BENCHMARKING INTEGRATED GPU ACCELERATION OF REAL-TIME NEURAL AUDIO INFERENCE ON SNAPDRAGON

Avery Huang, Gautham Srinivasan
Northeastern University
Boston, USA
huang.av@northeastern.edu
srinivasan.ga@northeastern.edu

Akito van Troyer
Berklee College of Music
Boston, USA
avantroyer@berklee.edu

Victor Zappi
Northeastern University
Boston, USA
v.zappi@northeastern.edu

ABSTRACT

Neural audio models for real-time synthesis and processing are typically deployed on CPU. Whether the integrated GPU found on modern system-on-chip platforms can accelerate these models under streaming audio constraints remains an open question. In this work, we benchmark five models (from a 769-parameter oscillator to a 6-million-parameter convolutional autoencoder) deployed in a custom C++ audio engine on a Qualcomm Snapdragon SoC, comparing optimized CPU inference against GPU inference. We use Qualcomm's QNN SDK, sweeping over batch and block sizes. Our results reveal when integrated GPU acceleration offers meaningful gains over CPU, when per-call overhead negates the benefits, and how model size and architecture determine GPU suitability. Since the Snapdragon family shares both hardware architecture and software toolchain across embedded boards, smartphones and laptops, the performance trends and deployment workflow presented here are applicable beyond the specific device tested.

1. INTRODUCTION

Deep learning has become a central tool in audio processing and synthesis research. Neural networks now achieve state-of-the-art results in amplifier and effects emulation [1, 2, 3, 4], audio codec design [5, 6], speech synthesis [7, 8] and musical sound generation [9, 10, 11]. Within the DAFx community specifically, the last several years have seen a proliferation of causal, streamable neural architectures, i.e., models designed to operate sample-by-sample or on small blocks of audio within the strict timing constraints of a real-time audio callback. These span recurrent networks for nonlinear circuit emulation [1], temporal convolutional networks for audio effects modeling [12], differentiable digital signal processing (DSP) hybrids for synthesis [13], and streamable convolutional autoencoders for timbre transfer [14].

As these models grow in complexity—from fewer than a thousand parameters for a simple neural oscillator, to several million for a convolutional autoencoder—the computational cost of inference becomes a primary concern for deployment on resource-constrained hardware. Significant effort has gone into making CPU-based inference efficient for these models. Dedicated inference engines, like RTNeural [15] and ONNX Runtime[1], provide optimized C++ implementations and minimize dynamic memory allocation, enabling sample-level processing within audio callbacks. Collectively, these tools have made CPU deployment viable for many neural audio models, but they leave open the question of whether the GPU—the workhorse of neural network acceleration in virtually every other domain—can offer additional gains for real-time audio inference.

GPU acceleration for neural audio is, in principle, well established. Vocoders based on generative adversarial networks routinely achieve throughput factors exceeding 100× real-time on desktop GPUs [7, 16] and GPU-based training is standard practice for all neural audio models. However, there is a fundamental distinction between *throughput-oriented* and *latency-constrained* inference that is often obscured by inconsistent use of the term "real-time" across communities. In the machine learning literature, inference is typically called "real-time" if processing a buffer of N seconds takes fewer than N seconds—a *real-time factor* (RTF = compute time / audio duration) below 1.0. This metric is meaningful for offline or batch rendering, but it doesn't say much about whether a model can actually produce output within the deadline imposed by an audio callback at a given sample rate and buffer size. For interactive audio—whether in a digital audio workstation, a live performance system, or an embedded instrument—the relevant constraint is not throughput over a long buffer, but whether each individual block of samples (often 64–1024 samples at 48 kHz, i.e., 1.3–21.3 ms) can be processed before the next block arrives. Practically speaking, a model that achieves an impressive RTF on a 10-second batch may still fail when required to meet a 5 ms deadline on every single callback, due to per-call overhead, memory transfer costs, or GPU kernel launch latency.

Furthermore, the vast majority of GPU inference results reported in the literature are obtained on discrete NVIDIA GPUs [7, 16, 17, 18]—high-end hardware with dedicated VRAM and PCIe bandwidth that is absent from the devices where audio applications are most commonly deployed. Musicians and sound designers should not be expected to purchase expensive discrete GPUs to run neural audio models, and instrument designers and audio product developers typically target laptops or single-board computers equipped with integrated GPUs [19, 20, 21].

In this paper, we investigate whether the integrated GPU available on a Qualcomm Snapdragon SoC (system on a chip) can accelerate streaming inference of neural audio models and under what conditions. We benchmark five models (four of which previously published) that carry out neural audio synthesis and processing. They span three orders of magnitude in parameter count (769 to approximately 6 million), cover the two fundamental streaming regimes of neural audio (sample-by-sample and block-based architectures) and employ a variety of layer types, including dense, recurrent, convolutional and encoder/decoder. All models are de-

[1] https://github.com/microsoft/onnxruntime

ployed in C++ using three different inference approaches, two on CPU and one on GPU. For each configuration, we measure the average and standard deviation of per-sample inference time[2] across repeated test runs, sweeping over batch and block sizes to characterize how GPU utilization scales with the degree of available parallelism.

We choose Qualcomm's Snapdragon platform for several reasons. First, the specific board we chose (RB3 Gen 2) is representative of a class of embedded single-board computers increasingly used in audio hardware prototyping and product development (e.g., Rubik Pi 3[3], Arduino VENTUNO Q[4]). Second, the broader Snapdragon family of SoCs powers the vast majority of Android smartphones [20] as well as a growing number of laptops, making our results potentially indicative of behavior on related hardware (e.g., Snapdragons with different CPU clock speed and GPU version). Third, Snapdragon SoCs support Qualcomm's AI Runtime (QAIRT) SDK[5], which provides unified interfaces for deploying models on CPU, GPU (and also NPU, which we leave for future work) backends from a single model representation; this SDK offers a level of direct GPU access that is not straightforwardly available in comparable mobile/edge ecosystems such as Apple's CoreML or Google's TensorFlow Lite.

