Real-Time AI-Generated Sound Effects for Video Games

A Latency and Quality Analysis

Igor Szuniewicz

Supervisor: De Meulemeester Roel
Coach: Van der Kelen Cedric

Graduation work 2025-2026

howest hogeschool

DIGITAL ARTS & ENTERTAINMENT

Abstract & Keywords

ABSTRACT (ENG)

Real-time audio generation in video games presents a fundamental trade-off between output variation and system latency. While generative models (AudioGen, MMAudio, ElevenLabs) can produce contextually diverse sound effects from text prompts, their inference times (2-5 seconds per sample) significantly exceed acceptable latency thresholds for interactive gameplay (100-150 milliseconds).

This thesis evaluates the feasibility of AI-generated sound effects for real-time game audio by measuring latency, quality, and reliability across three generation approaches: procedural synthesis, local GPU inference, and cloud-based APIs. A hybrid system was developed connecting Unreal Engine 5 to multiple generative backends, incorporating caching mechanisms and fallback strategies.

Results indicate that uncached AI generation remains unsuitable for interactive sounds requiring frame-accurate timing (footsteps, impacts, UI feedback). However, combining AI generation with predictive caching and pre-generation enables practical deployment for ambient audio, music, and recurring sound effects. Cloud-based services (ElevenLabs) demonstrated superior audio quality but exhibited variable network latency (range: 1.8-8.3 seconds). Local GPU models achieved lower latencies (AudioGen: 2.2±0.4s, MMAudio Small: 1.4±0.3s) but required 2.4-3.1 GB VRAM and occasionally produced silent outputs (rate: 2-4%).

The study provides quantitative benchmarks for real-time AI audio implementation and demonstrates that strategic caching reduces perceived latency by 95-98%, enabling generative audio as a practical workflow tool for contemporary game development within hardware constraints typical of independent studios.

SAMENVATTING (NL)

Real-time audiogeneratie in videogames presenteert een fundamentele afweging tussen outputvariatie en systeemlatentie. Hoewel generatieve modellen (AudioGen, MMAudio, ElevenLabs) contextueel diverse geluidseffecten kunnen produceren vanuit tekstprompts, overschrijden hun inferentietijden (2-5 seconden per sample) aanvaardbare latentiedrempels voor interactieve gameplay (100-150 milliseconden) aanzienlijk.

Deze thesis evalueert de haalbaarheid van AI-gegenereerde geluidseffecten voor real-time gameaudio door latentie, kwaliteit en betrouwbaarheid te meten over drie generatie-aanpakken: procedurele synthese, lokale GPU-inferentie, en cloudgebaseerde APIs. Een hybride systeem werd ontwikkeld dat Unreal Engine 5 verbindt met meerdere generatieve backends, met cachemechanismen en fallback-strategieën.

Resultaten tonen aan dat niet-gecachte AI-generatie ongeschikt blijft voor interactieve geluiden die frame-accurate timing vereisen (voetstappen, impacten, UI-feedback). Het combineren van AI-generatie met predictieve caching en pre-generatie maakt echter praktische implementatie mogelijk voor omgevingsaudio, muziek en terugkerende geluidseffecten. Cloudgebaseerde services (ElevenLabs) toonden superieure audiokwaliteit maar vertoonden variabele netwerklatentie (bereik: 1.8-8.3 seconden). Lokale GPU-modellen bereikten lagere latenties (AudioGen: 2.2±0.4s, MMAudio Small: 1.4±0.3s) maar vereisten 2.4-3.1 GB VRAM en produceerden occasioneel stille outputs (percentage: 2-4%).

De studie levert kwantitatieve benchmarks voor real-time AI-audio-implementatie en toont aan dat strategische caching waargenomen latentie met 95-98% reduceert, waardoor generatieve audio een praktisch workflow-instrument wordt voor hedendaagse game-ontwikkeling binnen hardwarebeperkingen typisch voor onafhankelijke studios.

Keywords: generative audio, AI sound effects, Unreal Engine 5, real-time latency, caching, pre-generation, perceptual quality, hybrid cloud-edge

Preface

Preface

This year I worked on *Shadow Frames*, a horror game our team made as a group project. I handled all the audio, which meant finding and designing over 100 sound effects. Creaking doors, footsteps on different surfaces, ambient drones, paranormal whispers - the kind of sounds that make or break a horror game.

Figure 1: Shadow Frames - Horror game environment showcasing atmospheric lighting and detailed interior design that required comprehensive sound design

Most of my time went into searching. Scrolling through Soundly, Freesound, Splice. Listening to half a second of each sample before moving to the next. Some sounds were easy to find. Others took forever because the exact thing I needed simply was not in any library.

Around the same time, I started experimenting with AI audio tools like ElevenLabs. You type a description and it generates something. Most of the time the result is not quite what you want. But sometimes it surprises you with something unexpected.

That made me wonder: if AI can create sounds from text, could a game engine do the same thing during gameplay? Instead of loading pre-made files, it would generate the sound exactly when needed. No more searching. No more libraries. Just describe what you need, and the game creates it in real-time.

That question became this thesis. Through this research, I hope to provide concrete measurements and practical insights that will help other developers understand when - and if - AI audio generation is ready for real-time game audio.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem: Why Audio Matters

Interactive audio in video games serves as critical sensory feedback for player actions. In *Shadow Frames*, when a player's footsteps sync perfectly with character animation, or when a door creak triggers instantly on interaction, these millisecond-precise details maintain the illusion of presence. When these sounds are delayed by even 200 milliseconds, players notice - the game feels "laggy" or unresponsive. When sounds are missing or contextually wrong, immersion breaks.

Research on audio latency in interactive systems establishes concrete thresholds for acceptable delays. Kaaresoja et al. [6] recommend 20-70ms for audio feedback in interactive applications, while Halbhuber et al. [5] found that audio latency of 40ms in first-person shooters did not significantly affect player performance, but delays of 270ms and above caused measurable degradation in game experience. Game audio systems must therefore maintain consistent sub-100ms response times for interactive events such as footsteps, impacts, and UI feedback.

Traditional game audio workflows rely on pre-recorded sound libraries, requiring designers to manually search, edit, and implement hundreds of individual assets per project. Recent advances in generative AI models (AudioGen [2], MMAudio [3], ElevenLabs) offer an alternative approach: synthesizing sound effects from text descriptions at runtime. These models can produce contextually appropriate audio (e.g., "footstep on stone, close mi-

crophone", "metal impact, sharp transient") without requiring pre-existing recordings.

However, contemporary text-to-audio models present significant constraints for real-time deployment. Inference times typically range from 2-5 seconds per sample, GPU memory requirements reach 2-4 GB per loaded model, and reliability issues include silent output failures in 2-4% of generation attempts. For game engines operating at 60 frames per second (16.7ms per frame), these latencies exceed acceptable thresholds by orders of magnitude.

Note: The core tension: Generative AI offers parametric variation in audio output but introduces latencies incompatible with real-time interaction requirements. Can strategic caching and pre-generation bridge this gap?

This raises the central research question:

Can AI-generated sound effects be produced fast enough for real-time gameplay?

1.2 Research Questions and Hypotheses

This research question is operationalized through three specific sub-questions addressing latency, caching effectiveness, and quality trade-offs:

RQ1: How do latencies compare across procedural synthesis, local GPU inference, and cloud-based synthesis?
RQ2: How much do caching and pre-generation reduce perceived latency?
RQ3: What trade-offs exist between latency, reliability, and quality?

Based on preliminary testing and literature review, three hypotheses were formulated:

H1: Cloud synthesis (ElevenLabs) will provide superior audio quality but will not meet interactive latency budgets due to network overhead.

H2: Procedural DSP synthesis will meet latency requirements consistently but will yield lower perceived realism compared to AI-generated audio.

H3: Caching and pre-generation strategies can reduce perceived latency for repeated sounds to near-instantaneous levels.

1.3 What Real-Time Means

"Real-time" means different things for different sound types. I categorize game audio into three latency tolerance classes:

Category	**Latency Budget**
Interactive SFX	100-150 ms. Footsteps, impacts, UI confirmations must feel instant.
Ambience Loops	Several seconds acceptable. Transitions are crossfaded; pre-generation is practical.
Music/Long-Form	No strict limit. Can generate asynchronously in the background.

Table 1.1: Latency budgets by sound category

1.4 Contributions

This work contributes four things to the field:

1. **A working Unreal Engine 5 integration** connecting multiple generative models to a real-time game engine
2. **A measurement methodology** separating cold-start costs, warm inference, network overhead, and engine-side processing
3. **An evaluation framework** combining objective metrics (latency, VRAM, failure rate) with subjective quality assessment (perceptual quality ratings)
4. **Documented patterns** for caching, pre-generation, DSP variation, and fallbacks that make slow models usable

1.5 Thesis Structure

Chapter 2 reviews related work on game audio constraints, generative models, and real-time systems. Chapter 3 describes the research methodology, hardware setup, and evaluation approach. Chapter 4 documents the technical implementation. Chapter 5 presents results. Chapter 6 discusses findings and limitations. Chapter 7 concludes with recommendations for developers.

Chapter 2

Literature Study

2.1 Game Audio and Latency: Why Speed Matters

2.1.1 Human Perception of Audio-Visual Sync

Humans are surprisingly sensitive to audio timing in interactive contexts. Research on audio latency perception establishes that the just noticeable difference (JND) for audio feedback is approximately 49ms under optimal conditions [8]. Kaaresoja et al. [6] recommend 20-70ms as the acceptable range for audio feedback in interactive systems, with delays beyond this threshold becoming perceptible.

In competitive gaming contexts, Liu et al. [7] demonstrated that latency linearly degrades both quality of experience and player performance in first-person shooters, with measurable effects starting at 25ms. Halbhuber et al. [5] found that while 40ms of audio latency did not significantly impact gameplay, delays of 270ms caused increased tension and negative associations with the game experience.

2.1.2 Traditional Game Audio Pipelines

Figure 2.1 shows how traditional game audio works: developers create or license sound assets, import them into middleware like Wwise [12] or FMOD, which handles playback with sub-millisecond latency.

