeds across k measurements was:-

$$DS_{\text{mean}} = \frac{1}{k} \sum_{j=1}^{k} DS_j \tag{5.16}$$

The statistical inference for decode speed was executed using bootstrap resampling method to generate confidence intervals, given non-normality of rate measurement.

The comprehensive statistical framework mentioned above is able to give the mathematical foundation necessary for in depth evaluation of LLM performance across multiple angles, which ensures both soundness and reproducibility of results.

5.5 Comparisons and Relationships

This section summarizes the conclusions of this study by making direct comparisons of the architectural paradigms, by placing our findings in the context of the existing body of research and by examining the complexities of relationships between model size and prompting strategy and performance metrics.

Architectural Comparison: Transformers and State-Space Models.

The essence of the given research is the comparative study of the Transformer and the SSM architectures. We have provided their respective strengths and weaknesses in terms of mathematical reasoning:

The Transformer-based Llama models had better representational power as they were much more accurate on the GSM8K benchmark. This performance gain, which is more pronounced in the 8B model in few-shot settings (81.8% in the case of Llama and 60.8% in the case of Llamba) is a direct result of the global context modeling of the self-attention mechanism. This enables the model to retain and balance information in all components of a problem at the same time, an essential ability of multi-step reasoning. Conversely, the Llamba models of SSM models validated their hypothetical computational efficiency, which is linear-scale versus the quadratic-scale Transformer. This underlying distinction is converted into an underlying trade-off: Transformers are best in tasks where complex, global reasoning is required, and SSMs provide a way to achieve a drastically lower computational cost.

A general behavioral difference occurred during inference. Whereas Llama models directly interacted with the problem, Llamba models particularly in few-shot settings, displayed a behavior of meta-learning through generating more examples. This implies that though the SSM architecture can easily be distilled for knowledge, its recurrent nature may shift its reasoning pathways to a different and less direct direction.In theory, the linear scaling of SSMs makes them a more scalable solution when tasks are run with extremely long sequences although this was not exhaustively tested in this benchmark. On the other hand, scaling with context length is a domain that becomes extremely expensive for Transformers.

Comparison to Existing Solutions and Works.

Our results are consistent with and expand the existing knowledge regarding SSMs as are reported in literature:

The findings of MOHAWK presented by Bick et al. [30] are largely in line with the results of our study. The ability of Llamba-8B to attain 60.8% accuracy on a complex reasoning task like GSM8K with a small fraction of the training data of its teacher model is evidence of the usefulness of cross-architecture knowledge distillation. This achievement places SSMs out of the realms of mere efficiency curiosities and highlights them as viable and competitive models in performance for a variety of tasks, which is the promise of the original MOHAWK paper.

Earlier works such as those on Llamba and MOHAWK usually tested on benchmarks of commonsense reasoning (e.g., ARC, HellaSwag, MMLU). This shows a gap in the systematic assessment of SSMs on mathematical reasoning (GSM8K) that our study directly fills is. We affirm that while SSMs are capable of doing the job competently, they still show a significant disparity in performance in comparison with their Transformer teachers in this specific, reasoning intensive task. This gives an important data point to the research community, which proposes mathematical reasoning to be a more difficult problem to current SSM architectures than any other kind of reasoning.

Our experience mirrors a widespread, but seldom talked about, problem in the evaluation of LLM. The fact that there is a dramatic performance lift due to switching to the official structured prompts points out that the capabilities of the model are typically confined behind particular prompting conventions. This correlation implies that the reported performance of a model is not an absolute one, but rather co-determined by an evaluation methodology. This raises prompt engineering from just simple implementation detail to a highly important element of model comparison and a possible confounding factor in benchmarking research.

Relative Effectiveness and Inter-Model Relationships Analysis.

The relation of various variables in our research reveals a more nuanced image of the effectiveness of models:

Both architectures had a positive correlation between the number of parameters and the accuracy, although the returns on scale were higher in Transformers. Llama-3B exhibited an enormous leap beyond Llama-1B and Llama-8B cemented the lead. Scaling was also beneficial to Llamba, although the performance gains were slightly less, and the 8B model did not bridge the gap with its Llama counterpart. This suggests the Transformer architecture can be used more successfully to utilize increased capacity to perform complex reasoning.