2. RELATED WORK

GPU acceleration has been instrumental in advancing neural audio research, both for training and for high-throughput inference. GAN-based vocoders routinely achieve throughput factors of 25–167× real-time on discrete NVIDIA GPUs [7, 16, 18, 17] and GPU-based synthesis has been demonstrated for modular sound generation at over 16,000× real-time [22]. Deployment tools such as TensorRT [23] further optimize inference through layer fusion and reduced-precision quantization, but are restricted to NVIDIA hardware. These results share two characteristics that limit their applicability to interactive audio: they report throughput on large buffers using the real-time factor as the performance metric (see Section 1), and they rely on discrete GPUs absent from the devices where audio applications are typically deployed.

Skare [24] published GPU audio benchmarks at DAFx 2024, measuring kernel launch overhead and memory transfer costs on NVIDIA CUDA hardware. The findings quantify a fundamental challenge: the overhead of dispatching small workloads to the GPU can dominate the actual computation, making GPU acceleration counterproductive for the small buffer sizes typical of real-time audio. That study benchmarks GPU primitives rather than complete models; to our knowledge, only one work has directly addressed GPU-based *inference* for real-time audio. Publicly released in 2025, the GPU Audio SDK[6] runs over 100 simultaneous Neural Amp Modeler[7] (NAM) plugin instances on a single discrete NVIDIA GPU by batching processing requests into execution windows of 100–200 μs. While this demonstrates GPU audio in a production context, it requires a discrete GPU and achieves parallelism by aggregating many independent audio streams—a fundamentally different scenario from accelerating a single model on an integrated GPU.

A parallel line of work has focused on CPU deployment on resource-constrained hardware. Hoopes et al. [20] benchmarked neural audio effects on Android phones spanning 2014–2022, finding that RTNeural consistently outperformed ONNX Runtime due to lower per-call overhead, though GPU inference was not explored. Pelinski et al. [21] demonstrated deep learning deployment on the Bela embedded audio platform with strict real-time constraints.

In the speech domain, the LPCNet [8] family pioneered efficient on-device synthesis by combining linear prediction with compact recurrent networks, achieving real-time on a single phone CPU core. This line culminated in FARGAN [25], which reaches 600 MFLOPS while matching HiFi-GAN quality.

The choice of inference engine is critical, as framework overhead can exceed the cost of the neural computation itself. RTNeural [15] provides lightweight C++ inference with no dynamic memory allocation, targeting sample-by-sample processing of small recurrent and dense networks. NAMcore[8] offers hand-optimized Eigen-based inference for NAM's WaveNet architecture. ONNX Runtime is a general-purpose engine supporting multiple backends, including a QNN Execution Provider that can target Qualcomm's CPU, GPU and NPU—though we are not aware of any published deployment using this GPU capability for streaming audio or any other domain. ANIRA [26] addresses real-time safety by decoupling inference from the audio thread via static thread pools, motivated by the concern that engines like ONNX Runtime may violate real-time guarantees through dynamic memory allocation. The Neutone SDK [19] wraps PyTorch models for use in digital audio workstation plugins, handling buffer sizes and sample rate conversion on CPU.

Notably, the GPU Audio SDK reimplemented NAMcore's WaveNet for discrete GPU execution, demonstrating that GPU parallelization of these architectures is feasible—but only across multiple simultaneous instances on discrete hardware.

Beyond GPUs, FPGAs have been applied to audio-related workloads such as sparse LSTM acceleration [27] and WaveNet synthesis, but require hardware description language expertise and long development cycles. NPUs such as the Hexagon Tensor Processor on Qualcomm SoCs offer high-throughput inference for quantized models but require INT8/INT16 quantization, whose impact on audio quality has not been investigated. The integrated GPU, by contrast, supports standard floating-point arithmetic, is accessible through vendor SDKs and is present on virtually every Snapdragon device, making it a natural candidate for accelerating neural audio inference without sacrificing numerical fidelity.

3. MODELS

In this section we detail the five models used in our benchmarks. Table 1 summarizes their task, streaming regime, key architectural layers, sample rate and parameter count. To our knowledge, other architectures that are increasingly prominent in audio research (such as transformer-based and diffusion-based models) do not yet offer streaming implementations suitable for per-callback

[2] This is commonly referred to as 'latency' in machine learning, but we prefer the term 'inference time' to avoid confusion with the established meaning of latency in real-time audio.

[3] https://www.thundercomm.com/product/rubik-pi/

[4] https://www.arduino.cc/product-ventuno-q

[5] https://softwarecenter.qualcomm.com/catalog/item/Qualcomm_AI_Runtime_Community

[6] https://github.com/gpuaudio/gpuaudio-sdk

[7] https://github.com/sdatkinson/neural-amp-modeler

[8] https://github.com/sdatkinson/NeuralAmpModelerCore

execution at the buffer sizes typical of interactive audio. Therefore they are not included in our tests. Pre-trained versions of all models, along with Google Colab notebooks to load them in PyTorch, test inference and export them to ONNX are available online[9].

