SynthLang

Requirements Specification Document

April 30, 2026

Project Sponsor Dr. Benjamin V. Tucker

Faculty Mentor Isaac Shaffer

Members

Paul: Lead, Coder

Tyler: Recorder, Coder

Logan: Release Manager, Coder

Jalen: Architect, Coder

Accepted as baseline requirements for the project

For the client:_____ For the team:_____

Date: _____ Date: _____

Table of Contents

1 Introduction

An Augmentative and Alternative Communication (AAC) device allows users with speech impairments to speak. This could be a tablet with buttons, or drawing on a notebook to get ideas across. A technology that has been used as an AAC in recent history to great effect is text-to-speech (TTS). Deep learning technologies have vastly increased the quality and flexibility of TTS in the past decade, all while making TTS cheaper and more feasible to make and use. Something that would have been thought either impossible or so resource intensive that it might as well be is creating TTS models for underserviced languages, or training models to specific people's voices. Languages that don't have large amounts of high-quality training data, or speakers who will soon lose their voice due to disease, and therefore can't record 100 hours of training data.

Our sponsor is Dr. Benjamin Tucker here at the NAU Department of Communication Science & Disorders, and some of his research focuses specifically on lesser spoken and studied languages. He has also done numerous research papers on speech generation technologies such as text-to-speech. Our goal with this capstone project is to create a text-to-speech pipeline that will allow for training of new models with relatively low amounts of training data, and an interface that allows people to use those models. By doing this, we can create TTS models for languages that may not have high-quality models yet (Navajo/Diné Bizaad specifically). This will also be used for a few people who have come into Dr. Tucker's lab to record audio data of their voices. With our project, we hope to give a voice to those in the Diné community who have lost theirs and hope to enable them to be represented and heard in their communities.

2 Problem Statement

2.1 Application Context

The sponsor's current workflow for generating speech is based on a legacy Text-to-Speech (TTS) system using an outdated Tacotron2 implementation. This process involves collecting speech data, training models, and generating audio outputs through a series of manual and technically complex steps. While this workflow was previously functional, it is no longer sustainable due to changes in hardware compatibility and a lack of ongoing support for the underlying framework.

Currently, there is no unified system that integrates model training, management, and inference into a single streamlined pipeline. As a result, the process is fragmented and inefficient, requiring significant manual intervention and technical expertise at each stage.

2.2 Problem Overview

The primary issue is that the existing TTS system is outdated, difficult to maintain, and incompatible with modern computing environments. This significantly limits the sponsor's ability to continue developing and using speech models, particularly for low-resource languages.

2.3 Identified Deficiencies

The current system has several key limitations:

- **Outdated and Unsupported Technology**
  The Tacotron2 implementation is no longer maintained and cannot reliably run on modern hardware, creating long-term sustainability issues.
- **Lack of a Unified Pipeline**
  There is no integrated workflow for training, deploying, and using TTS models, leading to inefficiencies and repeated setup effort.
- **High Technical Complexity**
  The system requires advanced knowledge of machine learning frameworks and environment configuration, making it difficult to use or extend.
- **Limited Accessibility**
  There is no user-friendly interface for generating speech, preventing easy interaction with the models.

Deployment Restrictions

Institutional policies prevent the use of NAU's high-performance GPU cluster for always-online web servers and since there is no budget for cloud-based GPU hosting, our solution must be able to be deployed on machines without a dedicated GPU.

2.4 Proposed Solution

To address these challenges, the project proposes the development of a modern, modular TTS system consisting of two main components:

1. **TTS Training Pipeline**
   A flexible and modular pipeline built using Coqui TTS and PyTorch to support model training, fine-tuning, and experimentation with low-resource languages.
2. **Web-Based Application**
   A lightweight web interface built using FastAPI and HTMX that allows users to input text and generate speech through a simple and accessible interface.

This solution replaces the legacy system with maintainable, modern technologies while improving usability, performance, and accessibility. The architecture follows a server-side approach to comply with institutional constraints and maintain cost efficiency, while remaining flexible for future extensions.

