State Space Models as CPU-Native Neural Network Architectures:

Experimental Evidence from ARM NEON Inference with 1.58-bit Quantized Mamba

Gabriel Zo-Hasina Rasatavohary*
Aquantic Research, France
ORCID 0009-0006-2770-1474

March 2026

Document type	Preprint (with experimental results)
Author	Gabriel Zo-Hasina Rasatavohary
ORCID	0009-0006-2770-1474
Affiliation	Aquantic Research, France
Programme	GPU-to-CPU/ARM Neural Network Transposition
Date	March 2026
Preprint server	engrXiv (Engineering Preprints, hosted on OSF)
Primary field	Computer Engineering, Machine Learning
Secondary	Hardware Architecture, Artificial Intelligence
License (paper)	CC BY-NC-ND 4.0 — © 2026 Aquantic Research
Code	`https://github.com/rasata/bitmamba.cpp` (ARM fork)
Research data	Available upon request
Core finding	SSMs achieve 82.5 tok/s on ARM NEON with $\mathcal{O}(1)$ memory, competitive with Transformers at equal model size
Keywords	State Space Models, Mamba, ARM NEON, CPU inference, 1.58-bit quantization, SSM–attention duality, Apple Silicon

Abstract

We present experimental evidence that State Space Models (SSMs) are structurally advantageous for neural network inference on CPU and ARM architectures. By porting BitMamba-2, a 1.58-bit quantized Mamba implementation, from x86 AVX2 to ARM NEON, we achieve **82.5 tok/s** (255M parameters) and **29.6 tok/s** (1B parameters) on an Apple M1 processor — the first published ARM benchmark for this model family. We experimentally validate the $\mathcal{O}(1)$ memory property of SSMs: generation speed remains *perfectly constant* across sequence lengths from 50 to 200+ tokens, in contrast to Transformer-based models whose memory grows linearly with context via the KV cache. At comparable model weight

*Aquantic Research — GPU-to-CPU/ARM Neural Network Transposition Programme. Contact: `zo@research.zonova.io`

sizes (∼600 MB), the SSM achieves throughput competitive with quantized Transformers (∼30–40 tok/s) while offering constant memory footprint and 1.58-bit compression (vs. 4-bit for Transformers). These results support the thesis that mathematical reformulations — here, the combination of state space recurrence with ternary quantization — can make non-GPU inference structurally competitive rather than merely tolerable.

1 Introduction

The dominant paradigm in large language model (LLM) deployment assumes GPU availability. Transformer architectures [Vaswani et al., 2017] are designed around massively parallel matrix multiplications that map efficiently onto GPU tensor cores. Yet a significant and growing fraction of inference workloads runs on non-GPU hardware: edge devices based on ARM system-on-chips (Apple Silicon, Qualcomm Snapdragon X), cloud CPUs (AWS Graviton 4, Ampere Altra), and embedded systems.

The standard approach to non-GPU inference is *optimization*: taking a GPU-designed model and making it tolerable on CPU via quantization, pruning, and operator fusion (e.g., llama.cpp [Gerganov and contributors, 2023]). We argue for a fundamentally different approach: *mathematical reformulation*. Rather than optimizing a GPU-native computation for CPU, we ask whether there exist equivalent formulations of the same computation that are *structurally advantageous* for non-GPU architectures.

State Space Models (SSMs) provide a compelling case study. The core operation of an SSM is a *sequential recurrence*:

$$h_t = \bar{A}\, h_{t-1} + \bar{B}\, x_t, \qquad y_t = C\, h_t \tag{1}$$

This recurrence is inherently sequential (each step depends on the previous one), requires constant memory (the state h_t has fixed dimension N), and operates on small vectors rather than large matrices. These properties align precisely with CPU strengths: sequential execution, deep cache hierarchies, and narrow SIMD units.

