Technical Design Document

Synthetic Radio Host - Hinglish Podcast Generator

Contents

1. Project Overview

1.1 Project Description

The Synthetic Radio Host system is an automated podcast generation platform that:

- Fetches contextual information from Wikipedia for any given topic.
- Generates natural, conversational Hinglish scripts using Ollama LLM.
- Parses and structures dialogue between two hosts.
- Converts text to speech using ElevenLabs API with distinct voices.
- Combines audio segments into a complete podcast MP3 file.

The system demonstrates practical application of LLMs (Ollama) for natural language generation combined with Text-to-Speech APIs (ElevenLabs) to create engaging podcast content in Hinglish (Hindi-English mix).

1.2 Key Features

- Natural Hinglish conversation generation with phonetic spelling
- Emotional tags support ([laughs], [sighs], [giggles]) for realistic audio
- Automatic dialogue parsing with intelligent speaker identification
- Multi-voice podcast generation (male and female hosts)
- Robust error handling for API failures and edge cases
- User confirmation before audio generation to save API costs as Eleven labs APIs are costly.
- Comprehensive logging and progress tracking

1.3 Use Cases

- Educational content creation for topics requiring explanation
- Entertainment podcasts on trending topics
- Multilingual content generation for Indian audiences
- Automated radio show production
- Content creation for social media platforms

2. System Architecture

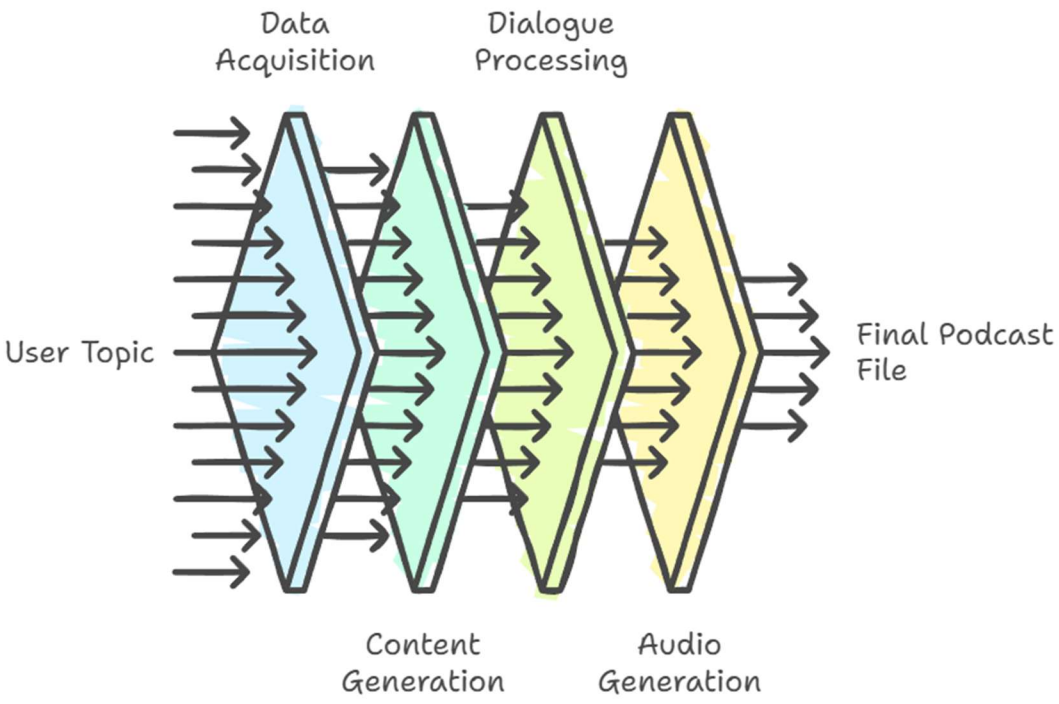
2.1 High-Level Architecture

The system follows a modular, pipeline-based architecture with clear separation of concerns.

