Cross-Platform Benchmark of State Space Model Inference:

ARM NEON vs x86 AVX2 vs GPU CUDA 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
Companion paper	Rasatavohary [2026] (ARM NEON results, engrXiv)
Preprint server	engrXiv (Engineering Preprints, hosted on OSF)
Primary field	Computer Engineering, Machine Learning
License (paper)	CC BY-NC-ND 4.0 — © 2026 Aquantic Research
Code	`https://github.com/rasata/bitmamba.cpp` (ARM fork, with x86 support)
Weights	`https://huggingface.co/rasatavohary/BitMamba-2-1B` (mirror)
Research data	Available upon request
Core finding	Cross-platform SSM inference spanning scalar to AVX-512: x86 scalar (0.58 tok/s for 1B), x86 AVX2 via WSL2 (4.2 tok/s), x86 Windows native (13.0 tok/s), ARM NEON (27.6 tok/s), x86 AVX-512 Xeon 4210R (47.6 tok/s for 1B, 113.1 tok/s for 255M), x86 AVX2 i3-12100F reference (∼53/146 tok/s). First-party AVX-512 server benchmark validates $\mathcal{O}(1)$ and achieves 1.72× ARM NEON for 1B, but below theoretical 4× ratio due to AVX2 kernel path
Keywords	State Space Models, Mamba, ARM NEON, x86 AVX2, AVX-512, GPU CUDA, cross-platform benchmark, 1.58-bit quantization

Abstract

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

We present a cross-platform benchmark of State Space Model (SSM) inference across ARM NEON (Apple M1) and four x86 configurations spanning three hardware generations and three SIMD levels (scalar, AVX2, AVX-512). Building on our companion preprint [Rasatavohary, 2026], which established the first ARM NEON port of BitMamba-2 at 82.5 tok/s (255M) and 29.6 tok/s (1B), we extend the analysis with x86 measurements of the same 1.58-bit quantized models under six environments: WSL2 (Linux on Windows), Windows native (MSVC), published native Linux AVX2 reference figures, a native Linux scalar baseline on a pre-AVX2 processor (Intel i5-3230M, Ivy Bridge), and a new native Linux AVX-512 server benchmark on an Intel Xeon Silver 4210R (Cascade Lake, 10 cores, AVX-512 + VNNI).

The results reveal a layered overhead structure for the 1B model: Windows native MSVC achieves 13.0 tok/s (3.1× faster than WSL2 at 4.2 tok/s), while published native Linux AVX2 reference reaches ∼53 tok/s (4.1× faster than Windows native). This decomposes the total WSL2-to-native-Linux gap (12.6×) into a WSL2 virtualization penalty (3.1×) and a compiler/OS penalty (4.1×, MSVC on Windows vs. GCC on native Linux). The native Linux scalar benchmark on the i5-3230M (Ivy Bridge, SSE4.2 only, no AVX2/FMA) yields 2.42 tok/s (255M) and 0.58 tok/s (1B), quantifying the SIMD contribution: the AVX2-to-scalar ratio reaches ∼60× for the 255M model. The Xeon 4210R AVX-512 benchmark achieves 113.1 tok/s (255M) and 47.6 tok/s (1B) using the AVX2 code path. We also implement and benchmark native AVX-512 kernels (512-bit, 64 int8/iteration), which are 18–31% *slower* due to Cascade Lake frequency throttling — a controlled negative result establishing that wider SIMD is not universally beneficial for ternary matmul.

We validate the $\mathcal{O}(1)$ memory property of SSMs across all six platforms and analyze the performance decomposition across virtualization, compilation, ISA, SIMD width, and hardware generation factors. These results confirm that SSM inference on native x86 with SIMD achieves interactive speeds (>30 tok/s) for models up to 1B parameters on consumer hardware. Comparison with cloud GPU APIs (Claude 3.5 Haiku, GPT-4o Mini) contextualizes throughput but is confounded by a 10–80× model size disparity; a normalized metric (tok/s per billion parameters) favors the CPU SSM approach. SIMD vectorization is the single most important factor for CPU SSM inference performance.

1 Introduction

In a companion preprint [Rasatavohary, 2026], we demonstrated that State Space Models (SSMs) with 1.58-bit ternary quantization achieve competitive inference on ARM hardware, reaching 82.5 tok/s for a 255M-parameter model on an Apple M1. That work established a key result: SSMs are not merely *tolerable* on non-GPU hardware but *structurally advantageous*, thanks to the alignment between sequential recurrence and CPU execution models.

However, the companion paper benchmarked a single platform (ARM NEON on Apple Silicon). Two critical questions remained unanswered:

1. **SIMD width effect**: How does x86 AVX2 (256-bit, 8-wide float) compare to ARM NEON (128-bit, 4-wide float) for the same ternary matmul kernel? The theoretical 2× advantage of wider SIMD may be offset by microarchitectural differences (decode width, issue queue depth, memory subsystem).
2. **GPU-to-CPU gap**: How far is CPU SSM inference from GPU SSM inference? If CPU throughput approaches a significant fraction of GPU throughput for the same architecture, the "mathematical reformulation" thesis is strongly supported.

This paper addresses the first question with controlled x86 AVX2 measurements, and provides context for the second using published reference benchmarks. The GPU CUDA baseline measurement was attempted but blocked by environment constraints (Python 3.8, CUDA 11.7, incompatible library versions under WSL2).

Contributions.

1. First x86 AVX2 benchmark of BitMamba-2 under WSL2 and Windows native (MSVC), enabling comparison with ARM NEON results;
2. First native Linux scalar benchmark on pre-AVX2 hardware (Intel i5-3230M, Ivy Bridge), quantifying the SIMD contribution to ternary SSM inference;
3. First-party AVX-512 server benchmark on Intel Xeon Silver 4210R (Cascade Lake, 10 cores, AVX-512 + VNNI), revealing the gap between AVX2 kernel code and AVX-512 hardware potential;
4. Decomposition of the WSL2-to-native performance gap into virtualization (3.1×) and compiler/OS (4.1×) factors;
5. Validation of $\mathcal{O}(1)$ memory on x86 (scalar, AVX2, and AVX-512) and ARM, across four distinct CPUs;
6. Cross-platform comparison across six environments: ARM NEON, x86 WSL2, x86 Windows native, published native Linux AVX2 reference, native Linux scalar (Ivy Bridge), and native Linux AVX-512 (Xeon 4210R);
7. Analysis of factors that affect cross-platform SSM inference: SIMD width, instruction set generation, virtualization, and compilation.

