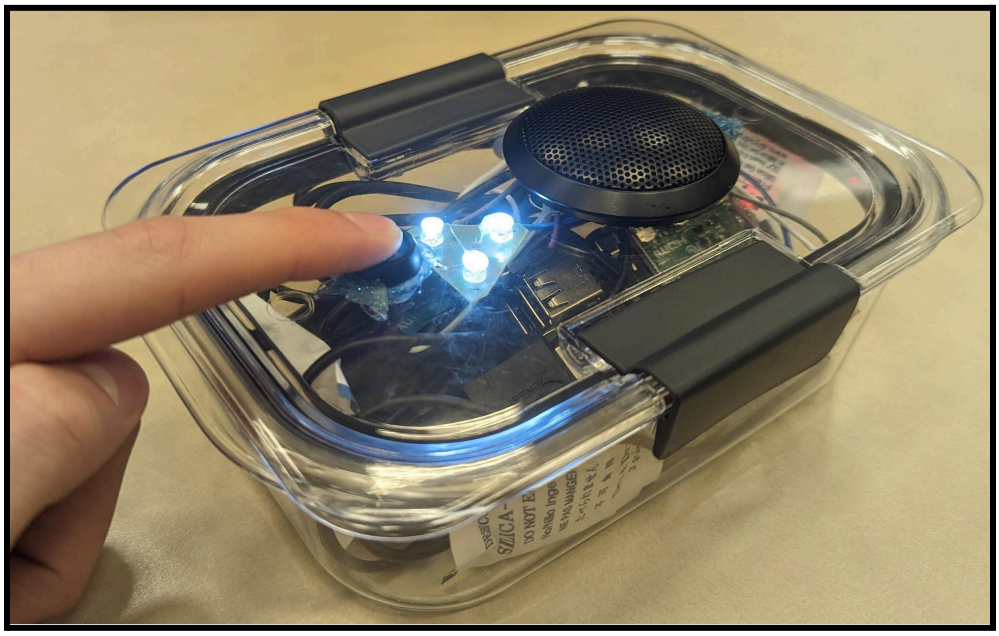
ShowerScribe — Final Report

Team number: 14 **Date**: Nov 26, 2023
Alex Aumais, Victoria Mazilu, Aayush Patel, Tyler Chen, Vincent Chung, Balaji Leninrajan
Code repository: https://git.uwaterloo.ca/aaumais/shower-scribe



Abstract

Sometimes, thoughts spontaneously appear while we are taking a shower. It then presents a challenge to hold onto that information until we come out of the shower and manually record it.

ShowerScribe aims to provide a service to users who have many thoughts in showers and would like to record those thoughts, whether random or not, for future purposes. ShowerScribe will create audio recordings of those intriguing thoughts with a simple button click. ShowerScribe additionally categorizes those ideas so that innovative and mind-blowing ideas will jump out instantly to the users. Each recording is given a generated title and recap, and a transcription is developed for easy reference back to the thought.

Shower Scribe utilizes a wide range of software and hardware while taking safety, privacy, and cost into consideration.

Testimonies

- “Yes I would [buy a Shower Scribe] lol if it were an actual product” — Alex’s Friend
- “I took a shower, shower thoughts, shower scribe 😲 [...] i will be your first customer” — An SE student

The Final Product

Software — Tech Stack

ShowerScribe is built with a tech stack that balances simplicity and performance. Our backend is powered by Flask. It provides a lightweight and flexible foundation for building web applications, allowing our team to structure routes, manage templates, and easily connect with other services written in python.

ShowerScribe uses SQLAlchemy, a Python SQL toolkit, to integrate a relational database into our Flask web application by providing an object-oriented approach to define models, query data, and manage database operations. Semantic search is handled by a Chroma vector database, again with a simple python integration.

Transcription is handled by AssemblyAI's transcription service, and summary and title generation is done with Cohere.

This tech stack runs on a Raspberry Pi 4 (it could have run on a Pi Zero W 2, but we had issues with USB connections, discussed further on). A special consideration was that the scribe should be able to run offline, so since we're using Bootstrap, a CSS framework, we had to download the entire thing locally.

