OTTER

Open Text Transcription Editing Resource

Table of Contents

Problem Statement

Editing spoken-word audio remains a challenging and finicky task. Traditional audio editing tools require users to scrub through waveforms measured in milliseconds, repeatedly listen to recordings, and visually interpret amplitude patterns in order to locate semantic content. This workflow is inefficient, difficult to learn, and poorly suited to speech-centric tasks such as podcast production, interviews, lectures, and research recordings.

Recent commercial tools, such as **Descript** and **Trebble**, demonstrate a more effective paradigm: transcript-driven editing, in which users edit text and the corresponding audio is modified automatically. These systems significantly reduce cognitive load by allowing users to work in terms of words, phrases, and sentences rather than raw time. However, these solutions are proprietary, cloud-first, and opaque. They require uploading media to external services, restrict offline use, and do not expose their internal data models or editing representations. As a result, they are unsuitable for privacy-sensitive recordings, reproducible research, and educational contexts where system transparency is essential.

Despite the availability of open-source speech recognition models capable of producing accurate word-level timestamps locally, there is currently no free and open-source system that integrates transcription, transcript-driven editing, and non-destructive audio manipulation into a cohesive, local-first editor. This gap limits access to modern editing workflows and prevents students and researchers from studying, extending, or adapting transcript-based media editing systems.

This project addresses that gap by designing and implementing **OTTER**, an open-source, local-first transcript-driven audio editor in which the transcript serves as a first-class editing surface. By explicitly linking text, time, and media through a shared timeline model, **OTTER** aims to make speech editing more accessible while also serving as an educational platform for exploring the intersection of human–computer interaction, media systems, and software architecture.

High-level system summary

OTTER is a free and open-source, local-first audio editor designed around a **transcript-first editing model**, in which text, not waveforms, is the primary representation of spoken-word media. Users edit recordings by selecting, deleting, or rearranging words and phrases in the transcript, and the system applies the corresponding non-destructive edits to the underlying audio. Audio waveforms and playback controls are presented as contextual aids, reinforcing alignment and precision without requiring users to reason directly about timestamps or samples.

At the core of **OTTER** is a **shared timeline model** that explicitly links transcript tokens to audio time ranges and editing operations. This model serves as the single source of truth for:

- the transcript view (timestamps and text-based navigation)
- the audio view (contextual waveform visualization and playback feedback)
- a non-destructive edit representation (an edit decision list or edit graph) that preserves the original media while enabling reversible, efficient, and reproducible edits.

OTTER is implemented as a desktop application to support reliable offline operation, local computation, and direct access to media resources. The system architecture consists of:

- a desktop user interface and local orchestration layer (e.g., Electron)
- a local transcription component based on open-source Whisper-family models to generate word-level timestamps
- a local media pipeline using FFmpeg for decoding, preview rendering, and final export, and
- a local project data layer that stores transcripts, timing metadata, edit operations, and media references.

This design prioritizes accessibility, transparency, and educational value, enabling users to work with spoken-word media at the level of language while providing students with a clear, inspectable model of how text, time, and audio are integrated in modern transcript-driven editing systems.

Proposal

Problem statement

Audio editing is time-consuming and skill-gated: conventional tools require users to locate content by listening and visually scanning waveforms. Modern tools like Descript/Trebble demonstrate a better workflow—**edit the transcript, and the media follows**—but these solutions are commercial and typically cloud-first, limiting accessibility, privacy, reproducibility, and extensibility for research/education.