Neural Oscillator. The simplest model in our benchmark: a single-hidden-layer feedforward network with ReLU activation (hidden size 256, 769 parameters), trained to approximate $\sin(x)$ by mapping a phase input $x \in [0, 2\pi)$ to a scalar audio sample. It operates strictly sample-by-sample and contains only dense layers. It was designed from scratch as the simplest—yet fully functional—neural audio architecture within the benchmark.

AutoGuitarAmp. A recurrent neural network for black-box guitar amplifier modeling, based on the architecture of Wright et al. [1, 2]. It processes one audio sample per time step through an LSTM cell (hidden size 20) followed by a dense output layer, totaling 1,861 parameters. Like the oscillator, it operates sample-by-sample, but its recurrent architecture makes successive samples inherently sequential, i.e., processing N samples requires N LSTM steps. This makes it an important test case, representing the irreducibly sequential regime where GPU acceleration can only help within each time step, not across time.

NAM. An open-source platform for amplifier and effects modeling, based on a feedforward WaveNet [28] variant with dilated causal 1D convolutions, gated activations and skip connections (13,802 parameters, "standard" model). Unlike the recurrent models, NAM is block-based; it processes a buffer of samples in parallel through the convolution stack, making it a natural candidate for GPU acceleration.

MS-Wavehax. A lightweight neural vocoder [29] (∼500K parameters) that converts mel spectrograms and F0 contours into audio. The architecture interleaves deterministic DSP stages with learned layers. On the input side, F0 generates a harmonic prior, which is split into sub-bands by learned analysis filters before an STFT converts each to a complex spectrogram. On the output side, an iSTFT and learned synthesis filters reconstruct the waveform. The core is a stack of 8 ConvNeXt-style blocks using 2D depthwise separable convolutions (7×7 kernels), operating on the time-frequency representation—structurally identical to mobile vision workloads and precisely the type of operation Qualcomm's GPU is optimized for. We created a 48 kHz configuration by doubling the hop length (to 480 samples) and per-sub-band FFT size (to 240), replaced reflect padding with causal zero padding, and retrained from scratch. We benchmark the neural core only; surrounding stages are detailed in Section 5.

BRAVE. A causal convolutional variational autoencoder for timbre transfer [11], redesigned from RAVE [10] for sub-10 ms latency (∼6M parameters). An encoder compresses audio at 128:1 via strided 1D convolutions, and a decoder reconstructs via transposed 1D convolutions, with multi-band filterbanks at input and output. Streaming requires 47 cache tensors (∼320 KB of state); valid block sizes must be multiples of 128. BRAVE is the only model trained at 44.1 kHz; retraining at 48 kHz would require redesigning the architecture. Caspe et al. originally implemented BRAVE inference in RTNeural, but this code was not publicly released. We instead export to ONNX using an explicit stateless cache formulation where all 47 tensors are passed as inputs and outputs at each call.

[9] `https://github.com/victorzappi/integrated-gpu-neural-audio/tree/main/models`

4. EXPERIMENTAL SETUP

All experiments are conducted on the Qualcomm RB3 Gen 2 development board, built around the QCS6490 SoC. The CPU is an 8-core Kryo 670 in a tri-cluster configuration: one Gold Prime core at 2.7 GHz (Cortex-A78), three Gold cores at 2.4 GHz (Cortex-A78) and four Silver cores at 1.9 GHz (Cortex-A55). The GPU is an Adreno 643 running at 812 MHz. The board has 8 GB of LPDDR4x RAM, shared between CPU and GPU in a unified memory architecture. The board runs Qualcomm Linux (OpenEmbedded, Yocto-based) with a standard non-preemptive kernel. All C++ code, including inference libraries, was cross-compiled using Qualcomm's Intelligent Multimedia Product SDK[10]. Software versions used were QAIRT SDK 2.42, ONNX Runtime 1.22.0 and the latest available versions of RTNeural (Eigen backend) and NAMcore at the time of writing.

To test models under realistic audio conditions, we built a minimal C++ audio engine[11] on top of Qualcomm's AudioReach [30] framework. AudioReach routes the audio signal through the onboard Hexagon DSP to the hardware codec for playback. This routing runs independently and does not consume CPU or GPU resources available to inference. The engine opens a playback stream at 48 kHz, registers a callback, processes audio and converts it into the format expected by the hardware. The callback thread runs at the highest scheduling priority with CPU affinity set to the fastest core forced to run at full speed (2.7 GHz). All models are mono; the output is duplicated to both channels. Within each callback, we log the wall-clock execution time of the inference call and thermal readings across board zones.

We compare three inference approaches for each model: *best available CPU*, *QNN CPU* and *QNN GPU*. The best available CPU baseline is the best CPU-based inference we could achieve, selected per model: RTNeural for the Neural Oscillator and AutoGuitarAmp, NAMcore for NAM and ONNX Runtime for MS-Wavehax and BRAVE, whose layer types are not supported by specialized engines. QNN CPU and QNN GPU both use QNN[12] from the QAIRT SDK introduced in Section 1, which provides a unified API for deploying models on different compute backends from a single model representation. All models are first exported to ONNX and then converted to Qualcomm's DLC format. The same DLC is loaded at runtime targeting either the CPU or GPU backend, accessed through a thin C API wrapper integrated into our audio engine.

Unlike the other engines, QNN allows *batching* multiple sample-level invocations into a single call by reshaping the input tensor to *batch size* > 1—this enables parallelism for sample-by-sample models unless they embed recurrent layers (e.g., AutoGuitarAmp). We parallelize whenever possible: batch size > 1, block size $= 1$ for sample-by-sample models, or batch size $= 1$, block size > 1 for block-based models. In both cases the audio buffer size of the engine equals the batch or block size, so each callback triggers exactly one inference call.

