ECE1512H Project B Report

Jangwon Suh
Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto
Toronto, Canada
j.suh@mail.utoronto.ca

I. STATE SPACE MODELS (SSMS)

A. Summary of the SSMs

Key points: [1] introduces Mamba, a new sequence modeling architecture leveraging SSMs. Mamba aims to combine the efficiency of structured SSMs with the ability to model content-specific reasoning effectively. It enables input-dependent parameterization of SSMs, addressing the transformer-like content-awareness issues. Mamba incorporates an efficient algorithm to overcome computational bottlenecks, scaling linearly with sequence length. Mamba eliminates reliance on traditional transformer components like attention layers or MLP blocks, yielding improved throughput and scaling. Mamba shows state-of-the-art performance across modalities such as language, audio, and genomics. It achieves efficiency and quality improvements, outperforming similar models like transformers in both pretraining and downstream tasks. Notable results include improved perplexity in language modeling, enhanced DNA sequence modeling, and superior audio generation quality.

Technical Contributions: Mamba introduces input-dependent parameterization for SSMs (e.g., modifying Δ, A B, C) to enable selective filtering or propagation of inputs, enhancing context-aware reasoning. The approach generalizes recurrent and convolutional frameworks, overcoming limitations of linear time-invariant models. It combines GPU memory hierarchy optimizations with a parallel scan algorithm for efficient state computation. The novel use of recomputation reduces memory consumption during backpropagation. The Mamba block integrates SSMs and gated MLPs, forming a homogenous stackable unit. It replaces traditional transformer components with simplified, high-throughput architecture. It demonstrates linear scalability in sequence length during training and provides 5× throughput over transformers in inference.

Areas for Improvement: First, despite hardware-aware optimization, the method's efficiency is heavily reliant on GPU-specific accelerations, potentially limiting accessibility for researchers with limited resources. Memory utilization during training, though optimized, could further be explored for edge hardware. Second, while benchmarks are impressive, applying Mamba to more diverse real-world tasks could validate its broader applicability. Most benchmarks focus on long-sequence modalities; testing performance on mixed data types or heterogeneous tasks might reveal new insights. Third, although comparisons with transformers and other SSM-based models are detailed, including other novel architecture might offer a fuller evaluation. Lastly, the theoretical grounding of the selection mechanism is strong, but its interpretability in practical settings could be enhanced. Exploring real-world data examples to illustrate selective filtering effects might clarify its utility.

B. Part 1: Accuracy improvements for SSMs

Incorporating pretrained embeddings, such as word embeddings for language tasks or domain-specific embeddings, can significantly enhance model accuracy by providing richer input representations, as shown in Figure 1. Pretrained embeddings encode semantic relationships and contextual nuances from large corpora, offering improved input representations that enable models to better understand linguistic or domain-specific patterns. This can lead to faster convergence during training, as the embeddings already encapsulate essential information, reducing the model's need to learn these features from scratch. In low-resource settings, pretrained embeddings are particularly beneficial, compensating for limited labeled data with their inherent knowledge. In language tasks, embeddings like Word2Vec, GloVe, or contextualized ones from BERT and GPT provide semantic depth, while domain-specific embeddings for areas like genomics, finance, or law capture unique data characteristics. Additionally, pretrained embeddings from models like ResNet, CLIP, or Wav2Vec can be used in visual, auditory, or sequential data tasks to encode features that improve downstream performance. However, challenges include ensuring compatibility between the embedding space and model architecture, addressing domain mismatches by fine-tuning embeddings, and managing computational costs associated with high-dimensional embeddings. When integrated into architecture like Mamba, pretrained embeddings can complement its efficiency and scalability, enhancing its ability to model complex dependencies across domains such as language, genomics, and audio.

