REAL-TIME NEURAL AUDIO ON APPLE SILICON: BENCHMARKING INFERENCE FRAMEWORKS UNDER REALISTIC DAW CONTENTION

Dharanipathi Rathna Kumar Balasubramaniam, Saravanabalagi Ramachandran and Joseph Timoney

Department of Computer Science
Maynooth University
Maynooth, Ireland
dharanipathirk@gmail.com, saravanabalagi@gmail.com, joseph.timoney@mu.ie

ABSTRACT

Neural network models are increasingly deployed in audio plugins across a wide range of applications, including amplifier emulation, effects modeling, and synthesis. These models typically run on the CPU, and libraries such as `RTNeural` and `anira` have emerged to support this, but their backends prioritize portability over deep integration with any single processor architecture. This leaves performance on the table on macOS, which is widely adopted in professional music production, and Apple's current Mac lineup which runs on Apple Silicon. Apple's `BNNSGraph` API compiles the entire model graph ahead of time, enabling operation fusion, weight repacking, and memory copy elimination while leveraging dedicated matrix co-processors on the chip. `BNNSGraph` is designed for real-time use cases, so it can run in the audio thread without the buffering and worker-thread scheme used by `anira`. We evaluate widely used inference options like `BNNSGraph`, `RTNeural`, `LibTorch`, `ONNX Runtime`, and `anira` on model architectures commonly used in neural audio plugins. We move beyond the common practice of reporting performance measured in isolation. Isolated benchmarks provide a useful first estimate but do not reflect the contention that arises when a neural plugin shares the CPU with other plugins in a mix. To capture these effects, we build a benchmark framework on Tracktion Engine that constructs mix sessions with configurable plugin loads running alongside the neural plugins. By sweeping buffer size and contention level, we examine whether backend rankings established in isolation hold under realistic workload pressure, and whether platform-specific inference translates to lower tail latency and fewer deadline violations in a full mix.

1. INTRODUCTION

The architecture of Digital Audio Workstations (DAWs) has remained largely unchanged since the mid-1990s [1]. The DAW registers a callback with the operating system's audio subsystem—Core Audio on macOS [2], WASAPI on Windows [3]—and the audio driver invokes that callback at fixed intervals determined by the buffer size and sample rate. Each invocation must return before the next hardware interrupt; if it does not, the output buffer underruns and the result is an audible glitch. Within each callback, the DAW traverses its plugin graph synchronously: every insert effect on every active track must finish processing its buffer before the callback can return [4]. Although offline bounce and freeze paths can sidestep this cost, any scenario involving live monitoring, parameter adjustment during playback, or master-bus printing requires every plugin to meet the synchronous callback deadline. Plugin developers have long designed around this constraint, but neural network inference introduces new pressure on the callback budget [5, 6, 7].

Analog audio equipment—tube amplifiers, tape machines, optical compressors—exhibits nonlinear, time-varying behaviour that is difficult to capture with closed-form DSP. Neural networks can learn these input–output relationships directly from recorded measurements, making them a practical tool for audio effect modeling [5, 8]. Recurrent architectures based on the Long Short-Term Memory (LSTM) [9] were among the first to achieve perceptually convincing guitar amplifier emulation in real time. Temporal Convolutional Networks (TCNs) [10, 11] and WaveNet-style architectures [12] followed, offering parallelizable convolutions in place of sequential recurrence. Open-source projects such as Neural Amp Modeler (NAM) [13] and GuitarML [14] have broadened access to these models, and a recent survey covers the broader deployment landscape [15]. Most deployed neural audio plugins run inference on the CPU, as offloading to a GPU or Neural Processing Unit (NPU) introduces latency incompatible with the real-time callback model. A round trip to the GPU involves kernel launch overhead, data transfer, and synchronization; for the small buffer sizes preferred for live performance (64–128 samples at $48\,\mathrm{kHz}$, i.e. $1.3\,\mathrm{ms}$ to $2.7\,\mathrm{ms}$), this overhead can exceed the deadline itself [16]. GPU Audio [17] has demonstrated GPU-accelerated processing at larger buffer sizes, but low-latency playability remains a challenge. Also, current consumer NPU APIs, to the best of our knowledge, do not expose a callback-safe path for low-latency plugin inference.

If the neural network inference engine is not real-time safe—because it allocates memory, acquires locks, or blocks on I/O—a common recourse is to move inference to a separate thread and bridge the audio callback with a lock-free buffer. This is the approach taken by `anira` [7, 18], which wraps `LibTorch` [19], `ONNX Runtime` [20], and TensorFlow Lite behind a thread pool and ring buffer. The audio thread never blocks, but the architecture adds at least one buffer of latency for the asynchronous handoff, which must be declared to the host's delay compensation graph. Neutone [21] takes a similar approach with additional provisions for variable buffer sizes and sample rate conversion. On the other hand, `RTNeural` [6] avoids the threading problem entirely by implementing inference as C++ templates compiled at build time, producing code that is real-time safe by construction. Its main limitation is a restricted set of supported layer types and activation functions, which constrains the architectures that can be deployed.

macOS is widely used in professional music production—

Avanzini et al. report 52% macOS usage among DAW users [22]. Apple's current Mac lineup runs on Apple Silicon processors, which include a matrix co-processor known informally as AMX (Apple Matrix Extensions) [23]. AMX is not a discrete accelerator with its own address space—it is a co-processor tightly coupled to the CPU core. Unlike a GPU or NPU, AMX incurs no kernel launch overhead, no data transfer, and no synchronization cost. It extends the CPU's arithmetic capabilities with wide outer-product operations on matrix tiles, where conventional SIMD (NEON on ARM) is limited to narrower dot-product and element-wise operations. ARM's Scalable Matrix Extension (SME) [24] standardizes a similar matrix unit in the public ARM architecture. Audio neural networks occupy a middle ground in operand size: an LSTM hidden state of 20–96 units or a convolutional layer with 8–48 channels is too small for a GPU to overcome its dispatch overhead, yet large enough for AMX to outperform scalar and NEON paths.