Software — Web App Control

Contributors: Victoria, Aayush, Alex and Vincent

HTTP Response Request System

The ShowerScribe web app uses HTTP to communicate, with GET requests for the pages, as well as for serving audio files, and POST requests to submit forms.

Sorting Database Entries

The objective of the data organization in the ShowerScribe web application is to present recorded information in a clear, user-friendly manner. By effectively **organizing data** retrieved from the database, we aim to enhance the user experience, allowing users to easily navigate and access their recordings, transcriptions, and related information.

Recordings are **grouped by date** to provide a chronological view for ShowerScribe users. Recordings within the same session are clustered to provide a more organized view. Clustering is especially beneficial for users who make multiple recordings in a session. Each recording is presented individually, allowing users to access detailed information, associated transcripts, and resumes. Individual pages are defined to display detailed information about a specific recording. Associated transcripts and resumes are fetched and presented on the recording page.

Semantic Search

Semantic search is a critical feature within the ShowerScribe web application, enhancing the user experience by enabling the retrieval of relevant recordings based on the content of AI-generated resume files. This functionality is powered by a combination of vector embedding and similarity matching, providing users with a convenient means to discover recordings related to specific search queries.

The get_n_closest_ids function primarily facilitates the semantic search functionality. The function utilizes a **Chroma vector database** to store and compare semantic embeddings of text, to query a specific text collection, employing vector-based search techniques to identify the n closest matching text IDs.

The **Flask** route "/search_results" handles semantic search queries from users. The route extracts the search query and the number of desired results from the URL parameters. It then calls the get_n_closest_ids function, passing the search query and the specified number of results, directing the semantic search functionality based on the content of the resume file associated with each recording.

Software — Transcription and LLM Services

Contributors: Alex and Vincent

The reason we chose to use the AssemblyAI's API transcription service is because of its speed. We desire to provide users with a fast transcription service that can process multiple audio files quickly if the user presses the button multiple times during recording. AssemblyAI also has a wide bandwidth of transcription, making the program less likely to crash due to errors in transcribing data.

Summary and title generation are handled by Cohere Generate, a Large Language Model (similar to ChatGPT), called with an easy-to-use python library with a prompt explaining what its job is and the format to answer in. Its answer is then parsed and stored.

Hardware — Raspberry Pi and Microphone

Contributors: Balaji and Tyler

PyAudio

We decided to use the PyAudio library for Python to record the input from the microphone. In order for the program to record for an indeterminate amount of time the user decides, the data was recorded in predetermined chunks and stored as bytes in a list. Since bytes are a default data type in Python, we used a built-in method to join the bytes together for conversion.

PyAudio records audio, expecting it to be exported in .wav files. This proved highly convenient, as multiple robust ways exist to handle .wav files—the provided wave module made exporting the audio almost as easy as text.

To make the recorder interface with the rest of the program properly, we had to make all the recording code non-blocking and give the rest of the program easy access to the state of the recorder. The code runs on its thread to not impede the rest of the program, and the recorder's state is made available by simply accessing a variable.

The ShowerScribe program is registered as a service on the Raspberry Pi and automatically started on boot up.

Software – Conductor

Contributors: Tyler

The different services and programs are linked together using a conductor script that starts and manages all the Raspberry Pi's resources. The conductor uses multiprocessing to allow for parallel execution of scripts, such as the flask server, the transcription processes, and the LLM services. The conductor also is responsible for handling input from the Raspberry Pi's GPIO pins. The worker processes are managed using a multiprocessing pool, which allows jobs to be sent and executed asynchronously.

Hardware — Enclosure and Wiring

Contributors: Alex

The ShowerScribe is contained inside an airtight, clear food container, giving us an easy-to-use vessel that's waterproof and easily modified by cutting the plastic. Inside the enclosure are four silica desiccant packs, used to remove all moisture from the air. Two holes were cut in the box lid to accommodate the button and wire for the microphone. Wires were passed through the holes and then sealed with hot glue to restore a watertight seal.

