EchoVerse – An AI-Powered Audiobook Creation Tool

# Project Description

EchoVerse is a generative AI-based audiobook creation system that transforms user- provided text into expressive, downloadable audio content. Designed for accessibility, convenience, and content reusability, the tool empowers students, professionals, and visually impaired users to convert written material into natural-sounding narrations with customizable tone and voice.

The system accepts either pasted text or uploaded .txt files and displays the original content within the interface. Upon selection of a desired tone—Neutral, Suspenseful, or Inspiring— the text is rewritten using the IBM Watsonx Granite large language model. Prompt chaining is used to ensure the tone-specific rewrite remains faithful to the original meaning while enhancing stylistic quality.

Once rewritten, the text is passed to IBM Watson Text-to-Speech (TTS), where the user- selected voice—such as Lisa, Michael, or Allison—is used to synthesize high-quality narration. The final audio is streamed in-app using Streamlit’s audio playback component, and users are also given the option to download the file in .mp3 format.

A side-by-side layout showcases the original and rewritten text, supporting content verification and tone comparison. Additionally, a session-based “Past Narrations” panel displays all previously generated results, allowing users to replay or re-download audio within the same session. No login or persistent user tracking is implemented.

The tool is built using Python and Streamlit, with IBM Watsonx and TTS services integrated through API calls. Configuration variables such as API credentials and endpoints are secured using .env files. Though all voice styles worked successfully during testing, tone consistency was moderately affected by token limitations inherent in the current Watsonx deployment.

EchoVerse delivers a complete, low-friction workflow from text ingestion to expressive narration output—demonstrating how generative AI can enhance accessibility and user experience in content consumption.

# User Scenarios

EchoVerse is designed to support a range of realistic user workflows that involve converting textual content into expressive audio output. The interface offers flexibility in input methods, tone and voice customization, and real-time audio playback. The following scenarios demonstrate the applicability of EchoVerse across diverse use cases:

# Scenario 1: Study Notes to Audiobook

A student uploads a .txt file containing summarised notes for exam preparation. They select the "Inspiring" tone and choose the voice “Lisa” for narration. EchoVerse rewrites the content with a motivational tone and generates downloadable audio. The side-by-side display of original and rewritten text ensures the student can verify accuracy. The output enables learning through listening, especially during commuting or revision breaks.

# Scenario 2: Blog to Podcast Adaptation

A blogger pastes a written article into the app and selects the "Suspenseful" tone with the voice “Michael.” The system rewrites the text to enhance dramatic appeal and synthesizes a podcast-style narration. The user previews the audio, downloads the .mp3 file, and publishes it as a spoken alternative to the original post. EchoVerse thus serves as a content adaptation tool for multi-format engagement.

# Scenario 3: Accessibility for Visually Impaired Users

A visually impaired user pastes a passage from a speech into the interface. Choosing a neutral tone and a clear voice such as “Allison,” the user plays the narration within the app. The audio can also be downloaded for offline access. EchoVerse provides an independent and accessible reading experience, eliminating the need for external narration support.

# Scenario 4: Exploring Voice and Tone Variations

A user experiments by generating multiple narrations of the same content using different tone and voice combinations. Each output is saved under “Past Narrations” within the session. The user can compare previous results, replay them, or download preferred versions—supporting creative iteration without the need to re-upload or re-enter text.

# Technical Architecture

EchoVerse is organized into a modular, end-to-end pipeline that processes user-provided text into narrated audio. The system integrates tone rewriting using IBM Watsonx Granite LLM and voice synthesis using IBM Watson Text-to-Speech API, all accessible through a user-friendly Streamlit interface. The architecture includes the following key components:

# Input Layer

Users begin by either pasting plain text into a textbox or uploading a .txt file. The uploaded text is read and displayed in its original form in the interface. Streamlit widgets handle both direct input and file parsing. Once submitted, the content is stored in memory for processing through subsequent layers.