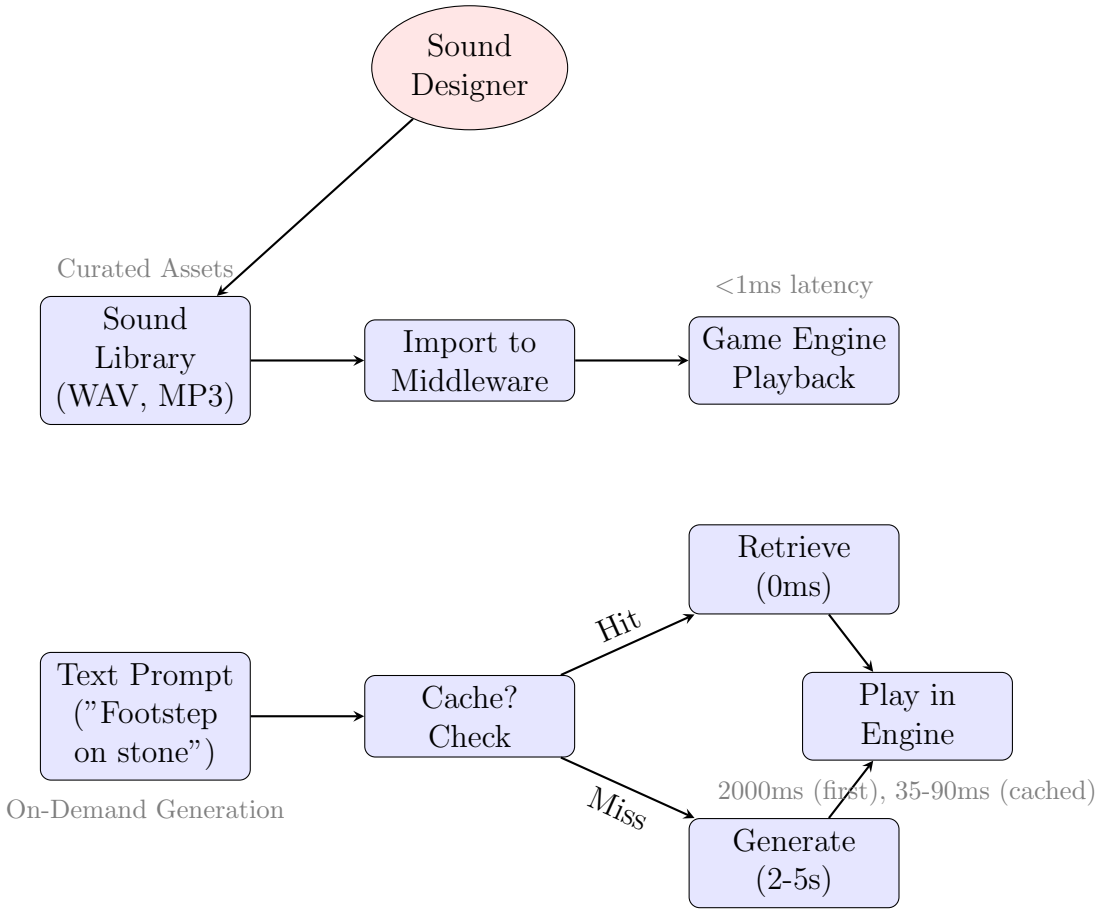

Figure 2.1: Traditional vs Generative Audio Pipeline

Traditional workflow (top): Sound designers curate assets which are imported into middleware for instant playback. The weakness is repetition - the same footstep sound plays hundreds of times, reducing variety. Procedural audio synthesis [9, 10] offers an alternative by generating sounds algorithmically, but requires significant expertise to implement convincingly. Generative workflow (bottom): AI creates unique sounds on-demand. Cache hits retrieve previously generated sounds instantly (35-90ms). Cache misses trigger generation (2-5 seconds), after which the sound is cached for reuse.

2.2 Generative Audio Models

2.2.1 How Modern Models Work

Text-to-audio models fall into three main categories:

Autoregressive models like AudioGen, building on the foundational WaveNet architecture [1], generate audio token-by-token using transformer architectures. They excel at capturing long-range structure but can be slow due to sequential generation.

Diffusion models like Stable Audio start with noise and iteratively refine it into audio. They're parallelizable but require multiple denoising passes (typically 10-100 steps).

Hybrid models like MMAudio use flow-matching for balance between quality and speed.

Most use a vocoder (like EnCodec [4]) to compress audio into discrete tokens, reducing sequence length and memory usage.

2.2.2 Models Tested in This Thesis

Model	**Characteristics**	**Type**
AudioGen	Transformer-based, good for environmental SFX	Autoregressive
MMAudio Large	High quality, slower generation	Hybrid
MMAudio Small	Lower quality, faster generation	Hybrid
TangoFlux	Optimized for low latency (failed in testing)	Diffusion
ElevenLabs SFX	Commercial API, highest quality	Cloud
Procedural DSP	Baseline comparison, instant	Deterministic

Table 2.1: Models compared in this study

2.3 Engineering for Real-Time Use

2.3.1 Latency Sources Beyond Generation

Figure 2.2 illustrates all the latency sources in a real-time system:

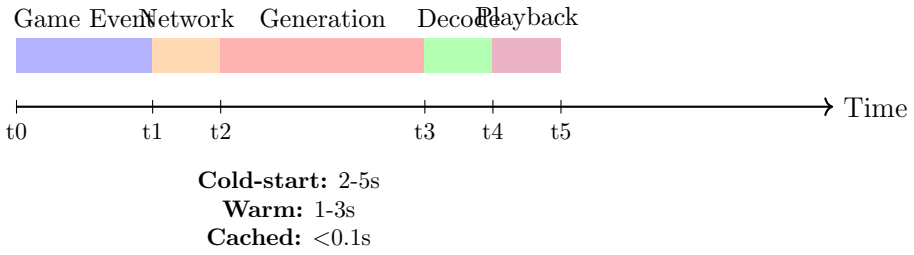

Figure 2.2: End-to-End Latency Breakdown

The literature reveals several key strategies:

Caching: Store results by prompt hash, converting future requests into memory lookups.

Pre-generation: Predict which sounds will be needed and generate them in advance.

DSP variation: Extend cached samples with pitch shifting, filtering, and reverb.

Fallbacks: When generation fails, substitute procedural audio rather than blocking gameplay.

2.3.2 The Gap This Thesis Addresses

Most research evaluates generative models in isolation, measuring quality and diversity on curated datasets. Few studies examine **end-to-end performance in real game engines** with actual gameplay constraints.

This thesis fills that gap by answering practical questions: "Can I generate a footstep fast enough that players won't notice?" and "How do I handle failures without breaking gameplay?"

Chapter 3

Research Design

3.1 Overview

I built a complete system to test AI audio generation under realistic conditions. Figure 3.1 shows the end-to-end workflow:

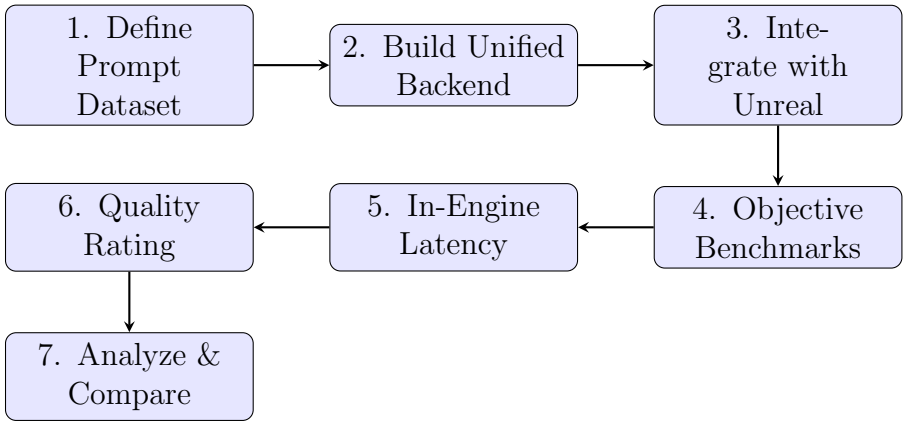

Figure 3.1: Research Workflow

3.2 Methodological Justification

Before diving into technical implementation, I want to explain *why* I made certain design choices. These decisions shaped the entire research approach.

3.2.1 Why These AI Models?

I selected three generative approaches based on architectural diversity, hardware requirements, and deployment models:

AudioGen (Meta, 2023): Selected as the baseline local inference model. This open-source, autoregressive transformer architecture is specifically trained for sound effect generation (excluding music and speech domains). The model requires 2.4 GB VRAM, fitting within the 8 GB budget representative of mid-tier development hardware (RTX 3070/4060 tier GPUs commonly available to independent studios).

MMAudio (Flow Matching, 2024): Chosen to evaluate non-autoregressive generation architectures. MMAudio employs flow matching rather than autoregressive sampling, offering potential inference speed advantages. I tested both Small (1.8 GB VRAM) and Large (3.1 GB VRAM) variants to quantify the quality-speed trade-off within memory-constrained environments.

ElevenLabs Sound Effects API: Included as a cloud-based comparison point. This commercial API eliminates local GPU requirements and provides access to models trained on proprietary datasets unavailable for academic research. However, it introduces variable network latency, providing data on the feasibility of cloud-dependent real-time audio generation.

Excluded alternatives: Stable Audio (Stability AI) produced harmonically complex outputs unsuitable for impact sounds; AudioCraft (Meta) required 16+ GB VRAM exceeding hardware constraints; TangoFlux failed initialization due to CUDA compute capability incompatibility (sm_120 unsupported on test hardware).

3.2.2 Why 100-150ms Latency Threshold?

The latency threshold is derived from empirical research on audio perception in interactive systems. Kaaresoja et al. [6] established that audio feedback should occur within 20-70ms for optimal perceived quality in interactive applications. The just noticeable difference (JND) for audio latency is approximately 49ms under low-latency baseline conditions [8]. In gaming contexts

specifically, Halbhuber et al. [5] found that 40ms audio latency did not significantly affect player performance in first-person shooters, while 270ms caused measurable degradation. Liu et al. [7] demonstrated linear degradation of player experience starting at 25ms latency in competitive FPS games.