The waterproofing was evaluated by dunking the empty box in water, and none came in. A visual confirmation of a seal is made before installation by ensuring the rubber sealing ring makes complete contact with the edges of the box.

The wiring for the project is a PCB connected to three LEDs taken from another product and a waterproof arcade button. Both are connected to the Raspberry Pi's GPIO pins on their one-loop.

The Original Plan — Pitfalls and Descoping

Initial Plan Versus Final Product and Anticipated Challenges

We achieved most of the objectives of the original MVP:

- being able to record audio when the button is held
- having an LED indicator for recording, a web app to serve audio and transcriptions,
- auto-startup

We decided to cut the charging port to increase the waterproofing of the container.

Hardware Challenge — Microphone

Initially, we purchased a microphone terminating into a 3.5mm jack and a 3.5mm to micro USB adapter to use with our Raspberry Pi Zero W 2. However, the adapter did not work. We soon discovered that the Raspberry Pi lacks a built-in DAC and thus can not receive audio input natively. We rectified this by buying a USB DAC, terminating in a USB type A and a USB type A to micro USB adapter. However, this also did not work. After more troubleshooting, we realized that the USB A to micro USB adapter did not function as intended. We did not have time to order another one, and thus resorted to using our backup Raspberry Pi 4 instead, connecting the DAC to one of the built-in USB type A ports.

Hardware Challenge — Button and LED in series

We intended to have the LEDs connected to the button in series, meaning there would be no link between the LEDs and the Raspberry Pi. We designed the LEDs this way for privacy, the rationale being that the LEDs will be a reliable indication of recording even if the Raspberry Pi is compromised. Furthermore, we soon realized that this would have little bearing on privacy, as the button and microphone can operate independently. While unfortunate, it at least reduced our workload as instead of wiring the LEDs and Buttons in series, which would have involved us solving the challenge of powering the LEDs independently, we connected the LEDs to a GPIO pin, providing us more control and reliability.

Software Challenge — Integration

Integration was a challenge due to fairly lax documentation of our methods, as well as being bottle necked by having to wait for the conductor script to be finalized by a single person. In the end, our solution was to call a group meeting that lasted 6 hours until the project was finished, with great success. We've learned that this was an effective strategy, and that it pays off to make clear requirements in advance.

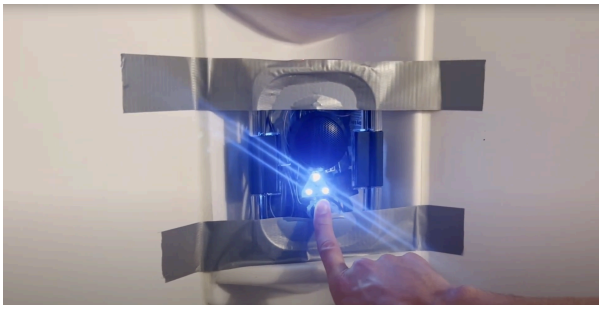
Software Challenge — Development environments

Establishing a consistent development environment was a challenge. We had members on MacOS, Windows, various distributions of Linux, as well as the final product running PiOS. This created some difficulties when it came to a consistent python version, dependencies, filing systems, and especially drivers when it came to audio recording.