3 Solution Vision

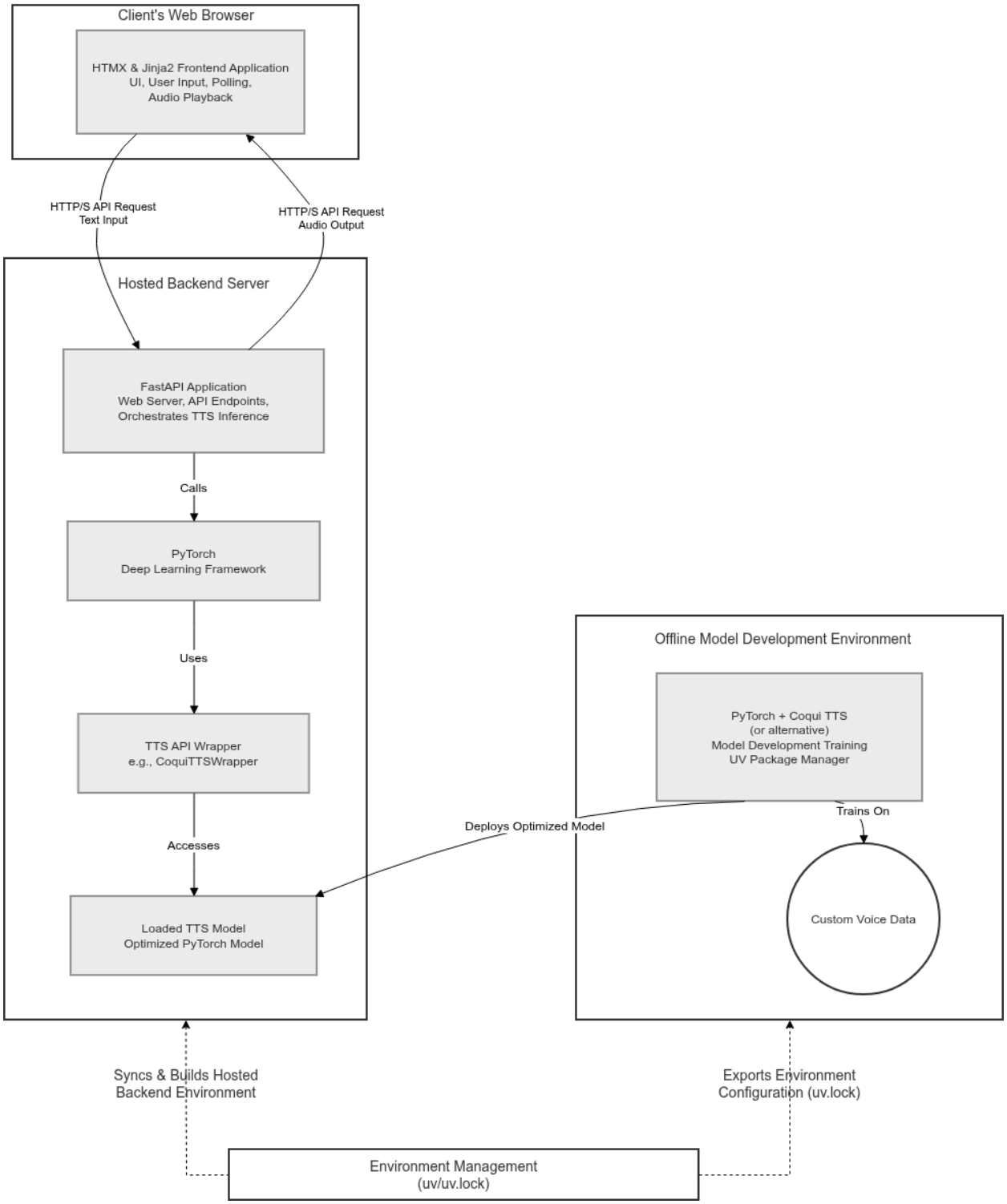
We will be making a new web-enabled Text-to-Speech (TTS) training and inference pipeline for Dr. Tucker. The purpose of this project is to provide a modern, maintainable platform for our client to train and deploy voice models for under-resourced languages like Diné Bizaad. Features will include:

- Web-Based Synthesis: An intuitive interface for users to input text and trigger real-time TTS inference.
- Training model pipeline offline on client's Titan GPUs.
- Framework Agnostic Architecture: A modular backend that supports seamless replacement or upgrading of the underlying TTS models.

The operational workflow is strategically split into two to balance performance with accessibility. While high-intensity model training is performed offline on the client's dedicated Titan GPUs, the resulting optimized models are deployed to a production server for inference. To ensure the tool remains usable across various academic environments, inference is performed on the server-side CPU. While this introduces a minor trade-off in synthesis latency when compared to GPU-based processing, the system provides real-time progress feedback to the user, ensuring a transparent and responsive experience. By utilizing modern dependency management (via uv) and a modular design, we provide Dr. Tucker with a maintainable ecosystem. The high-level system diagram can be viewed on the next page. The system will allow our client to focus on the data gathering and analysis of trained models instead of the technical details of a given TTS framework.

