Capstone Milestone 01

Project Refinement & Technical Strategy

Arad Fadaei, Mahboobeh Yasini, Johnpaul Tamburro

Version 1.2, January 24, 2026

Table of Contents

Chapter 1. Refined Project Charter

1.1. Problem Statement & Goal

The Problem: Traditional text-based adventure games and early interactive fiction rely heavily on rigid, verb-noun command structures (e.g., "look north," "get key"). This mechanic often breaks player immersion by forcing them to memorize specific syntax or guess the parser's vocabulary. This cognitive load shifts the player's focus from the narrative to the interface itself.

The Goal: Our objective is to create a "Voice First" adventure game where the user interface is effectively invisible. By leveraging Large Language Models (LLMs) and speech recognition, players will simply speak their intentions in natural language. The system will act as an AI Dungeon Master, translating those natural intents into strict game logic and responding with dynamic audio narration.

1.2. Scope Adjustments

We have solidified the project as a **multimodal web application** rather than a standalone desktop executable. The core loop involves accepting raw audio input via the browser and generating synthesized audio narration in real-time.

1.3. In Scope

- **Speech-to-Text (STT) Pipeline:** Real-time ingestion of microphone input via the Web Audio API.
- **Intent Classification:** Using an LLM to translate natural language into structured JSON function calls (e.g., transforming "I want to smash that goblin" into `attack(target="goblin", type="heavy")`).
- **Game State Management:** Using a modern frontend framework (Svelte) to track inventory, health, and world state based on the JSON inputs.
- **Dynamic Output:** Procedural text generation converted to audio via Text-to-Speech (TTS).

1.4. Out of Scope

- **Complex 3D Graphics:** The visual fidelity will remain minimal (DOM-based UI or simple Canvas) to prioritize the AI narrative engine and reduce asset production overhead.
- **Multiplayer Networking:** The project is strictly single-player to focus on the latency challenges of the AI pipeline.
- **Mobile Browsers:** We are targeting Desktop Chrome/Firefox initially to ensure stable Web Audio API support without mobile OS restrictions.

Chapter 2. Data Acquisition & Initial Exploration

2.1. Data Strategy

As a generative application relying on pre-trained foundation models, we do not require a traditional training dataset for model creation. Instead, our data strategy focuses on **validation and evaluation**. We are creating a "Simulated Evaluation Dataset" to benchmark the accuracy of our intent classification prompt engineering.

2.2. Plan for Generation

We will generate a dataset of approximately 200—300 pairs of "User Command" to "Correct Function Call." This will be done using a stronger model (e.g., GPT-4) to synthesize diverse phrasings of common commands (e.g., "Drink potion," "Chug the red flask," "Heal me") to ensure our smaller runtime model handles variety correctly.

2.3. Example Data Pairs

The following table illustrates the types of natural language inputs we expect and their corresponding structured outputs:

User Command	Target Function Call	Purpose
"Slice him!"	`attack(target='enemy', type='slash')`	validate aggressive combat intent with weapon specification
"Chug the red flask"	`use_item(item='red_potion', target='self')`	test synonym recognition and color-based item identification
"Head to the door on my left"	`move(direction='left')`	validate natural language direction parsing with contextual clues
"What's in my backpack?"	`open_inventory()`	test question-based commands for inventory access
"Look around"	`inspect(target='room')`	test environment observation with implicit target
"Cast fireball at the orc"	`cast_spell(spell='fireball', target='orc')`	validate magic system intent with explicit targeting

2.4. Initial Findings & Cleaning Strategy

- **Homophones & Variance:** Initial testing shows that STT engines often misinterpret fantasy jargon or struggle with punctuation (e.g., interpreting "knights" vs. "nights").
- **Preprocessing Pipeline:** To mitigate this, we plan to implement a text normalization layer that lowercases input and strips filler words (e.g., "um," "uh") before sending the payload to the LLM. This reduces token count and improves JSON parsing reliability.

Chapter 3. Proposed Technical Approach

3.1. Dual-Stack Strategy

To maximize both commercial viability and research value, we are implementing a decoupled architecture that supports two distinct backends: 1. **Commercial SaaS Stack:** A hybrid model using APIs for heavy lifting (Narrative/STT) and local lightweight models for logic. 2. **Self-Hosted Research Stack:** Optimized for privacy and offline capability using local models for everything.