2 Background

We refer the reader to Rasatavohary [2026] for a complete treatment of SSM foundations (S4, Mamba, RWKV, SSD duality, 1.58-bit quantization). Here we summarize only what is needed for the cross-platform analysis.

2.1 SSM Recurrence

The core operation of a discrete SSM is the sequential recurrence:

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

This requires $\mathcal{O}(N)$ state and $\mathcal{O}(N)$ work per step, independent of sequence length — the source of the $\mathcal{O}(1)$ memory property.

2.2 Ternary Matmul

With 1.58-bit weights $w \in \{-1, 0, +1\}$, the matrix-vector product reduces to:

$$(W \cdot x)_i = \sum_{j:w_{ij}=+1} x_j - \sum_{j:w_{ij}=-1} x_j \tag{2}$$

This eliminates floating-point multiplications. The kernel is implemented differently on each SIMD ISA:

- **AVX2**: `_mm256_sign_epi8` (single instruction, 32 int8 lanes);
- **NEON**: explicit mask + negate (`vceqq_s8`, `vandq_s8`, `vnegq_s8`; 16 int8 lanes);
- **GPU CUDA**: standard FP16 matmul via `mamba-ssm` selective scan kernel (not ternary).

2.3 SIMD ISA Comparison

Table 1 summarizes the key differences between the target ISAs.

Table 1: Instruction set comparison for SSM inference kernels.

Property	**NEON (ARM)**	**AVX2 (x86)**	**CUDA (GPU)**
Register width	128-bit	256-bit	32-bit (per thread)
Float32 lanes	4	8	1 (SIMT)
Int8 lanes	16	32	—
Ternary matmul	3 insns	1 insn	FP16 matmul
Parallelism model	SIMD	SIMD	SIMT (thousands)
Memory hierarchy	L1–L2–DRAM	L1–L2–L3–DRAM	Registers–L1–L2–HBM
Typical decode width	8 (M1)	4–6	N/A (warp-based)

3 Experimental Setup

3.1 Hardware Platforms

Table 2: Hardware platforms used for the benchmark.

	ARM (A)	**x86 WSL2 (B)**	**x86 Win native (C)**
CPU	Apple M1	Intel Core i9-10980HK @ 2.40 GHz	
Cores	8 (4P + 4E)	8 cores / 16 threads	
SIMD	NEON 128-bit	AVX2 256-bit	
RAM	16 GB (unified)	23 GB (WSL2)	64 GB
OS	macOS Darwin 25.2	Ubuntu 20.04 (WSL2)	Windows 10 Pro (19045)
Compiler	AppleClang 17.0	GCC 9.4.0	MSVC (VS 2022)

3.2 Models

- **BitMamba-2 255M**: 246 MB weights, 1.58-bit ternary quantization;
- **BitMamba-2 1B**: 614 MB weights, 1.58-bit ternary quantization;
- Both models from `https://huggingface.co/rasatavohary/BitMamba-2-1B` (mirror of Zhayr1/BitMamba-2), inferred via bitmamba.cpp [Zhayr1, 2025].

3.3 Protocol

Identical across both platforms:

- Prompt: `"The future of artificial intelligence in computing is"` (8 tokens)

- Generation lengths: 50 and 200 tokens
- Sampling: temperature 0.7, top-p 0.9, top-k 40, repetition penalty 1.1
- Metric: tokens per second (tok/s), per-token latency, prefill time
- Single cold-start run (model loading included)
- x86 benchmarks run under WSL2 (Windows Subsystem for Linux 2)

4 Results

4.1 ARM NEON Results (from Companion Paper)

Table 3 reproduces the ARM results from Rasatavohary [2026] for reference.

Table 3: BitMamba-2 inference on Apple M1 (ARM NEON). From Rasatavohary [2026].

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

4.2 x86 AVX2 Results (WSL2)

Table 4 presents the x86 AVX2 results under WSL2.

Table 4: BitMamba-2 inference on Intel i9-10980HK (x86 AVX2, WSL2).

Model	Tokens	Speed (tok/s)	Latency/tok	Prefill
BitMamba-2 255M	50	5.52	181.1 ms	836.1 ms
BitMamba-2 255M	200	5.78	173.0 ms	1238.6 ms
BitMamba-2 1B	50	4.21	237.5 ms	1700.5 ms
BitMamba-2 1B	200	3.05	327.8 ms	2381.7 ms

$\mathcal{O}(1)$ memory validation (x86). For the 255M model, the speed variation between 50 and 200 tokens is +4.7% (5.52 $\rightarrow$ 5.78 tok/s), consistent with $\mathcal{O}(1)$ memory. For the 1B model, the speed *decreases* by 27.6% (4.21 $\rightarrow$ 3.05 tok/s), suggesting that the 1B model under WSL2 encounters additional overhead at longer sequence lengths, possibly due to WSL2 memory management or filesystem I/O.

4.3 x86 AVX2 Results (Windows Native, MSVC)

To isolate the WSL2 overhead, we compiled and ran the same bitmamba.cpp on the *same hardware* (i9-10980HK) under Windows 10 natively using MSVC (Visual Studio 2022). Table 5 presents the valid results.

Table 5: BitMamba-2 inference on Intel i9-10980HK (x86 AVX2, Windows native MSVC). Only the 1B/50-token measurement is valid; other runs terminated prematurely (wall time <0.1 s), likely due to an MSVC-specific tokenizer or I/O issue.