Few-shot learning effectiveness was very sensitive to the architecture. Llama performed better when there were more examples, and the accuracy was greatest in the 2-shot and 8-shot configurations. In the case of Llamba, however, more shots did not always result in better performance and even introduced behavioral instability (e.g. the spurious example generation). This indicates that the typical few-shot prompting paradigm is more consistent with the pre-training goal of the Transformer than the distilled and recurrent nature of Llamba.

It was found that there was strong positive correlation between the compliance of a model with the structured output format and the final accuracy. Higher format compliance models (e.g. Llama-8B at 94% in 8-shot) also performed the best in accuracy. This connection suggests that the capability of strict adherence to directions is not only a superficial metric, but is also intertwined with the reasoning power of the model. A model capable of structuring its output reliably is also likely to be following the logical structure of the problem in a better way.

To conclude, the comparisons made in this paper show that the decision between Transformers and SSMs is never a binary one but must be determined by a set of factors such as task demands, resource availability and deployment conditions. Transformers are the absolute leaders in the category of highest performance, while SSMs, with the help of the advanced methods of distillation such as MOHAWK, prove to be a strong and efficient option in the category of practical and resource aware applications. The correlations that were found between scaling, prompting, and model behavior present a good guide in future studies that will seek to optimize and deploy these models.

5.6 Discussions

This study tries to understand how much viability does the Mamba architecture hold, specifically the State-Space Models (SSMs) such as Llamba which is a very lightweight and efficient alternative of Transformer-based models in the area of mathematical reasoning. The comparative analysis on the GSM8K benchmark reveals rather intertwining trade-offs between accuracy, efficiency and model behavior. The discussion below provides a way to interpret the findings, situating them within the context of making LLMs more resource-efficient and accessible.

The core findings of this study supports the established paradigm: Transformer-based models, represented by the Llama family, currently deliver better accuracy on complex reasoning tasks like GSM8K. Across all model sizes (1B, 3B and 8B) and few-shot settings (0-shot, 2-shot and 8-shot), Llama models consistently outperform all of its Llamba counterparts. This is evident in 2-shot and 8-shot scenarios, where Llama-8B achieved impressive accuracies of 81.8% and 80.1% respectively, which significantly surpasses Llamba-8B's peak 60.8%.

This gap in performance between both models largely attributes to the fundamental architectural strengths of the Transformer. With the help of its self-attention mechanism, it excels at establishing global dependencies and complex reasoning which are key to solve intricate word problems in GSM8K. The quadratic complexity nature of the attention being computationally expensive, gets the representational capacity required which is crucial for high-stakes reasoning accuracy.

However, this does not tell the whole story. Our analysis uncovered a critical methodological flaw that artificially deflated Llamba's reported accuracy. The Llamba-8B model exhibited a unique behavior in few-shot settings: after correctly solving the target problem, it would often generate additional, self-created example problems and solve those as well. Our answer extraction algorithm, designed to capture the final numerical answer, would then incorrectly parse the solution to this last generated example (e.g., a consistently generated "bookshelf" problem resulting in "40 books") instead of the correct answer to the original problem. This suggests that the model's actual mathematical reasoning capability is higher than the quantitatively reported accuracy of 3.6% in the 8-shot setting indicates. The model was able to learn meta-learning behavior of demonstrating its understanding with multiple problem demonstrations, which is something that a simple exact-match evaluation

fails to capture. This showcases challenges in evaluating novel architectures that does not conform to the expected output pattern of their Transformer-based teachers.

The Llamba architecture lagged in accuracy compared to Llama but still demonstrated its hypothesized efficiency advantages. The linear-time complexity of the Mamba-based SSM translates directly to practical benefits. It is outlined in the architectural comparison. Quantified in the final results section, the Llamba takes substantially lower memory footprints and faster inference times, especially for longer sequences. This efficiency is SSM's most compelling value.