3.2. Implementation A: Commercial Cloud Stack (SaaS Target)

Designed for public web deployment, prioritizing unit economics by hosting lightweight logic locally.

Component	Technology	Model / API	Responsibility
STT	**Groq**	`whisper-large-v3-turbo`	Extreme speed (~300ms) transcription via LPU inference.
Logic Router	**Function Gemma**	`function-gemma-it` (Local)	We host this lightweight model on the backend server itself. It handles the JSON logic generation, saving significant API costs per turn.
Narrative	**Groq**	`llama-3.3-70b`	High-fidelity prose generation for the Dungeon Master's descriptions. We use the API here because quality and context window matter most.
TTS	**Lemonfox.ai**	OpenAI-Compatible	Cost-effective voice synthesis (significantly cheaper than ElevenLabs/Deepgram).

3.3. Implementation B: Fully Self-Hosted Stack (Research/Open Source)

Designed for local execution (Docker) without external dependencies.

Component	Technology	Model / Library	Responsibility
STT	**Faster-Whisper**	`whisper-medium` (CTranslate2)	Local GPU-accelerated transcription running on the user's hardware.

Component	Technology	Model / Library	Responsibility
Logic Router	**Function Gemma**	`function-gemma-it` (GGUF)	The same model used in the SaaS stack, but running on the user's GPU.
Narrative	**Llama 3**	`Llama-3-8b-Instruct` (GGUF)	A quantized version of Llama 3 running via `llama.cpp` to generate the creative, storytelling elements of the game.
TTS	**Piper TTS**	`en_US-libritts-high` (ONNX)	A lightweight, neural text-to-speech engine optimized for real-time local synthesis.

3.4. Shared Core Infrastructure

Regardless of the selected AI stack, the core application infrastructure remains identical:

Component	Technology	Responsibility
Frontend	**Svelte 5** (TypeScript)	Manages the UI, Game State Stores, and Web Audio API recording.
Backend	**Python** (FastAPI)	Acts as the Interface Adapter. It detects which stack is active and routes requests accordingly. It also hosts the `llama.cpp` instance for Function Gemma in both modes.
Transport	**WebSockets**	Handles the bi-directional stream of audio chunks and text events.

3.5. Data Flow (The *Turn* Loop)

1. **Input:** Svelte records audio → WebSocket → FastAPI.
2. **Transcribe:**
   - *Cloud:* API Call to Groq.
   - *Local:* `Faster-Whisper` process.
3. **Intent (Logic):** Text sent to LLM Router.
   - *Both Modes:* **Function Gemma** (Local/Backend) outputs JSON.
4. **Execute:** FastAPI updates Game State (e.g., `hp -= 10`).
5. **Narrate (Creative):** Result sent to Narrative LLM.
   - *Cloud:* Llama 3.3 (Groq).
   - *Local:* **Llama 3** (llama.cpp).
6. **Speak:** Text sent to TTS.

    - *Cloud:* Lemonfox API.
    - *Local:* Piper TTS.

3.6. Infrastructure & Deployment Strategy

Our dual-stack approach requires two distinct deployment configurations:

3.6.1. Commercial Cloud Stack Deployment

- **Frontend Hosting: Vercel**. Chosen for its seamless integration with SvelteKit/Svelte.
- **Backend Hosting: Railway** or **Render**.
    - *Justification:* Unlike standard serverless functions (like AWS Lambda) which time out after a few seconds, our WebSocket architecture requires **persistent TCP connections**. Railway/Render provide simple Docker container hosting that allows the FastAPI server to maintain long-running socket connections with the browser.
    - *Resource Requirements:* The backend must host Function Gemma locally, requiring ~2-4GB RAM (with quantization). This is still more cost-effective than per-token API pricing for logic operations.
- **Containerization: Docker**. We will use a `Dockerfile` to package the Python backend, Function Gemma model files, and system dependencies (like `ffmpeg` for audio processing).
- **Environment Variables:** All API keys (Groq, Lemonfox) stored in backend environment only.

