SoundSage - Documentation

Introduction
This document discusses the development of an AI-based audio processing tool. The tool is designed to bridge the gap between Language Model Learning (LLM) and audio processing, providing a streamlined and efficient workflow for music production.

Product Overview
The product is an AI-based audio processing tool that uses LLM to enhance the audio processing workflow. The tool is designed to be user-friendly and efficient, allowing users to quickly and easily process audio files.

Product Features
The product features include:

AI-based audio processing: The tool uses AI to analyze and process audio files, improving the quality and efficiency of the audio processing workflow.
User-friendly interface: The tool features a simple and intuitive interface, making it easy for users to navigate and use the tool.
Efficient workflow: The tool is designed to streamline the audio processing workflow, allowing users to process audio files quickly and easily.
Operating Environment
The product is designed to operate in a standard computing environment. It requires a computer with sufficient processing power to run the AI algorithms and enough storage space to store the audio files.

User Interaction
Users interact with the product through a simple and intuitive interface. They can upload audio files, process them using the AI algorithms, and download the processed files.

System Features
The system features include:

Audio file upload: Users can upload audio files to the system for processing.
AI audio processing: The system uses AI algorithms to analyze and process the audio files.
Processed file download: Users can download the processed audio files from the system.
User Features
The user features include:

File upload: Users can upload audio files for processing.

File processing: Users can process the uploaded audio files using the AI algorithms.
File download: Users can download the processed audio files.

We're on a journey to create an AI-based audio processing model for home studio recording and professional music production. The ultimate goal is to create an AI-powered tool that can assist users in creating professional-quality audio productions from the comfort of their own homes. This tool should be accessible to users of all skill levels, from beginners to experienced professionals.

Here are some potential capabilities of our end goal:

Advanced Audio Processing: The virtual producer should be able to perform a wide range of audio processing tasks, from basic functions like gain control and EQ to more advanced tasks like compression, reverb, and noise reduction. It should also include a variety of mastering tools to help users finalize their productions.

Intelligent Assistance: The virtual producer should be able to provide intelligent assistance to users. This could include suggesting improvements to the audio, identifying issues with the mix, and providing tips and recommendations. It could also include a learning feature that adapts to the user's preferences and style over time.

Natural Language Processing: The virtual producer should be able to understand and respond to natural language commands. This would allow users to interact with the tool in a more intuitive and user-friendly way.

Collaboration Features: The virtual producer should include features that allow multiple users to collaborate on the same project. This could include real-time collaboration, version control, and the ability to leave comments and feedback.

Integration with Other Tools: The virtual producer should be able to integrate with other popular audio production tools and software. This would allow users to use their preferred tools alongside the virtual producer.

Accessible and User-Friendly Interface: The virtual producer should have an interface that is easy to use and accessible to users of all skill levels. This could include a simple and intuitive layout, customizable workflows, and a variety of learning resources and tutorials.

Cloud-Based Operation: The virtual producer could be cloud-based, allowing users to access their projects from any device and location. This would also allow for easy updates and improvements to the tool.

High-Quality Output: Above all, the virtual producer should be capable of producing high-quality audio that meets or exceeds industry standards. This is the ultimate measure of the tool's success.

Genre-Specific Personalities: Each personality could specialize in a particular genre of music. For example, a "Rock Star" personality could provide tips and suggestions for creating a powerful rock sound, while a "Pop Maestro" could offer advice on crafting catchy pop melodies.

Celebrity Personalities: As you suggested, we could also have personalities based on real-life musicians, producers, or other celebrities. These personalities could offer advice and suggestions based on the style of their real-life counterparts. For example, a "Dr. Dre" personality could provide tips on creating hip-hop beats, while a "Taylor Swift" personality could offer advice on writing pop-country songs.

Licensing Brands: We could partner with real-life producers, musicians, and brands to create licensed personalities. This could provide a source of revenue for the project and also increase its appeal to fans of these personalities.

Learning and Adapting/Customizable Personalities: Over time, the virtual producer could learn from the user's preferences and adapt its personality accordingly. This could make the tool feel more personalized and engaging.Users could have the option to customize their virtual producer's personality. This could include adjusting its level of enthusiasm, its approach to giving feedback (e.g., more direct vs. more supportive), and its musical preferences.