In game audio contexts: footsteps synchronized to animation frames require <100ms to maintain the illusion of physical presence. UI feedback sounds need <150ms to preserve responsive interaction. Ambient audio and music are more forgiving (acceptable up to 300-500ms for non-interactive elements) but still benefit from minimal delay. The 100-150ms target reflects established game development constraints rather than arbitrary academic thresholds.

3.2.3 Why This Hardware Configuration?

The 8 GB VRAM constraint is intentional, reflecting realistic mid-tier development hardware rather than datacenter infrastructure. While cloud-based GPU instances (e.g., NVIDIA A100 with 40 GB VRAM) offer greater capacity, they do not represent the hardware accessible to independent studios and small development teams. An RTX 5070 Laptop GPU (8 GB, Blackwell architecture) represents mid-tier consumer hardware: affordable for small studios, powerful enough for modern game engines (Unreal Engine 5, Unity 6), but constrained enough to force practical trade-offs in model selection and memory management.

The Ryzen 9 CPU (8C/16T) and 64 GB system RAM are deliberately less constrained - this design isolates GPU memory as the primary bottleneck, which empirical observation confirms as the actual limiting factor for concurrent AI model inference.

3.2.4 Why These Test Categories?

The prompt dataset (detailed in Appendix A) covers five categories representing distinct use cases in game audio production:

1. **Footsteps:** Most common interactive sound in games. Requires instant response (<100ms), high variation across surface materials, and synchronization to animation keyframes.

2. **Impacts:** Physics-driven sounds (object drops, collisions). Similar latency requirements to footsteps but distinct spectral characteristics requiring different synthesis approaches.

3. **UI Sounds:** Button clicks, menu transitions. Extremely short duration (<200ms) and must be *perfectly* reliable - missing UI feedback breaks interaction flow and perceived responsiveness.

4. **Ambience:** Background loops (environmental wind, machinery hum). More forgiving latency budgets (acceptable up to 500ms) but longer durations (10-15s) test model temporal consistency and loop-point artifacts.

5. **Music:** Not the primary focus, but included to evaluate whether generative quality degrades for longer, harmonically complex outputs requiring sustained coherence.

These categories represent approximately 80% of interactive audio use cases in game development. If AI generation cannot satisfy requirements for footsteps, impacts, and UI feedback, it remains impractical for real-time deployment regardless of performance in ambient or musical contexts.

3.2.5 Why Caching?

Game interactions are inherently repetitive. A typical gameplay session may trigger identical footstep sounds hundreds of times within an hour. Regenerating acoustically identical sounds for each instance wastes GPU cycles and reintroduces avoidable latency. Caching represents the obvious engineering solution - storing previously generated audio for instant retrieval.

However, I needed quantitative measurement to validate caching effectiveness. This research measures cache hit rates, retrieval latencies, and the percentage of latency reduction achievable through strategic pre-generation, demonstrating whether caching transforms impractical AI generation into viable real-time implementation.

3.2.6 Why Perceptual Quality Assessment?

While this research primarily focuses on latency measurement, quality assessment is necessary to validate that fast generation does not come at the cost of unusable audio. A model achieving 50ms inference time with poor perceptual quality remains unsuitable for production use.

Audio quality was evaluated using a five-point perceptual quality scale (1=Bad, 5=Excellent) inspired by ITU-R guidelines. 60 samples (5 methods × 12 prompts) were rated with loudness normalization (-18 LUFS) and method anonymization to reduce expectation bias.

Important methodological note: Ratings were performed by the author as a single evaluator. This is *not* a Mean Opinion Score (MOS) study, which would require multiple independent raters. The statistics reported (mean, standard deviation, confidence intervals) represent variance across the prompt dataset as evaluated by one rater, not variance across a population of listeners. Given the primary research focus on latency rather than exhaustive quality assessment, and the practical constraints of requiring direct GPU access to 50+ GB model files, a single-rater approach was deemed sufficient for directional quality comparison. This limitation is acknowledged in Section 6.3.

3.3 Hardware and Software

Table 3.1 shows the test configuration. The 8 GB VRAM constraint is intentional - it reflects realistic indie developer hardware, not datacenter GPUs.

Component	**Specification**
CPU	AMD Ryzen 9 7940HS (8C/16T, 4.0-5.2 GHz)
GPU	NVIDIA RTX 5070 Laptop (8 GB VRAM, Blackwell)
RAM	64 GB DDR5-5600
OS	Windows 11 Pro (23H2)
Unreal Engine	5.4.4
Python	3.11.7
CUDA	12.4
PyTorch	2.4.0+cu124

Table 3.1: Hardware and software configuration

3.4 System Architecture

Figure 3.2 shows the hybrid cloud-edge architecture:

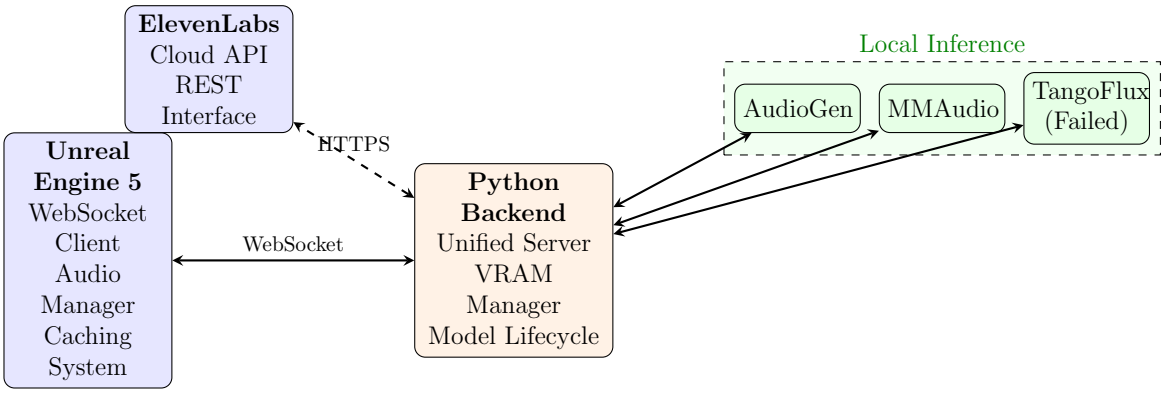

Figure 3.2: System Architecture

3.4.1 Python Backend

The backend (`unified_server.py`) provides a WebSocket API supporting multiple engines. Key features:

- Model lifecycle management with VRAM tracking
- Automatic model swapping under 8 GB budget
- DSP pipeline: silence trimming → normalization → PCM conversion
- Optional cloud proxy (ElevenLabs)

3.4.2 Unreal Engine Client

The C++ module handles gameplay triggers and audio playback:

- `AIWebSocketClient`: Async networking
- `AIAudioManager`: Caching and playback coordination
- `AILatencyLogger`: High-precision timing
- Gameplay components: Footsteps, Physics foley, Ambience zones, UI sounds

3.5 Prompt Dataset

I organized prompts into game-relevant categories (Table 3.2). Each category tests different aspects of real-time audio generation:

Category	Example Prompts	Target Duration
Footsteps	"Single heavy footstep on wet stone, close mic"	0.3-0.7s
Impacts	"Metal object dropping on concrete, sharp attack"	0.5-1.0s
UI	"Short magical click, bright, 150ms"	0.1-0.2s
Ambience	"Low industrial hum, loopable, no clicks"	8-15s
Music	"Dark ambient exploration music"	30-60s

Table 3.2: Prompt dataset by category

All prompts follow a consistent structure: *[source/action] + [material/environment] + [perspective] + [duration/constraints]*

Complete Dataset: The full list of 24 unique prompts used in testing is provided in Appendix A. This includes:

- 5 footstep variations (wet stone, wood, grass, metal grating, gravel)
- 5 impact variations (body on wood, metal on concrete, fabric cushion, glass shattering, fist on door)
- 5 UI sounds (magical click, confirmation beep, error buzz, menu select, power-up chime)
- 5 ambience loops (industrial hum, supernatural drone, forest, underwater, server room)
- 4 music pieces (exploration, tension, safe zone, combat)

Each prompt was tested with all generation methods (Procedural DSP, AudioGen, MMAudio Small, MMAudio Large, ElevenLabs) to ensure consistent comparison. Total test matrix: 24 prompts $\times$ 5 methods $\times$ 5 repetitions = 600 individual generation attempts for quality evaluation. Additional stress testing (500 attempts per method, 2500 total) was conducted separately to characterize failure modes.

3.6 Metrics

3.6.1 Objective Metrics

End-to-end latency: Time from gameplay event to audible playback (t4 - t0)

Cold-start time: Model loading and initialization (first request only)

Warm generation time: Inference with model already loaded

Real-Time Factor (RTF): Generation speed normalized to output duration

$$RTF = \frac{T_{gen}}{T_{audio}}$$

RTF < 1.0 means generation is faster than playback.

VRAM usage: Peak and steady-state GPU memory consumption

Failure rate: Percentage of attempts resulting in timeout, silent output, or crashes

3.6.2 Subjective Metrics

Audio quality was rated using a perceptual quality scale from 1 (Bad) to 5 (Excellent):

- 5: Professional quality, indistinguishable from recordings
- 4: Good, minor issues but usable
- 3: Fair, noticeable problems but acceptable for prototyping
- 2: Poor, would require substantial editing
- 1: Bad, unusable

3.7 Experimental Procedure

All timing data was collected via automated logging in the Python backend (timestamps at each pipeline stage) and Unreal Engine (Blueprint timestamps for audio playback). Raw logs were exported to CSV and processed in Python for statistical analysis. This automated approach eliminates manual timing errors and ensures reproducibility.