Apple's Core ML [25] framework provides an on-device machine learning runtime for all Apple platforms, and `BNNSGraph` [26] is its ahead-of-time (AOT) graph compiler, which exposes AMX through graph compilation. A CoreML model is compiled once at load time, producing an execution plan tailored to the specific chip variant—the compiler knows the cache hierarchy, the available AMX operations, and the optimal tile sizes for the target core. The BNNSGraph's compiled graph fuses adjacent operations, repacks weights for cache-line alignment, and eliminates intermediate memory copies. At inference time, a single call processes the entire audio buffer. Apple states that this call performs no heap allocation, acquires no locks, and makes no blocking system calls when used with a pre-compiled graph and pre-allocated workspace [27]; we verified this with RealtimeSanitizer [28] as described in Section 3. In contrast, `RTNeural` fixes layer shapes at compile time via C++ templates and does not perform chip-specific optimization at load time. `LibTorch`, when built against Apple's Accelerate framework, reaches AMX indirectly through Accelerate's BLAS [29] routines but does not fuse operations or repack weights for the target chip the way `BNNSGraph` does. `ONNX Runtime` uses its own built-in math library, MLAS [30], which contains processor-optimized GEMM kernels using NEON intrinsics directly rather than the platform BLAS—meaning it does not reach AMX on Apple Silicon. In our tests we did not observe AMX usage by `ONNX Runtime`.

The models used in neural audio plugins are typically stateful: LSTMs carry hidden and cell state across buffers, while TCN and WaveNet models carry dilated convolution buffers. This paper focuses on time-domain stateful models, which process every buffer without FIFO accumulation [31]. NAM [13], the most widely deployed neural amp modeler, is a stateful WaveNet. These models must preserve internal state across host-initiated buffer size changes. In `RTNeural`, this is managed by hand at the C++ level; `BNNSGraph` supports it natively through CoreML state tensors declared as `InOut` arguments.

In this paper, we propose a methodology for measuring the real-time performance of neural network models used inside audio plugins under realistic conditions. We claim that the common practice of reporting isolated real-time factor (RTF) is insufficient: a backend may report an RTF well below 1.0 when measured alone, yet fail when sharing CPU cores, cache, memory bandwidth, and callback budget with dozens of other plugins in a realistic mix session. To demonstrate this, we construct a configurable DAW environment using Tracktion Engine [32] with up to 36 tracks of conventional DSP running alongside the neural plugin, driven by a Core Audio callback through the BlackHole virtual audio driver. Using this methodology, we benchmark `BNNSGraph`, `RTNeural` (Eigen and XSIMD backends), `LibTorch`, `ONNX Runtime`, and `anira` (wrapping both `LibTorch` and `ONNX Runtime`) across three architectures—LSTM, TCN, and WaveNet—at three model sizes. We report the per-model and callback utilization distribution, and examine whether backend rankings established in isolation hold under realistic workload pressure. The remainder of this paper is organised as follows: Section 2 reviews related work and introduces `BNNSGraph`. Section 3 describes the experimental design. Section 4 presents isolated and contention results. Section 5 discusses the findings, and Section 6 concludes.

2. BACKGROUND

2.1. Related Work

The three model families evaluated in this work—LSTM, TCN, and WaveNet—represent the dominant architectures in deployed audio plugins. Wright et al. [5, 8] and Damskägg et al. [11] established these for guitar amplifier and distortion modeling. Comunità et al. recently compared LSTM, TCN, GCN, and S4D across multiple nonlinear effects [33], and Simionato and Fasciani compare LSTM, LRU, and S4D for virtual analog modeling [34]. Beyond these recurrent and convolutional families, state-space models based on S4 [35], Mamba [36], and the Linear Recurrent Unit [37] have been applied to audio effects [38, 39]. On the efficiency side, Südholt et al. achieve over 99% parameter removal with 3–4× inference speedup in LSTM guitar effects [40], and Clarke et al. apply structured pruning at inference time [41].

2.2. Inference Benchmarks

Several studies have compared inference engines for real-time audio, but all report performance measured in isolation. Chowdhury benchmarked `RTNeural` against `LibTorch` [6]. Ackva and Schulz compared `LibTorch`, `ONNX Runtime`, and TensorFlow Lite through `anira` [7, 18]. Hoopes et al. benchmarked `RTNeural` against `ONNX Runtime` on Android phones [42]. Stefani et al. compared four engines on embedded boards [43]. Esqueda and Murai [38] implemented LRU models in C++ with Eigen, explicitly rejecting `ONNX Runtime` due to real-time safety concerns; they report per-buffer processing times on an M2 Pro processor measured inside Ableton Live, but with only a single plugin active. Mitcheltree et al. describe the Neutone SDK, which wraps TorchScript models with automatic buffering, sample rate conversion, and delay compensation [21].

In all the aforementioned studies, a single model runs alone with no other processing competing for CPU time, cache, or memory bandwidth. In practice, however, an audio plugin rarely runs in isolation—it shares CPU cores, cache, and memory bandwidth with dozens of other plugins within the same synchronous callback. Inference under resource contention has been studied in adjacent domains: Gujarati et al. [44] design for predictable DNN inference latency by eliminating system-level sources of variance, and Zhou et al. [45] provide worst-case response time bounds for multi-DNN workloads. Even MLPerf [46], the industry-standard ML benchmark, does not include contention as a dimension. To our knowledge, no prior work has measured neural audio inference under DAW contention with real-time deadline pressure.

