Adapting Mamba Models for Deployment on Microcontrollers

Enabling Linear-Time Sequence Modeling on Ultra-Low-Power Edge Devices

Author: Bartosz Drabiński
b.drabinski@student.tudelft.nl

Responsible Professor: Qing Wang
Supervisor: Bo Yang

1. Introduction

TinyML focuses on deploying machine learning models on resource-constrained microcontrollers (MCUs)—devices typically limited to under 240 MHz CPUs and 512 KB of RAM. Because real-time applications like audio and video processing demand low latency, traditional approaches have relied on Convolutional or Recurrent Neural Networks [1] for their efficient memory scaling.

Recently, the **Mamba** [2] architecture has emerged as a compelling alternative, delivering Transformer-level performance with linear scaling relative to context length. However, deploying Mamba to MCUs remains a challenge, as critical optimization techniques like quantization and Low-Rank Factorization have not yet been fully explored for this architecture on edge hardware.

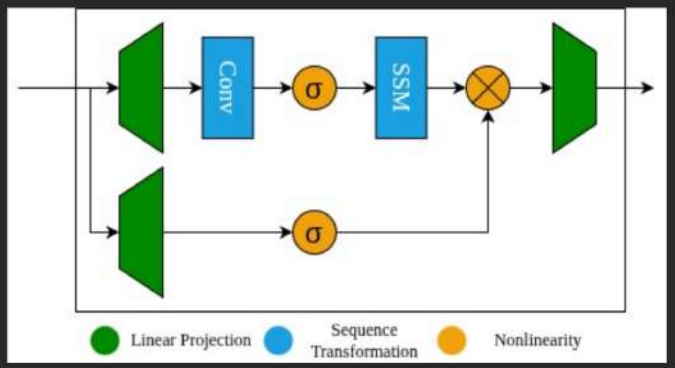
We address the following overarching research question: **How can the Mamba model inference for microcontrollers be improved?**

- What are the memory and computational bottlenecks of Mamba models on microcontrollers?
- How does quantization affect Mamba's performance on edge devices?
- To what extent can hardware-unfriendly components of the Mamba architecture be simplified or replaced?

2. Mamba architecture

The Mamba architecture utilizes a dual-path design featuring a gated path and a **Selective State Space Model** path. Unlike traditional, static SSMs, it achieves input-dependency by making its discretization step and transition matrices direct functions of the current input token.

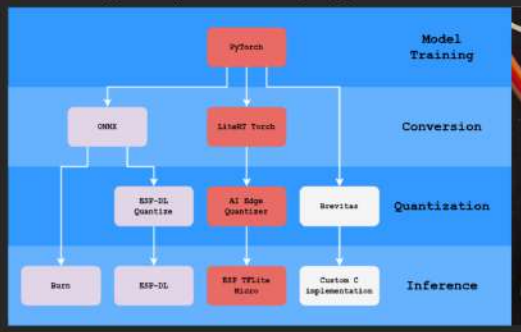
This **breaks Linear Time Invariance**, allowing the model to dynamically filter or compress information based on context. To mitigate the sequential bottlenecks of this recurrence on hardware, Mamba employs a hardware-aware parallel scan algorithm on the GPU, caching intermediate states in fast SRAM instead of slower HBM to maintain linear O(N) time complexity.

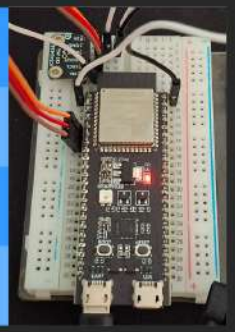


3. Model deployment

For this project, the **ESP32S3** was chosen as the deployment platform due to its high performance and its use in the MambaLite-Micro literature, which enables direct comparisons. The performance variations across different ESP32 models highlight how manufacturer-specific hardware instructions impact execution efficiency.

Among the various embedded deployment frameworks available (such as ONNX, ExecuTorch, or raw C), this work utilizes **LiteRT** (formerly **TensorFlow Lite for Microcontrollers**). LiteRT offers an ideal balance: it provides robust utilities for measuring memory usage and latency, and it is significantly easier to modify and deploy than a raw C implementation while maintaining widespread industry support.