Proposed solution

Build a locally hosted, FOSS application that:

1. Ingests audio files
2. Produces a transcript with word-level timestamps
3. Allows users to edit the transcript (delete/move text ranges)
4. Applies those edits to the audio non-destructively
5. Exports an edited audio file (and an edit list)

Target users

- Podcasters/students/educators editing spoken-word recordings or audio tracks from videos
- Researchers needing reproducible transcript-aligned edits
- Privacy-conscious users who cannot or do not wish to upload recordings to cloud services

Key features (MVP)

- Import audio (WAV/MP3/M4A)
- Local transcription with word timestamps
- Transcript view with word highlighting during playback
- Transcript-based edits:
  - delete selection
  - ripple delete behavior (close gaps)
  - undo/redo
  - Nudge/adjust word boundaries
- Waveform view synced to transcript
- Export edited audio via FFmpeg

Stretch goals (optional)

- Functionality
  - Auto-refine word boundaries
  - Intelligent “filler word” detection + one-click remove (um, ahh, etc.)
  - Crossfade smoothing at cut boundaries
  - Add recorded audio to existing tracks (provide new words that can be used to replace existing text).
- Much larger chunks
  - Speaker diarization & labeling
  - Video support (same transcript timeline)
  - Plugin API (effects, NLP cleanup, chaptering)
  - Collaboration (project file merge), not real-time multiuser
  - Add synthesized (cloned) audio to existing tracks based on new/changed words in the transcript.
- Optimizations:
  - Detailed Profiling
  - C++ implementation of relevant sections

Deliverables

- Well-organized GitHub repository using permissive open source license, e.g., MIT/Apache-2. Model license documented.
- Documentation:
  - architecture
  - build instructions
  - dataset/evaluation procedure
- Working desktop app (multi-platform if possible)
- Demo video + final report (metrics + user study)

Success criteria

Users can import audio → generate transcript → delete/move text → hear immediate preview → export final audio

- Transcript and waveform stay synchronized with <100ms drift in normal use
- Editing operations are reversible (undo/redo) and non-destructive

- Application runs fully offline after installation

Scope for one-semester build

Scope of operations for MVP

- Single track, single speaker
- Delete a transcript selection → remove audio segment(s)
- Move a transcript selection → reorder corresponding audio
- Undo/redo for all edits

Non-goals

- Diarization
- Studio-grade restoration (noise reduction UI)
- Automatic punctuation/casing perfection
- A full Descript clone
- Mobile apps

Recommended technology options to reduce risk

- Electron + React for UI
- Node for orchestration
- FFmpeg for audio manipulation
- Local Python service for transcription (open Whisper for simplicity)

Starting Point

The sponsor will provide a functional Proof-of-Concept (POC) which demonstrates how text transcription, audio playback, and timeline synchronization can work together with no cloud services or closed-source dependencies. It is implemented as a desktop app using Electron, with a JavaScript-based UI and a locally invoked transcription backend. This approach is suggested for the actual implementation also. The PoC also demonstrates possible architectural choices for modularity and expandability.