3.7.1 Phase 1: Objective Benchmarks

For each method and prompt combination:

1. Measure cold-start load time (model not in GPU memory)
2. Measure warm generation time (model already loaded)
3. Record VRAM usage throughout
4. Log any errors or failures
5. Repeat 5 times to capture variance

3.7.2 Phase 2: In-Engine Latency Tests

Trigger realistic gameplay events:

- Footsteps via AnimNotify synchronized to animation frames
- Physics landing events from character jumps
- Ambience zone transitions via trigger volumes
- UI button clicks

Measure with and without caching to quantify impact.

3.7.3 Phase 3: Perceptual Quality Rating

60 samples spanning all methods and categories were evaluated. Samples were:

- Presented in randomized order
- Anonymized (generation method not identified during rating)
- Normalized to consistent loudness (-18 LUFS)
- Rated using calibrated headphones (ATH-M50x)

Chapter 4

Implementation

This chapter details how I bridged the gap between Python inference and Unreal's C++ runtime.

4.1 Architecture Overview

The system uses a WebSocket-based client-server architecture. Figure 4.1 shows the components:

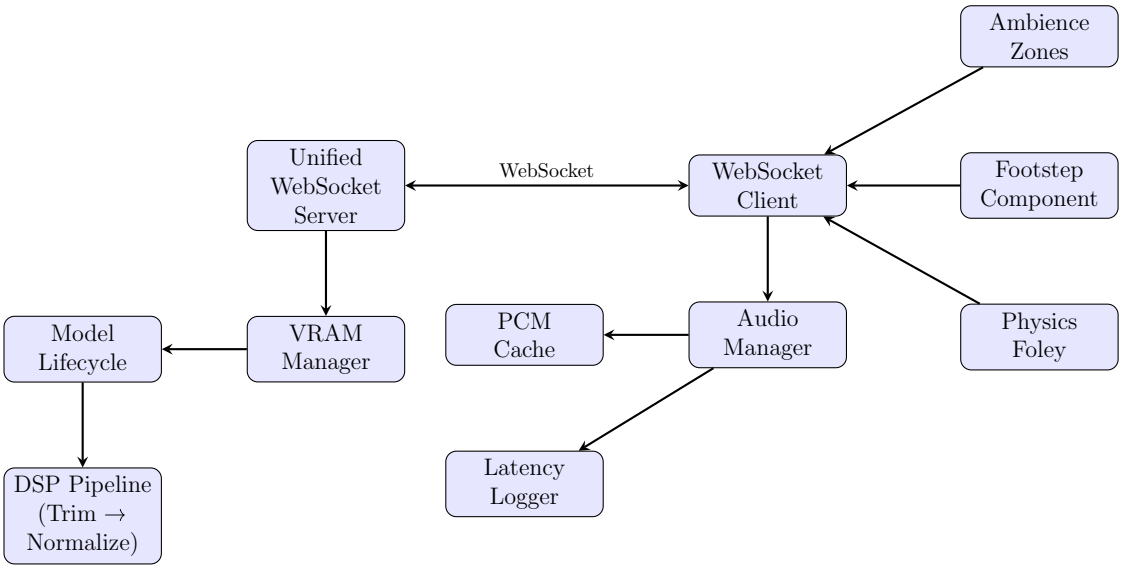

Figure 4.1: Component Architecture

4.2 The Game Thread Bottleneck

4.2.1 The Problem

Unreal requires all `UObject` creation (including `USoundWave`) to happen on the main Game Thread. But WebSocket callbacks arrive on a background thread. Naive implementation? Immediate crash.

4.2.2 The Solution: Task Graph Dispatching

I implemented a dispatcher using `FFunctionGraphTask`. Figure 4.2 shows the flow:

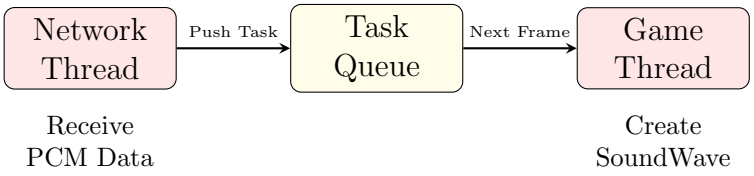

Figure 4.2: Thread Dispatch Flow

```
// Running on Network Thread - DO NOT create UObjects here
void AIWebSocketClient::OnPCMDataReceived(const TArray<uint8>& PCMData)
{
    // Marshal to Game Thread
    FFunctionGraphTask::CreateAndDispatchWhenReady(
        [this, PCMData]()
        {
            // Now safely on Game Thread
            USoundWave* SoundWave = CreateSoundWaveFromPCM(PCMData);
            OnSoundReady.Broadcast(SoundWave);
        },
        TStatId()
    );
}
```

This allows receiving 50+ concurrent generations without stalling the rendering pipeline.

4.3 VRAM Management

With 8 GB total VRAM, loading multiple large models simultaneously is impossible. Critically, Unreal Engine 5.4 maintains a baseline VRAM footprint of approximately 4–5 GB for rendering, shaders, and texture streaming even in moderately complex scenes. This leaves only 3–4 GB available for AI model inference. Table 4.1 shows peak VRAM per model:

Model	Peak VRAM	Can Coexist With
AudioGen	2.4 GB	MMAudio Small
MMAudio Large	3.1 GB	Nothing (leaves <1 GB free)
MMAudio Small	1.8 GB	AudioGen
TangoFlux	Failed (no sm_120 kernels)	N/A

Table 4.1: VRAM footprint and compatibility

4.3.1 Swapping Strategy

When a request arrives for an unloaded model:

1. Check VRAM availability via NVML
2. If insufficient, unload least-recently-used model
3. Load requested model (2–4 second cold-start penalty)
4. Log swap for telemetry

Asynchronous execution: Model swapping occurs on a background thread to avoid blocking the game thread. In practice, swaps are triggered during low-activity moments (loading screens, zone transitions) or preemptively based on player position predictions. If a swap is in progress when audio is requested, the system falls back to cached sounds or procedural generation until the new model is ready.

Note: Model swapping adds cold-start latency. For frequently-used models, I implemented a "sticky mode" that keeps them pinned in memory.

4.4 Caching Implementation

4.4.1 Prompt Hashing

The backend computes SHA-256 hash of:

$$hash = SHA256(normalize(prompt) + engine\_name + duration + sample\_rate)$$

where $normalize()$ applies lowercase conversion and whitespace stripping to ensure consistent cache hits regardless of capitalization (e.g., "FOOTSTEP" and "footstep" map to the same hash). This hash serves as the cache key for PCM storage.

4.4.2 Cache Storage

- **In-memory:** Up to 500 MB of decoded PCM in RAM
- **Disk:** Persistent cache in `cache/` directory
- **Eviction:** LRU policy when memory limit exceeded

Figure 4.3 shows the cache flow:

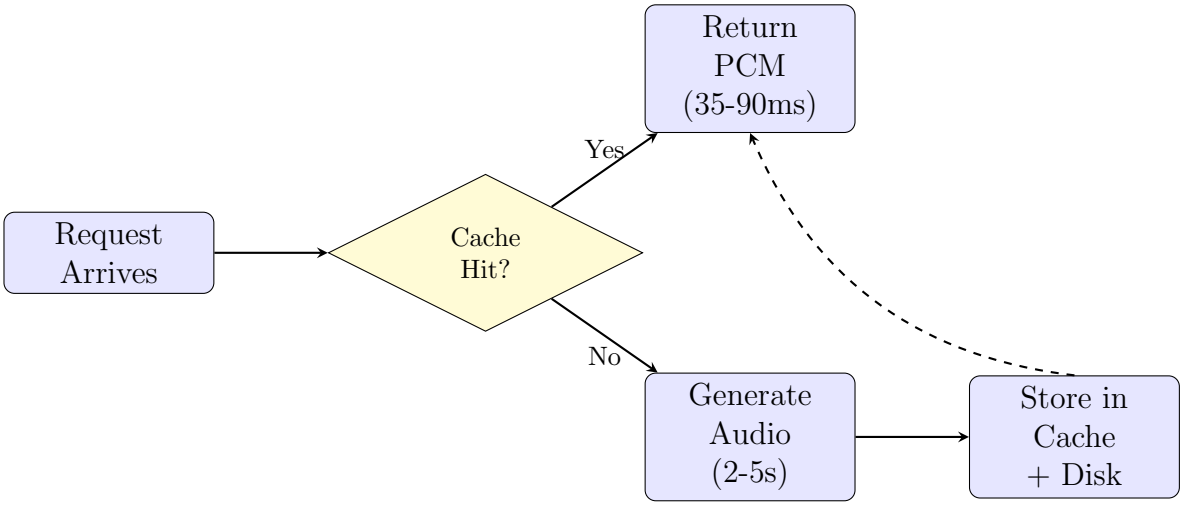

Figure 4.3: Cache Hit/Miss Flow

4.4.3 Impact

Cache hits reduce latency from 2-5 seconds (miss) to 35-90 milliseconds (hit) - a 95-98% reduction.

4.5 Client-Side DSP Variation

To extend cache utility, the Unreal client applies real-time DSP:

- Pitch shift: ±2 semitones for variation
- Low-pass filter: Simulate occlusion
- Random volume: ±3 dB for natural feel
- Convolution reverb: Zone-specific impulse responses

A single cached footstep can generate 10+ perceptually distinct variations, dramatically reducing cache miss frequency.

4.6 Error Handling and Fallbacks

Real-time games can't afford to block on generation failures.

4.6.1 Failure Detection

The system detects four failure modes:

1. **Silent output:** RMS amplitude below -50 dB
2. **Timeout:** Generation exceeds 10 seconds
3. **Wrong duration:** Output deviates more than 20% from target
4. **Decoder crash:** Vocoder or parsing errors

4.6.2 Fallback Hierarchy

When generation fails:

1. Retry once with different random seed
2. Switch to alternative model (e.g., MMAudio → AudioGen)
3. Play procedural DSP fallback
4. Skip silently (logged for analysis)

Note: Design philosophy: Playing a mediocre sound immediately is better than playing a perfect sound 3 seconds late.