This gap is especially notable for `BNNSGraph`, which has received no independent academic evaluation since Apple introduced it at WWDC 2024 as part of the Accelerate framework, updated at WWDC 2025 [47, 48]. Apple has not submitted `BNNSGraph` results to MLPerf or any other public benchmark suite. The API accepts a compiled CoreML model (`.mlmodelc`) and supports dynamic input shapes, allowing a single compiled graph to serve any buffer size without recompilation. This work is the first independent academic evaluation of `BNNSGraph`.

3. EXPERIMENTAL DESIGN

We target macOS on Apple Silicon because it is the dominant platform for professional music production [22] and the only platform where `BNNSGraph` is available. All experiments were conducted on a MacBook Air with an Apple M3 (8-core CPU, 24 GB unified memory) running macOS Tahoe 26.4. Code was compiled with Apple Clang 21.0.0 in Release mode. `LibTorch` 2.4.1 and `ONNX Runtime` 1.19.2 were used. The benchmark suite is publicly available.[1]

3.1. Models

We evaluate three stateful neural network architectures at three size tiers—Small, Medium, and Large—summarized in Table 1. All models process mono audio at 48 kHz and maintain internal state across buffer boundaries.

Table 1: Model architectures. H: LSTM hidden size. C: conv channels. L: layers.

Arch.	Tier	Configuration	Params
LSTM	S	$H=20$	1,861
	M	$H=40$	6,921
	L	$H=96$	38,113
TCN	S	$C=16$, $L=4$, dil. 1–8	4,337
	M	$C=32$, $L=8$, dil. 1–128	33,633
	L	$C=48$, $L=10$, dil. 1–512	93,745
WaveNet	S	$C=8$, $L=3$, dil. 1–4	841
	M	$C=16$, $L=10$, dil. 1–512	10,609
	L	$C=32$, $L=10$, dil. 1–512	41,697

The **LSTM** uses a single recurrent layer followed by a dense output layer ($H \to 1$). The **TCN** uses dilated causal convolutions with PReLU activation and residual connections. The **WaveNet** follows the RTNeural-NAM architecture [13, 49] with tanh-only activation (not gated tanh $\times$ sigmoid).

All models use random weight initialization. Since none of the architectures contain data-dependent branching, conditional sparsity, or early-exit paths, inference time is determined by topology and parameter count rather than weight values.

3.2. Backends

Table 2 summarizes the seven backends. Each backend processes the full audio buffer with a single framework call per callback, matching the buffer-at-a-time convention used by audio plugin hosts.

Table 2: Inference backends and their properties.

Backend	RT	AMX	Lat.	Cont.
`BNNSGraph`	✓	native	0	✓
`RTNeural-Eigen`	✓	—	0	—
`RTNeural-XSIMD`	✓	—	0	✓
`LibTorch`	—	BLAS	0	—
`ONNX Runtime`	—	—	0	—
`anira-LibTorch`	✓*	BLAS	+1 buffer	✓
`anira-ONNX RT`	✓*	—	+1 buffer	✓

RT = real-time safe. Lat. = added latency. Cont. = included in contention benchmark. *Audio thread is RT-safe; inference runs on background thread.

`BNNSGraph` processes all architectures buffer-at-a-time via a single `BNNSGraphContextExecute` call with dynamic shapes. `RTNeural` processes LSTM per-sample and TCN/WaveNet buffer-at-a-time, following the RTNeural-NAM pattern. `LibTorch` and `ONNX Runtime` are included in the isolated benchmark but excluded from contention tests because they are not designed for real-time use. `ONNX Runtime` is stateful for LSTM via explicit state I/O, but TCN and WaveNet are effectively stateless in our implementation.

`anira` decouples inference onto background worker threads, keeping the audio thread allocation-free at the cost of additional buffering latency [7, 18]. The added latency is computed automatically by `anira` from the host buffer configuration and model input/output geometry. When fresh output is not available, `anira` returns zeroed output; we count these events as inference underruns.

3.3. Real-Time Safety Verification of BNNSGraph

`BNNSGraph` is closed-source, so Apple's claim that the compiled graph performs no heap allocation, acquires no locks, and makes no blocking system calls [27] cannot be verified by source inspection. To independently check this claim, we compiled a minimal `BNNSGraph` inference loop with RealtimeSanitizer (RTSan) [28], which flags calls to non-deterministic functions (`malloc`, `pthread_mutex_lock`, etc.) in code annotated as real-time. Using `-fsanitize=realtime` (LLVM 22) and the `[[clang::nonblocking]]` attribute, RTSan reported zero violations over 100 consecutive invocations. This is consistent with Apple's claim, though it does not constitute a formal proof for all configurations.

3.4. Isolated Benchmark

The isolated benchmark measures raw inference speed without audio device overhead. Five backends are tested—`BNNSGraph`, `RTNeural-Eigen`, `RTNeural-XSIMD`, `LibTorch`, and `ONNX Runtime`—across 3 models, 3 sizes, and 6 buffer sizes (32–1024 samples), yielding 270 unique callback configurations with 3 repetitions each (810 measurements), plus 45 throughput configurations (one per backend/model/size). For each configuration, we pre-generate a deterministic random input stream with a fixed seed. After 100 warmup iterations, we adaptively select the iteration count to achieve at least 5 s of measurement time. Per-iteration wall-clock durations are recorded via

[1] `https://github.com/dharanipathirk/neural-audio-bench`

`mach_absolute_time`. We report the median, p95, p99, p99.9, min, max, and standard deviation of per-iteration wall-clock duration. For comparability with prior work, we also report the conventional real-time factor (RTF = median duration / deadline), though the contention results in Section 4 will show that this single summary statistic can be misleading.