The architecture consists of four main layers:

- Data Acquisition Layer: Fetches context from Wikipedia based on user topic.
- Content Generation Layer: Creates Hinglish conversation script using Ollama LLM ollama3.2
- Processing Layer: Parses and structures dialogue.
- Audio Generation Layer: Converts text to speech and combines audio to prepare a final podcast file.

Architecture Diagram:



2.2 Component Diagram

The system consists of the following main components and their interactions:

Component Details:

Wikipedia Context Fetcher: Fetches topic information from Wikipedia API

Prompt Generator: Creates structured prompts for LLM with Hinglish instructions using one-shot inference.

Ollama LLM Integration: Generates conversational scripts using local/remote Ollama

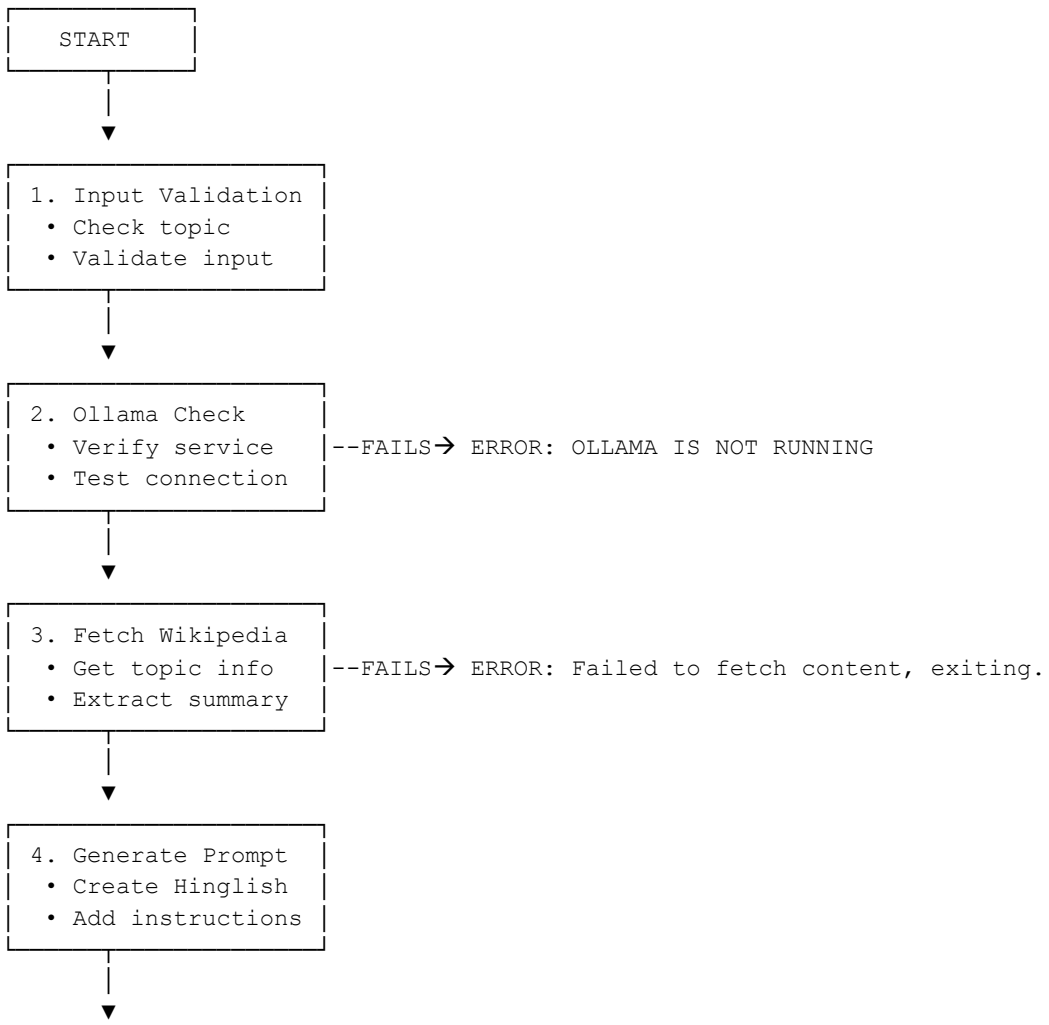
Script Parser: Parses and structures dialogue with speaker identification