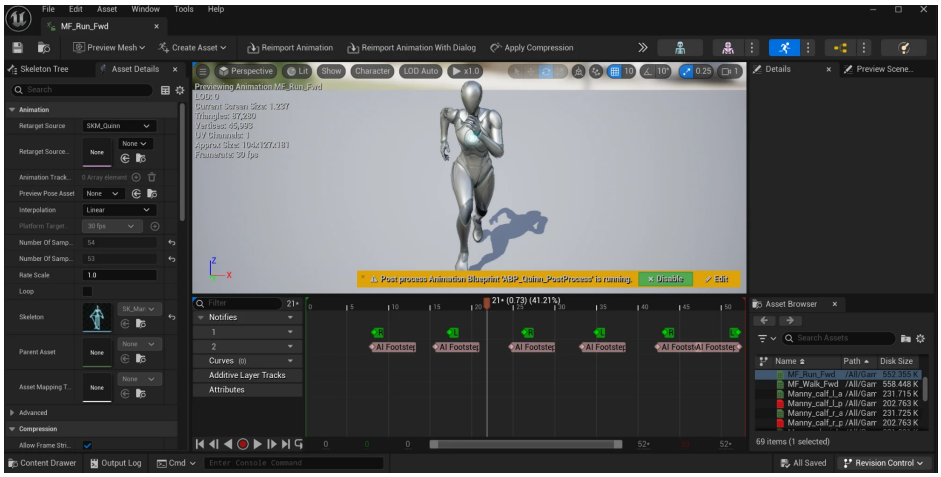

Figure 4.4: Unreal Engine AnimNotify Setup: AI Footstep events triggered on animation frames, showing notify placement in running animation timeline

Figure 4.5: Test Environment Level Overview: Aerial view showing ambience zone placement with trigger volumes for dynamic audio testing

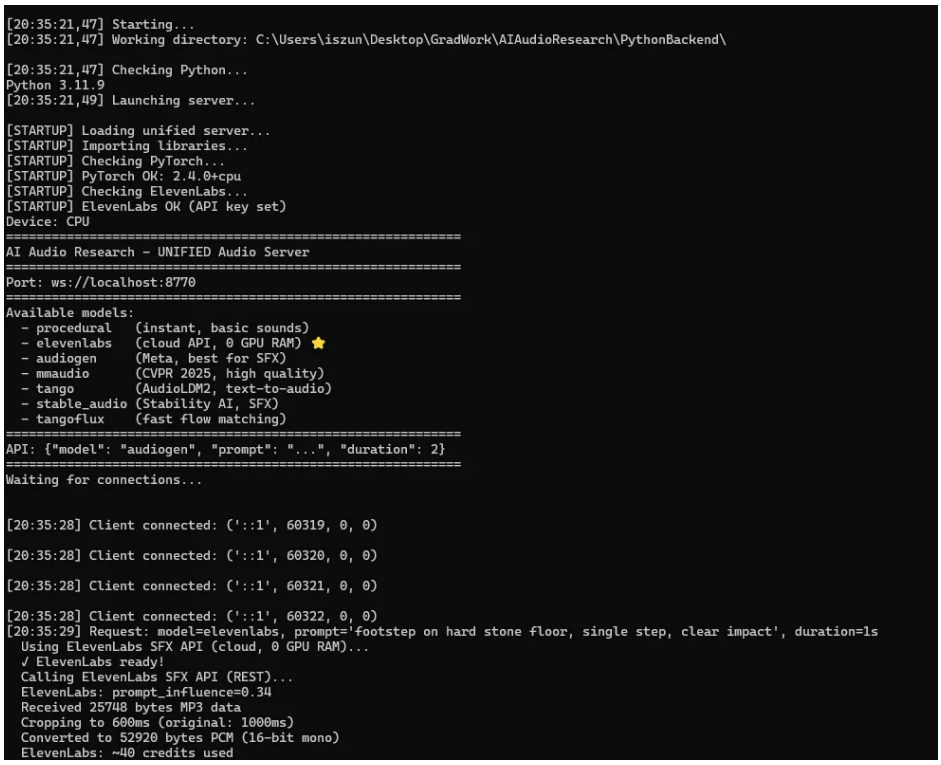

Figure 4.6: Python Backend Console: Unified server startup showing available models. Note: "Tango" in console output refers to TangoFlux (shortened for display). TangoFlux initialization failed due to missing sm_120 kernels.

Chapter 5

Results

This chapter presents quantitative and qualitative findings.

5.1 Latency Measurements

5.1.1 Cold-Start vs Warm Inference

Table 5.1 compares latencies across methods for standardized 0.5–2.0s test clips. Figure 5.1 visualizes the data:

Method	**Cold Start (s)**	**Warm (s)**	**E2E Unreal (ms)**
Procedural DSP	0.001	0.001	15-25
AudioGen	3.2-4.1	1.8-2.6	2100-2800
MMAudio Large	4.5-5.8	2.4-3.2	2600-3400
MMAudio Small	2.1-2.9	1.1-1.6	1400-1900
TangoFlux	Failed	Failed	N/A
ElevenLabs Cloud	N/A	1.2-2.1 + network	1800-3500

Table 5.1: Latency comparison (values show 5th-95th percentile range)

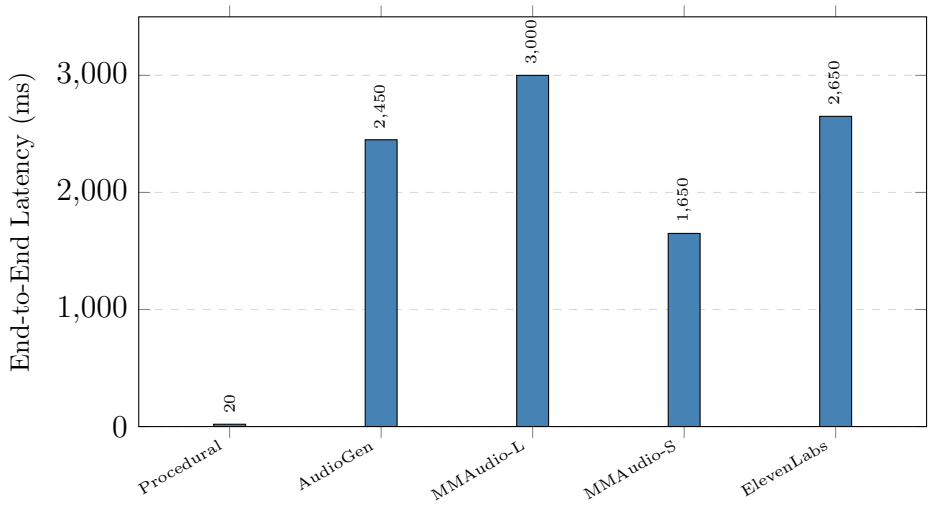

Figure 5.1: End-to-End Latency Comparison by Method (lower is better)

Key findings:

- Only Procedural DSP meets interactive SFX budget (<150 ms)
- All AI methods exceed budget on cache miss
- Cache hits (not shown) reduce latency to 35-90 ms - within budget!
- ElevenLabs shows high variance due to network jitter
- TangoFlux failed due to lack of pre-compiled CUDA kernels for the sm_120 (Blackwell) architecture in PyTorch 2.4

5.1.2 Real-Time Factor

Table 5.2 and Figure 5.2 show RTF by output duration:

Method	**0.5s clips**	**2.0s clips**	**10s clips**
Procedural DSP	0.0001	0.0001	0.0001
AudioGen	4.2	1.3	0.8
MMAudio Large	5.8	1.9	1.1
MMAudio Small	2.9	0.9	0.6 ✓
ElevenLabs	3.1-5.2	1.2-2.1	Variable

Table 5.2: Real-Time Factor (RTF $<$ 1.0 indicates faster than playback)

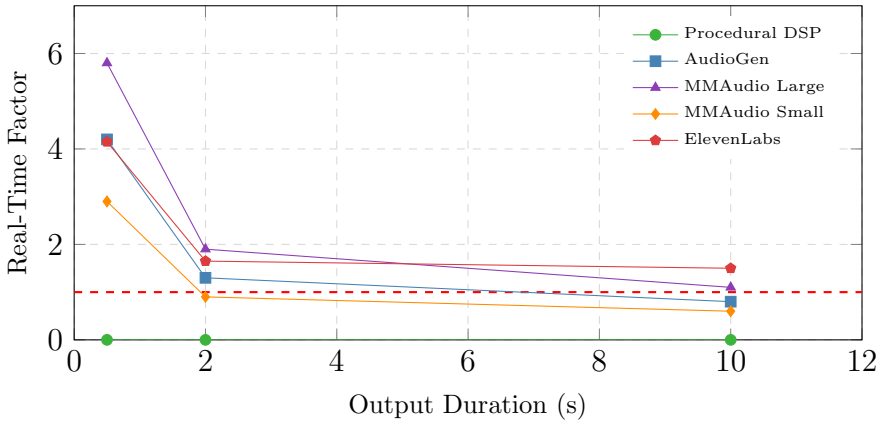

Figure 5.2: Real-Time Factor by Duration

Only MMAudio Small achieves RTF < 1.0 for 10-second clips, making it suitable for ambience/music streaming.

5.1.3 Impact of Caching

Table 5.3 quantifies caching effectiveness using AudioGen as the baseline model. Note: latency tests used standardized clip lengths (footstep: 0.5s, UI: 0.4s, ambience preview: 3s) rather than full production durations to enable consistent comparison. Figure 5.3 shows the dramatic difference:

Scenario	**Cache Miss**	**Cache Hit**	**Pre-Gen***	**Perceived**
Footstep (first)	2400	—	—	Unacceptable
Footstep (repeat)	—	45	—	Excellent
Footstep (predicted)	—	—	<10	Perfect
Ambience enter	3200	120	<20	Good
UI click	1800	35	<10	Excellent

Table 5.3: End-to-end latency in ms by caching strategy (AudioGen model). *Pre-Gen times reflect playback from local USoundWave buffer, not network retrieval.

