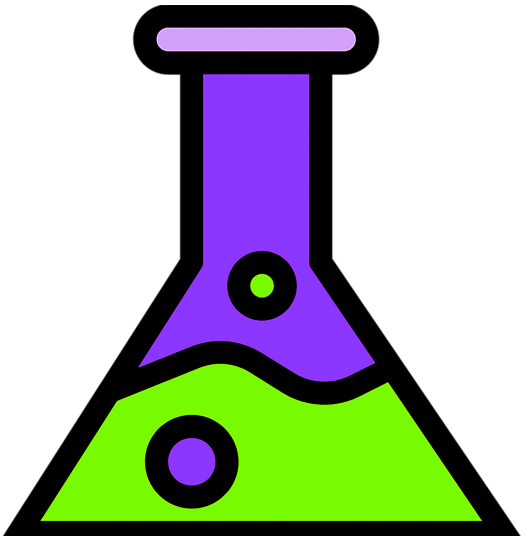
**Group Members:**
Hubert Liu, Noah Newton, Tien Nguyen

# 1. Project Overview

Puke Studio is a visual programming language for audio processing, designed with the goal of eventually being free and open source. It allows users to create patches by connecting nodes that modify input and output in different ways, similar in concept to software like Max 8. Approaching the software as new users, we focused on exploring its functionality and limitations, with particular attention to implementing autotune and pitch shifting features. We also worked on producing documentation to make the system more accessible to future users.