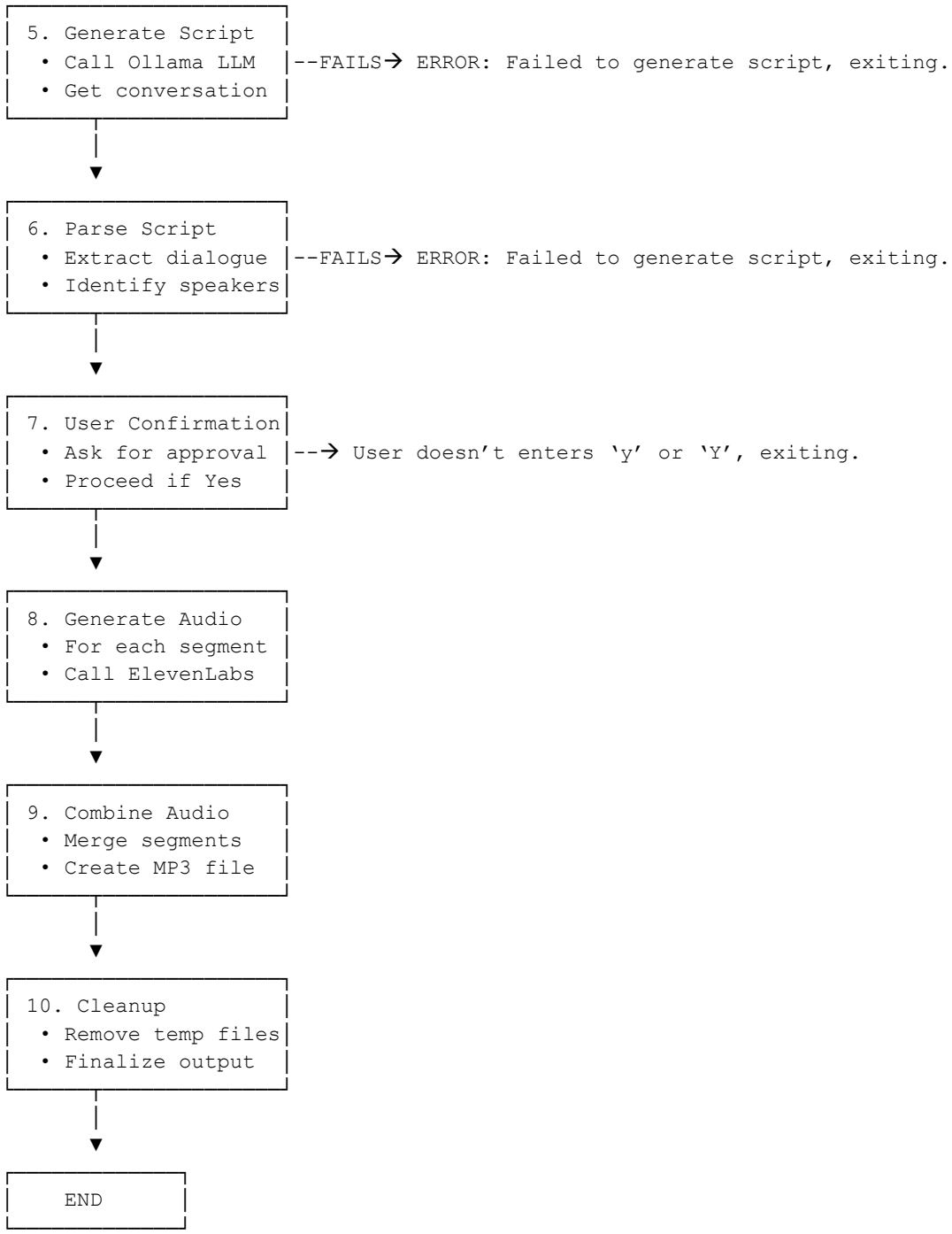
ElevenLabs TTS Engine: Converts text segments to audio with emotion support