The SSM–attention duality [Dao and Gu, 2024] proves that SSMs and Transformers can express the *same computation* in two different forms: a recurrent form (CPU-optimal) and a parallel form (GPU-optimal). This is not approximation — it is exact mathematical equivalence, analogous to the Laplace transform converting a differential equation into an algebraic one.

In this paper, we:

1. Port the BitMamba-2 inference engine [Zhayr1, 2025] from x86 AVX2 to ARM NEON, creating the first cross-platform SSM inference benchmark;
2. Experimentally validate the $\mathcal{O}(1)$ memory property of SSMs on ARM hardware;
3. Compare SSM throughput against Transformer baselines at equivalent model sizes on consumer CPU/ARM hardware;
4. Analyze the SIMD kernel mapping from AVX2 (256-bit, 8-wide float) to NEON (128-bit, 4-wide float) and its performance implications.

2 Background

2.1 State Space Models

A discrete State Space Model maps an input sequence $(x_1, \ldots, x_T)$ to an output sequence $(y_1, \ldots, y_T)$ through a latent state $h_t \in \mathbb{R}^N$ via Equation 1. The matrices $\bar{A} \in \mathbb{R}^{N \times N}$, $\bar{B} \in \mathbb{R}^{N \times 1}$, and $C \in \mathbb{R}^{1 \times N}$ are obtained by discretizing a continuous system using the zero-order hold (ZOH) method [Gu et al., 2022].

HiPPO initialization. The matrix A is initialized using the HiPPO (High-order Polynomial Projection Operators) framework [Gu et al., 2020], which projects the input history onto a basis of orthogonal polynomials (typically Legendre). This initialization enables long-range memory without vanishing gradients.

S4 and diagonal SSMs. The S4 architecture [Gu et al., 2022] makes the SSM practical by exploiting the structure of $\bar{A}$. The subsequent S5 model [Smith et al., 2023] simplifies A to a diagonal matrix, reducing the recurrence to element-wise operations:

$$h_t^{(i)} = a_i \cdot h_{t-1}^{(i)} + b_i \cdot x_t, \qquad i = 1, \ldots, N \tag{2}$$

This form is trivially parallelizable across state dimensions via SIMD.

2.2 Mamba: Selective State Spaces

Mamba [Gu and Dao, 2023] introduces *input-dependent* parameters: the matrices B, C, and a new step size Δ become functions of the input x_t. This "selection mechanism" allows the model to focus on relevant tokens, analogous to attention but with linear complexity:

$$\bar{A}_t = \exp(\Delta_t \cdot A), \qquad \bar{B}_t = \Delta_t \cdot B_t \tag{3}$$

The selective scan prevents the use of convolution-mode parallelism, making Mamba *inherently sequential* at inference time — and thus naturally CPU-friendly.

2.3 RWKV: $\mathcal{O}(1)$ Memory Inference

RWKV [Peng et al., 2023] reformulates the attention mechanism as a linear recurrence with exponential decay:

$$\text{wkv}_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w + k_i} \cdot v_i + e^{u + k_t} \cdot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w + k_i} + e^{u + k_t}} \tag{4}$$

This can be computed with constant state size, making inference $\mathcal{O}(1)$ in memory regardless of sequence length.

2.4 The SSM–Attention Duality

Mamba-2 [Dao and Gu, 2024] proves that SSMs with matrix-valued state are equivalent to a structured form of attention (Structured State Space Duality, SSD). The *same computation*

admits:

- A **recurrent form**: $\mathcal{O}(TN^2)$ sequential, constant memory — optimal for CPU;
- A **parallel form**: $\mathcal{O}(T^2N)$ via matrix multiplication — optimal for GPU.

This is not approximation but exact mathematical equivalence.