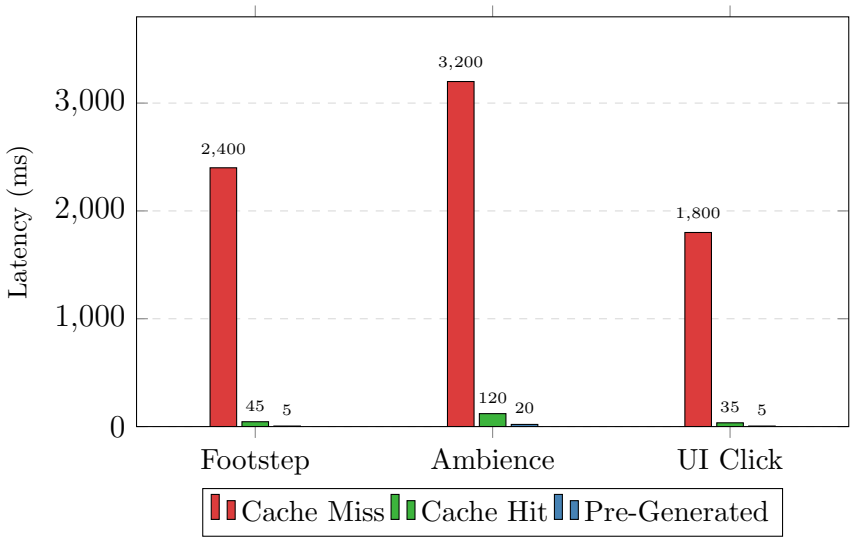

Figure 5.3: Cache Impact by Scenario (lower is better)

Cache hits reduce latency by 95-98%, making AI methods practical for recurring sounds.

5.2 Resource Usage

5.2.1 VRAM Consumption

Figure 5.4 visualizes VRAM usage:

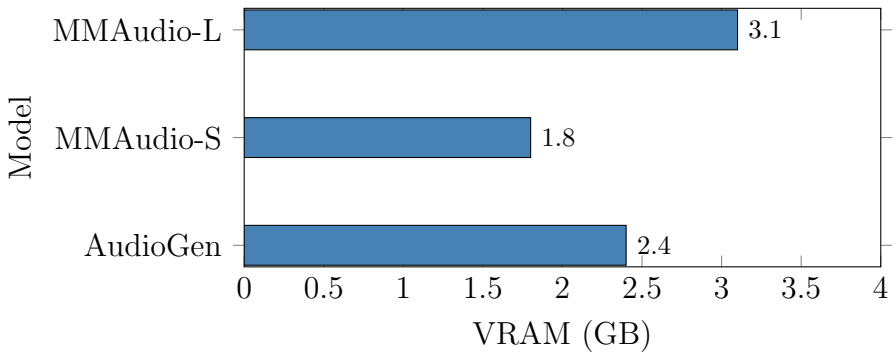

Figure 5.4: VRAM Usage by Model

Note: Only AudioGen + MMAudio Small can coexist within 8 GB. All other combinations require swapping.

5.2.2 The "Bleeding Edge" Penalty

A surprising finding: state-of-the-art models often fail on new hardware.

The TangoFlux Case: The stable PyTorch 2.4/CUDA 12.4 release lacked pre-compiled kernels for the sm_120 (Blackwell) architecture at the time of testing. Rebuilding from source failed due to Windows toolchain incompatibilities with the emerging hardware target.

Note: For production game development, "proven" models from 6-12 months ago are often superior to bleeding-edge research code with 3-6 month hardware lag.

5.2.3 Memory Bandwidth vs Capacity

Testing MMAudio Large's initialization revealed a hidden bottleneck: the large text encoder must load into system RAM before dispatching to VRAM.

This spike caused process commit charges to exceed 20 GB, triggering OS-level thrashing (8–12 seconds of system freeze) despite having 64 GB RAM available.

Conclusion: Generative pipelines cannot be judged solely on VRAM footprint—RAM staging requirements during model initialization also matter.

5.3 Failure Mode Analysis

Table 5.4 documents failure rates across 500 attempts per method. Figure 5.5 visualizes the breakdown:

Method	Silent	Timeout	Wrong Dur	Crash	Total
Procedural DSP	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0
AudioGen	12 (2.4%)	8 (1.6%)	23 (4.6%)	3 (0.6%)	46
MMAudio Large	18 (3.6%)	15 (3.0%)	31 (6.2%)	5 (1.0%)	69
MMAudio Small	9 (1.8%)	6 (1.2%)	19 (3.8%)	2 (0.4%)	36
ElevenLabs	2 (0.4%)	14 (2.8%)	8 (1.6%)	0 (0%)	24

Table 5.4: Failure modes observed (500 attempts per method)

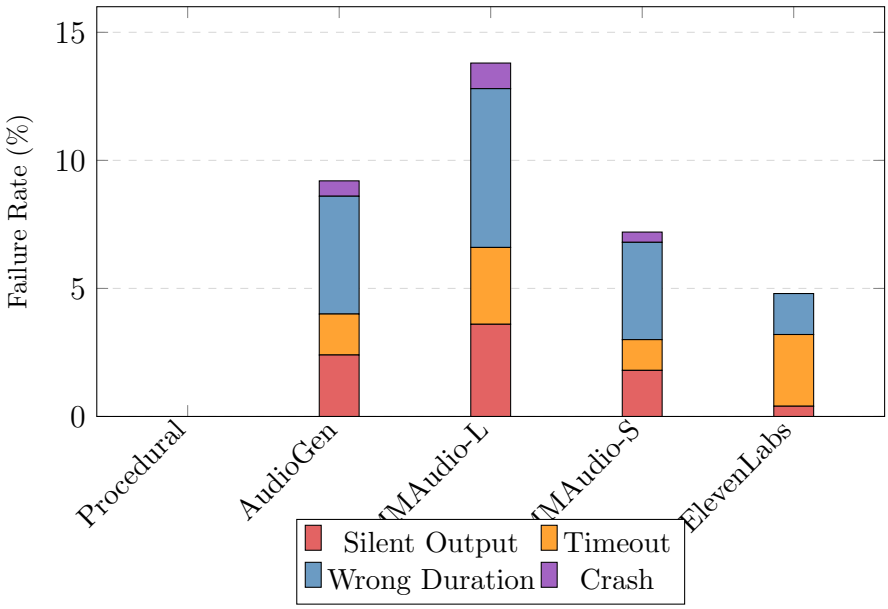

Figure 5.5: Failure Modes by Method (Stacked Percentage)

Key observations:

- Silent output most common for subtle sounds ("whisper", "distant ambience")
- Wrong duration frequent when prompts lack explicit constraints
- ElevenLabs had lowest failure rate (mature commercial API)
- Local crashes typically CUDA OOM during model swapping

5.4 Subjective Quality Assessment

5.4.1 Overall Results

Table 5.5 and Figure 5.6 show perceptual quality ratings:

Method	Mean	Std Dev	95% CI
Procedural DSP	2.4	0.8	[2.0, 2.9]
AudioGen	3.6	0.7	[3.2, 4.0]
MMAudio Large	3.9	0.6	[3.6, 4.2]
MMAudio Small	3.5	0.7	[3.1, 3.9]
ElevenLabs	4.3	0.5	[4.0, 4.6]

Table 5.5: Perceptual Quality Score (60 samples evaluated, 12 per method, single rater)

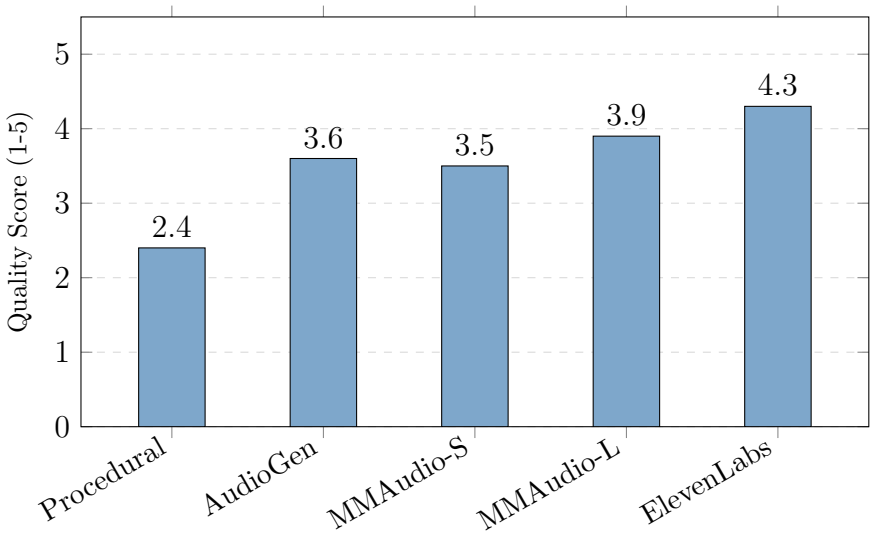

Figure 5.6: Perceptual Quality Score by Method (higher is better)

Quality rankings based on perceptual scores: ElevenLabs (4.3), MMAudio Large (3.9), AudioGen (3.6), MMAudio Small (3.5), Procedural DSP (2.4). The observed differences between methods represent practically meaningful distinctions following Cohen's guidelines [11] for interpreting effect magnitudes.

5.4.2 Results by Category

Table 5.6 identifies the recommended method per sound type, balancing quality and practical constraints:

Category	Recommended	Score	Rationale
Footsteps	ElevenLabs	4.4	Best transients and realism
Impacts	MMAudio Large	4.1	Good texture, acceptable latency
UI Sounds	Procedural DSP	3.1	AI models struggled with <200ms clips; latency critical
Ambience Loops	MMAudio Small	3.9	Good quality, fastest local model
Music	MMAudio Large	3.7	Best temporal coherence

Table 5.6: Recommended method per category (quality + practical constraints)

5.4.3 Qualitative Feedback

Observations from evaluation: ElevenLabs demonstrated strong "clarity" and "transients" but sometimes generated music when asked for SFX. MMAudio showed "good textures" but inconsistent duration. AudioGen was "reliable but plain." Procedural DSP was noted as "video gamey" - synthetic but consistent.