The success of the MOHAWK distillation framework in creating Llamba models that achieve competitive, albeit lower, performance using less than 0.1% of the teacher's training data cannot be overstated. It proves that knowledge can be effectively transferred from the resource-intensive Transformer paradigm to a more efficient SSM architecture. This can be a promising path toward creating models that are not only faster and lighter but also capable enough for many practical applications where state-of-the-art accuracy is not a strict requirement.

Our iterative journey in prompt engineering advocates a universal truth in LLM evaluation that model performance is inextricably linked to how it is prompted and evaluated. The shift from handcrafted, instructional prompts to the official and tokenized conversational prompts used by Llama authors was a breakthrough. This transition led to a dramatic increase in both accuracy and structured format compliance for both model families. It shows that aligning the prompt structure with the model's training-time conversational format is essential for unlocking its full potential for mathematical reasoning tasks. This emphasizes that for a fair comparison, evaluation methodologies must be meticulously optimized for each architecture, since suboptimal prompting can obscure a model's true capabilities.

Furthermore, the anomalous behavior of Llamba in few-shot settings reveals a limitation of current automated evaluation pipelines. Reliance on regex-based extraction can be efficient and reproducible but quickly becomes brittle in the face of unexpected but valid model behaviors. Future work must develop more robust, semantics-aware evaluation methods that can distinguish between a model failing to reason or reason in an unanticipated format.

For applications where accuracy is paramount and cloud-based inference is feasible, Transformer-based models like Llama remain the undisputed choice, however, for edge computing, real-time applications on consumer hardware and scenarios where strict latency or privacy requirements are a must, SSM-based models like Llamba present a compelling alternative.

Their lower computational and memory demands directly address the societal, environmental, and economic concerns that often rise in using AI. By enabling local execution on less powerful devices, SSMs can increase access to LLM technology, reduce reliance on energy-intensive data centers and lower the economic barrier to entry. In critical sectors like education and healthcare within developing regions,

hardware is often limited, thus, a model that is modestly as accurate as a Transformer but can run efficiently on available devices can be far more valuable than a state-of-the-art model that cannot be deployed at all.

This study provides valuable insights but it is evident that there are several limitations that should be acknowledged. The first one being that the evaluation was limited to the GSM8K dataset. Despite the fact that it is excellent for mathematical reasoning, its generalizability to other fields like code generation, logical deduction, or scientific QA has not been explored further. Then secondly, while the use of regex-based answer extraction can be effective in the majority of situations, it was insufficient for Llamba's particular few-shot behavior. Thirdly, hardware constraints limited the scale of our efficiency profiling and prevented the use of more advanced, LLM-based evaluation techniques. Last but not least, the research centered on a single SSM architecture, Llamba and compared it to a single Transformer model family, namely Llama. Other SSM variants or Transformer models could have produced different outcomes.

To conclude, this research projects the idea that while Transformer models currently maintain a significant advantage in accuracy for mathematical reasoning, SSMs like Llamba offer a promising path toward efficiency and accessibility. The choice between them is based on which is best for a given set of constraints and priorities, not which is universally superior. We offer a number of recommendations and suggestions for future research based on our findings. At first, more robust, context-aware evaluation pipelines, possibly employing LLM-as-a-judge or rule-based semantic parsers for answer extraction, must be developed to handle divergent model behaviors without penalizing valid reasoning. Then the second recommendation is that in order to help close the accuracy gap while maintaining efficiency, future research should investigate hybrid models that strategically combine SSM layers for effective long-context processing with a limited number of attention layers for crucial reasoning steps. Last but not least, in order to fully comprehend the capabilities of SSM, evaluation must extend to additional reasoning-intensive tasks that make use of their linear-time strength. These tasks include long-context applications like the analysis of legal documents and the generation of long codes, as well as other domains like common sense and code generation.

Therefore, more research is needed on specialized prompt engineering and fine-tuning methods made just for SSMs, that can include teaching them to use their selective scanning mechanisms for reasoning more effectively or suppressing redundant example generation in few-shot settings. By continuing to explore and refine alternative architectures like SSMs, the research community can work towards a future where powerful language models are not only capable but also practical and sustainable.