To verify that the conversion pipeline preserves output quality, we confirmed that PyTorch and ONNX outputs are numerically identical within floating-point precision. Informal listening confirmed no audible difference across all four engines.

[10] `https://github.com/qualcomm-linux/meta-qcom-qim-product-sdk`

[11] `https://github.com/victorzappi/ar-audioengine`

[12] `https://www.qualcomm.com/developer/software/qualcomm-ai-engine-direct-sdk`

Table 1: Summary of the five benchmark models.

Model	Task	Regime	Sample Rate	Key Layers	Parameters
Neural Oscillator	Synthesis	Sample-by-sample	48 kHz	Dense, ReLU	769
AutoGuitarAmp	Amp modeling	Sample-by-sample	48 kHz	LSTM, Dense	1,861
NAM	Amp modeling	Block-based	48 kHz	Dilated Conv1d, 1×1 Conv	13,802
MS-Wavehax	Vocoding	Block-based	48 kHz	2D Depthwise Separable Conv, Conv1d	∼500K
BRAVE	Timbre transfer	Block-based	44.1 kHz	VAE, Strided Conv1d, ConvTranspose1d	∼6M

Each test consists of a 10-second audio run repeated 5 times, with per-sample inference times concatenated before computing mean and standard deviation. At 48 kHz, this yields up to 2,400,000 data points for sample-by-sample models and approximately 1,700 for the largest block size tested (1,440 samples), ensuring that warm-up transients have negligible impact on the reported statistics. For sample-by-sample models on the best available CPU engine, the buffer size is fixed at 256 samples (5.3 ms). For all other configurations, it matches the batch or block size. The minimum buffer supported by the RB3 Gen 2 is 64 samples (1.3 ms).

We assess real-time viability by comparing mean per-sample computation time against the sample period ($1/48{,}000 \approx 20.83\,\mu$s). This is a necessary but not sufficient condition; in practice, format conversion, data transfer and scheduling indeterminacy consume part of the budget [20]. We additionally track audio underruns—callbacks whose inference time exceeds the buffer period—as a complementary indicator.

5. RESULTS

Thermal monitoring across all board zones showed a peak average temperature of approximately 81 °C on the hottest CPU sensor, well below the SoC's passive throttling threshold. No thermal throttling was observed during any test run, confirming that all inference time measurements reported below reflect sustained performance rather than thermally degraded operation; mobile consumer devices may throttle earlier though, due to enforced skin-temperature limits. In the following subsections, we present per-sample inference time results for each model, organized by inference engine and batch or block size. Each subsection also details model-specific implementation choices, including supported batch/block sizes, audio configuration (synthesis vs. processing of an input signal) and any adaptations required for deployment. The C++ code for each test as well as the exported models (JSON, ONNX, and DLC files) are available online[13], under `/projects` and `/models` respectively.

5.1. Neural Oscillator

The Neural Oscillator is configured as a 440 Hz sine synthesizer, receiving a phase ramp as input and producing audio samples as output. No input audio is processed.

For the best available CPU baseline, the model runs sample-by-sample via RTNeural. This type of small, fixed-architecture network is precisely the use case where RTNeural excels [15, 20, 26]. For QNN, we test batch sizes of 1, 64, 128, 256, 512 and 1024 samples. A batch size of 1 isolates the per-call overhead of the QNN runtime with minimal computation, while batch sizes of 64–1024 represent typical low-latency real-time audio buffer sizes (1.3–21.3 ms at 48 kHz) and allow the overhead to be amortized across more samples. Figure 1 shows the mean per-sample inference time with standard deviation for each configuration.

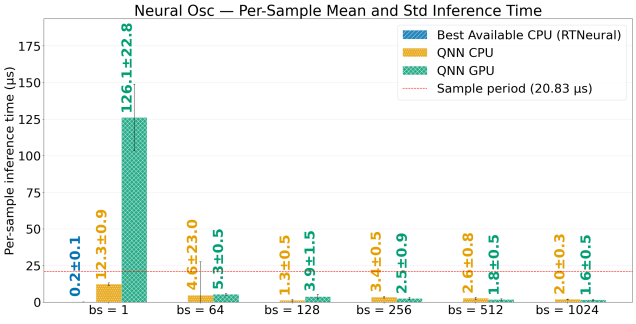

Figure 1: Mean per-sample inference time for Neural Oscillator, across inference engines and QNN batch sizes. The dashed line indicates the sample period ($1/48{,}000 \approx 20.83\,\mu$s).

RTNeural dominates with a mean per-sample time of 0.2 μs, an order of magnitude faster than any QNN configuration. At batch size 1, the QNN GPU backend performs poorly (126.1 μs), well above the real-time threshold, confirming that per-call overhead dwarfs the actual computation for this tiny model. Both QNN backends improve substantially as batch size increases. GPU drops from 126.1 to 1.6 μs and CPU from 12.3 to 2.0 μs across the tested range. All configurations meet the real-time threshold except QNN GPU at batch size 1, but for a model of this size, RTNeural remains the clear choice.

5.2. AutoGuitarAmp

AutoGuitarAmp processes an uncompressed mono audio file through the LSTM-based amplifier model. As with the Neural Oscillator, RTNeural serves as the best available CPU baseline, running sample-by-sample. As discussed in Section 3 and unlike the previous model, no batching is possible here (all three engines run with a batch size of 1). Table 2 reports the mean per-sample inference time for each engine.