Interactive Dialogue: The virtual producer could use natural language processing to engage in interactive dialogue with the user. This could make the tool feel more like a real-life collaborator and less like a piece of software.

Learning from Past Work: The virtual producer could analyze the user's past work to learn about their musical style and preferences. This could involve analyzing the musical elements of their songs (e.g., melody, rhythm, harmony), as well as the production techniques they've used (e.g., EQ settings, compression, reverb).

Reading Bios and Other Information: The virtual producer could also read the user's bios and other information to learn more about their background and influences. This could help the virtual producer to understand the user's musical journey and provide more personalized assistance.

Customizable Avatar/Voice: The virtual producer could have a customizable avatar and voice. This could make the tool more engaging and enjoyable to use, and could also allow the user to create a virtual producer that matches their personal style.

AR/VR Integration: While the market for AR/VR in music production may not be large at the moment, it's possible that this could change in the future. Integrating AR/VR capabilities into the virtual producer could provide a more immersive and interactive experience for users.

Efficiency and Inspiration: The ultimate goal of the virtual producer should be to help users work more efficiently and stay inspired. By handling the technical aspects of music production, the virtual producer could allow users to focus on the creative aspects of their work. It could also provide inspiration by suggesting new ideas and techniques.

As for market assessment there a few claiming to be using “AI” to mix/Master but no one has created a comprehensive toolset based on LLM’s and deep learning models in the future.

This is all great but it's a hefty task and will take a considerable amount of time and resources to successfully accomplish. Notably the first tool, AutoGain, is already in place, which is a great start. It sets the clip gain level to industry standards, ensuring a consistent output for all users regardless of recording volume.

Let's talk about the minimum viable product (MVP). For our project, the MVP could be a basic version of the LLM (Language Learning Model) integrated with the AutoGain tool and maybe an EQ that is similar to Gullfoss as well as a dataset of traditional EQ presets and the ability to get a little crazy if the over all sound is better/ or specified by the user. The EQ would require a lot more scripting and testing due to its nature, a simplified version would be just telling it simple commands for EQ instead of having it make the decisions you could directly tell it “HPF250hz” or “sound sage please high shelf, boost 3db from 8k and high cut 18khz sharp knee”, would be the simplest way to do it now however i believe we should focus on the LLM integration with AutoGain for testing and Datasets for industry standard production and engineering practices. There are many tools we will make such as AutoEQ: This tool could provide users with a range of equalization options, allowing them to adjust the frequency response of their audio files.

AutoCompressor: This tool could allow users to control the dynamic range of their audio files, reducing the volume of loud sounds and increasing the volume of quiet sounds.

AutoLimiter: This tool could prevent the volume of audio files from exceeding a certain level, ensuring that the audio does not distort or clip.

AutoReverb: This tool could add a reverb effect to audio files, creating a sense of space and depth.

AutoDelay: This tool could add a delay effect to audio files, creating echoes and enhancing the sense of space.

AutoNoise Reduction: This tool could reduce unwanted noise in audio files, improving the overall sound quality.

AutoEnhancer: This tool could enhance certain aspects of the audio, such as clarity, warmth, or presence.

AutoStereo Widener: This tool could increase the stereo width of audio files, creating a wider and more immersive soundstage.

AutoPitch Corrector: This tool could correct the pitch of audio files, ensuring that the audio is in tune.

AutoTime Stretcher: This tool could change the speed of audio files without affecting the pitch, allowing users to slow down or speed up their audio.

However for now until the LLM/AutoGain bridge is connected we will push those tools aside so we can focus on that.

Here's a potential roadmap to get us from where we are now to the MVP:

Integration of LLM with AutoGain: This is the next immediate step. We need to ensure that the LLM can effectively control the AutoGain tool. This will involve creating an interface between the two and ensuring that they can communicate effectively.

Development of Basic Command Understanding: The LLM needs to understand basic audio processing commands. This could be as simple as "set gain to -18 LUFS" or “do the gains staging”. And

Implementation of Error Handling and Progress Update Scripts: These scripts will ensure that the system can handle errors gracefully and keep the user informed about the progress of their commands. This will involve catching exceptions, logging them, and providing useful error messages to the user.

Development of a Simple GUI: The MVP will need a simple graphical user interface (GUI) that users can interact with. This could be as simple as a command line where users can type their commands and see the system's responses.