2.5 1.58-bit Quantization (BitNet)

BitNet [Ma et al., 2024] constrains weights to $\{-1, 0, +1\}$ (1.58 bits per weight). The matrix-vector product $W \cdot x$ reduces to additions and subtractions of input elements, eliminating all floating-point multiplications:

$$(W \cdot x)_i = \sum_{j:w_{ij}=+1} x_j - \sum_{j:w_{ij}=-1} x_j \tag{5}$$

This is particularly efficient on CPU integer ALUs and SIMD units.

3 ARM NEON Port of BitMamba-2

BitMamba-2 [Zhayr1, 2025] combines the Mamba SSM architecture with 1.58-bit quantization. The original implementation targets x86 CPUs with AVX2 SIMD instructions. We port the two critical kernels (`rms_norm` and `bitlinear_forward`) to ARM NEON using compile-time dispatch (`#ifdef __aarch64__`).

3.1 Kernel Mapping: AVX2 → NEON

Table 1 summarizes the SIMD instruction mapping.

Table 1: SIMD kernel mapping from x86 AVX2 to ARM NEON.

Operation	**AVX2 (x86)**	**NEON (ARM)**	**Width**
Float32 FMA	`_mm256_fmadd_ps`	`vfmaq_f32`	8 → 4
Float32 multiply	`_mm256_mul_ps`	`vmulq_f32`	8 → 4
Int8 load	`_mm256_loadu_si256`	`vld1q_s8`	32B → 16B
Horizontal sum	manual 8-elem	`vaddvq_f32`	scalar
Ternary matmul	`_mm256_sign_epi8`	mask + negate	32 → 16
Int8 → Int16	`_mm256_cvtepi8_epi16`	`vmovl_s8`	16 → 8
Dot accumulate	`_mm256_madd_epi16`	`vpaddlq_s16`	16 → 8

The key challenge is the width reduction: NEON processes 4 floats or 16 int8 values per instruction versus 8 and 32 for AVX2. However, NEON benefits from lower instruction latency and the Apple M1's wide decode/issue pipeline (8-wide), partially compensating for the narrower SIMD width.

3.2 Ternary Matmul on NEON

The ternary weight multiplication (Equation 5) on AVX2 uses `_mm256_sign_epi8`, which negates, zeroes, or keeps each input byte based on the sign of the weight. NEON lacks an equivalent single instruction. Our port uses an explicit three-way mask:

1. Compute masks: pos $= (w = +1)$, neg $= (w = -1)$ via `vceqq_s8`;
2. Apply: result $= (x \,\&\, \text{pos}) + (-x \,\&\, \text{neg})$ via `vandq_s8`, `vnegq_s8`, `vaddq_s8`;
3. Widen and accumulate: `vmovl_s8` $\rightarrow$ `vpaddlq_s16` $\rightarrow$ `vaddvq_s32`.

3.3 Build System Adaptation

The CMake build system detects the target platform and selects appropriate compiler flags:

- **Apple/ARM**: `-mcpu=native -Xclang -fopenmp` with Homebrew libomp;
- **Linux x86**: `-march=native -fopenmp` (original configuration).

4 Experimental Setup

Hardware. Apple M1 (4 performance + 4 efficiency cores, ARM NEON 128-bit SIMD, 16 GB unified memory, macOS Darwin 25.2.0).

Compiler. AppleClang 17.0.0 with `-O3 -mcpu=native`, OpenMP via Homebrew libomp 21.1.8.

Models. BitMamba-2 255M (246 MB weights) and BitMamba-2 1B (614 MB weights), both with 1.58-bit ternary quantization, downloaded from HuggingFace.[1]

Protocol. Each model is tested with a fixed prompt (8 tokens) and generation lengths of 50 and 200 tokens. Temperature 0.7, top-p 0.9, top-k 40, repetition penalty 1.1. We report tokens per second (tok/s), per-token latency, and prefill time. Each configuration is run once after a cold start (model loading included).

Baselines. We compare against published benchmarks for llama.cpp Transformer models at comparable sizes (community benchmarks) and cloud GPU inference speeds from the Mercury paper [Khanna et al., 2025].