RTNeural achieves a mean per-sample time of 0.43 μs, comfortably below the real-time threshold. QNN CPU is roughly 44× slower at 19.06 μs—practically not enough for real time. QNN GPU performs worst at 167.59 μs, over 8× above the real-time threshold, with high variance (std = 31.87 μs). This is the expected outcome for an irreducibly sequential model. The GPU cannot parallelize across time steps, so the per-call overhead dominates with no opportunity for amortization through batching.

[13] `https://github.com/victorzappi/integrated-gpu-neural-audio`

Table 2: AutoGuitarAmp: mean and standard deviation of per-sample inference time at batch size 1.

Engine	Batch Size	Mean (μs)	Std (μs)
Best CPU (RTNeural)	1	0.43	0.16
QNN CPU	1	19.06	0.75
QNN GPU	1	167.59	31.87

5.3. NAM

NAM processes the same mono audio file as AutoGuitarAmp. The best available CPU baseline uses NAMcore, the dedicated Eigen-based inference engine optimized for NAM's WaveNet architecture.

As a block-based convolutional model, NAM naturally processes multiple samples in parallel within each forward pass. All three engines are tested at block sizes of 64, 128, 256, 512 and 1024 samples. Figure 2 shows the mean per-sample inference time for each engine and block size.

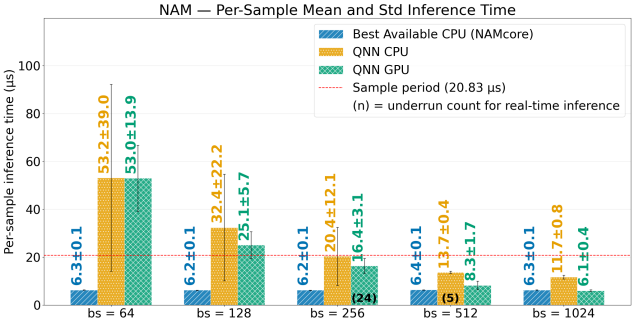

Figure 2: Mean per-sample inference time for NAM across inference engines and block sizes. The dashed line indicates the sample period (20.83 μs). Underruns are shown only where real-time performance is achieved.

NAMcore is remarkably stable, delivering a mean per-sample time of approximately 6.3 μs regardless of block size, well below the real-time threshold. Both QNN backends start poorly at block size 64 (QNN CPU: 53.2 μs; QNN GPU: 53.0 μs) but improve substantially with increasing block size. The GPU backend benefits most, dropping from 53.0 to 6.1 μs at block size 1024—an 88% reduction—effectively matching NAMcore performance. QNN CPU also improves (53.2 to 11.7 μs, a 78% reduction) but remains roughly twice as slow as NAMcore at the largest block size. This is the first model in our benchmark where the GPU backend approaches the best available CPU baseline, suggesting that NAM's dilated 1D convolutions offer enough parallelism for the GPU to amortize its overhead at larger block sizes.

5.4. MS-Wavehax

MS-Wavehax synthesizes audio from mel spectrogram and F0 inputs. The best available CPU baseline uses ONNX Runtime, as the model's 2D depthwise separable convolutions would require reimplementation from scratch in RTNeural.

As described in Section 3, we benchmark the neural core only. The core is exported to ONNX as a standalone module that receives mel conditioning and prior sub-band spectrograms as inputs, and produces refined sub-band spectrograms as outputs. The surrounding stages are excluded because they would run faster on CPU than on GPU, and including them in the GPU path would artificially penalize the GPU measurements. In particular, the deterministic stages (harmonic prior, STFT/iSTFT) are best served by optimized native C++ DSP libraries such as NE10[14], while the learned analysis and synthesis filters are well suited to inference via RTNeural, given their small size and simple topology [20].

Block sizes are constrained to multiples of the hop length (480 samples). We test 480, 960 and 1440 samples—the closest values to the range used for the other models (10, 20 and 30 ms at 48 kHz). Figure 3 shows the mean per-sample inference time for each engine and block size.

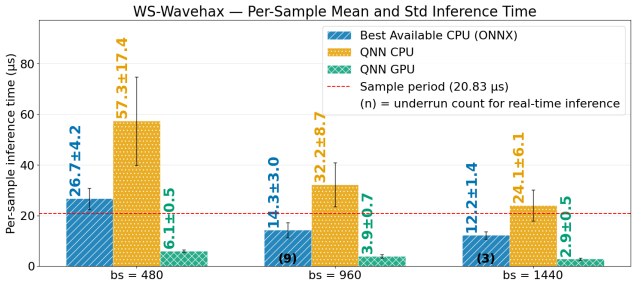

Figure 3: Mean per-sample inference time for MS-Wavehax (neural core) across inference engines and block sizes. Underruns shown for real-time performance only.

MS-Wavehax is the first model in our benchmark where the GPU backend decisively outperforms all CPU options. ONNX Runtime exceeds the real-time threshold at block size 480 (26.7 μs) and improves with increasing block size (a 54% reduction to 12.2 μs at 1440), but still incurs underruns at 960 and 1440 samples. QNN CPU follows a similar trend, starting worse (57.3 μs) and ending worse (24.1 μs), never reaching real-time operation. In contrast, the QNN GPU backend runs comfortably in real-time from the smallest block size (6.1 μs at 480) and continues to improve with block size, reaching 2.9 μs at 1440—a 52% reduction. The 2D depthwise separable convolutions at the core of MS-Wavehax are an excellent match for the Adreno 643's compute architecture. It's worth noting that the GPU leaves substantial headroom within the audio callback for the surrounding stages (harmonic prior, STFT/iSTFT, analysis and synthesis filters) that must also execute each frame—headroom that neither CPU-based engine can provide.