By lowering the barrier to entry for high-quality speech synthesis, we are giving our client the tools to help languages like Diné Bizaad and isiXhosa stay 'audible' in a digital world. Our modular approach directly addresses the primary deficiency of our clients previous system: its tendency to break when dependencies are updated. This will enable Dr. Tucker in his mission to provide language tools for under-resourced communities. Additionally, the creation of these models could help these communities develop language teaching tools to maintain the health of their own languages for generations to come.

System Overview



4 Web Application Requirements

4.1 Domain Level Requirements

D1. Input and Output:

- The website will provide text input and TTS-generated audio output: the website needs to provide an interface where the user can submit text and receive audio in return.

D2. Model Selection:

- The website will provide the user with the ability to select different pre-trained models for different voices and languages: multiple different acoustic models will need to be selectable and usable by the user.

D3. Dynamic Interface:

- The website will inform the user of the progress of the audio generation as it happens.

D4. Backend Integration:

- The backend of the website will be able to store and communicate with pre-trained models that have been uploaded.

D5. Hostability:

- The website will be maintainable and easily hostable for NAU ITS.

D6. Website Accessibility:

- The website will be accessible for both mobile and desktop users.

4.2 Functional Requirements

F1. Simple Hosting:

- The FastAPI backend will handle asynchronous requests, and the Jinja2 templating engine shall serve HTMX attributes directly to the frontend to eliminate the need for a separate Javascript build step.

F2. Single-page application frontend:

- The website will have a text input box: A new user should be able to easily select a language and input text with no scrolling.
- The website will provide a drop-down menu for model selection.
- The website will provide a button to start synthesis.
- The website will provide an output area with the following controls:
    - **Play**
    - **Pause**
    - **Slow down**
    - **Speed up**
    - **Replay**
    - **Download audio**
- The website will provide downloaded audio in a standard format, such as .wav.
- The website frontend shall utilize sentence-based chunking (as researched in Feasibility 3.3.1) to allow audio playback to begin while the remainder of the text is still being processed by the CPU.
    - To minimize time-to-first-audio, the backend shall implement sentence-level text chunking and utilize PyTorch's torch.compile() to optimize the inference graph for CPU execution.
- The frontend shall utilize HTMX polling to query synthesis status. This replaces the need for WebSockets, reducing server-side complexity and ensuring compatibility with standard NAU ITS hosting environments.

F3. Error handling:

- The Website shall leverage FastAPI's native asynchronous error handling to ensure that synthesis failures do not hang the user's browser session.

4.3 Performance Requirements

P1. Initial Load Time:

- The web interface shall load and become interactive within 2 seconds on a standard broadband connection (25 Mbps) to ensure immediate accessibility.

P2. Model Switching Latency:

- Switching between different TTS models via the dropdown menu shall update the application state and be ready for new text input within 2 seconds.

P3. Progress Feedback Frequency:

- To manage user expectations during slow CPU-based synthesis, the progress bar shall update at least once every 1 second while a synthesis task is active.

P4. Concurrent User Stability:

- The web application shall remain responsive for navigation and model selection for up to 5 concurrent users, even if a synthesis task is currently being processed by the backend.

P5. Client-Side Resource Footprint:

- The web interface shall maintain a memory footprint of less than 100MB in the client's browser to ensure stability on lower-end mobile and desktop devices.

P6. Audio Playback Readiness:

- Once synthesis is complete, the generated audio file shall be loaded into the browser's player and be ready for playback within 1 second.

P7. CPU Bottleneck:

- The system shall maintain a synthesis throughput of at least 30 characters per second on server-side CPUs to ensure a 300-character passage completes in under 10 seconds.

4.4 Environmental Requirements

E1. ITS Hosting with CPU:

- The web application will be optimized to work on CPU, and needs to conform to NAU hosting rules.

E2. Python version 3.12:

- PyTorch shall be the primary engine for inference, utilizing torch.compile() to optimize the model for CPU-based execution.
- To work with the version of PyTorch and CoquiTTS, the backend must run on a specific Python version to match our version of CoquiTTS. UV package manager will be used to guarantee reproducibility and parity across all environments.

E3. Low library dependence:

- The application needs to work into the future with no changes or refactoring. UV package manager and an effort to not use unnecessary libraries will allow this.