Model	Tokens	Speed (tok/s)	Latency/tok	Wall time
BitMamba-2 1B	50	**13.01**	76.9 ms	3.84 s
BitMamba-2 1B	200	(invalid)	—	0.031 s
BitMamba-2 255M	50	(invalid)	—	0.027 s
BitMamba-2 255M	200	(invalid)	—	0.028 s

Validity analysis. The 1B/50-token run completed in 3.84 s with a coherent 13.01 tok/s throughput. The other three runs terminated in <0.1 s without generating tokens, producing spurious throughput values (>1000 tok/s). We attribute this to a tokenizer path or binary compatibility issue under MSVC that causes early termination for certain model/token configurations. Only the valid measurement (13.01 tok/s) is used in subsequent analysis.

4.4 Published Reference: Native Linux x86

The original BitMamba-2 repository [Zhayr1, 2025] reports native Linux x86 performance on an Intel Core i3-12100F (4 cores, AVX2):

Table 6: BitMamba-2 reference performance on native Linux x86 (from Zhayr1 [2025]).

Model	Speed (tok/s)	RAM	Hardware
BitMamba-2 255M	∼146	252 MB	Intel i3-12100F (native Linux)
BitMamba-2 1B	∼53	621 MB	Intel i3-12100F (native Linux)

4.5 Native Linux Scalar Benchmark (Ivy Bridge)

To isolate the contribution of SIMD vectorization, we deploy the benchmark on an Intel Core i5-3230M (Ivy Bridge, 2013), which has SSE4.2 but *lacks* AVX2 and FMA. The bitmamba.cpp codebase was modified to add compile-time SIMD detection: the AVX2 kernel path is now gated behind `__AVX2__` and `__FMA__` preprocessor checks; on pre-AVX2 hardware, the code falls through to a pure scalar implementation. Deployment was automated via Ansible (role-based playbook with rsync, remote compilation, and result retrieval).

Table 7: BitMamba-2 performance on native Linux x86 **without AVX2** (Intel i5-3230M, Ivy Bridge, scalar fallback).

Model	Tokens	Speed (tok/s)	Latency/tok	Prefill
BitMamba-2 255M	50	2.42	413 ms	2 789 ms
BitMamba-2 255M	200	2.48	403 ms	2 786 ms
BitMamba-2 1B	50	0.58	1 724 ms	12 108 ms
BitMamba-2 1B	200	0.58	1 724 ms	12 077 ms

Hardware details. Intel Core i5-3230M @ 2.60 GHz (turbo 3.20 GHz), 2 cores / 4 threads, 15 GB DDR3, Ubuntu 24.04 LTS, kernel 6.8.0-106, GCC 13.3.0. SIMD: SSE4.2 only (no AVX2, no FMA). Compiled with `-O3 -march=native -fopenmp`.

4.6 Native Linux AVX-512 Benchmark (Xeon Cascade Lake)

To evaluate the effect of wider SIMD and server-class hardware, we deploy the benchmark on an Intel Xeon Silver 4210R (Cascade Lake, 2019). This CPU supports AVX-512 (512-bit vectors) and VNNI (Vector Neural Network Instructions for int8 dot products). Deployment follows the same Ansible automation as the scalar benchmark.

We run *two* configurations on the same hardware: (1) the original AVX2 kernel path (256-bit intrinsics, compiled with `-march=native` which enables AVX-512 at the compiler level but the kernel code only uses AVX2 intrinsics), and (2) new native AVX-512 kernels we developed for this experiment, using 512-bit intrinsics (`_mm512_*`) throughout all three hot paths: `rms_norm`, activation quantization, and the ternary matmul.

Hardware details. Intel Xeon Silver 4210R @ 2.40 GHz (turbo 3.20 GHz), 10 cores / 20 threads, 78 GB DDR4, Ubuntu 24.04 LTS, kernel 6.8.0-106, GCC 13.3.0. SIMD: AVX-512F/BW/DQ/VL + VNNI. Compiled with `-O3 -march=native -fopenmp`.

4.6.1 AVX2 Kernel Path on AVX-512 Hardware

Table 8: BitMamba-2 on Xeon 4210R using the **AVX2 code path** (256-bit intrinsics on AVX-512 hardware).

Model	Tokens	Speed (tok/s)	Latency/tok	Prefill
BitMamba-2 255M	50	113.1	8.8 ms	39 ms
BitMamba-2 255M	200	104.5	9.6 ms	38 ms
BitMamba-2 1B	50	47.6	21.0 ms	123 ms
BitMamba-2 1B	200	47.0	21.3 ms	124 ms

4.6.2 Native AVX-512 Kernels

We implemented native AVX-512 kernels for all three critical paths:

- **rms_norm**: 16 floats per iteration using `_mm512_fmadd_ps` and `_mm512_reduce_add_ps` (2× wider than AVX2's 8-float path);
- **Activation quantization**: vectorized float→int32→int16→int8 packing via `_mm512_cvt_roundps_epi32`, `_mm512_cvtepi32_epi16`, and `_mm256_cvtepi16_epi8` (16 values per iteration);
- **Ternary matmul**: 64 int8 elements per iteration using AVX-512 masked operations. Since `_mm512_sign_epi8` does not exist in AVX-512, we replace the AVX2 sign trick with explicit mask comparison: `_mm512_cmpeq_epi8_mask` identifies $w = +1$ and $w = -1$ positions, then `_mm512_mask_mov_epi8` blends x, $-x$, or 0. Accumulation uses `_mm512_cvtepi8_epi16` (256→512 widening) and `_mm512_madd_epi16` for horizontal pairwise int16→int32 reduction, with a final `_mm512_reduce_add_epi32` horizontal sum.

Table 9: BitMamba-2 on Xeon 4210R using **native AVX-512 kernels** (512-bit intrinsics, opt-in via `-DBITMAMBA_USE_AVX512`).

Model	Tokens	Speed (tok/s)	Latency/tok	Prefill
BitMamba-2 255M	50	92.4	10.8 ms	44 ms
BitMamba-2 255M	200	86.0	11.6 ms	43 ms
BitMamba-2 1B	50	32.6	30.7 ms	177 ms
BitMamba-2 1B	200	32.6	30.7 ms	176 ms

4.6.3 AVX-512 Frequency Throttling: A Negative Result

Table 10 directly compares both configurations on the same hardware.

Table 10: AVX2 path vs. native AVX-512 on the *same* Xeon 4210R — wider SIMD is *slower* due to frequency throttling.