# Tone Rewriting Layer (LLM Integration)

After text input, users select a tone from available options: Neutral, Suspenseful, or Inspiring. A prompt is dynamically constructed using the selected tone and passed to the IBM Watsonx Granite model. The system uses prompt chaining to preserve the semantic meaning while adjusting stylistic features. The response from the LLM is parsed and displayed adjacent to the original text, allowing users to observe how the tone has been modified. Rewritten content is capped within token limits to ensure model compatibility and reliable generation.

# Text-to-Speech Layer (Audio Generation)

Once the rewritten text is prepared, users select a voice from IBM’s TTS options such as Michael, Lisa, or Allison. The rewritten content is then sent to the IBM Watson Text-to-Speech API. The resulting audio stream is returned in real time, rendered via Streamlit’s built-in audio player, and also made available for download in .mp3 format. Audio is cached in memory for current session reuse.

# Session Management and History Tracking

EchoVerse maintains session-level state to track generated outputs. A “Past Narrations” panel displays all audio previously created during the active session. Each entry includes the selected voice, tone, and the associated rewritten text. These entries can be played back or downloaded again, allowing users to compare variations or revisit earlier outputs without repeating input steps. Persistent user accounts are not implemented; all data is cleared once the session ends.

# Frontend and Interface Layer (Streamlit)

The user interface is constructed using Streamlit layout elements such as containers,

columns, and selection widgets. It includes components for text input or upload, tone and voice selection, a dual-column display for original and rewritten text, audio playback, download buttons, and session history. The layout is optimized for real- time interactivity with minimal navigation and clearly labeled actions. Styling is handled through inline HTML and CSS for background images and layout control.

# Environment Configuration and Security

API credentials such as IBM\_API\_KEY, IBM\_PROJECT\_ID, and service endpoints are stored in a .env file. These variables are loaded securely using the python-dotenv package and used to initialize the Watsonx and TTS clients. This configuration ensures separation of secrets from application logic and supports secure local development.

# Pre-requisites

* + **Python Programming**: <https://docs.python.org/3/>

Core programming language used for backend logic, API integration, and Streamlit app development.

* + **Streamlit Framework**: <https://docs.streamlit.io/>

Used to build the user interface, manage real-time input/output interactions, and render audio playback.

* + **IBM Watsonx.ai (Granite Foundation Model)**: <https://www.ibm.com/cloud/watsonx>

Used for rewriting text in user-specified tones using prompt-chained LLM calls.

* + **IBM Watson Text-to-Speech (TTS)**: https://cloud.ibm.com/apidocs/text-to- speech

Used to convert rewritten text into expressive audio using selected voice styles.

* + **IBM Watson Machine Learning SDK (Python)**: [https://ibm.github.io/watson-](https://ibm.github.io/watson-machine-learning-sdk/foundation_models/) [machine-learning-sdk/foundation\_models/](https://ibm.github.io/watson-machine-learning-sdk/foundation_models/)

Provides Python-based access to Watsonx models and TTS API through secure credential-based authentication.

* + **Python Dotenv for Environment Variables**: [https://pypi.org/project/python-](https://pypi.org/project/python-dotenv/) [dotenv/](https://pypi.org/project/python-dotenv/)

Used to securely load IBM credentials and configuration parameters from a .env file.

* + **Base64 Encoding (Python Standard Library)**: <https://docs.python.org/3/library/base64.html>

Used to embed and style the background image of the Streamlit interface.

* + **Local Development Environment**: Visual Studio Code –

<https://code.visualstudio.com/>

Preferred editor used during development and debugging of the application. Compatible with Windows 11.

* + **Command Line Execution**:

All components executed locally via the terminal using streamlit run main.py.

# Project Workflow

The EchoVerse system was developed using a milestone-based workflow to ensure modularity, clarity, and verifiability at each stage of the pipeline. The methodology followed a bottom-up approach—starting from backend integration of AI services, followed by layered enhancements in logic, interface design, and session control. The system evolved through four major milestones.