- UV package manager ensures "Environment Parity" by using a uv.lock file to prevent version conflicts between dev and production.

5 Training Pipeline Requirements

5.1 Domain Level Requirements

D1. Accessible Model Creation:

- The system will enable Dr. Tucker to create TTS acoustic models without requiring advanced software engineering expertise.

D2. Support for Fine Tuning

- The system will support model fine-tuning on small, noisy, or incomplete datasets typical of under-resourced languages.

D3. Configurable Training Process

- The system will allow the client to customize training behavior without modifying internal source code.

D4. Reproducibility of Experiments

- The system will support reproducibility of model training runs.

D5. Modular and Extensible Architecture

- The system will allow integration with multiple TTS frameworks and future technologies.

D6. Transparent Training and Debugging

- The system will provide visibility into training progress, errors, and outputs.

5.2 Functional Requirements

This section outlines the core functionality of our system insofar as it relates to the training pipeline.

5.2.1 Dataset management

F1. Dataset Ingestion

The syste

F2. Dataset Validation

- The system will validate dataset structure and report inconsistencies.

- The system will detect missing or mismatched audio-text pairs.
- The system will identify invalid audio formats or corrupt files.

F3. Dataset Preprocessing

- The system will normalize text input (e.g., casing, punctuation handling).

5.2.2 Training management

F4. Training Configuration

- The system will provide a programmatic interface for defining training configurations.
- The system will allow users to specify model parameters (e.g. epochs, batch size).
- The system will allow users to select training strategies (e.g. fine-tuning vs. training from scratch).
- The system will allow adjustment of optimization parameters (e.g. learning rate).
- The system will allow configuration of preprocessing options.

F5. Configuration Persistence

- The system will allow saving and loading of training configurations.
- The system will associate configurations with specific training runs.

F6. Training Execution

- The training pipeline shall be implemented using the PyTorch deep learning framework to allow for consistent model formats across both the training and inference environments.
- The system will initiate model training using a provided dataset and configuration.
- The system will support execution on GPU.

F7. Training Logs

- The system will record training metrics (e.g., loss values).
- The system will log errors and warnings encountered during training.

F8. Progress Monitoring

- The system will provide real-time updates on training progress.
- The system will report estimated time remaining for training completion.

5.2.3 Model output and management

F9. Model Export

- The system will output trained models in a reusable format.

- The system will associate trained models with their training configurations and datasets.

5.2.4 Framework abstraction

F10. Framework Integration

- The system will support integration with at least Coqui TTS.
- The system will abstract framework-specific details from the user interface.

F11. Framework Switching

- The system will allow substitution of underlying TTS frameworks without requiring major changes to user workflows.

5.3 Performance Requirements

This section details the performance requirements of our system including specific values for execution time.

5.3.1 Training performance

P1. Training Initialization Time

- The training pipeline shall target the 15x–17x speedup observed during feasibility testing (Feasibility 3.2.5) by utilizing CUDA-accelerated PyTorch workflows.

P2. Resource Utilization

- The training pipeline shall utilize CUDA 12.x to leverage the FP16/BF16 tensor cores of the client's NVIDIA Titan GPUs, targeting a 15x–17x speedup over CPU-based training as identified in feasibility benchmarks.

5.3.2 Dataset handling performance

P3. Dataset Validation Time

- The system will validate datasets of up to 10,000 samples within 2 minutes.

P4. Preprocessing Throughput

- The system will preprocess audio at a minimum rate of 1x real-time (i.e., 1 hour of audio in ≤1 hour).

5.3.3 Usability performance

P5. Learnability

- A user with basic Python knowledge will be able to initiate a training run within 30 minutes of first use.

P6. Configuration Complexity

- The system will allow a minimal training configuration to be specified in ≤10 lines of code.

5.3.4 Reliability and reproducibility

P7. Reproducibility

- Given identical inputs and configurations, training runs will produce equivalent model outputs within acceptable variance.

P8. Failure Recovery

- The pipeline shall implement automated checkpointing, saving the model state and optimizer weights to a .pth file every 1000 steps (or as defined in the training config), allowing training to resume from the last stable epoch in the event of a system crash.

5.3.5 Monitoring Responsiveness

P9. Logging Latency

- Training logs will be updated with a delay of no more than 5 seconds.

5.4 Environmental Requirements

The following constraints are outside of the design process and cannot be changed without changing the nature of the project.

5.4.1 Software Constraints

E1. Programming Language

- The project shall utilize uv to manage a deterministic environment, ensuring that the exact versions of PyTorch and Coqui TTS used in training are replicated in the production inference server via a uv.lock file.