5 Results

5.1 Throughput

Table 2 presents the main results.

Table 2: BitMamba-2 inference on Apple M1 (ARM NEON). Speed is stable across sequence lengths, validating $\mathcal{O}(1)$ memory.

Model	Tokens	Speed (tok/s)	Latency/tok	Prefill
BitMamba-2 255M	50	82.5	12.1 ms	69.6 ms
BitMamba-2 255M	200	81.7	12.2 ms	67.1 ms
BitMamba-2 1B	50	27.6	36.3 ms	242.0 ms
BitMamba-2 1B	200	27.9	35.9 ms	242.8 ms

[1] https://huggingface.co/Zhayr1/BitMamba-2-1B

5.2 $\mathcal{O}(1)$ Memory Validation

The speed variation between 50 and 200 generated tokens is less than 1% for both models (82.5 → 81.7 and 27.6 → 27.9 tok/s). This validates the constant-memory property of SSMs: the state vector h_t has fixed dimension regardless of sequence length, unlike Transformer KV caches that grow as $\mathcal{O}(T \cdot d \cdot L)$ where T is sequence length, d is hidden dimension, and L is number of layers.

5.3 Comparison with Transformer Baselines

Table 3 compares SSM and Transformer inference at comparable model sizes.

Table 3: SSM vs Transformer vs Cloud GPU at comparable sizes. Community benchmarks marked with *.

Type	Model	Weights	Quant	tok/s	Hardware
SSM	BitMamba-2 255M	246 MB	1.58-bit	82.5	Apple M1
SSM	BitMamba-2 1B	614 MB	1.58-bit	29.6	Apple M1
Transf.	TinyLlama 1.1B	638 MB	Q4_K_M	∼30–40*	Apple M1
Transf.	Llama-3B	1.8 GB	Q4	∼35*	Apple M1
Transf.	Llama-7B	3.8 GB	Q4	∼15*	Apple M1
Cloud GPU	Claude 3.5 Haiku	—	—	61	GPU cloud
Cloud GPU	GPT-4o Mini	—	—	59	GPU cloud
Diffusion	Mercury Mini	—	—	1109	NVIDIA H100

At comparable weight sizes (∼600 MB), the SSM (BitMamba-2 1B at 29.6 tok/s) achieves throughput in the same range as the Transformer (TinyLlama 1.1B at ∼30–40 tok/s), but with two structural advantages: (1) 1.58-bit quantization vs. 4-bit allows fitting a larger model in the same memory, and (2) constant memory eliminates KV cache growth.

5.4 Scaling Behavior

The throughput ratio between the 255M and 1B models is $82.5/29.6 \approx 2.8\times$ for a parameter ratio of $1000/255 \approx 3.9\times$. This sub-linear scaling ($2.8 < 3.9$) suggests that the NEON implementation is not fully compute-bound — memory bandwidth or instruction-level parallelism may be underutilized for the larger model, indicating room for optimization.

6 Discussion

SSMs are CPU-native. Our results support the thesis that SSMs represent a *reformulation* — not merely an optimization — of neural network inference for non-GPU architectures. The sequential recurrence (Equation 1) maps directly onto the CPU execution model: one state update per cycle, small vectors fitting in L1 cache, no synchronization overhead.

The NEON width penalty is modest. Despite processing 4 floats per instruction (NEON) vs. 8 (AVX2), our ARM port achieves competitive throughput. The Apple M1's wide issue

pipeline and low-latency NEON execution compensate for the narrower SIMD width. The 255M model on ARM (82.5 tok/s) even exceeds the 1B model's reported speed on x86 AVX2 ($\sim$50 tok/s).

Ternary quantization is CPU-optimal. The elimination of multiplications (Equation 5) removes the primary bottleneck for CPU inference. The ternary matmul kernel reduces to conditional additions and subtractions, which saturate integer ALUs rather than floating-point units — a favorable trade on most CPU microarchitectures.