3.5. Contention Benchmark

The contention benchmark requires a DAW-grade audio engine that exposes Core Audio callbacks under controlled conditions. We use Tracktion Engine [32], which provides per-track plugin hosting, group buses, and the same Core Audio real-time thread scheduling as commercial DAWs. We route audio output through the BlackHole virtual audio driver, which delivers IOProc callbacks on a real-time thread with deadline pressure, without requiring physical hardware. In our configuration, Tracktion Engine processes tracks in parallel across its default thread pool. Four RT-safe backends participate: `BNNSGraph`, `RTNeural-XSIMD`, anira-LibTorch, and anira-ONNX RT.

Session layout. Each dimension uses a different session configuration. Dimension A models a realistic rock/pop mix across 36 tracks using real macOS system Audio Units: 10 drum, 2 bass, 4 guitar (one guitar track hosts the neural model), 2 keys, 6 vocals, 7 group buses, and 5 FX returns. Each track carries a chain of AUParametricEQ, AUDynamicsProcessor, and, where appropriate, AUMatrixReverb instances. Dimension B creates only neural plugin tracks with no conventional DSP. Dimension C uses a single track with lightweight custom DSP plugins (biquad EQ, compressor, reverb, delay) surrounding the neural plugin.

Dimension A—Mix contention. One neural plugin with conventional plugin chains active on a variable number of the 36 tracks: $\{0, 8, 24, 36\}$. Four buffer sizes: $\{64, 128, 256, 512\}$ samples. This measures how a single neural plugin degrades as the surrounding mix grows.

Dimension B—Instance count. Multiple neural plugin instances on separate tracks $\{1, 2, 4, 8, 16\}$, no conventional DSP, buffer size fixed at 128. The reported neural-plugin timing is taken from one representative instance (the first created in the session), not averaged across all instances. Aggregate session load is reflected by hardware xruns and, for `anira`, inference underruns summed across all instances. This reveals the per-backend scaling limit.

Dimension C—Serial depth. A single track with one neural model instance surrounded by lightweight conventional DSP (EQ, compressor, delay, reverb), varying the total insert-chain depth $D \in \{1, 3, 5, 7\}$ at buffer size 128. Unlike Dimension B, which scales neural instances in parallel, Dimension C tests whether serial plugins on the same track affect neural plugin timing.

Timing instrumentation. A `CallbackStartPlugin` on every active track records the earliest `mach_absolute_time` per callback via atomic compare-and-swap. A `CallbackEndPlugin` on the master bus captures the callback end time. Per-callback utilization is (duration / deadline) $\times$ 100%. We report percentiles of this distribution—p50 (median), p95, p99, p99.9, and max—where pN denotes the value at or below which N% of callbacks fall. When a callback exceeds its deadline, the audio driver reports a hardware xrun (buffer underrun), resulting in an audible glitch. Xruns are counted via Core Audio's `getXRunCount` API. For `anira` backends, we additionally count inference underruns—callbacks where the background thread had not finished in time.

Protocol. Each configuration runs in four stages that separate steady-state measurement from startup and thermal transients. *(i) Warmup:* set the device buffer size and 48 kHz sample rate, build the session with plugins and 30 s noise clips, and play for $\geq$3 s to reach steady state. *(ii) Reset boundary:* stop, reset the timing loggers, restart, settle 200 ms, and only then snapshot the xrun baseline, so the restart transient corrupts neither the timing distribution nor the xrun count. *(iii) Measure:* record for 15 s—long enough to populate the tail percentiles, yet short enough to limit thermal drift within a configuration. *(iv) Trim and reduce:* discard the initial callbacks (at least 300 ms worth, up to 10% of the run, capped at half) to exclude residual settle-period outliers, then compute statistics and write the CSV. At the outer level, a 45 s all-core CPU warmup precedes each benchmark phase to reach thermal steady state, so the first configuration does not benefit from a cold CPU, and a 120 s cooldown between phases restores a consistent thermal baseline.

4. RESULTS

4.1. Isolated Throughput

Table 3 reports throughput in multiples of real time ($\times$ RT) for each backend and model, computed from the per-sample cost over 10 seconds of audio. Higher is better.

`BNNSGraph` dominates TCN and WaveNet at all sizes, and the advantage grows with both model complexity and buffer size. For TCN-Large, the speedup over `RTNeural-XSIMD` increases from 3$\times$ at buffer 32 to 25$\times$ at buffer 1024; for WaveNet-Large, from 1.3$\times$ to 12$\times$. Across model sizes, `BNNSGraph` is 5$\times$ faster than the best `RTNeural` variant for TCN-Small, 19$\times$ at medium, and 25$\times$ at large. This scaling is consistent with `BNNSGraph` amortizing per-call dispatch overhead and exploiting AMX matrix instructions across the full buffer, whereas `RTNeural` relies on NEON SIMD without access to AMX.

For LSTM, `RTNeural-Eigen` outperforms `BNNSGraph` at small and medium sizes (154$\times$ vs 149$\times$ at small; 64$\times$ vs 21$\times$ at medium). This is expected because LSTM inference is inherently sequential—each timestep depends on the previous hidden state—so buffer-level graph compilation cannot parallelize across samples. At small and medium hidden sizes (H=20, 40), RTNeural's template-inlined per-sample loop avoids graph dispatch overhead entirely, and the matrix operations are too small for AMX to provide a meaningful advantage. At large (H=96), the per-sample matrix multiplications are wide enough for AMX to overcome this overhead, and `BNNSGraph` takes the lead (19$\times$ vs 13$\times$).