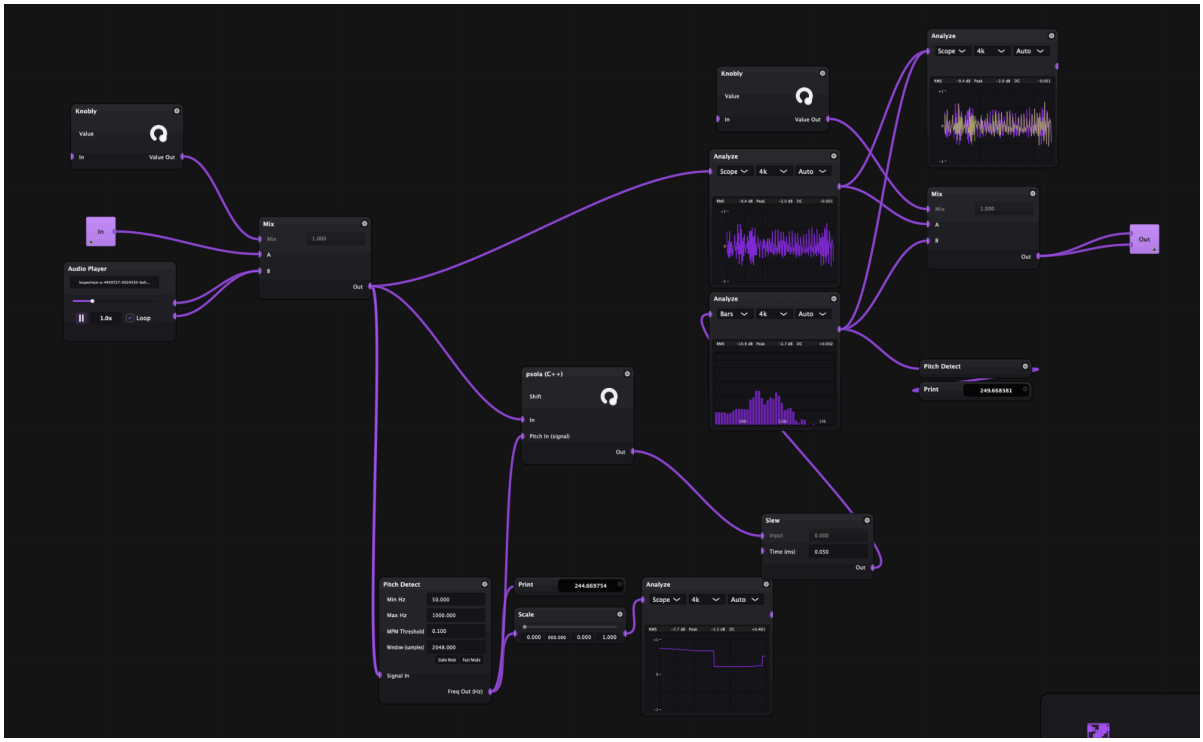


# 2. What We Did

We began by learning how to use Puke Studio and its patch-based workflow, experimenting with different types of audio input to understand how signals move through the system. As we worked, we identified several areas where new users might struggle. A primary area where we struggled (and decided to create documentation for) was the creation of custom C++ nodes, of which we built three: a PSOLA node for pitch shifting, a standalone autotune node for pitch correction, and a radio selection node for signal routing. Using these tools, we built three separate autotune patches, each exploring a different approach and achieving partial success.

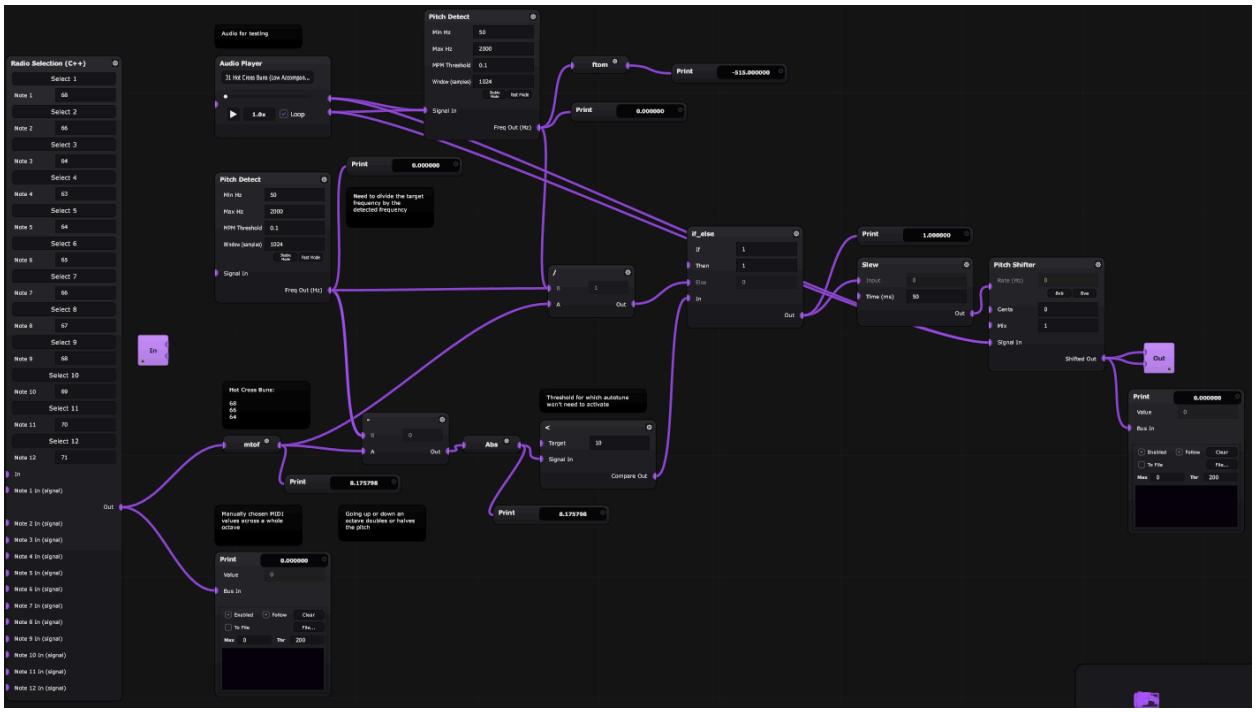
### 2.1 Autotune with custom PSOLA

This patch implements the Pitch Synchronous Overlap and Add (PSOLA) algorithm as an attempt to improve the built-in Pitch Shifter node. Instead of using a delay-based approach, PSOLA works by detecting signal peaks and rearranging them to modify the perceived frequency of the sound. This has the benefit of not requiring scaling and resampling of the input signal.