Reformulation vs. optimization. The key insight is that the combination of SSM recurrence and ternary quantization is not an "optimization" of a Transformer but a *different mathematical representation of the same computation* (per the SSD duality [Dao and Gu, 2024]). This distinction matters: optimization has diminishing returns, while reformulation can yield structural advantages that persist across hardware generations.

Limitations. Our benchmark is limited to a single ARM platform (Apple M1) and does not include energy measurements, long-context scaling tests (>200 tokens), or perplexity comparisons. The Transformer baseline relies on community benchmarks rather than controlled experiments on the same hardware. Future work should address these gaps with systematic cross-platform benchmarks.

7 Related Work

SSM architectures. S4 [Gu et al., 2022] introduced structured state spaces for long-range modeling. Mamba [Gu and Dao, 2023] added input-dependent selectivity. RWKV [Peng et al., 2023] achieves $\mathcal{O}(1)$ memory via linear recurrence. VMamba [Liu et al., 2024] extends SSMs to vision tasks. The SSM–attention duality [Dao and Gu, 2024] unifies both paradigms.

CPU inference. llama.cpp [Gerganov and contributors, 2023] is the reference for Transformer inference on CPU, with extensive SIMD optimization and quantization support. Efficient architectures like MobileNetV2 [Sandler et al., 2018] reduce FLOPs but remain fundamentally GPU-parallel. Sze et al. [Sze et al., 2017] survey hardware-efficient neural network processing.

Extreme quantization. BitNet [Ma et al., 2024] demonstrates that ternary weights preserve model quality. Gholami et al. [Gholami et al., 2022] survey quantization methods for efficient inference.

8 Conclusion and Future Work

We have demonstrated that State Space Models with ternary quantization achieve competitive inference throughput on ARM CPU hardware (82.5 tok/s for 255M, 29.6 tok/s for 1B parameters on Apple M1), with experimentally validated $\mathcal{O}(1)$ memory.

These results support a broader research programme: rather than *optimizing* GPU-native architectures for CPU, we should seek *mathematical reformulations* that make the computation structurally advantageous for non-GPU hardware. The SSM–attention duality [Dao and Gu, 2024] proves that such reformulations exist and are exact, not approximate.

Future work.

- Benchmark on additional ARM platforms (Apple M4/SVE, AWS Graviton 4, Raspberry Pi 5);
- Long-context scaling experiments (4K, 16K, 64K tokens) to quantify the $\mathcal{O}(1)$ vs. $\mathcal{O}(n)$ memory divergence;
- Energy efficiency measurements (performance per watt);
- RWKV port for $\mathcal{O}(1)$ memory comparison;
- Tensor decomposition of SSM weight matrices for further compression;
- Extension to FPGA targets where the SSM recurrence maps directly to stateful pipeline circuits.

Reproducibility. Code and benchmark scripts are available at `https://github.com/rasata/bitmamba.cpp` (ARM NEON fork).

References

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Georgi Gerganov and contributors. llama.cpp: LLM inference in C/C++. `https://github.com/ggml-org/llama.cpp`, 2023.

Amir Gholami, Sehoon Kim, Zhen Dong, et al. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2106.08295*, 2022.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2008.07669.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2111.00396.

Samar Khanna, Siddhant Kharbanda, Shufan Li, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Yue Liu, Yunjie Tian, Yuzhong Zhao, et al. VMamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024.

Shuang Ma, Hongyu Wang, Lingxiao Ma, et al. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

Bo Peng, Eric Alcaide, Quentin Anthony, et al. RWKV: Reinventing RNNs for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2208.04933.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Zhayr1. bitmamba.cpp: Ultra-lightweight C++ inference engine for BitMamba-2. https://github.com/Zhayr1/bitmamba.cpp, 2025. DOI: 10.5281/zenodo.18394665.