`LibTorch` is the slowest backend for LSTM at all sizes (about 3–4$\times$ RT), but performs better on TCN at medium and large sizes (24$\times$ and 12$\times$, respectively), surpassing both tested `RTNeural` variants in our isolated runs. For WaveNet, `LibTorch` trails `RTNeural` at small and medium sizes but roughly matches it at large size (about 12$\times$). This architecture-dependent pattern suggests that the bottleneck is specific to `LibTorch`'s CPU LSTM execution path rather than a generic framework-dispatch overhead. `ONNX Runtime` performs well for small models (108$\times$ for LSTM-Small and 398$\times$ for WaveNet-Small), but its TCN and WaveNet results must be interpreted with the caveat that ONNX TCN/WaveNet are stateless in our implementation (Section 3.2).

Table 4 reports RTF at buffer size 128 (deadline 2.67 ms) for

Table 3: Isolated throughput (× RT, ↑). Best in **bold**. † stateless.

Backend	LSTM S	LSTM M	LSTM L	TCN S	TCN M	TCN L	WaveNet S	WaveNet M	WaveNet L
BNNSGraph	149	21	**19**	**531**	**284**	**117**	**456**	**129**	**132**
RTNeural-Eigen	**154**	**64**	13	73	12	5	222	31	11
RTNeural-XSIMD	127	50	11	106	15	5	426	40	12
LibTorch	4	4	3	100	24	12	140	22	12
ONNX Runtime	108	47	10	135†	23†	9†	398†	48†	17†

Table 4: RTF at buf. 128, 48 kHz (↓). ≥ 0.10 in orange.

Backend	LSTM-L	TCN-L	WN-L
BNNSGraph	0.053	0.022	0.021
RTN-Eigen	0.078	0.222	0.092
RTN-XSIMD	0.089	0.211	0.085
LibTorch	0.346	0.230	0.209
ONNX Runtime	0.100	0.115	0.063

large models. All backends achieve RTF < 1.0, with BNNSGraph using at most 5.3% of the deadline (LSTM-Large, RTF 0.053). However, these isolated measurements do not reflect the cache, memory bandwidth, and scheduling pressure present in a real mix session. The following contention results quantify this gap.

4.2. Contention: Dimension A (Mix Contention)

The contention benchmark uses RTNeural-XSIMD as the RTNeural variant, as it achieved higher overall isolated throughput than the Eigen variant (Table 3). Table 5 reports p99 neural-plugin audio-thread utilization and hardware xrun counts for large models at buffer size 128, comparing zero contention ($c = 0$) against maximum contention ($c = 36$). We focus on large models because they place the most pressure on the callback budget, and buffer size 128 (2.67 ms deadline at 48 kHz), a typical low-latency setting for live monitoring. Results at buffer size 64 show the same trends with larger magnitudes (e.g. RTNeural-XSIMD TCN-Large reaches 164.7% p99 with 888 xruns at $c = 0$).

Table 5: Dimension A: neural-plugin utilization (%) and hardware xruns for large models, buffer 128. p99 values exceeding 50% in orange, exceeding 100% in red. †Audio-thread cost only; inference offloaded to background thread.

Backend	Mdl	$c=0$ p50	$c=0$ p99	$c=0$ xr	$c=36$ p50	$c=36$ p99	$c=36$ xr
BNNSGraph	LSTM	3.5	15.6	0	13.9	17.1	0
BNNSGraph	TCN	3.2	13.7	0	11.3	14.3	0
BNNSGraph	WN	6.2	10.9	0	8.4	11.4	0
RTNeural	LSTM	8.5	79.9	0	8.5	81.7	0
RTNeural	TCN	21.5	158.5	355	75.5	88.0	15
RTNeural	WN	8.0	65.5	0	8.0	66.3	0
anira-LT†	all	*0.1*	*≤0.5*	0	*0.1*	*≤0.4*	0
anira-OX†	all	*0.1*	*≤0.9*	0	*0.1*	*≤0.7*	0

The central result is the gap between isolated and contention performance. RTNeural-XSIMD TCN-Large has an isolated RTF of 0.211 (Table 4), suggesting 79% of the deadline budget remains unused. Under a Core Audio callback even with *zero contention tracks*, however, p99 utilization reaches 158.5% with 355 hardware xruns—a model that appears to run at 5× real time in isolation produces hundreds of audible glitches under the actual audio driver. Figure 1 visualizes this gap for all large models.

BNNSGraph TCN-Large, by contrast, maintains p99 at 14.3% even with 36 contention tracks, leaving 86% of the deadline budget available. Across all architectures and contention levels, BNNSGraph never exceeds 20% p99 utilization for large models at buffer 128, with zero hardware xruns.

BNNSGraph's p99 stays essentially flat across all contention levels for all three architectures. For RTNeural TCN-Large, p99 decreases from 158.5% at $c = 0$ to 88.0% at $c = 36$, while the median rises from 21.5% to 75.5% (Table 5). This suggests that the tail distribution narrows under higher contention, though the cause is not fully understood and needs further investigation. RTNeural LSTM-Large and WaveNet-Large remain between 65–82% p99 across all contention levels.

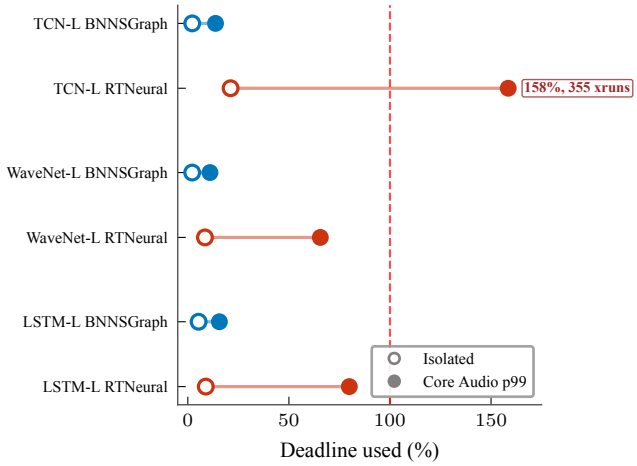

Figure 1: Isolated utilization vs. Core Audio p99 neural-plugin utilization at buffer 128, zero contention. Open circles: isolated RTF × 100. Filled circles: p99 under Core Audio. RTNeural TCN-Large shifts from 21% to 158.5%, producing 355 xruns.