GitHub Repository: otter-poc

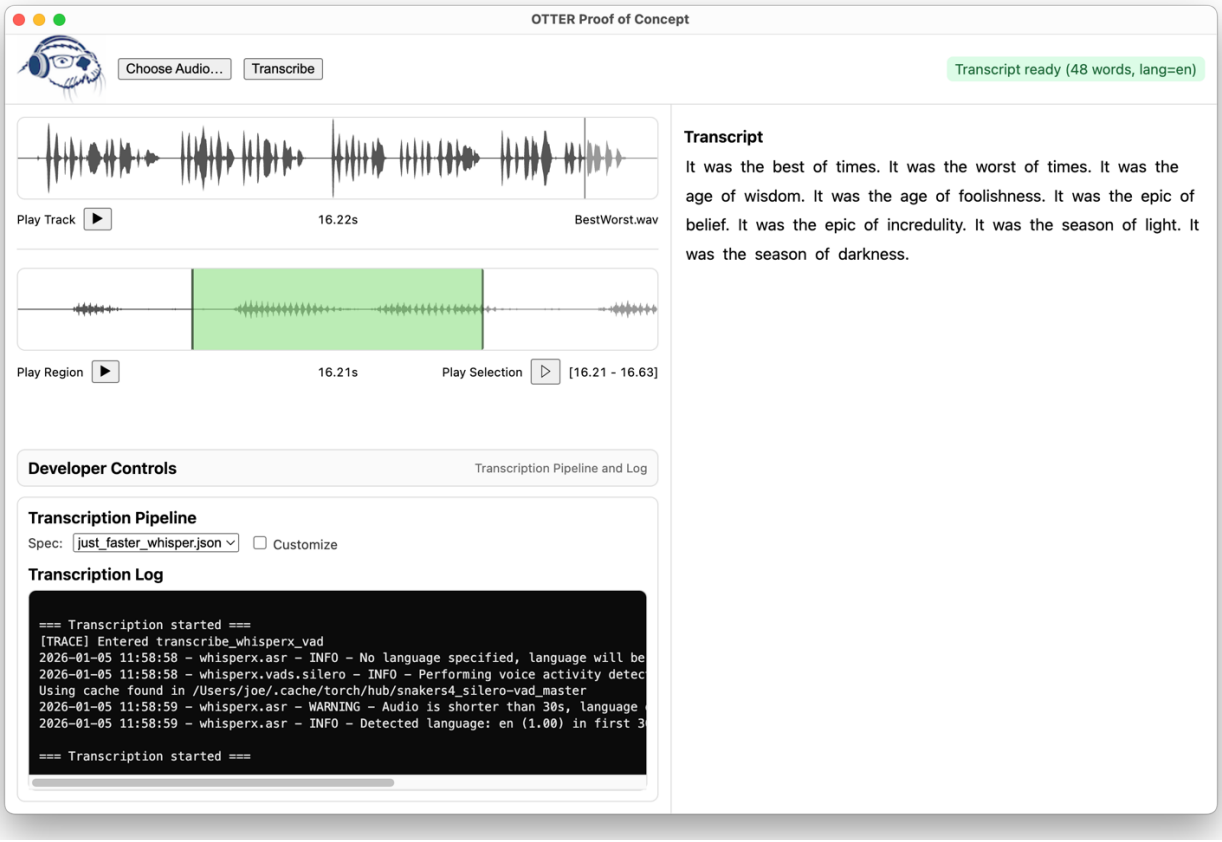

Figure 1- PoC Interface

Team Structure Options

Minimum viable team: **8 students**

The team will ultimately be responsible for dividing the project up amongst themselves and balancing the workload as the project progresses. More students may be added to the project by extending into the stretch goals (see above).

Suggested role split:

- (3) UI/UX design, specification, and implementation of the Transcript editor. Usability testing.
- (2) Waveform + playback sync
- (2) Transcription + local services (Whisper runner, job queue, caching)
- (1) Media pipeline + export (FFmpeg, project format, integration tests)

- ALL will be expected to perform:
  - Contribute to user stories
  - Testing/QA
  - Documentation
  - Intra-team coordination / communication

Sample Week-by-week implementation plan

Assuming 15 weeks; compress if needed.

Week 1 — Project kickoff

- Finalize requirements + non-goals
- Choose stack (Electron+React recommended)
- Repo setup: CI, formatting, basic issue tracker
- Define project file format (JSON) and timeline model
- Draft initial high-level user stories (~10)

Week 2 — End-to-end “thin slice”

- Import audio → FFmpeg decode → play audio
- Generate waveform peaks (basic)
- Integrate local transcription service stub (even mocked)
- Refine user stories

Week 3 — Transcription integration

- Whisper-based transcription running locally
- Produce transcript with word timestamps
- Store transcript + timing in project file

Week 4 — Transcript UI (read-only) + playback sync

- Transcript rendering with word highlighting during playback
- Click word → seek playback
- Basic search in transcript

Week 5 — Timeline model + segment mapping

- Implement non-destructive edit model:

    - segments = list of (source_start, source_end) in order
- Map words → segments reliably
- Unit tests for mapping

Week 6 — Editing op 1: delete selection

- Select words range → delete → rebuild segments
- Ripple behavior (close gap)
- Undo/redo infrastructure starts