4. Mamba-MCU Optimizations

To evaluate performance, a Mamba model was trained on the **Human Activity Recognition** (HAR task) and **Google Speech Commands V2** (KWS task) datasets. The model architecture consists of a simple linear input layer, one Mamba layer, pooling and linear layers for the output. It was trained in PyTorch. We improved upon the previous works by:

- Replacing the hardcoded C inference with an easier to use LiteRT python deployment pipeline
- Applying **INT8** weight and activations post training **quantization**
- **Removing loop unrolling** that the LiteRT introduces, which made the model duplicate inference graph nodes for the S6 step.

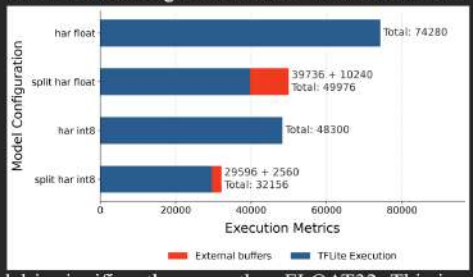
5. Results and Discussion

After training we test accuracy achieved for both datasets for all steps in the deployment process. The accuracies achieved are not far off the state of the art models, that are usually significantly larger. The SOTA for HAR is ≈96%, and ≈94% for KWS.

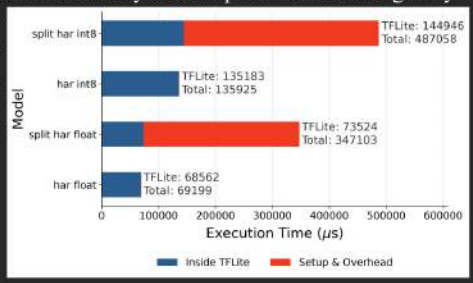
Table 1: Accuracy results for full and split models.

Dataset	Float32	Int8
HAR PyTorch	93.89%	-
HAR TFLite Full	93.89%	93.59%
HAR TFLite Split	93.89%	92.37%
KWS PyTorch	88.96%	-
KWS TFLite Full	88.96%	85.06%
KWS TFLite Split	88.96%	84.10%

As expected we see that the quantization of the entire model to INT8 **significantly (by 34.9%) reduces its peak memory** usage. While this is less than the theoretical 4x improvement, this is common because of the overhead of the scale and zero point which is even larger for models with such small dimentions as ours. A more surprising discovery was that the splitting operation implemented to circumvent LiteRT limitations also reduces peak memory usage.



The latency of the INT8 model is significantly worse than FLOAT32. This is mainly caused by the MUL operator implementation for INT8 requiring multiple additions and bitshifts. The latency of our split model is also greatly hindered by the need to manually move the data between 3 different models, which adds huge overhead.



6. Conclusion

LiteRT deployment with splitting **reduces peak RAM by ≈75%** compared to the C-based MambaLite-Micro [3] baseline. Uniform INT8 quantization by itself **cuts model size by ≈60%** and peak RAM by ≈33%, but increases execution latency on ESP32 hardware lacking native INT8 vector instructions. Decoupling the execution graph (Pre-SSM, SSM-step, Post-SSM) successfully bypasses TFLM's sequence unrolling limits to lower peak memory and enable long-context workloads, though it amplifies static quantization sensitivity.

Next-step optimizations should focus on integrating the **loop-partitioned SSM** step directly into LiteRT as a **native custom operator**. This implementation will streamline runtime execution, eliminate pipeline complexity, and make the architecture fully practical for edge deployment.

7. References

[1] M. Tri Lê, P. Wolinski, and J. Arbel, "Efficient Neural Networks for Tiny Machine Learning: A Comprehensive Review," ACM Trans. Intell. Syst. Technol., vol. 17, no. 4, pp. 1–41, Aug. 2026, doi: 10.1145/3798276.

[2] A. Gu and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," May 31, 2024, arXiv: arXiv:2312.00752. doi: 10.48550/arXiv.2312.00752.

[3] H. Xu, J. Xia, W. Yang, Y. Sui, and S. Xia, "MambaLite-Micro: Memory-Optimized Mamba Inference on MCUs."

Code available at: https://github.com/drabart/MCUFriendlyMamba