Audio Combiner: Merges individual audio segments into final podcast

2.3 Flow Diagram

The system follows this execution flow:





Detailed Flow Steps:

Step 1: Input Validation: User provides topic → System validates topic is non-empty

Step 2: Ollama Connection Check: System verifies Ollama service is running and accessible

Step 3: Wikipedia Context Fetch: System retrieves topic summary from Wikipedia (max 2000 chars)

Step 4: Prompt Generation: System creates Hinglish conversation prompt with context and instructions

Step 5: Script Generation: Ollama LLM generates 2-minute Hinglish conversation script

Step 6: Script Parsing: System parses script into structured dialogue segments with speaker tags

Step 7: User Confirmation: System prompts user for audio generation confirmation

Step 8: Audio Generation: For each dialogue segment, ElevenLabs API generates audio

Step 9: Audio Combination: All audio segments are concatenated into final MP3 file

Step 10: Cleanup: Temporary audio files are removed

3. Setup and Deployment Instructions

3.1 Prerequisites

- Python 3.12 or higher
- Ollama installed and running (local or remote) with model 3.2
- ElevenLabs API key

3.2 Installation Steps

Step 1: Install Python Dependencies

Install required packages:

```
pip install wikipediaapi ollama requests python-dotenv
```

Step 2: Install and Configure Ollama

1. Download Ollama from https://ollama.com/

2. Install Ollama on your system

3. Start Ollama service:

```
$ ollama serve
```

4. Pull the required model:

```
$ ollama pull llama3.2
```

Step 3: Configure Environment Variables

Create a .env file with your ElevenLabs API key:

```
ELLEVENLABS_API_KEY=your_api_key_here
```

Note: Update the path in syntheticRadioHostScript.py line 17 to point to your .env file

Step 4: Verify Voice IDs

The system uses predefined voice IDs from ElevenLabs. Current configuration:

- Male Host: Mihir voice
- Female Host: Aura voice

Voice IDs are defined in the VOICES dictionary (lines 570-576). You can modify these to use different voices from your ElevenLabs account.

3.3 Running the Application

1. Ensure Ollama is running:

```
$ ollama serve
```

2. Run the main script:

```
$ python syntheticRadioHostScript.py <api-key> <output_file_path_name>
```

Command also works without any user entries as given below:

```
$ python syntheticRadioHostScript.py
```

In this case api-key will be read from env file at location

```
C:/Users/vineet.srivastava/venvGenAIStudy/.env
```

And output file will be `C:/LEARNING/LANGCHAIN/ollama_podcast.mp3`

3. The script will:

- Ask to use existing api-key for user confirmation if no api-key is provided.
- Fetch Wikipedia context
- Generate script using Ollama
- Parse dialogue segments
- Ask for user confirmation before audio generation
- Generate and combine audio files

3.4 Output Location

Default path for the final podcast MP3 file is:

```
C:/LEARNING/LANGCHAIN/ollama_podcast.mp3
```

This path can be modified in the main() function (line 646).

This path gets over-ridden with the user provided path once provided as an argument while running the script file.

4. Code Explanations and Assumptions

4.1 Core Functions

4.1.1 fetch_wikipedia_context()

Purpose: Retrieves topic information from Wikipedia.