# Milestone 1: IBM Watsonx Integration and Tone Rewriting Engine Activity 1.1 – Credential Setup and API Client Configuration

The IBM Watson Machine Learning SDK (ibm-watsonx-ai) was installed and integrated using environment variables stored in a .env file. Variables such as IBM\_API\_KEY, IBM\_PROJECT\_ID, and IBM\_URL were loaded using python-dotenv and injected into the LLM client configuration. Secure handling of credentials enabled consistent and authenticated access to Watsonx services.

# Activity 1.2 – Prompt Chaining with Tone Control

The system incorporated dynamic prompt chaining logic to guide the Granite LLM in rewriting input text according to a selected tone. A standardized prompt format was constructed for each tone, embedding the original text along with clear instructions (e.g., “Rewrite the following in a suspenseful tone while preserving meaning”). This allowed stylistic variation without semantic drift. Token truncation safeguards were added to ensure responses stayed within model limits.

# Activity 1.3 – Tone-Specific Rewriting and Output Rendering

The prompt was submitted to Watsonx Granite using the generate\_text() call, with decoding settings set to ensure factual and moderately creative responses (greedy decoding, temperature 0.5, token limit 300). Returned results were parsed, formatted, and displayed alongside the original input in the Streamlit interface, using a responsive dual-column layout for comparison.

# Milestone 2: IBM Watson TTS Integration and Voice Control Activity 2.1 – Voice Selection and API Integration

Voice options—such as Lisa, Michael, Allison, and Kate—were made available through a Streamlit dropdown menu. On generation trigger, the rewritten text and selected voice ID were passed to the IBM Text-to-Speech endpoint using IAM-authenticated HTTP POST requests. The system dynamically embedded voice parameters into each API request.

# Activity 2.2 – Audio Stream Handling and MP3 Conversion

The audio returned from IBM TTS was received as a binary audio stream and written into an in-memory MP3 format. This audio was then rendered using Streamlit’s built-in audio player component (st.audio()) for immediate playback. Additionally, the MP3 stream was linked to a st.download\_button allowing direct file download, with no need for filesystem writes.

# Activity 2.3 – Error Management and Fallback Responses

Voice generation failures—such as empty input, malformed responses, or invalid characters—were caught using try/except logic. Appropriate error messages were displayed within the interface, prompting users to recheck their input or reduce content size. This validation ensured that the interface remained stable and fail-safe across different use conditions.

# Milestone 3: Streamlit UI Design and Feature Assembly Activity 3.1 – Layout Design and Sectionalization

The interface was divided into distinct logical blocks using Streamlit layout components: Title Section (logo and app name), File Upload and Text Input, Tone and Voice Selection, Rewriting Display, Audio Playback, Download, and Past Narrations. A fixed-width, center- aligned layout was used for uniformity, and each section was separated using containers and headings for readability.

# Activity 3.2 – File Upload Logic and Parsing

The file upload module allowed users to drag-and-drop or browse .txt files. On upload, the file was read using UTF-8 encoding and processed into plain string format. Large files were automatically trimmed to avoid LLM token overflow, and content preview was shown to the user. Uploaded data and manually pasted text were processed identically in downstream steps.

# Activity 3.3 – Dual-View Rewriting Display

To facilitate comparison between original and AI-rewritten content, a two-column layout was implemented. The left column displayed the original input, and the right showed the rewritten version. This format allowed real-time visual inspection of stylistic transformations and helped confirm that essential meaning had been preserved.

# Milestone 4: Session State Handling and Narration History Activity 4.1 – Narration Storage in Streamlit Session State

Each time a user generated audio, the corresponding text, voice, tone, and audio stream were stored in a session-scoped dictionary within st.session\_state. This design allowed persistence of output during the session and enabled users to revisit previous narrations instantly without reprocessing.

# Activity 4.2 – Past Narrations Display and Interaction

An expandable “Past Narrations” panel was implemented, displaying all previous outputs in reverse chronological order. Each entry included rewritten text, an audio playback component, and a repeat download option. The panel allowed users to compare results across tones and voices, supporting creative and educational use cases.