The anira backends report near-zero audio-thread utilization (≤0.9% p99), as expected: the audio thread performs only lock-free ring buffer operations while inference runs on a background thread. This metric, however, does not reflect whether the background thread keeps pace with the callback rate; Section 4.3.1 examines this.

4.3. Contention: Dimension B (Instance Scaling)

Table 6 reports the p99 timing of one representative neural plugin instance within the multi-instance session, together with session-level failure metrics (hardware xruns and, for anira, total inference underruns across all instances).

As N increases, RTNeural TCN-Large produces hardware xruns starting at $N = 4$ (3 xruns), rising to 312 at $N = 16$, despite an isolated RTF of 0.211 per instance. BNNSGraph scales to $N = 16$ with p99 at 27.1% and only 1 xrun (Table 6). anira-LibTorch maintains zero audio-thread xruns at all instance counts, but at $N = 16$ its background thread accumulates 14,022 inference underruns for TCN-Large. For LSTM-Large at $N = 16$, the underrun count rises to 44,324 (Section 4.3.1).

Table 6: Dimension B: per-instance neural-plugin utilization (%) and xruns for TCN-Large, buffer 128. N = number of neural plugin instances. p99 is from one representative instance, not averaged across instances. inf. ur. = total `anira` inference underruns summed across all instances.

	`BNNSGraph`		`RTNeural`		`anira-LibTorch`	
N	p99	xrun	p99	xrun	p99	inf. ur.
1	4.0	0	27.1	0	0.1	0
2	11.7	0	75.4	0	0.1	0
4	19.7	0	85.3	3	0.1	0
8	24.4	0	86.5	99	0.1	0
16	27.1	1	78.3	312	2.1	14,022

4.3.1. The Anira Tradeoff

`anira` moves inference off the audio thread, reducing the neural plugin's audio-thread cost to near zero, but shifts the failure mode: when the background inference cannot keep pace, the current implementation returns zeroed output for that callback and records an inference underrun. For a single large-model instance in Dimension A, `anira-LibTorch` reports zero inference underruns.

At $N = 16$ (Dimension B), however, `anira-LibTorch` accumulates 44,324 LSTM-Large underruns and 14,022 TCN-Large underruns (aggregated across all 16 instances over approximately 5,193 callbacks). This is consistent with saturation of `anira`'s shared inference capacity: same-configuration sessions share a singleton `anira` context, a global worker pool, and a shared `LibTorch` processor. On our test machine, this corresponds to four parallel workers and model instances by default. At buffer size 128, 16 instances submit approximately 6,000 inferences per second, leaving under 0.67 ms per worker inference—close to the isolated `LibTorch` TCN-Large cost of 0.61 ms.

By contrast, `anira-ONNX RT` reports zero inference underruns at $N = 16$ for all three architectures, consistent with its lower isolated per-inference cost (0.31 ms for TCN-Large).

4.4. Contention: Dimension C (Serial Depth)

Table 7 reports neural-plugin and full-callback p99 utilization for TCN-Large as the serial insert-chain depth D increases.

Table 7: Dimension C: TCN-Large at buffer 128. Neural-plugin p99 and full-callback p99 (%) as serial depth increases.

	Neural p99		Callback p99	
Backend	$D=1$	$D=7$	$D=1$	$D=7$
`BNNSGraph`	4.0	3.3	4.1	4.4
`RTNeural`	27.1	26.5	27.2	27.7
`anira-LibTorch`†	0.06	0.06	0.11	0.59
`anira-ONNX RT`†	0.08	0.08	0.11	0.59

†Audio-thread cost only; inference on background thread.

Dimension C has only a small effect on the neural plugin cost itself. Across all large-model cases, neural-plugin p99 changes by less than 1 percentage point between $D = 1$ and $D = 7$ (Table 7). The full-callback span increases modestly with depth, as the additional serial plugins contribute extra work on the same track: for TCN-Large, full-callback p99 rises from 4.1% to 4.4% for `BNNSGraph` and from 27.2% to 27.7% for `RTNeural`. No inference underruns occur in Dimension C, and only isolated single xruns appear for `RTNeural` TCN-Large at depths 5 and 7.

5. DISCUSSION

The central result is that isolated performance is not a reliable proxy for real-time robustness under DAW contention. A backend may report a comfortable isolated RTF while still producing large tail latencies and hardware xruns once it shares CPU cores, caches, memory bandwidth, and scheduling budget with a realistic plugin graph.

Dimension A demonstrates this most clearly. Under a 36-track mix session with real system Audio Units, `BNNSGraph` remains below 20% p99 for all large models at buffer 128, while `RTNeural` TCN-Large reaches 158.5% p99 even at zero contention.