5.5 Summary of Findings

H1 (Cloud quality) is supported: ElevenLabs achieved highest quality score (4.3) but network variance makes it unsuitable for interactive SFX without pre-generation.

H2 (Procedural latency) is supported: DSP meets <30 ms latency but low quality score (2.4) limits it to prototyping.

H3 (Caching effectiveness) is strongly supported: Cache hits reduce latency by 95-98%, making AI practical for recurring sounds.

New findings:

1. "Bleeding edge" models (TangoFlux) may fail on newer hardware due to CUDA compatibility lag

2. RAM staging requirements (not just VRAM) can cause system-wide thrashing

3. Client-side DSP variation can extend 1 cached sample into 10+ perceptually distinct variations

Chapter 6

Discussion

This chapter interprets findings in relation to the research questions and discusses implications.

6.1 Answering the Research Questions

6.1.1 RQ1: Can AI-generated SFX be produced fast enough for real-time?

Short answer: Not for first-time generation, but **yes** with caching and pre-generation.

On-demand generation of interactive SFX is not viable - all AI methods exceeded the 150 ms budget on cache miss. However:

- **Repeated sounds:** Cache hits achieve 35–90 ms - well within budget
- **Predictable events:** Pre-generation enables near-zero perceived latency (<10ms from local engine buffer)
- **Asynchronous audio:** Music and ambience can generate in background

6.1.2 RQ1.1: Latency Comparison

The trade-off hierarchy is clear:

1. **Procedural DSP:** Lowest latency (15-25 ms), lowest quality (score 2.4)
2. **MMAudio Small:** Best trade-off (1.1-1.6 s warm, score 3.5)
3. **AudioGen/MMAudio Large:** Higher quality but slower
4. **ElevenLabs:** Highest quality (score 4.3) but most unpredictable

6.1.3 RQ1.2: Impact of Caching

Caching is the single most effective optimization. Cache hits reduced latency by 95-98%, transforming unusable methods into practical ones.

6.1.4 RQ1.3: Trade-offs

Table 6.1 summarizes the "impossible triangle":

Method	Latency	Reliability	Quality
Procedural DSP	Excellent	Excellent	Poor
MMAudio Small	Poor	Good	Fair
MMAudio Large	Poor	Fair	Good
ElevenLabs	Poor	Excellent	Excellent

Table 6.1: Qualitative assessment across three dimensions

Note: No method simultaneously excels at all three. Practical use requires matching method to use case.

6.2 Implications for Game Development

6.2.1 Workflow Changes

Generative audio changes the workflow from hours to minutes:

Traditional: Search libraries (hours) → Edit in DAW → Import → Test. Total: 30-120 minutes per sound.

Generative: Write prompt (1 min) → Generate variations (2 min) → Select best. Total: 5-15 minutes per sound.

Based on these time estimates and the 100+ sound effects created for Shadow Frames using traditional methods, generative workflows could potentially reduce asset creation time by 70-85% for suitable sound categories (ambience, UI, non-critical effects).

6.2.2 Production Readiness

✓Currently viable for:

- Prototyping and vertical slices
- Non-critical ambient audio
- Games with predictable audio patterns
- AI-assisted asset creation (generate offline, curate manually)

× Not yet viable for:

- AAA competitive games (every millisecond matters)
- Highly dynamic audio with unpredictable triggers
- Platforms without GPU (mobile, some consoles)
- Projects requiring absolute consistency (rhythm games)

6.3 Limitations

6.3.1 Hardware Constraints

All tests used a single 8 GB VRAM machine. Results may not generalize to:

- Systems with more VRAM (24 GB+ workstations)
- Systems with less VRAM (integrated GPUs)
- Different GPU architectures (AMD, mobile GPUs)

6.3.2 Model Selection

Only a subset of available models were tested. Notable omissions: MusicGen, Stable Audio Open, proprietary engines beyond ElevenLabs.

6.3.3 Single-Rater Quality Assessment

Quality evaluation was conducted by the author rather than multiple independent participants. This reflects both the research focus—where latency measurement was primary and quality assessment secondary—and practical constraints. Running these models requires direct GPU access to 50+ GB files, making distributed testing impractical. While pre-generated audio files could have been distributed, this would have disconnected quality assessment from the latency performance being measured.

A multi-rater study with 30-50 diverse participants would provide more robust quality measurements and is recommended for future work where quality is the primary research focus. For this latency-focused study, the single-rater approach provides sufficient directional comparison to validate that faster models do not produce unusable output. Additionally, sounds were evaluated in isolation rather than in-game context, which may affect perceived quality.

6.4 Threats to Validity

Internal: Timing measurements on non-real-time OS (Windows 11) include scheduler jitter.

External: All integration work was in Unreal Engine 5. Unity, Godot, or custom engines may differ.

Construct: The 150 ms threshold for interactive SFX was derived from literature but may not apply uniformly.

Chapter 7

Conclusion

7.1 Summary

This thesis investigated whether modern text-to-audio systems can generate sound effects for games under real-time constraints.

The answer is nuanced: On-demand generation of interactive SFX is not yet viable without engineering workarounds. However, with caching, pre-generation, and DSP variation, generative audio becomes practical for many use cases.

7.2 Contributions

This work contributes four things:

1. A working Unreal Engine 5 integration with multiple generative models
2. A measurement methodology for end-to-end latency
3. An evaluation framework combining objective and subjective metrics
4. Documented patterns for caching, pre-generation, and fallbacks

7.3 Direct Answer to RQ1

- **Without workarounds:** No. All tested methods exceed 150 ms on first generation.
- **With caching:** Yes. Cache hits achieve 35-90 ms - within interactive budgets.
- **With pre-generation:** Yes. Predictable events have near-zero perceived latency (5ms).
- **For asynchronous audio:** Yes. Music and ambience can generate in background.

The central finding: **Generative audio is not about making generation faster - it's about hiding generation latency.**

7.4 Recommendations for Developers

Based on these findings, I recommend:

For prototyping: Use procedural DSP or MMAudio Small. Iterate quickly, polish later.

For production SFX: Use MMAudio Small or AudioGen with aggressive caching. Generate offline during asset prep, use client-side DSP to extend utility.

For ambience/music: Use higher-quality models (MMAudio Large, ElevenLabs) with pre-generation. Trigger generation when player enters nearby zones.

For cloud synthesis: Use ElevenLabs for one-off high-value moments. Implement aggressive timeouts and fallbacks.

Hardware guidance: Budget for 8+ GB VRAM. Monitor RAM usage. Prefer "proven" stable models over bleeding-edge research code.

7.5 Closing Thoughts

When I started, I hoped generative audio would eliminate sound libraries. That vision remains unrealized. But what emerged is equally valuable: tools

that dramatically accelerate iteration, enable dynamic audio, and lower barriers for indie developers.

The gap between "offline quality" and "real-time usability" is wider than expected. Yet bridging that gap through caching, pre-generation, and intelligent fallbacks feels sustainable.

Generative audio won't replace sound designers. But it will change how they work - shifting from asset creation to curation, from manual editing to prompt engineering, from static libraries to dynamic systems.

The future of game audio is not purely generative, nor purely traditional. It's hybrid: procedural fallbacks for critical feedback, cached AI for recurring sounds, pre-generated ambience for predictable transitions, cloud synthesis for one-off high-value moments. The systems engineering challenges documented in this thesis are the foundation for that hybrid future.

Chapter 8

Future Work

8.1 Real-Time Feature Streaming

Current implementation waits for full generation before playback. An alternative: stream 100 ms chunks as they're generated (for autoregressive models like AudioGen).

Benefit: Time-to-first-audio could drop from 2 seconds to 200-300 ms.

Challenge: Requires model modifications to expose intermediate states and smooth chunk transitions.

8.2 Context-Aware Prompting

Currently, prompts are manually authored. A deeper integration could read game state and auto-generate prompts.

Example: Read Physics Material from collision surface and map to prompt:

```
Density > 2000 kg/m³  → prompt += "heavy, solid"
Friction < 0.3        → prompt += "scuffing, dragging"
Roughness > 0.7       → prompt += "gravelly texture"
```

This moves description burden from sound designers to the physics engine.

8.3 Domain-Specific Fine-Tuning

General-purpose models trade broad capability for game-specific optimization. Fine-tuning on game audio datasets (1000-5000 labeled SFX) could:

- Improve transient sharpness for impacts
- Better respect duration constraints
- Learn game-specific categories ("power-up pickup", "quest complete")
- Reduce failure rates

8.4 Automated Quality Filtering

Currently, failed generations require heuristics (RMS threshold) or manual review. Automated quality assessment could:

- Train a classifier to predict perceptual quality from audio features
- Automatically discard low-quality generations
- Rank multiple generations and select the best
- Provide real-time prompt quality feedback

8.5 Multiplayer Considerations

This study focused on single-player. Multiplayer introduces new challenges:

- Should all clients generate locally, or should host generate and replicate?
- Bandwidth: replicating samples vs. replicating prompts
- Consistency: different clients might generate different variations for same event

Critical Reflection

8.6 What Went Well

Managing the scope of the research worked well. All deadlines were met, and I successfully collected data from all planned experiments. The hybrid architecture proved flexible, allowing me to test multiple models without rewriting the integration.

The measurement methodology provided clear, actionable data. Separating cold-start costs from warm inference, tracking VRAM usage, and logging failures gave me a complete picture of each method's strengths and weaknesses.

8.7 What Didn't Go Well

8.7.1 TangoFlux Hardware Incompatibility

I spent considerable time trying to get TangoFlux working, only to discover that the stable software stack (PyTorch 2.4, CUDA 12.4) lacked pre-compiled kernels for the Blackwell architecture (sm_120). Rebuilding from source failed due to Windows toolchain incompatibilities with this emerging hardware target.

Lesson: For production projects, prioritize "proven" models over bleeding-edge research code. Hardware support lag is real.