# Activity 4.3 – Final Integration and Cross-Component Testing

All modules—rewriting, TTS, UI, session state—were integrated and tested end-to-end. Multiple use cases were validated, including tone-voice switching, large file upload, and rapid reprocessing. Final debugging focused on exception handling, dynamic styling, and component coordination. The system was confirmed to be fully functional and visually stable across multiple sessions and text lengths.

# Project Output

The completed EchoVerse system provides a streamlined, responsive interface for converting raw text into downloadable audiobooks using AI-based tone rewriting and voice synthesis. The output spans multiple components—each tightly integrated to support usability, expressiveness, and session continuity. The following components reflect the actual, working output as implemented and tested during the final deployment.

# Component 1: Text Input and File Upload

At the top of the interface, users are presented with two options: drag-and-drop .txt file upload or direct text input. The file uploader accepts plain text files up to 200MB in size, and visual feedback is provided to confirm successful parsing. If users prefer manual input, a dedicated textbox allows them to paste any content for transformation. Both options are processed in real time and serve as the base for downstream rewriting and synthesis tasks. The layout prioritizes ease of access for all user types—whether uploading large reading material or experimenting with short snippets.



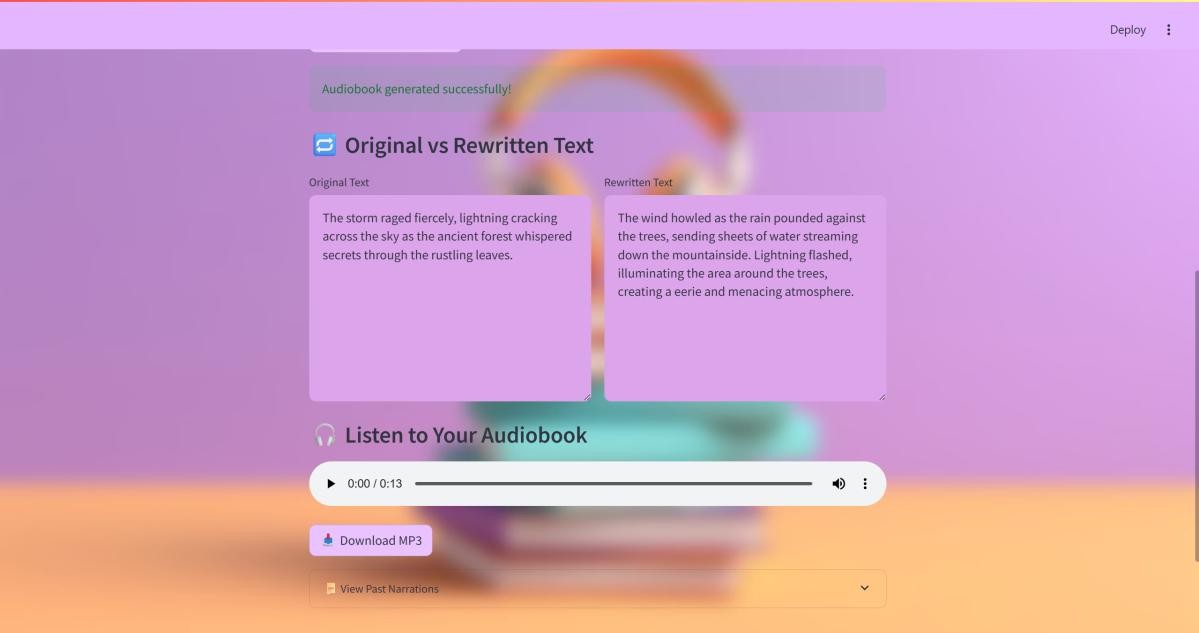
# Component 2: Tone and Voice Selection

Users can then select their preferred narrative tone (Neutral, Suspenseful, or Inspiring) and voice style (Lisa, Michael, Allison, Kate) via dropdown menus. These selections control the behavior of the Granite LLM and Watson TTS pipelines, respectively. The dropdowns are designed to retain selection states and clearly reflect the active choices. Internally, these values are used to construct the tone-specific prompt and determine the final narration voice, allowing users to experiment with various tone–voice pairings to match mood or context.