Next we aim to integrate the AutoGain Python program with OpenAI's API for ChatGPT. This will enable automatic Gain Staging of specified audio files from any given folder/directory on any computer, using a chat interface with OpenAI's LLM.

The process should unfold as follows:

The User prompts: Gain-Stage* all* of the audio files* in this specific folder**.
The LLM Extracts values from User prompt
- “Gain-Stage” = AutoGain
- “audio files” = .wav .mp3 .flac .aaif etc..
- “All” = select entire folder to import the input/specified files
- “Specific folder” = select destination/output folder for final processed files
- “Specific folder” = “Specified Folder”/ Folder Name (for renaming the “New” folder

post process.)
The LLM creates a list of commands in the correct order to produce the final processed files.
problem/unprocessed files+solution/command list = result/processed files.
The LLM sends a signal to a python script that searches all directories for the specified folder
The python script checks the contents of the folder to ensure the files are audio files and that they are the correct audio files based on any specifications, if any. If not, the script returns false.
The python script navigates to the specified folder.
The python script copies specified audio files within the specified folder.
The python script navigates to the "PreProcess" folder within SoundSage's AutoGain folder
The python script pastes the copied specified audio files to the “PreProcess” folder
The python script initiates the beginning of the file selection process for AutoGain by setting the inputs and outputs:
The python script selects input files from the "PreProcess" directory within the AutoGain UI.
The python script chooses the output folder: "PostProcess" within the AutoGain UI.
The python script initiates Audio Processing: (runs AutoGain by executing AudioAnalysis.py, and AudioProcessor.py.)
The python script creates a “New” folder within the original specified folder
The python script navigates to the “PostProcess" folder within SoundSage's AutoGain folder
The python script copies specified audio files within the “PostProcess" folder.
The python script pastes the copied processed audio files to the “New” folder
The python script sends a signal to The LLM to respond that the process is complete and to check the new folder : “Path/To/Folder/Files”

The LLM returns a response to the user, including information about where the processed files are located and how to retrieve them. The response should also contain a log of all processing applied, analysis results, and recommended actions for improved audio quality/industry-standard results. Essentially, it should provide a comprehensive report on everything the process discovered about the selected audio files.
To achieve this, we may need to write additional scripts, potentially two or three. One script would integrate the LLM with AutoGain, and another would allow the LLM to manage files locally on a computer, granting it access to the entire hard drive.

This integration will be accomplished by bridging the LLM Chat Interface with AutoGain, creating a series of commands. These commands will be used in a specific order to achieve the desired result, which in this case is the automated Gain staging of specified audio files.

Before this, the AI needs to locate the correct directory with the specified audio files to run the provided script. Additionally, we need to develop a chat interface GUI to replace the original UI.

Current Project directory::

SoundSage * Folder

main.py
AutoGain_controller.py
openai_controller.py
AutoGain *Folder
  audio_analysis.py
  audio_processor.py
  PreProcess *Folder
  PostProcess *Folder

But it will need the following scripts before it is complete Command Parsing Script: This script would be responsible for interpreting the user's command and extracting the necessary information. It would likely involve natural language processing techniques, which could be implemented using libraries such as NLTK or SpaCy.

File Verification Script: This script would check the contents of the specified folder to ensure that it contains valid audio files. It could use a library like os or glob to list the files in the directory, and a library like mutagen or pydub to check if the files are valid audio files.

File Management Script: This script would handle the copying and moving of files between directories. It could be implemented using the shutil or os libraries.

AutoGain Processing Script: This script would interact with the AutoGain program to perform the Gain Staging process. It would need to set the input and output directories, initiate the program, and handle any output or errors.

User Notification Script: This script would generate a response for the user once the process is complete. It would need to gather information about the processed files and format it into a user-friendly message.

Error Handling Script: This script would handle any errors that occur during the process. It would need to catch exceptions, log them for debugging purposes, and provide useful error messages to the user.

Progress Update Script: This script would provide ongoing feedback to the user during the process. It could be implemented using a library like tqdm for console-based progress bars, or it could involve updating a GUI with progress information.

Chat GUI Script: develop a graphical user interface, we would need a script to create and manage the GUI. This could be implemented using a library like tkinter or PyQt.