Code used in this report is provided in: https://github.com/j-suh-1998/ECE1512H-Project-B.git

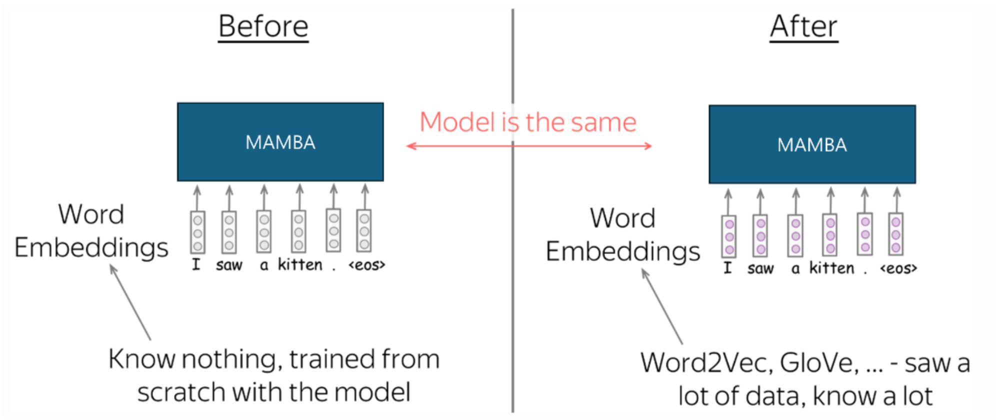

Figure 1. Diagram showing the method of incorporating pretrained embeddings, edited from [2]

C. *Part 2: Efficiency improvements for SSMs*

Mamba's efficiency can be significantly improved through mixed precision training. Mixed precision refers to using both 16-bit (half-precision) and 32-bit (full-precision) floating-point numbers during computations in deep learning. It optimizes model training and inference by leveraging the performance benefits of lower-precision arithmetic while maintaining numerical stability for critical operations. Modern GPUs (e.g., NVIDIA Volta, Ampere architectures) have specialized Tensor Cores optimized for half-precision arithmetic, which can perform operations faster than full precision. Half-precision variables use half the memory of full-precision variables, allowing for larger batch sizes and models or reducing memory constraints during training. Mixed precision carefully selects operations that can safely use lower precision without significant degradation in model accuracy. Mixed precision involves dividing operations into two categories. The first one is float16, which is used for most operations, including matrix multiplications, convolutions, and activation functions. It also reduces memory and computational requirements. And the other one is float32, which is used for critical operations where precision is necessary, such as gradient updates in optimizers and accumulation of small values to prevent underflow/overflow issues.

I used Mamba-130M as the pretrained model and set a sequence length of 32 with a batch size of 2 to minimize memory usage. I measured the memory consumption with gradients turned off, as shown in Table 1. The results clearly demonstrate that using mixed precision reduces memory usage. While the difference in memory usage between the original model and the mixed precision model may not appear significant for a single iteration, it becomes increasingly impactful when the model is iterated multiple times.

Table 1. Comparison of memory usage with original model and model with mixed precision

	Original model	Model with mixed precision	Difference
Memory usage (MB)	1747.43	1741.29	6.14

II. VISION LANGUAGE MODELS (VLMS)

A. *Summary of VLMs*

Key points of Qwen2-VL [2]: First, Qwen2-VL is a novel approach for processing images of varying resolutions into different numbers of visual tokens, improving efficiency and accuracy in visual representation. Second, Qwen2-VL introduces Multimodal Rotary Position Embedding (M-RoPE), which is a sophisticated method that enhances positional information fusion across text, images, and videos. Third, Qwen2-VL implements a cohesive framework for handling both images and videos, allowing a more comprehensive visual understanding. Fourth, it also investigates how scaling model parameters and training data impacts performance, with models at 2B, 8B, and 72B parameter scales. Fifth, Qwen2-VL models achieve state-of-the-art or near state-of-the-art results across numerous multimodal benchmarks. Lastly, Qwen2-VL supports advanced reasoning, decision-making, and multilingual visual understanding, catering to diverse applications like document parsing, video comprehension, and agent-based interactions.