Model	Kernel	SIMD width	tok/s	Δ
255M	AVX2 path	256-bit (32 int8)	**113.1**	baseline
255M	AVX-512 native	512-bit (64 int8)	92.4	**−18**%
1B	AVX2 path	256-bit (32 int8)	**47.6**	baseline
1B	AVX-512 native	512-bit (64 int8)	32.6	**−31**%

Analysis. The native AVX-512 kernels are 18–31% *slower* than the AVX2 path on the same hardware. This is a direct consequence of Intel's AVX-512 frequency licensing on Cascade Lake (microarchitecture family 6, model 85): when the CPU detects 512-bit instructions, it applies a *mandatory* frequency reduction of approximately 20–30% to stay within the thermal design power (TDP) envelope. This throttling affects the *entire core*, not just the AVX-512 instructions, so even surrounding scalar and AVX2 code runs slower during AVX-512 execution windows.

The theoretical throughput advantage of AVX-512 for the ternary matmul is:

$$\text{AVX-512 speedup (ideal)} = \frac{64 \text{ int8/iter (512-bit)}}{32 \text{ int8/iter (256-bit)}} = 2.0\times \tag{3}$$

However, the observed penalty exceeds the gain:

$$\text{Net effect} = \frac{2.0\times \text{ (width gain)}}{1.25\text{–}1.45\times \text{ (throttle penalty)}} = 1.38\text{–}1.60\times \text{ (expected)} \tag{4}$$

The measured result is even worse (0.69–0.82×), suggesting additional factors beyond frequency throttling:

1. **Missing `_mm512_sign_epi8`**: the AVX2 ternary multiply uses a single `_mm256_sign_epi8` instruction (sign propagation). AVX-512 lacks this intrinsic, requiring a 4-instruction sequence (negate, two mask comparisons, two masked moves) that adds latency;
2. **Port contention**: the mask-based approach generates more μops that compete for execution ports, reducing throughput even at the same clock;
3. **Thermal steady-state**: the 1B model (longer execution) shows a larger penalty (−31% vs. −18%) because sustained AVX-512 execution reaches deeper thermal throttling states.

Practical conclusion. Based on these results, AVX-512 native kernels are **not beneficial on Cascade Lake** for ternary SSM matmul. We therefore gate AVX-512 behind an opt-in compile flag (`-DBITMAMBA_USE_AVX512`) and default to the AVX2 code path even on AVX-512 hardware. The results reported in all other sections of this paper use the AVX2 path (113.1/47.6 tok/s).

AVX-512 may become beneficial on newer microarchitectures where frequency throttling is reduced (Intel Ice Lake client, Sapphire Rapids) or eliminated (AMD Zen 4/5 with AVX-512 at full frequency). This is a priority target for future benchmarks.

4.7 Cross-Platform Comparison

Table 11 consolidates all available results.

Table 11: Cross-platform comparison: scalar → AVX2 → AVX-512 → ARM NEON. Models at 1.58-bit ternary quantization.

Platform	**Environment**	**SIMD**	**1B, 5**
x86 scalar (i5-3230M)	native Linux	SSE4.2 (scalar)	
x86 AVX2 (i9-10980HK)	WSL2 (GCC 9.4)	AVX2 256-bit	
x86 AVX2 (i9-10980HK)	Win native (MSVC)	AVX2 256-bit	
ARM NEON (Apple M1)	native macOS	NEON 128-bit	
x86 AVX-512 (Xeon 4210R)	**native Linux**	**AVX2 path on AVX-512**	
x86 AVX2 (i3-12100F)	native Linux (ref.)	AVX2 256-bit	
External baselines (different models — see caveats below)			
Apple M1	TinyLlama 1.1B Q4 [Zhang et al., 2024]	NEON	
Cloud GPU	Claude 3.5 Haiku [Anthropic, 2024] †	GPU	
Cloud GPU	GPT-4o Mini [OpenAI, 2024] †	GPU	

*Community benchmark via llama.cpp [Gerganov and contributors, 2023]. †Throughput from Artificial Analysis [Artificial Analysis, 2026] (API measurement, March 2026).

Caveat on model sizes: BitMamba-2 models are 255M and 1B parameters with 1.58-bit ternary quantization. Claude 3.5 Haiku and GPT-4o Mini are *proprietary* models with undisclosed sizes, widely estimated at ∼8–20B parameters [Anthropic, 2024, OpenAI, 2024] — **10–80× larger**. TinyLlama is 1.1B (Q4, 4-bit quantized Transformer). The throughput comparison across these models is therefore *not* parameter-normalized: higher tok/s for the smaller BitMamba-2 models is expected. The comparison is included to contextualize "interactive speed" thresholds (>30 tok/s for conversational use), not to claim that a 255M SSM outperforms a multi-billion parameter Transformer at the *same task quality*.

5 Analysis

5.1 SIMD Width Effect: NEON vs AVX2

Theoretical prediction. AVX2 processes 2× more float32/int8 elements per instruction than NEON (8 vs 4 float lanes, 32 vs 16 int8 lanes). If the ternary matmul kernel is purely SIMD-bound, we expect:

$$\frac{\text{AVX2 throughput}}{\text{NEON throughput}} \approx 2\times \tag{5}$$

Observed ratio (native x86 reference vs ARM NEON). Using the published native x86 reference figures:

$$\frac{146 \text{ tok/s (AVX2, i3-12100F native)}}{82.5 \text{ tok/s (NEON, M1)}} = 1.77\times \tag{6}$$

The observed 1.77× ratio is close to the theoretical 2×, with the shortfall attributable to:

- The Apple M1's 8-wide decode pipeline vs. the i3-12100F's narrower decode;
- NEON's ternary matmul requires 3 instructions vs. AVX2's 1, partially offset by M1's superscalar execution;
- M1's large L1 cache (192 KB I + 128 KB D) vs. typical Intel L1 (32 KB I + 48 KB D).

For the 1B model: $53/27.6 = 1.92\times$, even closer to the theoretical 2×, suggesting that the larger model is more SIMD-bound (less cache/memory bottleneck).