3.6.2. Self-Hosted Stack Deployment

- **Single Container Architecture:** The Svelte frontend will be built into static assets and served directly by the FastAPI backend.
- **Model Storage:** Docker image includes all model files (Function Gemma, Llama 3, Whisper, Piper TTS).
- **Hardware Requirements:**
    - Minimum: 8GB RAM, 4-core CPU (CPU-only inference).
    - Recommended: 8GB+ VRAM GPU for optimal performance.
- **Deployment:** Users should be able to run the entire game locally with a single command.
- **Open Source:** Full source code released on GitHub.

3.7. Security & Cost Control

- **API Key Isolation:** All third-party keys (Groq, OpenAI, Lemonfox) will be stored strictly in the Backend environment variables. **No keys will ever be exposed to the Frontend.**
- **Usage Limits:** To prevent budget overruns during testing, the backend will implement a "Turn Limit" (e.g., max 50 turns per session) and a simplified Rate Limiter to prevent a single client from spamming the STT endpoint.

Chapter 4. Project Plan & Team Roles (Next 3 Weeks)

4.1. Team Roles

- **Member 1 (Frontend Lead):** Responsible for Svelte project architecture, UI component design, and the core `GameManager` logic (handling Health, Inventory, and State).
- **Member 2 (AI Engineer):** Responsible for the Python backend and integrating the local Function Gemma instance into the API pipeline.
- **Member 3 (Integration Specialist):** Responsible for Docker containerization and the audio pipeline (WebSockets/Piper/Lemonfox).

4.2. Task Breakdown

Week 4: Foundation & Tools

- **Goal:** Establish the dual-environment backend.
- **Tasks:**
  - create `AbstractLLMClient` class in Python.
  - Implement `GroqClient` (for Narrative) and `LocalClient` (for Function Gemma logic).
  - Download and quantize Function Gemma and Llama 3 models.

Week 5: The Voice Command Prototype

- **Goal:** End-to-end voice control.
- **Tasks:**
  - Connect Frontend Audio → Backend.
  - Verify that "Go North" works using the local Function Gemma model in both environments.

Week 6: Combat Vertical Slice

- **Goal:** A playable loop with audio feedback.
- **Tasks:**
  - Integrate TTS (switchable between Lemonfox/Piper).
  - Implement initial Combat Logic.

Chapter 5. Questions, Risks & Mitigations

5.1. Risk 1: Latency Accumulation

The proposed pipeline (Audio → Text → LLM → Logic → TTS) involves multiple network hops. **Specific Risk:** Switching to Lemonfox for TTS may introduce slightly higher latency than a streaming-first provider like Deepgram.

- **Mitigation:**
  - **Optimistic UI:** We will display the text narration on screen **immediately** as it is generated by the LLM, so the user can read while waiting for the audio to buffer.
  - **Parallel Processing:** Transcription and logic routing can happen simultaneously to reduce total pipeline time.
  - **Fallback Plan:** If the audio delay proves too immersion-breaking during the prototype phase, we will pivot back to Deepgram Aura for the final showcase.

5.2. Risk 2: Model Hallucination

The LLM might generate invalid function names or parameters that do not exist in the game code, causing the application to crash or malfunction.

- **Mitigation:**
  - Use **Strict JSON Mode** (or Grammar Constraints when using local models) to force the output to adhere to our defined schema.
  - Implement a "fallback" handler in Python that detects malformed JSON and asks the player to repeat the command naturally, rather than crashing the game.
  - Function Gemma's specialization in function calling should reduce this risk compared to general-purpose models.

5.3. Risk 3: Backend Resource Load

Hosting Function Gemma on the backend for the SaaS version means the server needs more RAM/CPU than a simple API proxy.

- **Mitigation:**
  - **Quantization:** We will use quantization for Function Gemma to keep memory usage extremely low.
  - **Hosting:** We will slightly upgrade the Railway service tier to ensure it can handle the inference load, which is still cheaper than paying per-token for logic.
  - **Load Testing:** Benchmark the backend under simulated multi-user load to determine if horizontal scaling is needed.

5.4. Risk 4: Stack Divergence

The behavior of Groq (Llama 70b) and Local (Llama 8b) might differ, causing bugs that only appear in one mode.