Week 7 — Editing op 2: move/reorder selection

- Cut/paste transcript range → reorder segments
- Handle edge cases (partial word boundaries)
- Update transcript view + waveform markers

Week 8 — Waveform UI integration

- Show waveform timeline aligned to edited audio timeline
- Selection in waveform highlights transcript (basic)
- Scrub + zoom/pan

Week 9 — Preview engine

- Smooth playback across multiple segments
- Optional micro-crossfade at boundaries
- “Play from cursor” correctness tests

Week 10 — Export pipeline

- Render edited audio via FFmpeg using segment list
- Export formats: WAV + MP3 (or M4A)
- Export EDL-like JSON for reproducibility

Week 11 — Reliability + UX pass

- Project save/load robustness
- Crash recovery basics
- Better selection behavior, keyboard shortcuts (delete, undo, redo)

Week 12 — Evaluation preparation

- Define test dataset (5–10 varied recordings)

- Create scripted tasks for usability study
- Instrumentation: timing, errors, subjective ratings

Week 13 — Usability testing + iteration

- Run sessions (5–10 participants)
- Prioritize fixes: discoverability, sync issues, export failures

Week 14 — Polish + documentation

- Installers/build artifacts
- README, architecture diagrams, contributor guide
- “Demo script” rehearsal

Week 15 — Final demo + report

- Live demo: import → transcribe → edit text → export
- Present metrics and lessons learned
- Backlog for future cohorts

Demonstrating and evaluating success

Demo Target

- Import a 3–5 minute spoken-word recording
- Run local transcription (or use precomputed transcript to save time)
- Click words to seek; playback highlights words
- Delete a paragraph by selecting text
- Move a sentence earlier (cut/paste)
- Export edited audio
- Play before/after + show exported file duration matches expected

Objective metrics

- **Transcript↔audio sync accuracy:** average absolute error of word highlight vs audio (sample-based or manual spot-check)
- **Edit correctness:** cut boundaries are within X ms of intended word spans
- **Latency:** time from edit action to audible preview start
- **Export success rate:** % exports that match expected duration + no corruption
- **Performance:** waveform generation time; memory usage; startup time

Usability testing

5–10 participants, 3 tasks:

- Remove filler section
- Reorder two sentences
- Export and verify

Record:

- task completion rate
- time on task
- number of “confusion events”
- System Usability Scale score or a short Likert survey

Sample MVP User Stories

Story 1: Import Audio

As a user, I want to import a spoken-word audio file into the application, **so that** I can begin editing it.

Acceptance Criteria

- The user can select a local audio file (e.g., WAV, MP3, M4A).
- The audio loads successfully and can be played back.
- The project remembers the audio file reference.
- Basic transport controls (play/pause/seek) work.

Story 2: Generate Local Transcript

As a user, I want to generate a transcript of the audio locally, **so that** I can edit the recording using text.

Acceptance Criteria

- Transcription runs entirely on the local machine.
- The transcript includes word-level timestamps.
- Progress is visible during transcription.
- The resulting transcript is stored with the project.

Story 3: Transcript-Based Navigation

As a user, I want to click on any word in the transcript and hear the audio from that point, **so that** I can navigate by reading instead of scrubbing.

Acceptance Criteria

- Clicking a word seeks playback to the correct timestamp.
- Playback begins with minimal latency.
- The active word is visually highlighted.
- The transcript auto-scrolls during playback to keep the cursor visible.

Story 4: Transcript-Driven Deletion

As a user, I want to delete words or sentences directly from the transcript, **so that** the corresponding audio is removed automatically.

Acceptance Criteria

- Selecting text and pressing *Delete* removes it from the transcript.
- Playback skips the corresponding audio segment.
- Remaining audio plays continuously (no silence gaps).
- The original audio file remains unchanged.

Story 5: Non-Destructive Editing with Undo

As a user, I want all edits to be undoable, **so that** I can experiment without fear of losing data.