AVX-512 hardware: AVX2 path vs. native kernels. The Xeon 4210R provides a unique opportunity to evaluate both paths on the same hardware. Using the AVX2 code path, it achieves $113.1/82.5 = 1.37\times$ ARM NEON for 255M and $47.6/27.6 = 1.72\times$ for 1B — below the AVX2 reference (1.77–1.92×), explained by lower clock speed (2.4 vs. 3.3 GHz). When switching to native AVX-512 kernels (512-bit intrinsics processing 64 int8/iteration), performance *drops* to 92.4/32.6 tok/s (−18%/−31%), due to Cascade Lake's aggressive frequency throttling (see Section 4.6). This establishes that **wider SIMD is not always better**: the frequency/width trade-off is architecture-dependent, and AVX2 remains optimal on pre-Ice Lake Intel hardware.

WSL2 overhead decomposition. The Windows native benchmark on the *same hardware* (i9-10980HK) allows us to decompose the total performance gap for the 1B model:

Table 12: Performance decomposition for BitMamba-2 1B (50 tokens) across six x86 environments.

Environment	tok/s	vs. WSL2	vs. scalar	Factor
x86 scalar Linux (i5-3230M, GCC)	0.58	0.14×	1.0×	no SIMD baseline
x86 WSL2 (i9-10980HK, GCC 9.4)	4.21	1.0×	7.3×	AVX2 + newer HW
x86 Windows native (i9-10980HK, MSVC)	13.01	3.1×	22.4×	+ no WSL2
x86 AVX-512 Linux (Xeon 4210R, GCC)	47.6	11.3×	82×	+ server HW + native
x86 native Linux (i3-12100F, GCC, ref.)	∼53	12.6×	91×	+ higher IPC clock

The total 12.6× gap decomposes into three layers:

1. **WSL2 virtualization**: 3.1× — measured directly by comparing WSL2 vs. Windows native on the *same* CPU. This includes Hyper-V memory virtualization and cross-kernel filesystem access (`/mnt/c/`);
2. **Compiler/OS**: 4.1× — MSVC on Windows vs. GCC on native Linux. This includes MSVC's potentially suboptimal AVX2 intrinsics codegen vs. GCC, and Windows vs. Linux OS-level differences (scheduler, memory allocator, I/O);
3. **Hardware**: the i3-12100F (12th gen Alder Lake) has higher IPC than the i9-10980HK (10th gen Comet Lake), but this should account for at most 1.2–1.5×, leaving the remainder attributable to software factors.

This finding is valuable: it establishes that **WSL2 introduces a 3× overhead for CPU inference**, and that the remaining gap is dominated by compiler optimization quality (GCC vs. MSVC) and OS-level efficiency (Linux vs. Windows). For valid benchmarking, native Linux with modern GCC or Clang is essential.

5.2 SIMD vs. Scalar: Quantifying the AVX2 Contribution

The native Linux scalar benchmark allows us to isolate the SIMD contribution for the first time. Comparing the i5-3230M scalar result against the published i3-12100F AVX2 reference (both on native Linux with GCC):

$$\frac{146 \text{ tok/s (AVX2, i3-12100F)}}{2.42 \text{ tok/s (scalar, i5-3230M)}} = 60.3\times \quad \text{(255M model)} \tag{7}$$

$$\frac{53 \text{ tok/s (AVX2, i3-12100F)}}{0.58 \text{ tok/s (scalar, i5-3230M)}} = 91.4\times \quad \text{(1B model)} \tag{8}$$

These ratios far exceed the theoretical 8× lane width ratio (AVX2 processes 32 int8 vs. scalar's 1 per instruction). The amplification factors are:

- **Lane width**: 8× (AVX2 256-bit integer);
- **Instruction specialization**: AVX2 provides `_mm256_sign_epi8` (ternary multiply in 1 instruction) and `_mm256_madd_epi16` (horizontal pairwise accumulation), which replace multi-instruction scalar sequences;
- **Hardware generation**: the i3-12100F (Alder Lake, 2022) has higher IPC, larger caches, and faster memory than the i5-3230M (Ivy Bridge, 2013) — estimated at 1.5–2×;
- **Residual**: after accounting for lane width (8×), instruction specialization (∼2–3×), and hardware generation (∼1.5–2×), the total predicted ratio is 24–48×, roughly consistent with the observed 60–91× when considering memory bandwidth differences.

This decomposition has a practical implication: **any CPU deployment of ternary SSMs must target AVX2 or NEON at minimum**. The scalar fallback is 60–91× slower and is not viable for interactive use. This motivates the compile-time SIMD detection added to bitmamba.cpp as part of this work.

5.3 $\mathcal{O}(1)$ Memory Across Platforms

Table 13 verifies speed stability across sequence lengths.

The 1B model validates $\mathcal{O}(1)$ on ARM NEON (+1.1%), scalar Linux (0.0% — perfectly stable), and the Xeon 4210R (−1.3%). The 255M model validates on all platforms except the Xeon, where we observe a −7.6% drop at 200 tokens. This is likely due to thermal throttling on the Xeon under sustained AVX workloads (the 1B model, being slower, generates less sustained heat). The 1B model on WSL2 shows anomalous degradation (−27.6%), attributed to WSL2 memory pressure.

The scalar result is particularly significant: $\mathcal{O}(1)$ holds even without SIMD (0.0% variation), confirming it is a property of the SSM recurrence, not of the SIMD implementation.

Table 13: Speed stability across sequence lengths ($\mathcal{O}(1)$ validation).

Platform	50 tok	200 tok	Δ	$\mathcal{O}(1)$?
ARM NEON (255M)	82.5	81.7	−1.0%	✓
x86 AVX2 WSL2 (255M)	5.52	5.78	+4.7%	✓
x86 scalar Linux (255M)	2.42	2.48	+2.5%	✓
x86 AVX-512 Xeon (255M)	113.1	104.5	−7.6%	∼
ARM NEON (1B)	27.6	27.9	+1.1%	✓
x86 AVX2 WSL2 (1B)	4.21	3.05	−27.6%	×
x86 scalar Linux (1B)	0.58	0.58	0.0%	✓
x86 AVX-512 Xeon (1B)	47.6	47.0	−1.3%	✓

5.4 Scaling Behavior

Table 14: Parameter scaling: 255M → 1B throughput ratio.