Technical Contributions of Qwen2-VL: First, Qwen2-VL enhances adaptability by removing the fixed image resolution constraint present in conventional models. Second, the introduction of M-RoPE significantly boosts the model's ability to handle spatial and temporal details across different modalities. Third, it uses 3D convolutions and dynamic resolution adjustments to optimize training efficiency for long video and complex multimodal data. Fourth, it implements fine-tuning using diverse datasets, enabling complex, instruction-driven tasks across multiple modalities. Lastly, it provides open access to model weights, encouraging further research and development in the field.

Areas for Improvement of Qwen2-VL: First, although competitive, the model underperforms compared to leading systems like GPT-4o in specific challenges (e.g., MMMU benchmark). There is room for improvement in terms of performance in certain benchmarks. Second, the current token limit per video affects performance for extended durations. Improving support for longer sequences could enhance video comprehension. Third, while versatile, the model could benefit from further fine-tuning for tasks requiring extreme precision, such as detailed spatial reasoning or 3D navigation. Fourth, using smaller model sizes, instead of large model sizes like 72B parameters, might remove the necessity of substantial computational resources, potentially widening broader adoption. Lastly, testing and validation across more diverse and complex real-world scenarios could provide deeper insights into robustness and adaptability.

B. *Efficiency Bottlenecks*

To profile the efficiency bottleneck of Qwen2-VL model, I have made a code which integrates PyTorch's profiling tools to measure the performance of its key computational steps. It details the names of operations or layers, total time spent on CPU and CUDA for each operation, and resource utilization insights to identify bottlenecks.