Acceptance Criteria

- All transcript edits are non-destructive.
- Undo and redo restore previous transcript and audio states.
- Undo/redo works across multiple edits.
- The system never modifies the source media.

Story 6: Contextual Waveform Visualization

As a user, I want to see a waveform aligned with the transcript, **so that** I can visually confirm timing and pauses without editing by waveform.

Acceptance Criteria

- A waveform is displayed contextually alongside transcript regions.
- The waveform highlights the currently playing segment.
- Waveform view stays synchronized with transcript playback.
- Waveform is read-only in MVP.

Story 7: Reordering Transcript Sections

As a user, I want to move a selected sentence or paragraph to a different position in the transcript, **so that** I can rearrange the audio content.

Acceptance Criteria

- Transcript text can be cut and pasted.
- Audio playback reflects the new ordering.
- Word-to-audio alignment remains correct.
- Operation is undoable.

Story 8: Save and Load Projects

As a user, I want to save and load my project, **so that** I can stop working and later resume editing without losing any progress.

Acceptance Criteria (Clear and Testable)

- The application allows the user to save the current project state to disk.
- The saved project captures all essential state, including:
  - Reference to the source media file(s)
  - Transcript data (including word-level timing)
  - Any user-refined word boundaries or regions
  - Edit decisions made so far (e.g., cuts, rearrangements, muted sections)
- A previously saved project can be loaded at a later time.
- After loading, the project appears in the same logical state as when it was saved:
  - Transcript content and ordering are preserved
  - Refined boundaries and selections are restored
  - The editor timeline reflects all prior edits
- Saving and loading does not modify the original media files.

Story 9: Export Edited Audio

As a user, I want to export the edited audio to a standard format, **so that** I can use it outside the application.

Acceptance Criteria

- The edited result can be exported (e.g., WAV or MP3).
- Export reflects all transcript edits.
- Export completes successfully for multi-minute audio.
- The exported duration matches the edited transcript.

Technology Stack and Licensing

OTTER is implemented as a local-first desktop application built using modern web technologies, allowing students to focus on interaction design and system architecture while maintaining reliable access to local compute and media resources. The user interface is developed using **JavaScript/TypeScript**, standard **HTML/CSS**, and a modern UI framework such as **React**, packaged as a desktop application using either **Electron**. This approach enables a document-style, transcript-first editing interface while providing controlled access to the local filesystem, background processing, and audio playback. Inter-process communication mechanisms (e.g. **Electron IPC**) are used to coordinate between the user interface and local services responsible for transcription and media processing.

Speech-to-text transcription is performed entirely on the user's machine using open-source **Whisper**-family models, producing transcripts with word-level timestamps suitable for fine-grained, transcript-driven editing. **Python** may be used to drive the Whisper model. Audio decoding, preview rendering, and final export are handled locally using **FFmpeg**, enabling support for common audio formats and non-destructive editing workflows. Waveform visualization will use a package such as **wavesurfer.js**. Project state is stored locally in a structured project file (**JSON**) that captures transcript data, timing metadata, and edit operations rather than modifying source media directly. All components are selected to ensure offline usability, transparency, and reproducibility, while exposing students to real-world tools and design challenges encountered in modern media and human–computer interaction systems.

OTTER will have a permissive Open Source License such as MIT or Apache 2.0. It will document use of third-party licenses and models. It will not require any proprietary services nor have any cloud dependencies.

Technology Summary:

- JavaScript/TypeScript, Python
- HTML/CSS
- React
- Electron / Electron IPC
- Whisper
- FFmpeg
- Wavesurfer.js
- JSON

Skills

Existing Skills

These are prerequisites to ensure the team can make progress and come to a successful completion. Along with these skills, success will depend on ***enthusiasm, motivation, and communication***.