Chapter 6

Conclusion

6.1 Summary of Findings

This research undertook a systematic comparative study of State-Space Models (SSMs) and Transformer-based models on mathematical reasoning with respect to the model family of Llamba and Llama model at 1B, 3B, and 8B scale parameters. The test has been done on the GSM8K benchmark, where accuracy and inference efficiency as well as model behavior are evaluated. The major conclusions can be summed up as follows:

Transformer models always performed better than SSM models in all model sizes and in few-shot settings(0-shot, 2-shot, 8-shot). The gap between the performance increased with scale of the model and few-shot examples, with Llama-8B reaching the highest accuracy of 81.8% in the 2-shot scenario, much higher than the highest of Llamba-8B, which was 60.8%. This confirms that the Transformer self-attention mechanism,despite its quadratic complexity, offers a representational advantage to complex, multi-step reasoning problems.

While less precise, Llamba models showed the theoretical and practical advantages of SSMs, mainly, the linear complexity of its computations and reduced memory footprint. This qualifies them as a strong candidate to be deployed in resource-constrained settings where the high precision of Transformers is not a mandatory consideration, which is in line with the goals of on-device AI and democratization of LLMs.

The research found that prompt structure can easily influence model performance. The introduction of official, tokenized conversational prompts adopted by Llama authors led to an increase in performance of both model families dramatically. This underscores the fact that when a model is prompted in a suboptimal way, its true performance can be seriously misrepresented. Because of this, prompt engineering plays a vital role for performing a fair architectural comparison.

A notable observation was that the Llamba-8B model demonstrates unique and systematic behavior in few-shot settings. Once the target problem was solved correctly, the model would tend to produce more example problems which were self-generated. This caused the automated evaluation pipeline to extract the answer to last example

generated, artificially deflating the accuracy reported by Llamba. This shows that the true mathematical reasoning skill of Llamba is greater than the quantitative outcomes indicate and that there is a weakness in conventional regex-based assessment frameworks when faced with unforeseen model behavior.

The effectiveness of the MOHAWK framework has been established. The Llamba models, which were distilled from their Llama instructors with less than 0.1 percent of the original training data, were able to perform competitively. This shows that it is possible to transfer knowledge from the Transformer architecture to more efficient SSM architectures, paving the path to develop highly capable, but efficient, models.

To sum up, the results provide a clear picture of the existing landscape: Transformers are the architecture for the best reasoning accuracy, while SSMs can be the viable and efficient alternative in the situations when the computational resources are the main constraint. It is not a matter of which architecture is universally superior, but rather which is more applicable to certain application needs, and strikes the right balance between accuracy, efficiency and accessibility.

6.2 Contributions to the Field

This research makes several significant contributions to the ever evolving field of natural language processing and the development of accessible large language models. Our work extends beyond a simple performance comparison, it provides methodological, empirical, and practical insights that can guide future research work and development.

While State-Space Models have been benchmarked on many common reasoning tasks, their performance on complex and multi-step mathematical reasoning has remained rather unexplored. This study provides one of the first systematic and empirical evaluations of SSMs (specifically the Llamba family) on the challenging grade school math GSM8K benchmark. We establish a crucial baseline and demonstrate that while there is a significant accuracy gap that exists compared to Transformers in SSMs, they are still capable of non-trivial mathematical reasoning, achieving up to 60.8% accuracy in a zero-shot setting with the Llamba-8B model. This fills a critical gap in the understanding of SSM capabilities and provides a valuable point of reference for the community.

The Llamba models often generate correct solutions followed by self-created examples, a behavior that confuses the regex-based answer extraction process and highlights a fundamental limitation in current benchmarking practices. This finding serves as a critical caution to future researchers that automated metrics can severely underestimate the true capabilities of models that deviate from expected output patterns. It shows the need for more capable, semantically-aware evaluation frameworks as alternative architectures continue to emerge.

Our results provide strong and independent validation of the MOHAWK framework's effectiveness. We demonstrate that it is possible to successfully distill knowl-

edge from a state-of-the-art Transformer teacher (Llama) into a fundamentally different and more efficient SSM student (Llamba), achieving competitive performance with a minuscule fraction (less than 0.1%) of the original training data. However, our work also contextualizes this success by showing the limits of this distillation. We found a performance gap on a hard task like GSM8K that reveals that certain reasoning capabilities of the Transformer are not fully transferred, which means there is room for improving distillation techniques.