For convolutional models, `BNNSGraph` is markedly more resilient under contention. This is consistent with ahead-of-time graph compilation, operation fusion, and chip-specific optimization for Apple Silicon. We verified AMX instruction presence in `BNNSGraph` and `LibTorch` hot paths by sampling instruction streams with `lldb` and decoding the patterns documented by Johnson [23]; `RTNeural` and `ONNX Runtime` execute NEON-only kernels. Since Apple exposes no public AMX performance counters, we cannot isolate AMX's quantitative contribution, but the results are consistent with AMX-backed kernels completing the same work in fewer cycles, leaving more margin for scheduler jitter and cache interference. The scaling results reinforce this: `BNNSGraph`'s advantage grows with buffer size, and under Dimension B it scales sub-linearly with instance count (p99 rises from 4.0% at $N = 1$ to 27.1% at $N = 16$). For LSTM, small and medium models favour `RTNeural` in isolation, but the large-model contention results show `RTNeural-XSIMD` LSTM-Large at 80–82% p99 across contention levels, leaving limited practical margin. For convolutional models, `RTNeural` shows the largest contention gap. The `anira` results reveal a different tradeoff. By moving inference off the audio thread, `anira` reduces audio-thread cost to near zero but replaces deadline pressure with inference underruns and added latency. The underrun behaviour at $N = 16$ is consistent with saturation of `anira`'s default shared worker pool, and the contrast between `anira-LibTorch` (44,324 LSTM-Large underruns) and `anira-ONNX RT` (zero underruns) shows that the backend's per-inference cost, not `anira`'s scheduling, determines this failure mode. Asynchronous wrappers must therefore be evaluated with underrun metrics in addition to xruns.

Dimension C shows that serial insert depth on a single track has minimal effect on neural plugin timing. Compared with Dimension B, where parallel instance scaling produces large p99 increases and hardware xruns, the dominant contention mechanism in this benchmark is scaling the number of neural models in parallel across tracks, not placing lightweight plugins around a neural model on one track. Stacking multiple neural models *in series* on one track (a chain of neural effect and amplifier models) would impose serial inference dependencies rather than the parallel instance load of Dimension B, and we expect it to discriminate backends more sharply than the lightweight serial DSP of Dimension C; we leave this to future work.

5.1. Limitations

1. **Mono session.** The benchmark uses mono audio. Stereo processing would increase CPU pressure.
2. **Flat topology.** Group buses and returns are modelled as independent load rather than a fully routed hierarchical mix graph.
3. **Virtual audio driver.** BlackHole provides Core Audio callback scheduling but does not include DMA or bus behaviour of a physical interface.
4. **Single machine.** All results are from one Apple M3 chip. Different M-series generations have different cache hierarchies and matrix-unit implementations.
5. **Thermal state.** We used warmup and cooldown phases to reduce thermal drift but did not directly log chip temperature.
6. **Default `anira` configuration.** Results reflect `anira`'s default shared worker-pool behaviour, not a per-model tuned configuration.
7. **ONNX statefulness.** `ONNX Runtime` TCN and WaveNet are effectively stateless across calls in our implementation.
8. **Memory not measured.** We report timing only; peak memory footprint (higher for convolutional models than for stateful recurrent models, and interacting with buffer size) is a complementary axis left to future work.

6. CONCLUSION

This paper addressed a practical question: do the neural network model and the backend that look strongest in an isolated benchmark still behave best inside a realistic DAW session? The answer is often no. A model–backend combination can appear comfortably real-time-capable in isolation and still produce large tail latencies and audible xruns under a real audio callback with realistic plugin contention. The main contribution is therefore methodological as well as empirical: neural audio models and inference backends should be evaluated under contention, not only in isolated loops.

`BNNSGraph` proved the most robust backend for convolutional models on Apple Silicon. It maintained low p99 neural-plugin utilization under Dimension A contention, scaled substantially better under Dimension B instance growth, and showed an isolated throughput advantage that increased with buffer size. For audio plugin developers, the practical message is straightforward. If Apple Silicon is the target platform, `BNNSGraph` is the strongest option for real-time neural audio models. More importantly, backend and model selection should not be based on isolated real-time factor alone. Tail latency under realistic contention is what determines whether a model remains usable in a live session.

The benchmark framework introduced here is reusable beyond Apple Silicon and beyond the specific backends studied. The benchmark suite and analysis scripts are publicly available. Future work should apply the same methodology to more realistic stereo and routed mix graphs, while also improving the framework so that plugin developers can easily and reliably evaluate neural model architectures and backends.

7. REFERENCES

[1] A. P. Bell, *Dawn of the DAW: The Studio as Musical Instrument*. Oxford University Press, 2018.

[2] Apple Inc. Core Audio overview. Apple Developer Documentation. Accessed: Mar. 2026. [Online]. Available: https://developer.apple.com/documentation/coreaudio.

[3] Microsoft, "Windows Audio Session API (WASAPI)," accessed: 2026. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/coreaudio/wasapi

[4] R. Bencina, "Interfacing real-time audio and file I/O," in *Proc. of the Australasian Computer Music Conference (ACMC)*, 2014, pp. 21–28.

[5] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time guitar amplifier emulation with deep learning," *Applied Sciences*, vol. 10, no. 3, p. 766, 2020.

[6] J. Chowdhury, "RTNeural: Fast neural inferencing for real-time systems," *arXiv preprint arXiv:2106.03037*, 2021.

[7] V. Ackva and F. Schulz, "ANIRA: An architecture for neural network inference in real-time audio applications," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2024, pp. 1–10.

[8] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, Sep. 2019, pp. 173–180.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," in *arXiv preprint arXiv:1803.01271*, 2018.

[11] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *Proc. IEEE Intl. Conf. Acoustics, Speech, Signal Process. (ICASSP)*, 2019.

[12] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu *et al.*, "WaveNet: A generative model for raw audio," in *arXiv preprint arXiv:1609.03499*, 2016.

[13] S. Atkinson. Neural Amp Modeler (NAM). Accessed: Mar. 2026. [Online]. Available: https://github.com/sdatkinson/neural-amp-modeler.

[14] K. Bloemer. Guitarml. Accessed: Mar. 2026. [Online]. Available: https://github.com/GuitarML/GuitarML.

[15] C. J. Steinmetz *et al.*, "Audio signal processing in the artificial intelligence era: Challenges and directions," *J. Audio Eng. Soc.*, vol. 73, no. 7/8, pp. 406–428, Jul. 2025.

[16] T. Skare, "GPGPU audio benchmark framework," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-24)*, Guildford, UK, 2024.