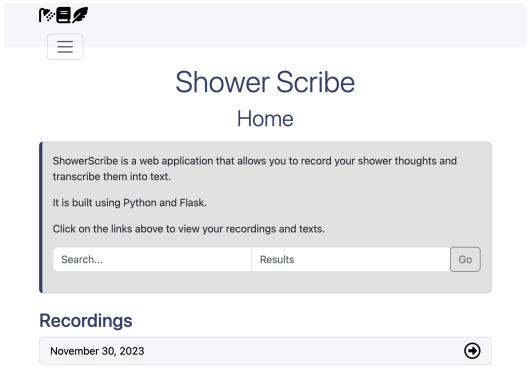
Evidence of it Working

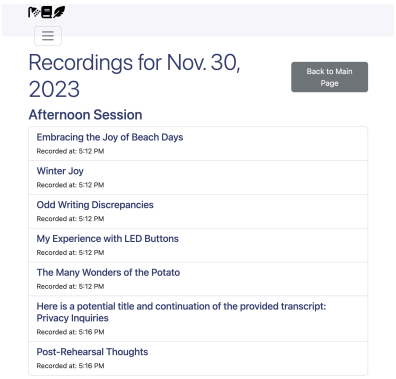
Demo Video https://youtu.be/pPOIAyJtj88

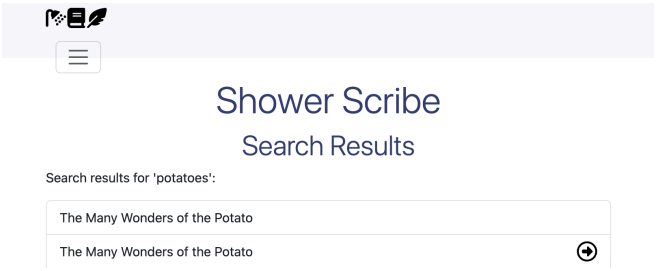
https://drive.google.com/file/d/1MCvdn-PLnfXQh6nwU3ENAvmPHRRmTqXR/view?usp=sharing



Screenshots







Key Performance Indicators

Recording Accuracy:

Since, at its core, the Shower Scribe is an audio recording device, having sound audio recordings is paramount. The recording quality test was defined qualitatively as a human being able to recognize every word in the recording, speaking at an average indoor volume (~60 dB at 50cm, evaluated with a cell phone's microphone) and with the microphone ~50 cm away from the user. The quality of recordings was evaluated in various environments such as a quiet room, a busy classroom, an echoey hallway, and a bathroom with the shower running. The device quickly **passed the test in all of these scenarios**.

Transcription Accuracy:

Transcription accuracy was evaluated by reading the first scene of “The Bee Movie”, while sitting in a bathroom with the shower running. The transcription was **96% accurate**, only missing a few times the reader spoke too fast or quietly. This is well enough to get an idea of what was talked about and surpasses our target of 80%.

Uptime:

There are minor issues with the power supply and crashing, but the Shower Scribe can auto-start from losing power within 30 seconds, which from our experience needs to happen once every 2 hours or so, which is **99.5% uptime**.

Speed:

While there are no requirements for speed outlined in the original requirements, per our benchmarks, recordings become available **less than a second** after finishing the recording, transcriptions about 5 to 10 seconds later, and AI summary and title about 1 second after that. Give that end users will record while in their show, and check after, this is a totally acceptable speed.

SE 101 Considerations

Safety, Privacy and the Code of Ethics

Safety, privacy and the PEO Code of Ethics must be considered when building a project. We had to address the safety concerns posed by having electrical components in the shower and ensure that the wires did not establish contact with water, by utilizing a waterproof casing, sealing any drilled holes securely, and having the only components outside the casing be waterproof. With the given precautions, we prevent any unsafe contact with water.

Similarly, there were possible privacy issues with a recording device for which we had to make provisions. We implemented a user-controlled recording system to ensure the user felt comfortable that there would be no recordings without consent. The system only records while

the user is pushing the recording button, and an LED is lit to indicate that the machine is recording. Also, we ensured that each recording is anonymous and can only be seen and viewed on the user's end.

Additionally, we seek the user's consent to use recording transcription software and LLM services. Users can opt out of sending their recordings for transcription or transcripts to Cohere for Summary by turning the transcript and LLM services off in the website's Settings section.

Intellectual Property

Since our system mainly uses two AI models, AssemblyAI for transcription AI models and Cohere for LLM services, we should adequately credit the use of these AI models and respect their terms and conditions and ownership of these trained data. All other libraries we used are open source and are listed on the repository. The icons that compose our logo are under a free license (like Bootstrap). An inspection of the licenses of our dependencies indicate that there should be no restrictions on commercial use.

References

- https://flask.palletsprojects.com/en/3.0.x/
- https://cohere.com/
- https://www.assemblyai.com/
- https://getbootstrap.com/
- https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
- https://www.sqlalchemy.org/
- https://www.trychroma.com/