# Text2Synth: A Generative Audio Plugin

Proposal Overview: I will be building a VST3 audio plugin that utilizes machine learning to generate synth presets based on a text prompt. All of the machine learning will be done using PyTorch within Google Colab Pro. I have moved beyond simple architectural investigation and settled on a cross-modal architecture that bridges text and synthesizer parameters. Specifically, I am utilizing Residual Feed-Forward Networks to map semantic embeddings to knob values, leveraging the LAION-CLAP model to interpret semantic inputs.

Additionally, I will utilize LibTorch to translate everything into JUCE, where I will be using C++ to build the main plugin frontend and backend. This project will provide me with a base knowledge of machine learning, as well as cementing my knowledge of real-time audio processing in C/C++. I will use my own private library of 22,000 Serum .fxp files to train my model.

Technical Implementation Strategy: To achieve the connection between text and audio, my implementation follows a three-stage pipeline that I have already begun validating.

First is Data Ingestion. I am parsing my dataset of 22,000 .fxp files using DawDreamer (a Python VST render engine). Unlike standard parsing, I am rendering "Smart Sweeps"—brief audio clips of every preset—to capture the true sonic character of the synth. I then use LAION-CLAP (Contrastive Language-Audio Pretraining) to generate vector embeddings of these audio clips. This allows the model to understand the relationship between a prompt like "warm analog pad" and the actual timbre of the sound, rather than relying solely on text metadata.

Second is the Model Architecture. I am training the model in PyTorch to predict synthesizer parameter values based on the input CLAP embeddings. Instead of a standard VAE, I am utilizing a Generator with Residual Blocks. This architecture takes the text embedding vector and passes it through a series of dense layers with residual connections, allowing the network to learn complex, non-linear mappings between language and synthesizer knob positions without suffering from vanishing gradients.

Third is the C++ Integration. I will build a subtractive synthesizer engine in JUCE that mirrors the parameter structure of my training data (matching oscillators, envelopes, and filters). I will then export my trained PyTorch model to TorchScript and load it inside the plugin using LibTorch. When a user types a prompt, the model will run inference in real-time and update the synthesizer knobs instantly.

Mentorship: To support the multidisciplinary nature of this project, I have identified three mentors to serve as technical references for the audio, machine learning, and mathematical components, respectively.

- Digital Signal Processing (DSP) & Audio Programming: I will consult with Professor Mike Frengel from Northeastern University’s Music Technology department. His academic background in audio programming will be a key resource for designing the architecture of the subtractive synthesis engine and ensuring C++ best practices within the JUCE framework.

- Machine Learning Engineering: For PyTorch integration, I will consult Ethan Wee, a software engineer at AMD currently working directly on the PyTorch framework. His industry-level knowledge of the library will be invaluable for troubleshooting the complex integration of LibTorch within a real-time audio environment.
- Mathematical Theory: For theoretical verification, I will consult Jemma Schroder, a PhD candidate in Pure Mathematics at the University of Texas at Austin. Her expertise will be utilized to review the mathematical soundness of the neural network architectures and the linear algebra involved in the tensor transformations.

Project Goals: My goal is to have a rudimentary audio plugin that generates synth presets based on a user's text prompt. It will not be state of the art, as I am just one person working on this in a short timeframe. However, given the amount of training data I have, I believe that I can build something functional that works as I described. I will focus on the core functionality of the prediction pipeline first, prioritizing a stable build that does not crash the Digital Audio Workstation (DAW) over complex sound design features. I will likely have to adjust my scope to account for the computational constraints of real-time audio, potentially simplifying the neural network architecture or the synthesizer engine if the CPU overhead becomes prohibitive.

Specific success metrics include:

- Successfully loading the LibTorch model within a JUCE environment without build errors.
- Achieving a loss rate during training that indicates the model is learning distinct categories of sounds rather than outputting random noise (overcoming "Mode Collapse").
- Creating a user interface that allows for text entry and displays the resulting parameter changes visually.

Project Resources: I will utilize my Google Collab Pro account to access the A100 GPU for training my model, as well as Google Gemini to debug and help me when I get stumped. Additionally, I will use MIT open course ware such as their Intro to ML notes, and YouTube videos that I will cite in my final LaTeX report.

I will also be using Open Source Software (OSS) to facilitate the development:

- DawDreamer: To load VST plugins in Python, render audio sweeps, and extract parameter values from the .fxp files correctly.
- LAION-CLAP: To tokenize text and generate audio embeddings, bridging the gap between language and sound.
- JUCE Framework: For the cross-platform audio plugin architecture.
- LibTorch: The C++ distribution of PyTorch for running the model within the plugin.

Risk Management: There are several technical risks associated with this project that I have accounted for. The primary risk is the file size and performance overhead of LibTorch, which can be heavy for a real-time audio thread. To mitigate this, I will run the inference (the prediction calculation) on a background thread so that the audio stream is never interrupted, preventing clicks and pops. A secondary risk is the "Black Box" nature of neural networks, where the model might output chaotic or silent

parameters. I will implement "safety clamping" in my C++ code to ensure that the generated values always stay within valid, audible ranges (e.g., ensuring volume does not exceed 0dB).

Project Timeline:

- Week 1-2: Data Preparation. Writing Python scripts using DawDreamer to batch process the 22,000 .fxp files. This involves accurately extracting knob data and rendering audio sweeps for embedding generation.
- Week 3-5: Model Training. Developing and training the model in Google Colab. I have allocated extra time here to experiment with hyperparameters and ensure the Residual Block architecture converges properly without mode collapse.
- Week 6-8: C++ Audio Engine. Building the subtractive synthesizer in JUCE. This includes programming the oscillators, filters, envelopes, and modulation routing that the model will control.
- Week 9: Integration. Connecting the trained PyTorch model to the C++ engine using LibTorch. This involves handling the complex linking process between the Python-exported model and the C++ environment.
- Week 10: UI Implementation. Building the frontend interface, specifically the text input field, and mapping the backend parameter changes to visual knobs and sliders.
- Week 11: User Testing. Distributing the beta version of the plugin to a small group of users to identify bugs, test stability across different DAWs, and verify that the text prompts produce expected sonic results.
- Week 12: Final Polish and Report. Fixing bugs found during testing, optimizing CPU performance, writing the final LaTeX report, and organizing the GitHub repository for public release.

Project Deliverables:

1. A functional VST3 audio plugin that generates synth presets from a text prompt, with a straightforward UI and parameters to manipulate the preset.
2. A LaTeX report, outlining my entire process for this project, including mathematical explanations of the model used.
3. A GitHub repository that contains the entire project in full so it is available for anyone to access online.
4. A short demo video demonstrating the text-to-sound workflow to prove functionality.