[17] GPU Audio Inc. GPU Audio: GPU-powered audio processing. Accessed: Mar. 2026. [Online]. Available: https://www.gpu.audio.

[18] V. Ackva and F. Schulz, "anira: An architecture for neural network inference in real-time audio applications," *arXiv preprint arXiv:2506.12665*, 2025.

[19] A. Paszke, S. Gross, F. Massa *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[20] Microsoft. ONNX Runtime. Accessed: Mar. 2026. [Online]. Available: https://onnxruntime.ai/.

[21] C. Mitcheltree, B. Teleaga, A. Fyfe, N. Masuda, M. Schäfer, A. Bradic, and N. Tokui, "Neutone SDK: An open source framework for neural audio processing," *arXiv preprint arXiv:2508.09126*, 2025.

[22] F. Avanzini, R. Della Longa, D. Fantini, L. A. Ludovico, G. Presti *et al.*, "A study on the adoption and usage of digital audio workstations," in *Proc. Sound and Music Computing Conf. (SMC)*, 2025, pp. 136–143.

[23] D. Johnson. Apple AMX instruction decoder. Accessed: Mar. 2026. [Online]. Available: https://gist.github.com/dougallj/7a75a3be1ec69ca550e7c36dc75e0d6f.

[24] Arm Ltd. (2023) Scalable matrix extension (SME). Arm Architecture Reference Manual Supplement. Accessed: Mar. 2026. [Online]. Available: https://developer.arm.com/documentation/ddi0616/latest.

[25] Apple Inc., "Core ML," 2024, apple Developer Documentation. Accessed: 2026. [Online]. Available: https://developer.apple.com/documentation/coreml

[26] ——, "BNNSGraph," 2024, apple Developer Documentation. Accessed: 2026. [Online]. Available: https://developer.apple.com/documentation/accelerate/bnnsgraph

[27] ——. (2024) Support real-time ML inference on Apple Silicon. WWDC 2024, Session 10211. Accessed: Mar. 2026. [Online]. Available: https://developer.apple.com/videos/play/wwdc2024/10211/.

[28] D. Trevelyan, A. Barker, and C. Apple, "Realtime Sanitizer," GitHub Repository, accessed: Apr. 2026. [Online]. Available: https://github.com/realtime-sanitizer/rtsan

[29] Apple Inc., "BLAS," apple Developer Documentation. Accessed: 2026. [Online]. Available: https://developer.apple.com/documentation/accelerate/blas

[30] Microsoft, "MLAS (Microsoft Linear Algebra Subprograms)," processor-optimized GEMM kernels and platform-specific threading code; accessed: 2024. [Online]. Available: https://github.com/microsoft/onnxruntime/tree/main/onnxruntime/core/mlas

[31] F. Caspe, J. Shier, M. Sandler, C. Saitis, and A. McPherson, "Designing neural synthesizers for low-latency interaction," *J. Audio Eng. Soc.*, 2025.

[32] Tracktion Corporation. Tracktion Engine. Accessed: Mar. 2026. [Online]. Available: https://github.com/Tracktion/tracktion_engine.

[33] M. Comunità, C. J. Steinmetz, and J. D. Reiss, "Differentiable black-box and gray-box modeling of nonlinear audio effects," *Frontiers in Signal Processing*, 2025.

[34] R. Simionato and S. Fasciani, "Comparative study of state-based neural networks for virtual analog audio effects modeling," *EURASIP J. Audio, Speech, Music Process.*, 2025.

[35] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[36] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[37] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De, "Resurrecting recurrent neural networks for long sequences," in *International Conference on Machine Learning*. PMLR, 2023, pp. 26 670–26 698.

[38] F. Esqueda and S. Murai, "Antialiased black-box modeling of audio distortion circuits using real linear recurrent units," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-25)*, Ancona, Italy, 2025.

[39] D. Dallinger, M. Bittner, D. Schnöll, M. Wess, and A. Jantsch, "Piano-SSM: Diagonal state space models for efficient MIDI-to-raw audio synthesis," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-25)*, 2025.

[40] D. Südholt, A. Wright, C. Erkut, and V. Välimäki, "Pruning deep neural network models of guitar distortion effects," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 31, pp. 256–264, 2023.

[41] C. J. Clarke *et al.*, "Inference-time structured pruning for real-time neural network audio effects," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-25)*, 2025.

[42] J. Hoopes, B. Chalmers, and V. Zappi, "Neural audio processing on Android phones," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-24)*, Guildford, UK, 2024.

[43] D. Stefani, S. Peroni, and L. Turchet, "A comparison of deep learning inference engines for embedded real-time audio classification," in *Proc. Intl. Conf. Digital Audio Effects (DAFx-22)*, Vienna, Austria, 2022, pp. 256–263.

[44] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.

[45] H. Zhou, S. Bateni, and C. Liu, "S^3DNN: Supervised streaming and scheduling for GPU-accelerated real-time DNN workloads," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2018, pp. 190–201.

[46] V. J. Reddi *et al.*, "MLPerf inference benchmark," in *Proc. Intl. Symp. Computer Architecture (ISCA)*, 2020.

[47] Apple Inc. (2025) What is new in BNNSGraph. WWDC 2025, Session 276. Accessed: Mar. 2026. [Online]. Available: https://developer.apple.com/videos/play/wwdc2025/276/.

[48] ——. (2025) Discover machine learning & AI frameworks on Apple platforms. WWDC 2025, Session 360. Accessed: Mar. 2026. [Online]. Available: https://developer.apple.com/videos/play/wwdc2025/360/.

[49] J. Chowdhury, "RTNeural-NAM," 2024, gitHub repository. Accessed: 2026. [Online]. Available: https://github.com/jatinchowdhury18/RTNeural-NAM