5.5. BRAVE

BRAVE performs timbre transfer on a streaming mono audio file. As discussed in Section 3, the best available CPU baseline uses ONNX Runtime.

Due to the 128:1 compression ratio, valid block sizes must be multiples of 128 samples, hence we test 128, 256, 512 and 1024 on all three engines. Although BRAVE was trained at 44.1 kHz (see Section 3), we run it at 48 kHz in our benchmarks to keep results directly comparable across all five models. This may affect synthesis quality, but evaluating that impact is beyond the scope of

[14] https://projectne10.github.io/Ne10/

this work. Figure 4 shows the mean per-sample inference time for each engine and block size.

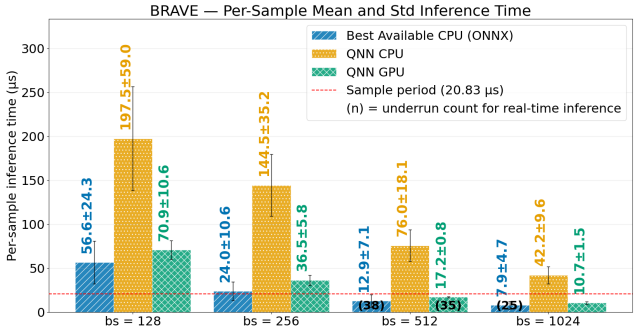

Figure 4: Mean per-sample inference time for BRAVE across inference engines and block sizes. Underruns shown for real-time performance only.

BRAVE shows a more nuanced picture. ONNX Runtime benefits substantially from larger block sizes, with mean per-sample time crossing below the real-time threshold at block size 512, and dropping from 56.6 μs at 128 to 7.9 μs at 1024. This 86% reduction speaks to the quality of the streaming redesign made by Caspe et al. [11]. QNN CPU improves with block size (197.5 to 42.2 μs) too, but never approaches real-time operation.

The GPU backend starts on par with ONNX Runtime at block size 128 (70.9 vs. 56.6 μs), suggesting that the strided 1D convolutions in BRAVE's architecture offer meaningful parallelism from the outset. It improves steadily from 70.9 to 10.7 μs at block size 1024—an 85% reduction—and also crosses the real-time threshold at block size 512. At first glance, GPU appears slightly slower than ONNX Runtime at the larger block sizes. However, the underrun counts indicate that mean inference time alone does not capture the full picture. ONNX Runtime incurs underruns at both 512 (38) and 1024 (25) indicating that, despite a low mean, its inference spikes periodically violate the callback deadline. The GPU backend, by contrast, produces zero underruns at block size 1024, making it the only engine that achieves reliable real-time inference for this model.

6. DISCUSSION

The results reveal a clear pattern, i.e., integrated GPU acceleration is not universally beneficial for real-time neural audio inference. Its value depends on the interplay between model size, architectural parallelism and block size.

Unsurprisingly, for small models with limited parallelism, the GPU offers no advantage. For the Neural Oscillator and AutoGuitarAmp, optimized CPU engines such as RTNeural remain the clear choice and the added complexity of a GPU deployment pipeline is not justified. Without any opportunity for temporal parallelism, the recurrent case is particularly unfavorable to the GPU backend.

The picture changes as model complexity increases, but not simply as a function of parameter count. NAM's dilated 1D convolutions sit near the lower bound of GPU viability for the QCS6490's Adreno 643—competitive only at the largest block sizes. MS-Wavehax, by contrast, represents the sweet spot; its 2D depthwise separable convolutions provide enough native parallelism for the GPU to dominate at all tested block sizes, with headroom to spare for surrounding DSP stages. BRAVE, despite being the largest model, falls between these two cases due to its less parallelizable 1D architecture, requiring large block sizes to approach real-time operation.

6.1. Real-time Safety of GPU Inference

The underruns observed with ONNX Runtime on BRAVE are consistent with known limitations of general-purpose inference engines in real-time audio contexts [20, 26]. ONNX Runtime may perform dynamic memory allocations and other non-real-time-safe operations during inference, leading to occasional inference spikes that violate the audio callback deadline, even when the mean inference time is below the threshold.

The GPU backend is inherently more predictable in this regard. QNN pre-compiles the inference graph and pre-allocates all memory at initialization; during inference, execution proceeds as a deterministic sequence of kernel dispatches, isolated from the CPU's memory allocator and OS-level thread scheduling. BRAVE results provide direct evidence of this, i.e., comparable mean inference times, but zero underruns on GPU versus dozens on CPU. This suggests that even when GPU and CPU inference times are similar, the GPU may be the preferable backend for real-time audio due to its more predictable execution model.

6.2. Borrowed Parallelism

Our results consistently show that increasing the buffer size improves GPU inference time. While this may appear to be a straightforward optimization, it should be approached with caution. In real-time audio, buffer size directly determines interaction latency and latencies above 10 ms are already problematic for many musical applications [31]. Relying on larger buffers to compensate for insufficient architectural parallelism (what we term *borrowed parallelism*) is therefore not a viable deployment strategy.

The Neural Oscillator and NAM illustrate this. Large batch/block sizes dramatically reduce GPU inference time but either yield no net gain over CPU or push latency beyond acceptable limits. MS-Wavehax demonstrates the alternative; its 2D convolutional architecture provides enough native parallelism that the GPU wins at the smallest tested block size, with no need to inflate the buffer. These findings have direct implications for model design, as discussed in Section 6.5.

6.3. Inter-run GPU Variability

On the GPU, mean inference time can vary significantly between consecutive runs of the same test. While not visible in the aggregated bar plots of Section 5, this phenomenon is clearly depicted in per-run plots.