Platform	255M (tok/s)	1B (tok/s)	Ratio
ARM NEON (M1)	82.5	27.6	2.99×
x86 AVX2 native (ref.)	146	53	2.75×
x86 AVX-512 Xeon (4210R)	113.1	47.6	2.38×
x86 AVX2 Win native	—	13.01	—
x86 AVX2 WSL2	5.52	4.21	1.31×
x86 scalar Linux (i5-3230M)	2.42	0.58	4.17×

ARM NEON and native x86 AVX2 show comparable scaling (∼ 3×), well below the parameter ratio (3.92×), confirming sub-linear throughput scaling. The Xeon 4210R shows a similar ratio (2.38×), consistent with the same AVX2 code path. The WSL2 result (1.31×) is anomalously flat, further evidence of a non-compute bottleneck (I/O or memory virtualization). The scalar baseline shows the steepest scaling (4.17×), closer to the parameter ratio, suggesting that without SIMD parallelism the throughput is more directly proportional to model size.

5.5 GPU-to-CPU Transposition Ratio

Defining the transposition ratio. The transposition ratio measures whether CPU inference can approach GPU inference throughput:

$$\text{Transposition ratio} = \frac{\text{CPU throughput}}{\text{GPU throughput (reference)}} \tag{9}$$

Important caveat: model sizes are not comparable. A direct GPU-to-CPU comparison requires benchmarking the *same model* on both platforms. Our BitMamba-2 models (255M, 1B parameters) are too small to run on proprietary GPU APIs (Claude 3.5 Haiku, GPT-4o Mini), and these proprietary models (estimated ∼8–20B parameters [Anthropic, 2024, OpenAI, 2024]) cannot run on our CPU setup. We therefore present two distinct comparison frameworks:

(a) Same-family comparison: Mamba on CPU vs. Mamba on GPU. The most rigorous comparison uses Mamba models on both platforms. While we could not benchmark

BitMamba-2 on GPU (the ternary 1.58-bit format has no GPU runtime), published benchmarks for standard Mamba models (FP16) [Gu and Dao, 2023b] provide context. A Mamba-130M on a single A100 GPU achieves ∼1000–2000 tok/s in FP16, far exceeding any CPU result. However, this comparison conflates precision (FP16 vs. 1.58-bit) and hardware cost ($10K+ GPU vs. consumer CPU).

(b) Interactive speed comparison (parameter-aware). Rather than claiming CPU "beats" GPU, we compare against the *interactive speed threshold* for conversational use (∼30 tok/s). Cloud API throughput figures from Artificial Analysis [Artificial Analysis, 2026] — Claude 3.5 Haiku [Anthropic, 2024]: ∼61 tok/s, GPT-4o Mini [OpenAI, 2024]: ∼59 tok/s — contextualize what "interactive speed" means for production LLMs, but these models are **10–80× larger** than BitMamba-2.

The key finding is that CPU SSM inference achieves interactive speeds even for the 1B model:

- Native x86 (i3-12100F): 146 tok/s (255M), 53 tok/s (1B) — well above interactive threshold;
- x86 AVX-512 (Xeon 4210R): 113 tok/s (255M), 47.6 tok/s (1B) — interactive on server hardware;
- ARM NEON (M1): 82.5 tok/s (255M), 27.6 tok/s (1B) — interactive for the 255M model;
- TinyLlama 1.1B (Q4, Transformer) on M1: ∼33 tok/s [Zhang et al., 2024] — comparable to BitMamba-2 1B on ARM, but using 4-bit (not 1.58-bit) quantization and 4× more memory.

Normalized comparison: tok/s per billion parameters. To partially account for model size differences, we compute a throughput-per-parameter metric:

Table 15: Throughput normalized by model size. This metric is approximate and does not account for differences in model quality, quantization, or architecture.

Model	Platform	Params	tok/s	tok/s/B
BitMamba-2 255M	x86 AVX2 (i3-12100F)	0.26B	146	561
BitMamba-2 255M	ARM NEON (M1)	0.26B	82.5	317
BitMamba-2 1B	x86 AVX2 (i3-12100F)	1.0B	53	53
BitMamba-2 1B	ARM NEON (M1)	1.0B	27.6	27.6
TinyLlama 1.1B	ARM NEON (M1, Q4)	1.1B	∼33	∼30
Claude 3.5 Haiku†	Cloud GPU	∼8–20B	61	3–8
GPT-4o Mini†	Cloud GPU	∼8B	59	∼7

†Parameter counts are external estimates; throughput from Artificial Analysis [2026].

The normalized metric (tok/s per billion parameters) tells a different story: BitMamba-2 on CPU achieves 53–561 tok/s/B, while cloud GPU models achieve only 3–8 tok/s/B. This reflects the fundamental advantage of smaller, quantized SSM models: each parameter is *computationally cheaper* (ternary multiply = addition) and the sequential recurrence avoids the quadratic attention cost. However, this metric does *not* imply equivalent task quality — the cloud models are far more capable.

Conclusion on transposition. The GPU-to-CPU transposition is viable in the sense that **CPU SSM inference achieves interactive speeds for models up to 1B parameters**, using only consumer hardware and sub-700 MB memory. The comparison with cloud GPU APIs contextualizes the throughput but is not a like-for-like benchmark due to the 10–80× model size disparity. A rigorous GPU-to-CPU comparison of the *same* BitMamba-2 models on GPU CUDA remains future work.

6 Discussion

Is the GPU-to-CPU transposition viable? Our results support the viability of GPU-to-CPU transposition for SSM inference *at the sub-billion parameter scale*. The combination of SSM recurrence ($\mathcal{O}(1)$ memory, sequential computation) and ternary quantization (multiplications replaced by additions) creates a computation profile that is *natively aligned* with CPU architecture strengths. CPU SSM inference achieves interactive speeds (>30 tok/s) for models up to 1B parameters on consumer hardware. However, we stress that this does not mean CPU "beats" GPU in general: the comparison with cloud GPU APIs (Claude 3.5 Haiku, GPT-4o Mini) is confounded by a 10–80× model size disparity. The transposition thesis is that *for a given model size*, the right mathematical reformulation (SSM + ternary quantization) makes CPU competitive — not that small CPU models match the capability of large GPU models.