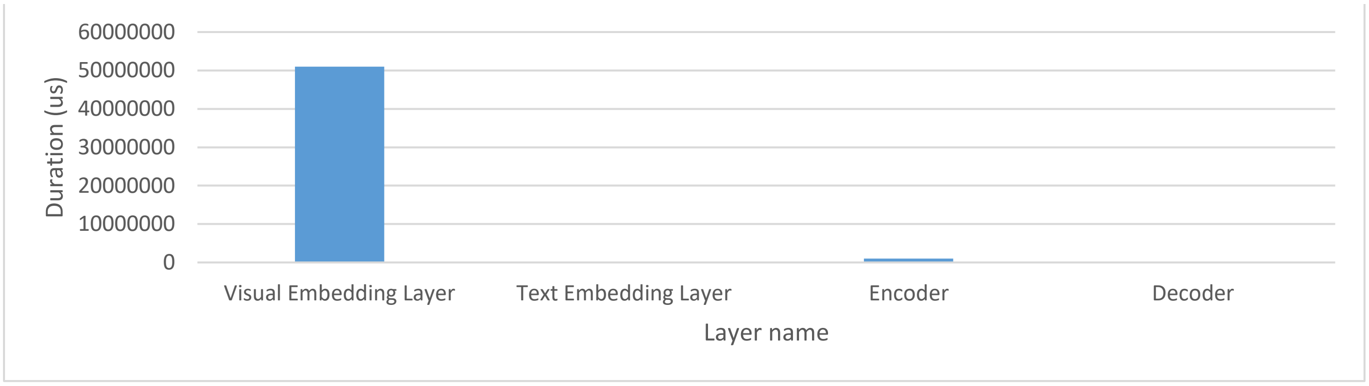

Figure 2. Computation time comparison among layers

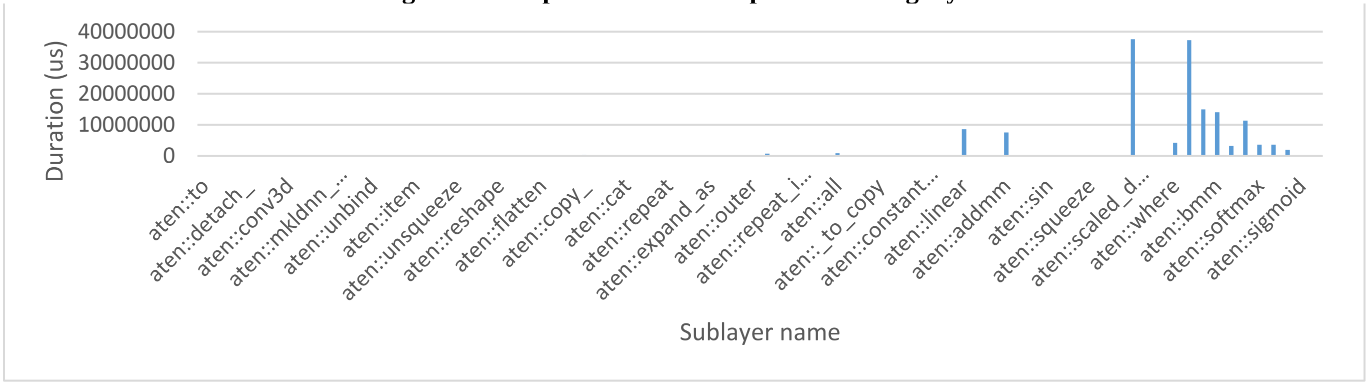

Figure 3. Computation time comparison among sublayers of visual embedding layer

As shown in Figure 2, the majority of computation time (97.9% of the total) is consumed in the visual embedding layer. Additionally, as depicted in Figure 3, most of the computation time in the attention layer is spent on the scaled dot product operation. These observations clearly indicate that the scaled dot product attention layer is the primary bottleneck affecting the model's efficiency.

To address this, I developed a new class, EfficientQwen2VLSdpaAttention, which builds upon the original Qwen2VLSdpaAttention class by introducing several efficiency improvements in its forward pass. This class introduces multiple enhancements to improve computational efficiency and reduce memory overhead. It unifies the projection of queries (Q), keys (K), and values (V) into a single linear layer (`self.qkv_proj`), eliminating the need for separate projections and reducing matrix multiplications. Instead of creating separate tensors, it slices the unified projection directly to obtain Q, K, and V, avoiding redundant operations and minimizing memory usage. The class ensures tensors (`query_states, key_states, value_states`) are made contiguous only when required, unlike the original class, which applies contiguity unconditionally, leading to unnecessary memory operations. Causal masks are handled more efficiently by slicing the attention mask directly, simplifying the logic for determining causality. Rotary position embeddings are managed modularly, with efficient updates for cosine and sine values. The class also minimizes the use of tensor transformations like permute and view, applying them only when necessary, thereby reducing computational overhead. Finally, it integrates PyTorch's scaled dot product attention with fewer conditional branches, streamlining operations and avoiding potential fallbacks for unsupported configurations. Together, these optimizations result in faster computations, reduced memory usage, and more streamlined tensor operations.

To evaluate these improvements, I compared the original and optimized attention mechanisms—Qwen2VLSdpaAttention and EfficientQwen2VLSdpaAttention—within a lightweight neural network classifier designed for the MNIST dataset. The models processed flattened 28x28 images using attention and classified them into 10 digits. Both models were trained for five epochs using the Adam optimizer and cross-entropy loss, followed by evaluation on the test set to measure accuracy and inference time.

Table 2. Comparison of accuracy and elapsed time of MNIST dataset classification between original model and efficiency improved model

	Original model	Efficiency improved model	Difference
Accuracy (%)	65.19	70.06	4.87
Elapsed time (s)	1.52	1.49	0.03

Table 2 summarizes the results, showing that the efficient classifier achieved a higher accuracy (70.06%) compared to the original classifier (65.19%), while also slightly reducing inference time (1.49 seconds vs. 1.52 seconds). While the difference in elapsed time might seem minor for a single iteration, the optimizations become increasingly impactful when the model is iterated multiple times. These enhancements not only improve computational efficiency but also contribute to better model performance, demonstrating the potential for efficiency improvements in larger vision-language models.

REFERENCES

[1] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024.

[2] https://lena-voita.github.io/nlp_course/transfer_learning.html

[3] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution, 2024.