- All Python dependencies for the training pipeline and web application must be bundled and managed by uv to guarantee reproducibility across different machines.

E2. Framework Dependency

- The system will initially integrate with Coqui TTS.
- The system shall implement a modular API wrapper for Coqui TTS to allow for the swapping between different acoustic models and vocoders' as outlined in Feasibility 3.1.3. Specifically with Speechbrain.

5.4.2 Hardware Constraints

E3. Execution Environments

- The system will operate on Linux-based and Windows based systems via a uv environment.
- The training system will support execution only on machines with GPUs.

5.4.3 Data Constraints

E4. Dataset Format

- The system will require datasets consisting of paired audio and text data.

E5. Storage Requirements

- The system will operate within available local storage without requiring distributed storage systems.

5.4.4 Integration Constraints

E6. Interface Compatibility

- The Python API shall expose a singleton Wrapper class for the TTS engine, following the Adapter design pattern to ensure the backend can interoperate with multiple TTS frameworks (such as Coqui TTS or SpeechBrain) without requiring changes to the web application's core logic.

6 Potential Risks

This section offers an analysis of the risks most relevant to the project’s success, focusing on the likelihood of occurrence and the specific impacts these risks have on the customer and the end-users. Relevance is determined by how these risks could compromise the development effort or the long-term utility of the linguistic research tools.

6.1 Infrastructure Constraints: CPU-Based Inference Latency

Relevance: *High Likelihood / High Impact*

As noted in the Technological Feasibility analysis, there is currently no budget for dedicated, persistent server hosting with GPU resources. Furthermore, university IT policy prohibits the use of high-performance faculty workstations as public-facing web servers. Consequently, the system must perform Text-to-Speech (TTS) inference on a server-side CPU.

Impact on the Customer:

The primary risk is **system abandonment** due to high latency. Benchmarks in the feasibility study show that a 319-character passage takes approximately 10 seconds to synthesize on a CPU, compared to less than one second on a GPU. For a student or researcher, this delay is a significant barrier to "real-time" interaction.

Worst-case scenario: If a user inputs a larger passage and the synthesis exceeds 30 or 60 seconds, the web browser may time out, or the user may assume the system has crashed and navigate away. For Dr. Tucker, the "mistake" of slow output means the tool fails to provide the seamless, low-friction experience required for educational integration, effectively rendering the revitalization assets inaccessible to the community.

6.2 Sustainability Risk: Open-Source Framework Obsolescence

Relevance: *Medium Likelihood / High Impact*

The project relies on Coqui TTS as its primary inference engine. However, the feasibility research confirmed that Coqui has ceased active corporate maintenance. This mirrors the exact problem the sponsor faced with his previous legacy TTS tools.

Impact on the Customer:

The risk to Dr. Tucker is **Long-Term Research Interruption.** If the system is built with a hard dependency on an unmaintained framework, the platform may break in 3 to 5 years when server operating systems or Python versions are upgraded.

Worst-case Scenario: the custom-trained voice models for under-resourced languages—which represent hundreds of hours of linguistic research—could become

trapped in an unrunnable software environment. This would force the sponsor into another costly "revitalization" cycle, wasting previous development efforts and potentially losing the ability to generate speech for those languages until a new replacement is built.

6.3 Technical Debt: Dependency Instability and "Environment Drift"

Relevance: *High Likelihood / Medium Impact*

A minor update to a sub-dependency (such as `torch` or `numpy`) can break the system, preventing it from starting or causing it to produce inconsistent audio output.

Impact on the Customer:

The impact is a **High Maintenance Burden and Deployment Failure.** Dr. Tucker's primary role is linguistic research, not full-time system administration. If the software is "fragile" and requires constant manual debugging of Python environments, the tool becomes a liability rather than an asset.

Worst-case scenario: When attempting to deploy the system to a new university server, a version conflict prevents the application from launching. The "mistake" of an inconsistent environment leads to a total loss of service. For the researcher, this means scheduled workshops or classroom sessions must be canceled because the tool is "down," damaging the professional credibility of the revitalization project.

6.4 Operational Risk: Server Resource Exhaustion

Relevance: Medium Likelihood / Medium Impact

Because synthesis is CPU-intensive and requires loading large PyTorch models into RAM, the server is vulnerable to resource spikes.

Impact on the Customer:

The risk is **Service Deniability** during peak usage. If the client introduces the web app to a large number of users who all try to access the service at once, the server's CPU and RAM could be overwhelmed.