SIMD vectorization is the dominant factor. The scalar baseline reveals that SIMD (AVX2 or NEON) is not merely an optimization but a *prerequisite* for viable CPU SSM inference. The 60–91× gap between scalar and AVX2 dwarfs all other factors (WSL2 overhead at 3.1×, compiler at 4.1×, hardware generation at 1.5–2×). This has implications for deployment targeting: pre-Haswell x86 CPUs (pre-2013) and ARM cores without NEON are effectively excluded from interactive SSM inference.

Which CPU ISA is better for SSM inference? Based on the native reference figures, x86 AVX2 achieves 1.77–1.92× the throughput of ARM NEON, close to the theoretical 2× SIMD width ratio. However, ARM has structural advantages: lower power consumption, unified memory architecture (no CPU-to-GPU data transfer), and wider deployment in edge/mobile devices. The "best" ISA depends on the deployment context:

- **Throughput-optimal**: x86 AVX2 on high-clock desktop CPUs; AVX-512 with native kernels (not yet implemented) could double this;
- **Server/multi-tenant**: x86 AVX-512 Xeon (47.6 tok/s for 1B with only AVX2 code path; native AVX-512 kernels would unlock the full potential);
- **Efficiency-optimal**: ARM NEON (lower perf/watt);
- **Edge/mobile**: ARM NEON (ubiquitous on phones, tablets, SoCs);
- **Not viable**: x86 pre-AVX2 (scalar), as demonstrated by the i5-3230M results.

AVX-512: wider is not always faster. The Xeon 4210R experiment is, to our knowledge, the first controlled comparison of AVX2 vs. native AVX-512 kernels for ternary SSM matmul on the *same* hardware. The result is a clear **negative**: native AVX-512 kernels are 18–31% slower

than the AVX2 path on Cascade Lake. This is caused by the compound effect of (a) frequency throttling (−20–30% clock reduction when executing 512-bit instructions), (b) the absence of `_mm512_sign_epi8` requiring a more expensive mask-based replacement, and (c) increased thermal throttling under sustained load (the 1B model, with longer execution, suffers more: −31% vs. −18%).

This result has broader implications beyond bitmamba.cpp: any workload where the compute kernel relies on AVX2-specific tricks (`_mm256_sign_epi8`, `_mm256_maddubs_epi16`) should *not* blindly port to AVX-512 on Cascade Lake. The correct strategy is to keep AVX2 as the default and gate AVX-512 behind a compile-time flag, enabling it only on microarchitectures where the frequency penalty is reduced (Ice Lake+) or absent (AMD Zen 4/5). We implement this strategy in bitmamba.cpp with the `BITMAMBA_USE_AVX512` define.

WSL2 overhead is measurable and decomposable. The Windows native benchmark quantifies the WSL2 overhead at 3.1× for the 1B model. Combined with the compiler/OS gap (4.1× MSVC Windows vs. GCC Linux), the total overhead reaches 12.6×. This decomposition is a practical contribution: practitioners can predict the expected performance loss when running under WSL2 or Windows native vs. native Linux.

Limitations.

- GPU CUDA baseline could not be measured directly due to Python/CUDA environment incompatibilities under WSL2 (Python 3.8, CUDA 11.7, missing mamba-ssm package);
- The Windows native MSVC build produced valid results only for the 1B/50-token configuration; the 255M model and 200-token runs terminated prematurely, likely due to a tokenizer path issue under MSVC — a complete native benchmark requires debugging the MSVC build;
- Published AVX2 reference figures (Zhayr1) were obtained on different x86 hardware (i3-12100F vs. our i9-10980HK), conflating hardware generation and SIMD effects;
- The AVX-512 negative result (native kernels slower than AVX2) is specific to Cascade Lake; it may not generalize to Ice Lake+ or AMD Zen 4 where throttling is reduced. Our AVX-512 ternary matmul uses a mask-based approach (4 instructions) instead of the single-instruction `sign_epi8` trick — an alternative AVX-512 strategy using lookup tables or VNNI may yield better results;
- Energy consumption was not measured;
- Four distinct x86 CPUs (i9-10980HK, i3-12100F, i5-3230M, Xeon 4210R) provide breadth but complicate direct comparison due to differing clock speeds, core counts, and memory subsystems.

7 Related Work

SSM architectures. S4 [Gu et al., 2022] introduced structured state spaces. Mamba [Gu and Dao, 2023a] added input-dependent selectivity. RWKV [Peng et al., 2023] achieves $\mathcal{O}(1)$ memory via linear recurrence. The SSD duality [Dao and Gu, 2024] unifies SSMs and attention.

CPU inference. llama.cpp [Gerganov and contributors, 2023] is the reference Transformer CPU runtime. BitMamba-2 [Zhayr1, 2025] targets SSM inference with ternary quantization. Our companion paper [Rasatavohary, 2026] provides the first ARM NEON port.

Quantization. BitNet [Ma et al., 2024] demonstrates ternary weight viability. Gholami et al. [Gholami et al., 2022] survey quantization.

SIMD benchmarking. Fog [Fog, 2024] provides detailed microarchitectural analysis of x86 CPUs. ARM NEON [Arm, 2023] and Intel AVX2 [Int, 2024] intrinsics guides document the ISA details.