- **Mitigation:**
  - We will use the **Self-Hosted Stack** as the "Source of Truth" for prompt engineering, as smaller models are less forgiving. If it works on Llama 8b, it will likely work on Llama 70b.
  - Maintain identical prompt templates between both stacks, only varying the model endpoint.
  - Implement comprehensive integration tests that run against both stacks to catch divergence early.

Chapter 6. Financial Analysis & Business Viability

6.1. Development Costs

During the development phase (Next 6 weeks), our goal is to keep fixed costs near zero by leveraging free tiers and credits.

- **Hosting (Railway/Render):** ~$5.00/month.
  - *Note:* The "Hobby" tier on Railway provides $5 of credit, effectively making it free for low-traffic dev work.
- **LLM (Groq):** ~$5.00 - $10.00 (Total).
  - *Note:* Development traffic is low volume. We will likely stay well under the initial credit limits.
- **STT/TTS (Groq/Lemonfox):** < $5.00 (Total).
  - *Note:* Lemonfox's cost is negligible for testing.
- **Hardware:** $0.00 (Existing Dev PCs with GPU support).
- **Total Estimated Dev Cost:** **<$25.00 CAD** for the entire capstone period.

6.2. Commercial Cloud Stack Economics

Key Innovation: By self-hosting Function Gemma on our backend server, we convert the "logic" cost from variable (per-token) to fixed (server RAM), dramatically improving unit economics.

6.2.1. Cost Per Turn Analysis

A "Turn" consists of: . **Input:** 5 seconds of audio (User command). . **Logic:** Function Gemma inference (self-hosted, zero marginal cost). . **Narrative:** ~100 tokens of prose generation. . **Audio Out:** ~200 characters of narrative speech.

Cost Breakdown (USD):

- **STT (Groq Whisper Turbo):** $0.04/hr → 5 seconds = **$0.000055**
- **Logic (Function Gemma):** Self-hosted = **$0.00** (absorbed into fixed server cost)
- **Narrative (Groq Llama 3.3 70B):** $0.79 per 1M tokens → 100 tokens = **$0.000079**
- **TTS (Lemonfox):** $2.50 per 1M characters → 200 characters = **$0.000500**

Total Variable Cost Per Turn: ~$0.00063 USD (~$0.0009 CAD).

6.2.2. Blended User Model ($5.00 USD/month subscription)

Scenario A: The "Heavy" User (The Loss Leader)

- **Usage:** 2 hours/day (every day) → ~3,000 turns/month.
- **Variable Cost:** 3,000 x $0.00063 = **$1.89 USD**.
- **Result:** +**$3.11 Profit** per user (vs -$0.40 in API-only approach).

Scenario B: The "Casual" User (The Profit Driver)

- **Usage:** 3 sessions/week (20 mins each) → ~250 turns/month.
- **Variable Cost:** 250 x $0.00063 = **$0.16 USD**.
- **Result:** +**$4.84 Profit** per user.

Scenario C: The Blended Reality (80/20 Split)

Assuming typical SaaS metrics where 20% of users are power users and 80% are casuals:

- **Weighted Average Variable Cost:** (0.2 x $1.89) + (0.8 x $0.16) = **$0.51 USD/User**.
- **Fixed Costs (Server):** ~$20/month ÷ 100 users = **$0.20/User**.
- **Total Cost:** $0.51 + $0.20 = **$0.71/User**.
- **Net Profit:** $5.00 - $0.71 = **$4.29/User**.

Conclusion: The project is **highly commercially viable**. By self-hosting Function Gemma, we've reduced variable costs by ~70% compared to an all-API approach, making even heavy users profitable.

6.3. Research & Portfolio Value

Regardless of the financial model, the project's primary value driver remains **Technical Research**.

- **Primary Objective:** Demonstrate advanced Full Stack capability by implementing a real-time, low-latency voice pipeline.
- **Key Research Question:** "Can a hybrid architecture (self-hosted logic + cloud narrative) achieve sub-1000ms conversational latency while maintaining commercial viability?"
- **Portfolio Value:** This architecture (Groq + Local LLM + WebSockets + Optimistic UI + Svelte 5) represents a Senior Level integration challenge and demonstrates expertise in cost optimization for AI applications.