This study provides a concrete and quantified trade-off between accuracy and efficiency for a practical task. We clearly delineate the deployment landscape that is for applications that requires peak accuracy, the Transformer models are still the easy choice, but if we look at a scenario where is resource-constrained environments where a model with modest accuracy is acceptable then the SSM offers a path to viable on-device deployment. This practical contribution can aid developers and researchers in making informed architectural choices based on specific application requirements.

Our iterative prompt engineering journey resulted in a dramatic performance boost by adopting the model's native conversational format learned during its training. It demonstrates that model performance is not an intrinsic property but is co-determined by the evaluation protocol. This finding elevates prompt engineering from an implementation detail to a critical component of rigorous and fair model comparison and also ensuring that architectural benchmarks are not confounded by suboptimal prompting strategies.

In summary, this thesis contributes a new benchmark for SSMs on mathematical reasoning tasks. It shows a critical methodological challenge in evaluating and validating the potential use of advanced distillation techniques. This work also provides a clear framework for deployment decisions and reinforces the importance of meticulous evaluation design.

6.3 Recommendations for Future Work

The Llamba model was distilled using the MOHAWK method from the Llama 3.x family models. The key achievement here was to be able to get such performance just performing distillation on 0.1% of the training data its parent was trained on. Therefore, in future with some modifications in the MOHAWK pipeline, the hybrid architecture of Transformer and Mamba blocks should be considered. A sweet spot can be explored considering the context scaling trade-offs in terms of the ratio of Transformer and Mamba based blocks or in terms of model weights. Besides this, other mathematical tasks can be explored where the teacher model has performed well in order to further test the effectiveness of this method. Furthermore, other tasks which require processing a longer context should be explored if a substantial result can be achieved to leverage its architecture.

Bibliography

[1] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015. arXiv: 1503.02531 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1503.02531

[2] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in nlp*, 2019. arXiv: 1906.02243 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1906.02243

[3] K. Cobbe et al., *Training verifiers to solve math word problems*, 2021. arXiv: 2110.14168 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2110.14168%7D

[4] S. Carreira, T. Marques, J. Ribeiro, and C. Grilo, *Revolutionizing mobile interaction: Enabling a 3 billion parameter gpt llm on mobile*, 2023. arXiv: 2310.01434 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.01434

[5] S. Dasgupta, T. Cohn, and T. Baldwin, "Cost-effective distillation of large language models," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 7346–7354. DOI: 10.18653/v1/2023.findings-acl.463 [Online]. Available: https://aclanthology.org/2023.findings-acl.463/

[6] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *Qlora: Efficient finetuning of quantized llms*, 2023. arXiv: 2305.14314 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2305.14314

[7] A. Vaswani et al., *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762

[8] G. Yang, D. Lo, R. Mullins, and Y. Zhao, "Dynamic stashing quantization for efficient transformer training," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7329–7336. DOI: 10.18653/v1/2023.findings-emnlp.489 [Online]. Available: https://aclanthology.org/2023.findings-emnlp.489/

[9] L. Yang et al., "Tc-hisrnet: Hyperspectral image super-resolution network based on contextual band joint transformer and cnn," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, pp. 1–16, Jan. 2023. DOI: 10.1109/JSTARS.2023.3323489

[10] W. Des. "Along comes a mamba: An evolution in sequence models based on state space models." Medium article about Mamba architecture and State Space Models, Accessed: Dec. 19, 2024. [Online]. Available: https://medium.com/@wilburdes/along-comes-a-mamba-an-evolution-in-sequence-models-based-on-state-space-models-2bd3d0e02d86

[11] D. Du et al., *Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation*, 2024. arXiv: 2402.10631 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.10631

[12] A. Grattafiori et al., *The llama 3 herd of models*, 2024. arXiv: 2407.21783 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2407.21783

[13] A. Gu and T. Dao, *Mamba: Linear-time sequence modeling with selective state spaces*, 2024. arXiv: 2312.00752 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2312.00752