Key Features:

- Validates topic is non-empty
- Checks if Wikipedia page exists
- Returns first 2000 characters of summary
- Handles API exceptions gracefully

Assumptions:

- Wikipedia API is accessible
- Topic exists in English Wikipedia

4.1.2 create_hinglish_prompt()

Purpose: Creates structured prompt for LLM with Hinglish conversation instructions.

Key Features:

- Includes phonetic spelling instructions
- Specifies emotional tags format
- Provides one-shot example for consistency
- Enforces 2-minute conversation length

Assumptions:

- LLM understands Hinglish and phonetic instructions
- Example format guides LLM output structure

4.1.3 check_ollama_connection()

Purpose: Verifies Ollama service is running and accessible.

Implementation:

- Attempts to list models using ollama.list()
- Returns True if response is not None
- Returns False on any exception

Assumptions:

- Ollama runs on default localhost:11434
- Network connectivity to Ollama service

4.1.4 generate_script_with_ollama()

Purpose: Generates Hinglish conversation script using Ollama LLM.

Key Features:

- Uses llama3.2 model by default
- Measures generation time
- Validates script length (minimum 50 characters)
- Returns tuple (script, elapsed_time)

Assumptions:

- llama3.2 model is available in Ollama
- LLM follows prompt instructions for format

4.1.5 parse_script()

Purpose: Parses generated script into structured dialogue segments.

Key Features:

- Handles both double newline (\n\n) and single newline (\n) separators
- Identifies hosts by name matching
- Assigns alternating [Indian English] and [Hindi accent] tags
- Handles intro/outro text and multi-line dialogues

Assumptions:

- Script follows "SpeakerName: text" format

4.1.6 generate_segment()

Purpose: Converts text segment to audio using ElevenLabs API.

Key Features:

- Uses eleven_v3 model for emotion support
- Detects emotional tags and adjusts voice settings
- Returns audio bytes on success (200 status)
- Returns None on API errors

Assumptions:

- Valid ElevenLabs API key with sufficient credits
- Voice IDs exist in ElevenLabs account

4.1.7 generate_podcast()

Purpose: Orchestrates complete podcast generation from dialogue.

Key Features:

- Sanity checks for inputs (API key, dialogue, voice IDs)
- Generates audio for each dialogue segment
- Saves temporary segments to disk
- Combines segments into final MP3
- Cleans up temporary files
- Handles partial failures gracefully

Assumptions:

- Sufficient disk space for temporary files
- Write permissions for output directory

4.2 Design Decisions

4.2.1 Why Ollama over Cloud LLMs?

Ollama was chosen for:

- Privacy: No data sent to external services
- Cost: Free for local usage

4.2.2 Why ElevenLabs for TTS?

ElevenLabs was chosen for:

- High-quality natural voices
- Emotion tag support for realistic conversations
- Indian accent support
- API-based integration for scalability

4.3 Error Handling Strategy

The system implements comprehensive error handling:

- Early validation of inputs
- Graceful degradation (returns None instead of crashing)
- User-friendly error messages

- Connection checks before API calls
- Partial failure handling (continues if some segments fail)

4.4 Limitations and Known Issues

- Hardcoded .env file path (line 49). Can be bypassed by providing eleven labs api key as an argument while running the script.
- Hardcoded output path (line 646). Can be bypassed by providing your system path as an argument while running the script.
- No retry mechanism for API failures
- No audio quality validation

5. Technology Stack

5.1 Core Technologies

Python 3.12+: Programming language

Ollama: Local LLM inference engine

llama3.2: LLM model for script generation

ElevenLabs API: Text-to-Speech service

Wikipedia API: Content source via wikipediaapi library

Audio File Format: MP3.

5.2 Python Libraries

- wikipediaapi: Wikipedia content retrieval
- ollama: Ollama LLM integration
- requests: HTTP requests for ElevenLabs API
- python-dotenv: Environment variable management
- re: Regular expressions for text parsing
- os: File system operations
- time: Timing and rate limiting