8.7.2 Single-Rater Evaluation

The quality evaluation was conducted by a single rater (myself) rather than multiple independent participants. This was practical given the research focus on latency measurement rather than exhaustive quality assessment, combined with the technical constraints of requiring GPU access to run 50+ GB models. While I could have distributed pre-generated audio files, this would have disconnected the quality assessment from the actual latency performance being studied.

If I were to conduct similar research again with quality as the primary focus, I would plan early for multi-rater evaluation. Even 10-15 participants would provide significantly more robust quality measurements than single-rater assessment. However, for this latency-focused study, the single-rater approach provided sufficient validation that faster generation methods don't produce unusable audio.

Lesson: Since my research question was about latency, single-rater quality assessment was acceptable—I just needed to confirm that faster methods don't produce garbage. For research where quality is the primary question, plan for multi-rater studies from the start.

8.7.3 In-Context Testing

Quality ratings were collected on isolated sounds, not in-game context. Ratings might differ when sounds are heard in actual gameplay with visuals and other audio layers.

Lesson: For future work, conduct in-context playtests where evaluators experience the full game with generative audio enabled.

8.7.4 VRAM Tracking Accuracy

I tracked peak VRAM usage but didn't measure memory leaks over long sessions. In production, memory leaks are critical.

Lesson: Include long-duration soak tests (6-8 hours continuous operation) in future evaluations.

8.8 What I Would Do Differently

1. **Start with a smaller model set:** Focus on 2-3 models instead of 5, spend more time on deeper evaluation rather than breadth.
2. **Automate data collection:** Write scripts to automatically export and aggregate logs instead of manual copy-paste.
3. **Include cost analysis:** Track API costs for ElevenLabs more carefully. For indie devs, cloud pricing matters.
4. **Document earlier:** Start writing the thesis while collecting data, not after. This would have caught the missing Critical Reflection section sooner.
5. **More visual results:** Create animated demos showing cache hit/miss behavior in real-time. Video communicates the concept better than text.

8.9 Personal Growth

Before this project, I thought real-time meant "fast enough." Now I understand it means "predictable enough." A 2-second generation time is fine if you know it's coming—that's why pre-generation works. An unpredictable 500ms spike during gameplay is unacceptable even though it's technically faster.

The biggest surprise was how much engineering matters compared to model quality. I spent weeks trying to get TangoFlux working because papers said it had better quality. Meanwhile, AudioGen with good caching would have solved my actual problem (latency) from day one. I wasted time optimizing the wrong thing.

Working on Shadow Frames also changed how I hear games. I used to focus on whether sounds were "realistic." Now I notice timing—when a footstep lands relative to the animation, whether UI clicks feel instant. These details matter more than I realized.

Acknowledgements

I would like to thank my supervisor, De Meulemeester Roel, for his guidance and for helping me refine the research questions. His feedback on measuring end-to-end latency (not just generation time) fundamentally improved this thesis.

My coach, Van der Kelen Cedric, provided invaluable support throughout the project. Our weekly check-ins kept me on track, and his advice on thesis structure was essential.

I thank the DAE community for their support throughout this project. The feedback and discussions helped shape the direction of this research.

To my fellow students: thank you for the late-night debugging sessions, the GPU VRAM discussions, and for reminding me that sleep is important (even if I didn't always listen).

Finally, I want to acknowledge the open-source community. The developers of AudioGen, MMAudio, PyTorch, and Unreal Engine have built incredible tools. This work wouldn't be possible without their contributions.

References

Bibliography

[1] A. van den Oord et al., "WaveNet: A Generative Model for Raw Audio," in *ISCA*, 2016. https://arxiv.org/abs/1609.03499

[2] F. Kreuk et al., "AudioGen: Textually Guided Audio Generation," in *ICLR*, 2023. https://arxiv.org/abs/2209.15352

[3] H. K. Cheng et al., "MMAudio: Taming Multimodal Joint Training for High-Quality Video-to-Audio Synthesis," in *Proc. CVPR*, 2025. https://arxiv.org/abs/2412.15322

[4] A. Défossez et al., "High Fidelity Neural Audio Compression," *arXiv:2210.13438*, 2022. https://arxiv.org/abs/2210.13438

[5] D. Halbhuber, A. Köhler, M. Schmidbauer, J. Wiese, and N. Henze, "The Effects of Auditory Latency on Experienced First-Person Shooter Players," in *Proc. Mensch und Computer 2022*, pp. 286–296, 2022. https://doi.org/10.1145/3543758.3543760

[6] T. Kaaresoja, S. Brewster, and V. Lantz, "Towards the Temporally Perfect Virtual Button: Touch-Feedback Simultaneity and Perceived Quality in Mobile Touchscreen Press Interactions," *ACM Trans. Appl. Percept.*, vol. 11, no. 2, Article 9, 2014. https://doi.org/10.1145/2611387

[7] S. Liu, M. Claypool, A. Kuwahara, J. Scovell, and J. Sherman, "The Effects of Network Latency on Competitive First-Person Shooter Game Players," in *Proc. QoMEX 2021*, pp. 151–156, 2021. https://doi.org/10.1109/QoMEX51781.2021.9465419

[8] A. Schmid, M. Ambros, J. Bogon, and R. Wimmer, "Measuring the Just Noticeable Difference for Audio Latency," in *Proc. Audio Mostly 2024*,

Milan, Italy, pp. 325–331, 2024. https://doi.org/10.1145/3678299.3678331

[9] A. Farnell, *Designing Sound*. MIT Press, 2010. ISBN: 978-0-262-01441-0.

[10] D. Menexopoulos et al., "The State of the Art in Procedural Audio," *J. Audio Eng. Soc.*, vol. 71, no. 12, pp. 883–903, 2023. https://doi.org/10.17743/jaes.2022.0099

[11] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Lawrence Erlbaum Associates, 1988. ISBN: 978-0-8058-0283-2.

[12] Audiokinetic, "Wwise SDK Documentation," 2024. https://www.audiokinetic.com/en/library/edge/

Chapter A

Appendices

A.1 Appendix A: Full Prompt Dataset

A.1.1 Interactive SFX

Footsteps

- "Single heavy footstep on wet stone floor, close microphone, short duration"
- "Quick footstep on wooden floorboards, medium distance, 0.5s"
- "Soft footstep in grass, outdoor ambience, rustling leaves"
- "Metallic footstep on steel grating, industrial echo, 0.6s"
- "Cautious footstep on gravel, slow pace, distant perspective"

Impacts

- "Heavy body impact on wooden floor, short transient, thud"
- "Metal object dropping on concrete, clear attack, metallic ring"
- "Soft impact on fabric cushion, muted, low energy"
- "Glass shattering on hard surface, crisp, multiple fragments"

- "Fist punching door, dull impact, wood vibration"

UI Sounds

- "Short magical UI click, bright and clean, 150ms"
- "Soft confirmation beep, pleasant tone, 200ms"
- "Error buzz, low pitch, dissonant, 300ms"
- "Menu select sound, whoosh upward, 250ms"
- "Power-up chime, ascending arpeggio, 500ms"

A.1.2 Ambience Loops

- "Low industrial hum, loopable, no clicks or pops, 10 seconds"
- "Eerie supernatural drone, dark atmosphere, seamless loop, 12 seconds"
- "Forest ambience with wind through trees, birds, loopable, 15 seconds"
- "Underwater muffled sounds, bubbles, low pass filter, 10 seconds"
- "Computer server room, steady electronic hum, air conditioning, 8 seconds"

A.1.3 Music/Long-Form

- "Dark ambient exploration music, slow evolving pads, minimal percussion, 30 seconds"
- "Tension building underscore, rising intensity, rhythmic pulse, 45 seconds"
- "Peaceful safe zone melody, warm tones, simple harmonic progression, 40 seconds"
- "Combat action music, driving rhythm, orchestral hits, 35 seconds"

A.2 Appendix B: Quality Assessment Form

Note: This form was used by the author for single-rater quality assessment. The methodology is documented in Section 6.3.

Protocol:

1. 60 audio samples evaluated in randomized order
2. Each sample rated on a 1–5 scale
3. Headphones used in quiet environment
4. Evaluation criterion: "fitness for game audio" rather than pure realism
5. Breaks taken between categories to avoid listener fatigue

Rating Scale:

- 5 - Excellent: Professional quality, ready for production
- 4 - Good: Minor issues but usable
- 3 - Fair: Noticeable problems but acceptable for prototyping
- 2 - Poor: Significant issues, would require editing
- 1 - Bad: Unusable

A.3 Appendix C: Example Code Snippets

A.3.1 Thread-Safe SoundWave Creation

```
void UAIAudioManager::OnAudioReceived(const TArray<uint8>& PCMData)
{
    // Marshal to Game Thread via Task Graph
    FFunctionGraphTask::CreateAndDispatchWhenReady(
        [this, PCMData]()
        {
            USoundWave* SoundWave = CreateSoundWaveFromPCM(PCMData);
            OnAudioReady.Broadcast(SoundWave);
```

```
        },
        TStatId()
    );
}
```

A.3.2 Python Unified Server

```
async def handle_request(websocket, path):
    async for message in websocket:
        request = json.loads(message)

        if request["type"] == "generate":
            # Route to appropriate engine
            if request["engine"] == "audiogen":
                audio = await generate_audiogen(request["prompt"])
            elif request["engine"] == "mmaudio_large":
                audio = await generate_mmaudio(request["prompt"], "large")

            # Apply DSP pipeline
            audio = trim_silence(audio)
            audio = normalize(audio)

            # Return metadata + PCM
            await websocket.send(json.dumps(metadata))
            await websocket.send(audio.tobytes())
```

A.4 Appendix D: Hardware Configuration

Component	**Details**
CPU	AMD Ryzen 9 7940HS (8C/16T, 4.0-5.2 GHz)
GPU	NVIDIA RTX 5070 Laptop (8 GB, sm_120 Blackwell)
RAM	64 GB DDR5-5600
OS	Windows 11 Pro 23H2
Unreal Engine	5.4.4
Python	3.11.7
CUDA	12.4
PyTorch	2.4.0+cu124

Table A.1: Complete system specifications