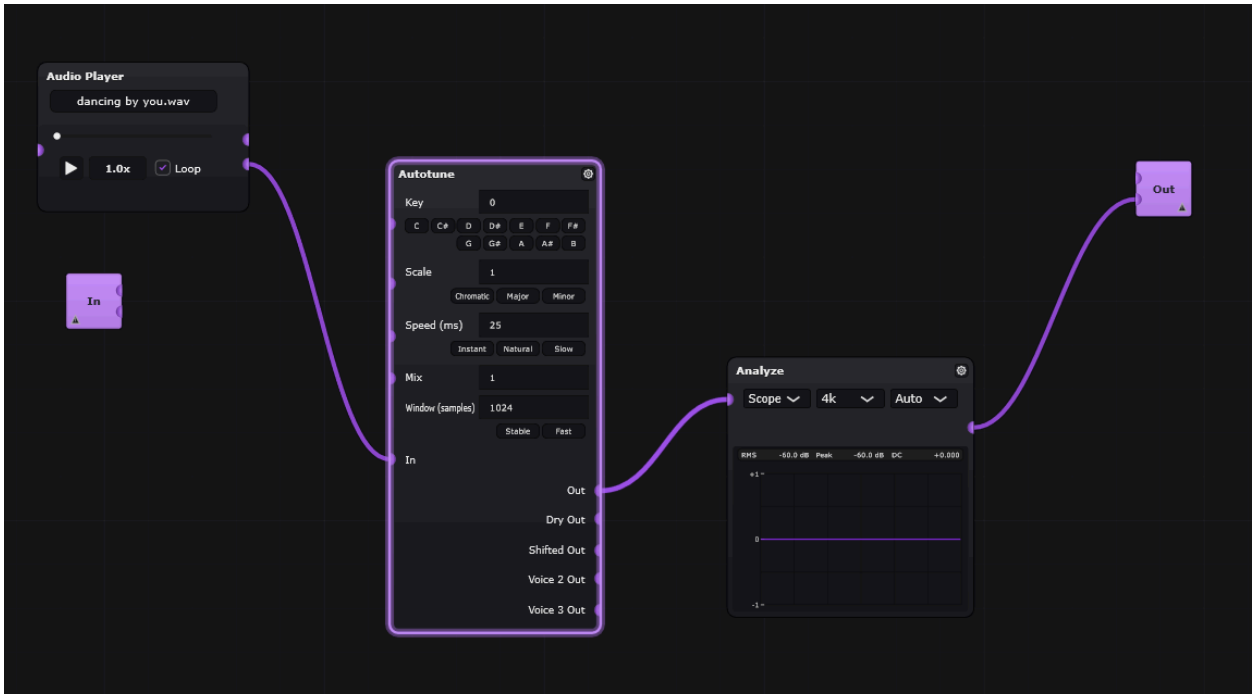

## 2.2 Autotune with built-in Pitch Shifter

This patch attempts to implement autotune with the built-in Pitch Detect and Pitch Shifter nodes, the custom C++ Radio Selection node in this case only being used to send multiple input values more quickly and easily. The patch detects what pitch the input audio has, and then tries to shift the pitch to a pitch value specified by the Radio Selection node, essentially a completely manual process where the user directly pitches their input at any given moment to a specific value.




### 2.3 Autotune with custom Autotune node

We added a new Autotune node to Puke Studio. Think of it like adding a new plugin (interface) to the app, users can find it in the effects menu, drop it into their patch, connect their audio through it, and it will automatically tune the incoming sound to the correct musical notes in real time. They can choose what key and scale they're playing in, how snappy or smooth the correction sounds, and how much of the effect to apply.