Figure 5 illustrates this for NAM at block size 256 on QNN GPU. The five runs cluster into two distinct performance modes, with runs 1 and 3 averaging approximately 13 μs and runs 2, 4 and 5 approximately 19 μs. Within each run, the variance is low; the large overall standard deviation (3.10 μs) is almost entirely driven by this inter-run mode switching rather than intra-run jitter.

This bimodal behavior was not observed on any CPU configuration. It suggests that GPU performance depends on initialization-time state (such as kernel compilation strategy, cache configuration, or memory layout) that is resolved once at startup and persists throughout the run. The "fast mode" ($\sim$13 μs) represents the true

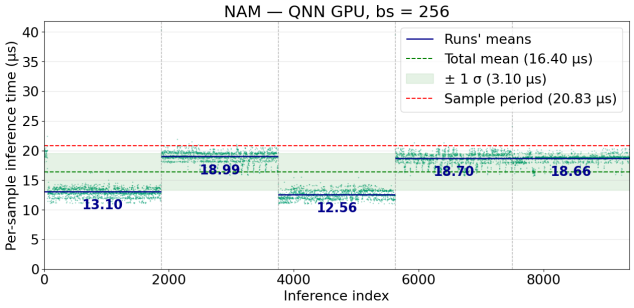

Figure 5: NAM's per-inference timing (QNN GPU, block size 256) across five consecutive 10-second runs. The GPU settles into one of two performance modes at initialization, with approximately 45% difference between them.

performance ceiling of the GPU for this configuration; the averaged result reported in the bar plots (16.4 μs) is a mixture of both modes and therefore underestimates the GPU's best-case capability. Understanding and reliably triggering this fast-mode initialization is an important direction for future work.

This behavior was not observed consistently across all GPU configurations though, and its magnitude varied. We attempted to eliminate it through QNN runtime settings (including performance profile and cache configuration options), but to no avail.

6.4. Transferability to Other Snapdragon Hardware

The QCS6490 SoC tested in this work is a mid-range chip in the Snapdragon family. A natural question is how these results transfer to the more powerful SoCs found in flagship smartphones (e.g., Snapdragon 8 Gen 3, 8 Elite) and laptops (e.g., Snapdragon X Elite, X2 Elite).

Moving up the Snapdragon family, both CPU and GPU performance increase substantially, but not proportionally. Single-core CPU performance roughly doubles from the QCS6490's Kryo 670 (2.7 GHz, Cortex-A78) to the Oryon cores found in flagship mobile and laptop SoCs (4.0–5.0 GHz, with wider pipelines). This means that optimized CPU baselines—RTNeural, NAMcore and similar engines—will become significantly faster on newer hardware. The GPU also improves dramatically, but with a caveat. The Adreno X1 in the Snapdragon X Elite delivers 4.6 TFLOPS compared to the Adreno 643's more modest throughput and the Adreno X2 roughly doubles that again. However, per-call overhead (the fixed cost of dispatching work to the GPU) may not shrink in proportion to raw compute gains.

The implication is that on more powerful SoCs, the GPU advantage threshold likely shifts to even larger and more complex models. And the borrowed parallelism problem may intensify; as CPUs get faster, the buffer sizes required for GPU parity increase, pushing developers further into latency territory that is unacceptable for interactive audio. Conversely, models that provide sufficient native parallelism—like MS-Wavehax's 2D convolutional core—may see even larger GPU advantages on these more powerful Adreno GPUs.

6.5. Implications for Model Design

Many neural audio models designed for streaming inference are deliberately lightweight, implicitly targeting CPU execution. This is a well-motivated strategy. Models such as LPCNet and its successors [8], FARGAN [25] and MS-Wavehax itself [29] were designed to minimize computational cost for single-threaded CPU deployment. The ecosystem of CPU inference engines (RTNeural, NAMcore, ANIRA) reinforces this paradigm.

This CPU-first design philosophy is particularly sensible when the alternative is a discrete GPU—a resource that is more powerful, but less widespread and penalized by explicit host-device memory transfers. Integrated GPUs share unified memory with the CPU, eliminating this cost and lowering the viability thresholds identified in our results. And, as previously discussed, they are also increasingly prevalent (from embedded boards to smartphones and laptops) and feature dedicated SDKs.

Our results suggest a complementary design direction that leverages this opportunity. Neural audio architectures that provide sufficient native parallelism can exploit integrated GPU acceleration even at small block sizes, achieving performance that CPU inference cannot match. The key architectural ingredients, as demonstrated by MS-Wavehax, are layer types with high data-level parallelism (e.g., 2D convolutions, depthwise separable convolutions) and sufficient model complexity to amortize per-call GPU overhead. Designing models with integrated GPU deployment in mind—rather than retrofitting CPU-oriented architectures onto the GPU—could unlock a new class of real-time audio applications in mobile and embedded domains.

7. CONCLUSION AND FUTURE WORK

We have presented a systematic benchmark of integrated GPU acceleration for real-time neural audio inference, evaluating five models across three backends on a Qualcomm Snapdragon SoC. GPU acceleration is not universally beneficial. Small and recurrent models remain best served by optimized CPU engines. However, models with sufficient size and native parallelism see the GPU decisively outperform all CPU alternatives, with more predictable execution that avoids the inference time spikes that often cause underruns in real-time callbacks. We caution against compensating for insufficient architectural parallelism by inflating buffer sizes, and argue that designing models for integrated GPU deployment is a promising direction for mobile and embedded real-time audio.