# Component 3: Original vs Rewritten Text Display

After processing is initiated via the “Generate Audiobook” button, the system rewrites the original input text according to the selected tone using Watsonx Granite LLM. The resulting rewritten version is displayed side-by-side with the original text in a dedicated dual-column block titled “Original vs Rewritten Text.” This layout allows the user to visually compare the structure, mood, and phrasing of both versions, and confirm that the emotional content has been successfully adjusted while maintaining semantic accuracy.



# Component 4: Audio Playback and MP3 Download

Once the text is rewritten, it is passed to the IBM Watson Text-to-Speech API, where the selected voice synthesizes a narrated version of the content. The audio output is returned as a binary .mp3 stream, which is rendered using Streamlit’s st.audio() player. Below the

player, a “Download MP3” button is made available for saving the file locally. This component is critical for users seeking offline access, such as students listening to notes on the move or creators embedding audio into external platforms. The playback functionality ensures that all output is immediately previewable without needing external tools.

# Component 5: Past Narrations Panel

EchoVerse tracks all narrations generated during a session and lists them under the “Past Narrations” expandable panel. Each narration record includes the rewritten text, the voice and tone used, a replayable audio player, and a re-download option. The feature enables comparative listening between different tones and voices applied to the same base input. It also enhances user productivity by eliminating the need to reprocess or re-upload content. Session state is maintained during runtime, though no long-term user tracking or login is implemented.

# Component 6: Visual Layout and Theming

The overall interface is designed with visual clarity and modern aesthetics in mind. A pastel- toned background image is embedded using base64 encoding and applied as a fixed, full- screen visual layer. All components are placed within semi-transparent Streamlit containers using carefully adjusted padding, margins, and font sizing. The color scheme and visual hierarchy ensure that critical operations—such as file input, tone selection, and audio playback—are easy to locate and operate. This design promotes user engagement while maintaining consistency across sessions.

# Component 7: System Responsiveness and Testing

During testing, the entire pipeline—from input to narration—was observed to complete within 6–8 seconds for standard-length text inputs. The tone rewriting via Watsonx Granite returned results with reasonable fidelity across all tones, though longer texts were occasionally truncated due to token limitations. Voice synthesis was consistently stable across all available voices, and audio download functionality worked without error. Edge cases like empty input, unsupported file types, and API timeouts were handled using built- in alerts and fallback messages, maintaining robustness under varied conditions.

# Conclusion

The EchoVerse project successfully demonstrates how generative AI can be applied to transform static text into expressive, customizable audio content. By integrating IBM Watsonx Granite for tone-aware rewriting and IBM Watson TTS for high-quality voice synthesis, the system delivers a complete pipeline for audiobook creation accessible to users with minimal technical knowledge.

From a functional perspective, the application achieved all primary goals: it allows users to input or upload text, apply stylistic tone transformations using an LLM, and generate downloadable narration in selected voice styles. The modular backend handled prompt chaining, voice rendering, and file processing with stability, while the frontend provided a clean, responsive interface for real-time interaction.

Session-level tracking of past narrations allowed users to experiment freely with tone and voice combinations, supporting creative use cases such as study material narration, content publishing, and accessibility enhancement. Voice generation was consistent across all supported styles, and while tone fidelity was affected in longer inputs due to model token constraints, the system maintained coherence and accuracy within practical limits.

Visually, the interface incorporated a stylized background, thematic color palette, and structured layout—contributing to usability and a polished user experience. The use of Streamlit accelerated development without compromising customizability.

In summary, EchoVerse meets its core objective of bridging AI-generated rewriting and voice synthesis to create an intuitive, expressive audiobook creation tool. It showcases the real-world potential of multimodal generative AI and stands as a strong example of accessible, AI-driven content transformation.