1. Core Programming & Software Engineering

- Proficiency in JavaScript/TypeScript, Python, or C++
- Ability to read and understand unfamiliar codebases
- Experience with Git and collaborative development
- Basic debugging and problem decomposition skills

2. Basic Web or UI Development

- Familiarity with HTML/CSS and event-driven programming
- Experience with at least one UI framework or toolkit
- Comfort building interactive interfaces
- Prototyping and Evaluation

2. UX, Human Factors Experience, User-Centered Design

- User-Centered Design (UCD) Principles
- Persona and Scenario Development
- User Story Creation and Management
- UI/UX Design Fundamentals
- Workflow and Interaction Design

3. Data Structures & Program State

- Understanding of structured data (arrays, objects, maps)
- Ability to reason about application state
- Experience passing data between components or modules

4. Applied Software Engineering Skills

- Communication and Documentation
- Ability to work from specifications and user stories
- Willingness to learn new tools and libraries

- Comfort asking questions and iterating on designs
- Ability to explain code and design decisions to others

Learning Objectives

These are **explicit learning outcomes** of the project and do not need to be known in advance.

1. Transcript-Driven Editing Models

- How word-level timestamps link text to audio
- Designing and maintaining a shared timeline model
- Understanding non-destructive editing concepts
- Representing edits as transformations rather than mutations

2. Media Systems & Audio Pipelines

- How digital audio is decoded, played, and rendered
- Using FFmpeg as a media processing tool
- Handling long-running media operations
- Managing performance and resource constraints

3. Applied Machine Learning Integration

- Running pre-trained speech recognition models locally
- Interpreting model outputs (timestamps, segments)
- Treating ML components as services rather than research artifacts
- Managing model size, performance, and tradeoffs

4. Desktop Application Architecture

- Structuring desktop apps using web technologies
- Managing privileged vs UI processes
- Inter-process communication
- Packaging and distributing applications

5. Synchronization Across Multiple Views

- Keeping transcript, waveform, and playback state consistent
- Defining a single source of truth
- Handling edge cases in time-based interaction
- Debugging synchronization errors

6. Human-Centered Design & Evaluation

- Translating user stories into usable interfaces
- Designing keyboard-driven and document-style workflows
- Conducting basic usability testing
- Interpreting feedback and iterating on design

11. Development Practices

- Writing documentation for external contributors
- Licensing and dependency awareness
- Designing for extensibility and transparency
- Communicating technical decisions clearly

FAQ

Can we choose our own name for the project, replacing **OTTER**?

- Yes. Please run the new name by the Sponsor before making the change official.
- You will also need to update the artwork associated with the project.

Can we change the artwork associated with the project?

- Of course. Consider it a placeholder.
- Please run the new artwork by the Sponsor before making the change official.

Who will we be working with on the project?

- The Sponsor is Joe Pasqua who is a member of the Industrial Advisory Board for the Computer Science Departments of CSUMB and Cal Poly. He has sponsored multiple Capstone projects at both universities.
- You can read more about his background on **LinkedIn**.

Will we be starting from scratch?

- No, the sponsor will provide a functional Proof-of-Concept (PoC).
- You can use this to help understand some of the underlying concepts and architecture.

Can we use existing code or subsystems?

- Yes! The team is encouraged to focus on the core value of the project and use applicable subsystems where possible.
- For example, you don't need to train an automatic speech recognition (ASR) model. You'll use an existing one.

Can we use AI?

- Yes! However, this isn't an exercise in vibe coding. You can use AI tools to help you learn new tools and libraries, but you are responsible for the code and architecture. If I ask you how something works or why you chose to do something a particular way, it's up to you to be able to explain it in detail.

How hard will this project be?

- Every Capstone project should be a challenge and **OTTER** is no different. However, the primary goal of the project is to be a good learning experience. We will tune the project to be sure we meet that goal.
- The project has sufficiently broad scope that we can scale the overall deliverables to make the project manageable in the time allowed, or expand it to include stretch goals.

How will we get started? In our first meeting we will:

- Introduce ourselves.
- Review the project proposal.
- Run through the pre-existing Proof-of-Concept app
- Start thinking about who will do what.
- Discuss logistics such as setting up a slack channel and github repo.