8 Conclusion

We present the first cross-platform benchmark of BitMamba-2 SSM inference spanning the full x86 SIMD hierarchy (scalar → AVX2 → AVX-512) and ARM NEON. Our key findings are:

1. **SIMD width ratio**: Native x86 AVX2 achieves 1.77–1.92× ARM NEON throughput, close to the theoretical 2× ratio, confirming that the ternary matmul kernel is SIMD-bound;
2. **AVX-512: wider is not faster on Cascade Lake**: The Xeon 4210R achieves 113.1 tok/s (255M) and 47.6 tok/s (1B) using the AVX2 code path. Native AVX-512 kernels (512-bit, 64 int8/iter), implemented and benchmarked as part of this work, are 18–31% *slower* due to frequency throttling, the absence of `_mm512_sign_epi8`, and thermal steady-state effects. AVX-512 is gated behind an opt-in flag for future architectures;
3. **SIMD vs. scalar**: The AVX2-to-scalar gap reaches 60–91×, far exceeding the 8× lane width ratio. This quantifies for the first time the combined effect of SIMD width, specialized integer instructions, and hardware generation on ternary SSM inference, establishing SIMD as the *dominant* performance factor;
4. **WSL2 overhead decomposition**: WSL2 adds a 3.1× penalty vs. Windows native (MSVC); the total gap to native Linux is 12.6×, decomposed into virtualization (3.1×), compiler/OS (4.1×), and hardware generation ($\sim 1\times$);
5. $\mathcal{O}(1)$ **memory**: Validated on ARM, x86 AVX2, x86 scalar, and x86 AVX-512 (255M and 1B models), with the 1B WSL2 anomaly attributed to virtualization overhead. The scalar result (0.0% variation) and Xeon result (-1.3%) confirm $\mathcal{O}(1)$ is a property of the SSM recurrence, independent of SIMD level;
6. **GPU-to-CPU transposition**: CPU SSM inference achieves interactive speeds (146 tok/s i3-12100F, 113 tok/s Xeon, 82.5 tok/s M1 for 255M) on consumer hardware. Cloud GPU APIs (Claude 3.5 Haiku: 61 tok/s, GPT-4o Mini: 59 tok/s) serve models 10–80× larger, so the raw tok/s comparison is not parameter-normalized. The transposition is viable for sub-billion SSMs: the normalized metric (tok/s per billion parameters) is 27–561 on CPU vs. 3–8 on cloud GPU;
7. **Deployment**: SSMs with ternary quantization are production-viable on consumer and server CPU hardware with SIMD, achieving interactive speeds (>47 tok/s for 1B on Xeon

server, >113 tok/s for 255M) with sub-700 MB memory footprint. Pre-AVX2 hardware (<1 tok/s for 1B) is not viable.

These results support the broader thesis of the GPU-to-CPU/ARM transposition programme: rather than optimizing GPU-native architectures for CPU, seeking *mathematical reformulations* that make the computation structurally advantageous for non-GPU hardware yields competitive or superior performance.

Future work.

- **AVX-512 on newer architectures**: our native AVX-512 kernels (implemented, benchmarked, slower on Cascade Lake) should be re-evaluated on Ice Lake, Sapphire Rapids (AMX), and AMD Zen 4/5 where frequency throttling is reduced or absent. VNNI `_mm512_dpbusd_epi32` for the ternary matmul remains untested;
- Native Linux AVX2 benchmark on modern x86 (i3-12100F or newer) to obtain first-party AVX2 measurements (the current AVX2 reference figures are from Zhayr1 [2025]);
- GPU CUDA baseline with compatible environment (Python 3.10+, CUDA 12+, mamba-ssm);
- AVX-512 benchmarks on Intel Sapphire Rapids (AMX) and Ice Lake to compare Cascade Lake VNNI vs. newer microarchitectures;
- ARM SVE/SME targets (Apple M4, AWS Graviton 4);
- Energy measurements (perf/watt), especially to compare the scalar baseline's energy cost vs. SIMD;
- Long-context scaling (4K, 16K, 64K tokens);
- RWKV cross-platform benchmark as a second SSM family.

Reproducibility. All code, build scripts, and benchmark automation are available at https://github.com/rasata/bitmamba.cpp. The benchmark scripts (`run-all-benchmarks.sh`, `run-linux-native.` reproduce the complete pipeline on WSL2/Linux with a single command. The scalar benchmark was deployed via an Ansible role (inventory, setup, benchmark, and result-fetch playbook) included in the research repository. Model weights are mirrored at https://huggingface.co/rasatavohary/BitMamba-2-1B and https://huggingface.co/rasatavohary/BitMamba-2-0.25B.

References

Anthropic. Claude 3.5 Haiku: Fast and cost-effective AI model. https://www.anthropic.com/claude/haiku, 2024. Estimated parameter count undisclosed; widely estimated at ∼8–20B parameters. Throughput ∼61 tok/s reported by Artificial Analysis (March 2026).

ARM NEON Intrinsics Reference. Arm Ltd., 2023. https://developer.arm.com/architectures/instruction-sets/intrinsics.

Artificial Analysis. Artificial analysis — LLM performance leaderboard. https://artificialanalysis.ai/leaderboards/models, 2026. Independent LLM throughput

benchmarks. Accessed March 2026. Reports output tok/s measured via API under standard load.

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

Agner Fog. The microarchitecture of Intel, AMD, and VIA CPUs. 2024. https://www.agner.org/optimize/microarchitecture.pdf.

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*, 2023a.

Albert Gu and Tri Dao. Mamba 130m and 790m pretrained models. https://huggingface.co/state-spaces/mamba-130m-hf, 2023b. FP16 Mamba checkpoints for research. See also mamba-790m-hf.

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.

Intel Intrinsics Guide — AVX2. Intel Corporation, 2024. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/.

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.

OpenAI. GPT-4o mini. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024. Estimated parameter count undisclosed; widely estimated at ∼8B parameters. Throughput ∼59 tok/s reported by Artificial Analysis (March 2026).

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

Gabriel Zo-Hasina Rasatavohary. State space models as CPU-native neural network architectures: Experimental evidence from ARM NEON inference with 1.58-bit quantized Mamba. engrXiv preprint, 2026. Aquantic Research. First ARM NEON port of BitMamba-2.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. https://github.com/jzhang38/TinyLlama, 2024. 1.1B parameter Transformer, 4-bit quantized (Q4_K_M) via llama.cpp. arXiv:2401.02385.

Zhayr1. bitmamba.cpp: Ultra-lightweight C++ inference engine for BitMamba-2. https://github.com/Zhayr1/bitmamba.cpp, 2025. DOI: 10.5281/zenodo.18394665. Weights mirrored at https://huggingface.co/rasatavohary/BitMamba-2-1B and https://huggingface.co/rasatavohary/BitMamba-2-0.25B.