[14] D. Gu et al., *Loongtrain: Efficient training of long-sequence llms with head-context parallelism*, Jun. 2024. DOI: 10.48550/arXiv.2406.18485

[15] S. Hayou, N. Ghosh, and B. Yu, *Lora+: Efficient low rank adaptation of large models*, 2024. arXiv: 2402.12354 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2402.12354

[16] R. Kumar, "Exploring state space models: The next evolution beyond transformers?" *Towards AI*, 2024. Accessed: Dec. 19, 2024. [Online]. Available: https://pub.towardsai.net/exploring-state-space-models-the-next-evolution-beyond-transformers-

[17] A. Kundu, Y. C. F. Lim, A. Chew, L. Wynter, P. Chong, and R. Lee, "Efficiently distilling LLMs for edge applications," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, Y. Yang, A. Davani, A. Sil, and A. Kumar, Eds., Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 52–62. DOI: 10.18653/v1/2024.naacl-industry.5 [Online]. Available: https://aclanthology.org/2024.naacl-industry.5/

[18] J. Lang, Z. Guo, and S. Huang, *A comprehensive study on quantization techniques for large language models*, 2024. arXiv: 2411.02530 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2411.02530

[19] S. Ma et al., *The era of 1-bit llms: All large language models are in 1.58 bits*, 2024. arXiv: 2402.17764 [cs.CL].

[20] D. Peng, Z. Fu, and J. Wang, "PocketLLM: Enabling on-device fine-tuning for personalized LLMs," in *Proceedings of the Fifth Workshop on Privacy in Natural Language Processing*, I. Habernal et al., Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 91–96. [Online]. Available: https://aclanthology.org/2024.privatenlp-1.10/

[21] R. Saha, N. Sagan, V. Srivastava, A. J. Goldsmith, and M. Pilanci, *Compressing large language models using low rank and low precision decomposition*, 2024. arXiv: 2405.18886 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.18886

[22] B. Shen et al., "Pruning large language models to intra-module low-rank architecture with transitional activations," in *Findings of the Association for Computational Linguistics: ACL 2024*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9781–9793. DOI: 10.18653/v1/2024.findings-acl.582 [Online]. Available: https://aclanthology.org/2024.findings-acl.582/

[23] R. Svirschevski, A. May, Z. Chen, B. Chen, Z. Jia, and M. Ryabinin, *Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices*, 2024. arXiv: 2406.02532 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2406.02532

[24] F. Tan et al., "MobileQuant: Mobile-friendly quantization for on-device language models," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 9761–9771. DOI: 10.18653/v1/2024.findings-emnlp.570 [Online]. Available: https://aclanthology.org/2024.findings-emnlp.570/

[25] W. Wang, Y. Mao, T. Dongdong, D. Hongchao, N. Guan, and C. J. Xue, "When compression meets model compression: Memory-efficient double compression for large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 16 973–16 983. DOI: 10.18653/v1/2024.findings-emnlp.988 [Online]. Available: https://aclanthology.org/2024.findings-emnlp.988/

[26] W. Yin, M. Xu, Y. Li, and X. Liu, *Llm as a system service on mobile devices*, 2024. arXiv: 2403.11805 [cs.OS]. [Online]. Available: https://arxiv.org/abs/2403.11805

[27] Z. Yu et al., *Edge-llm: Enabling efficient large language model adaptation on edge devices via layerwise unified compression and adaptive layer tuning and voting*, 2024. arXiv: 2406.15758 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2406.15758

[28] P. Zhang, G. Zeng, T. Wang, and W. Lu, *Tinyllama: An open-source small language model*, 2024. arXiv: 2401.02385 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2401.02385

[29] A. Bick, T. Katsch, N. Sohoni, A. Desai, and A. Gu, *Llamba: Scaling distilled recurrent models for efficient language processing*, 2025. arXiv: 2502.14458 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2502.14458

[30] A. Bick, K. Y. Li, E. P. Xing, J. Z. Kolter, and A. Gu, *Transformers to ssms: Distilling quadratic knowledge to subquadratic models*, 2025. arXiv: 2408.10189 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.10189