Worst-case scenario: The server process is "killed" by the operating system to prevent a total hardware hang. This results in a "500 Internal Server Error" for all active users. Instead of a user getting a "wrong answer," they get no answer at all.

6.5 Data Integrity: Phonetic Accuracy for Under-Resourced Languages

Relevance: *Low Likelihood / High Impact*

Neural TTS models, especially those trained on limited data, can occasionally "hallucinate" or mispronounce text based on subtle variations in punctuation or input.

Impact on the Customer:

The risk is the **Dissemination of Incorrect Linguistic Data**. For an under-resourced language, the primary goal is preservation. If the tool generates audio that is phonetically incorrect and that audio is used for language learning, the tool is inadvertently teaching the wrong pronunciation.

Worst-case Scenario: the software designed to save a language is actually polluting the phonetic record. Unlike a mere "inconvenience," this impact directly contradicts the project's mission of accurate language revitalization and stewardship.

Risk	Likelihood	Impact
Infrastructure Constraints	5	5
Sustainability Risk	3	5
Technical Debt	4	3
Operational Risk	3	3
Data Integrity	1	5

7 Project Plan

7.1 Overview

The project will be executed through a series of structured milestones that align with the system's functional requirements and technological feasibility. The approach emphasizes early validation through prototyping, followed by iterative development, integration, and

testing. This ensures that risks are identified early and that progress remains aligned with project goals.

7.2 Milestones and Timeline

The project is divided into the following key milestones:

- **Requirements Analysis and Technology Selection**
  Identify system requirements and constraints. Evaluate and select appropriate technologies, including Coqui TTS, PyTorch, FastAPI, and HTMX.
- **Prototype Development (Feasibility Validation)**
  Develop initial prototypes to test TTS model inference and validate performance on both CPU and GPU environments. This confirms that the selected technologies meet project needs.
- **Backend Development (FastAPI)**
  Implement the server-side application responsible for handling API requests, processing input text, and generating audio output using the TTS model.
- **Frontend Development (HTMX Interface)**
  Develop a simple and responsive user interface that allows users to input text, select options, and play generated audio.
- **TTS Training Pipeline Implementation**
  Build the pipeline for preprocessing data, training models, and supporting transfer learning for low-resource languages.
- **Performance Optimization**
  Improve system efficiency through PyTorch optimizations and evaluate inference speed to ensure acceptable user experience on CPU-based systems.
- **System Integration and Testing**
  Integrate all components and perform unit, integration, and usability testing to validate functionality and identify issues.
- **Deployment and Environment Setup**
  Deploy the application to a hosted server environment while ensuring compliance with institutional policies. Use a package manager (uv) to maintain consistent environments.
- **Final Validation and Refinement**
  Conduct final system testing, gather sponsor feedback, and implement necessary refinements before delivery.

7.3 Timeline Structure

1st Semester

TASK	START	DAYS	END	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Phase 1																	
Requirements Analysis	2/2/2026	14	2/15/2026														
Tech Selection	2/2/2026	14	2/15/2026														
Research & Setup Environment	2/16/2026	14	3/1/2026														
Phase 2																	
Backend Development	3/2/2026	21	3/22/2026														
Frontend Development	3/9/2026	21	3/29/2026														
TTS Training Pipeline Implementation	3/9/2026	28	4/5/2026														

2nd Semester

TASK	START	DAYS	END	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Phase 3																			
System Integration	8/24/2026	28	9/20/2026																
Testing Phase	9/21/2026	28	10/18/2026																
Deployment & Environement Setup	10/19/2026	21	11/8/2026																
Final Validation & Refinement	11/9/2026	28	12/7/26																

8 Conclusion

Synthlang wants to create a platform for speakers of underserved languages to find a natural-sounding voice to represent their thoughts. We also want those who have come into Dr. Tucker's lab and recorded their voices to be able to continue to speak their native language, even after degenerative conditions may take away their natural voices.

To do this, we will make a training pipeline using CoquiTTS and Pytorch that Dr. Tucker can use to train models and a website using FastAPI and HTMX where people can use those models. Through making this document, we finalized our tech stack and created demos to prove the feasibility of these technologies and ensure that our product is usable well into the future. These things wouldn't have been possible without careful examinations of the pros and cons of multiple choices of technologies at every step of the way. After creating a timeline, we are happy to say that SynthLang is making steady progress toward a working product, and we are very excited to see the direct impact that our product will have for the people that need it most.