Future work includes consolidating GPU performance by identifying the initialization parameters that control the inter-run variability observed in our benchmarks (Section 6.3) and profiling GPU core utilization to quantify how much compute capacity each model actually exploits. Most importantly, the QCS6490 also integrates a Hexagon NPU with a dedicated tensor accelerator rated at 12 TOPS. This accelerator was excluded from the present study because it requires INT8 or INT16 quantization, whose impact on neural audio quality warrants a dedicated investigation. NPU performance is scaling rapidly across the Snapdragon family, and benchmarking quantized neural audio models on these accelerators is a natural next step toward a complete assessment of on-chip acceleration for real-time audio.

8. REFERENCES

[1] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. 22nd Int. Conf. Digital Audio Effects (DAFx)*, Birmingham, UK, 2019.

[2] 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.

[3] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, 2019, pp. 471–475.

[4] C. J. Steinmetz and J. D. Reiss, "Efficient neural networks for real-time modeling of analog dynamic range compression," *arXiv preprint arXiv:2102.06200*, 2022, presented at AES Convention 152.

[5] N. Zeghidour, A. Luebs, A. Omber, J. Skoglund, and M. Visentin, "SoundStream: An end-to-end neural audio codec," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 495–507, 2022.

[6] A. Défosséz, J. Copet, G. Synnaeve, and Y. Adi, "High fidelity neural audio compression," *Transactions on Machine Learning Research (TMLR)*, 2023.

[7] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 17 022–17 033.

[8] J.-M. Valin and J. Skoglund, "LPCNet: Improving neural speech synthesis through linear prediction," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, 2019, pp. 5271–5275.

[9] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2020.

[10] A. Caillon and P. Esling, "RAVE: A variational autoencoder for fast and high-quality neural audio synthesis," *arXiv preprint arXiv:2111.05011*, 2021.

[11] F. Caspe, A. McPherson, and M. Sandler, "Designing neural synthesizers for low-latency interaction," *Journal of the Audio Engineering Society*, 2025.

[12] S. Miklanek and J. Schimmel, "Fast temporal convolutions for real-time audio signal processing," in *Proc. 25th Int. Conf. Digital Audio Effects (DAFx)*, Vienna, Austria, 2022.

[13] D. Südholt and C. Erkut, "Vocal timbre effects with differentiable digital signal processing," in *Proc. 26th Int. Conf. Digital Audio Effects (DAFx)*, Copenhagen, Denmark, 2023, pp. 363–366.

[14] A. Caillon and P. Esling, "Streamable neural audio synthesis with non-causal convolutions," in *Proc. 25th Int. Conf. Digital Audio Effects (DAFx)*, Vienna, Austria, 2022.

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

[16] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teber, J. Sotelo, A. de Brébisson, Y. Bengio, and A. Courville, "MelGAN: Generative adversarial networks for conditional waveform synthesis," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[17] S.-g. Lee, W. Ping, B. Ginsburg, B. Catanzaro, and S. Yoon, "BigVGAN: A universal neural vocoder with large-scale training," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.

[18] R. Prenger, R. Valle, and B. Catanzaro, "WaveGlow: A flow-based generative network for speech synthesis," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, 2019, pp. 3617–3621.

[19] C. Mitcheltree, N. Martikainen, M. A. Martinez Ramirez, A. Kuznetsov, C. J. Steinmetz, M. Comunita, and H.-H. Lim, "Neutone SDK: An open source framework for neural audio processing," *arXiv preprint arXiv:2508.09126*, 2025.

[20] J. Hoopes, B. Chalmers, and V. Zappi, "Neural audio processing on Android phones," in *Proc. 27th Int. Conf. Digital Audio Effects (DAFx)*, Guildford, UK, 2024.

[21] T. Pelinski, R. Diaz, A. L. Benito Temprano, and A. McPherson, "Pipeline for recording datasets and running neural networks on the Bela embedded hardware platform," in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Mexico City, Mexico, 2023, pp. 160–166.

[22] J. Turian, J. Shier, G. Tzanetakis, K. McNally, and M. Henry, "One billion audio sounds from GPU-enabled modular synthesis," in *Proc. 24th Int. Conf. Digital Audio Effects (DAFx)*, 2021.

[23] O. Shafi, C. Rai, R. Sen, and G. Ananthanarayanan, "Demystifying tensorrt: Characterizing neural network inference engine on nvidia edge devices," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 226–237.

[24] T. Skare, "General-purpose gpu audio benchmark framework," in *Proc. 27th Int. Conf. Digital Audio Effects (DAFx)*, Guildford, UK, 2024.

[25] J.-M. Valin, A. Mustafa, and J. Büthe, "Very low complexity speech synthesis using framewise autoregressive gan (fargan) with pitch prediction," *IEEE Signal Processing Letters*, vol. 31, pp. 2115–2119, 2024.

[26] V. Ackva and F. Schulz, "ANIRA: An architecture for neural network inference in real-time audio applications," in *Proc. IEEE Int. Symp. Industrial Electronics (IS2)*, 2024, industry Award.

[27] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 75–84, best Paper Award.

[28] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[29] R. Yoneyama, M. Kawamura, R. Terashima, R. Yamamoto, and T. Toda, "Comparative analysis of fast and high-fidelity neural vocoders for low-latency streaming synthesis in resource-constrained environments," in *Proc. Interspeech*, Rotterdam, The Netherlands, 2025.

[30] P. Lai, "Audioreach: An end-to-end solution," in *Proceedings of the 19th Linux Audio Conference*, 2025.

[31] A. P. McPherson, R. H. Jack, G. Moro *et al.*, "Action-sound latency: Are our tools fast enough?" in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*. Griffith University, 2016.