# 3. How We Did It

To get started, we studied existing patches and followed available tutorials to build a basic understanding of how the system works. From there, we experimented with both familiar and unfamiliar nodes, testing their behavior and limitations. We used the provided custom C++ node template to implement our own nodes, iterating on their functionality as needed. In parallel, we created multiple patches to test different autotune strategies and compared our custom solutions against the built-in Pitch Shifter node.

### 3.1 Autotune with custom PSOLA

Two implementations of PSOLA were tested, and ultimately we decided to do the following:

1. Take in the current pitch from an external pitch detection algorithm.
2. Naively assume that the frequency within the current buffer is unchanging and spread peaks evenly throughout.
3. Calculate locations of new peaks based off of the input frequency and the scaling factor
4. Iteratively copy source peaks onto destination peaks, overlapping and adding with a triangular window.

The PSOLA node is a custom C++ node which allows the user to shift the pitch of the input by an arbitrary amount. The performance of this node is limited by the small buffer size (512 samples, or 10.6 ms). This size limits the amount of peaks which can be stored within one buffer, which limits the performance of the PSOLA node. At typical speaking frequencies, this buffer only captures 2-3 peaks. Increasing the buffer size significantly improves the clarity of the output.

The task of detecting the input frequency was out of the scope of this project and delegated to an existing pitch detection node. If the circular buffer is added to the design, the pitch detection can be moved internally to increase its resolution.

This pitch is then used to generate equally spaced “peaks” along the input. These peaks are reconstructed on the output with different spacing using an overlap-and-add technique to achieve the pitch shifting effect.

Additionally, the triangular window used to perform the overlapping and adding can be updated to use a more “ideal” shape. This reduces ghosting effects where higher harmonics are erroneously introduced.

**3.2 Autotune with built-in Pitch Shifter**

Here’s a breakdown on the functionality of the patch:

1. Input audio pitch is detected with Pitch Detect node.
2. Target pitch is selected live by user with Radio Selection node and sent to the patch.
3. A pitch shift ratio is determined by comparing the target pitch to the detected pitch.
    a. A threshold is specified by the user to allow for the ratio to default to 1.0 if the detected pitch is close enough to the target pitch (no reason to modify if it near the desired pitch).
4. A Slew node smooths the resulting ratio before it is sent to Pitch Shifter node.
5. Pitch Shifter node modifies the audio and sends the result to output audio device.

The Radio Selection node is a custom C++ node that allows the user to specify 12 numbers in input fields. This is designed so that the node can hold a full octave of MIDI notes, which can all be pitched up or down any number of octaves with additive nodes. Once the 12 numbers are specified, the user can click the “Select” button above any number to change the output from the Radio Selection node to that number. This greatly improved the ease of changing signal

input values manually, which previously would have taken several seconds for a user to implement live.

*It is important to note that although the node has been named “Radio Selection,” it does not function exactly like a standard radio group (the intended functionality). Although the node only outputs the most recently selected value, selecting a new value doesn’t immediately deselect any previously selected values. This means that multiple values can technically be selected at any given moment, and if you wish for a value to be sent immediately into the patch upon a single click, it must be deselected first manually. This means that quick double quicks work best when using the Radio Selection node.

### 3.3 Autotune with custom Autotune node

Puke Studio has a built-in system for adding new nodes to the app. Every node is made of two parts:

- **Definition file** that tells the app what the node looks like its name, what parameters it has, and what ports it exposes
- **DSP adapter file** that contains the actual audio processing code that runs in real time.

We created both of these files from scratch for the Autotune node, following the same patterns used by existing nodes like the Harmonizer and Pitch Shifter.

The autotune node processes the incoming audio in four main steps:

1. **Detect the pitch**
   The node listens to a small window of recent audio samples and estimates the main pitch of the sound. This pitch is measured as a frequency in Hertz (Hz).
2. **Find the closest correct note**
   Convert Hz to MIDI: note = 69 + 12 x log2(Hz / 440). We try to find the nearest note in scale mask (12-bit integer, 1 bit per chromatic pitch). In other words, the detected frequency is converted into a musical note value. The node then checks the selected key and scale to find the nearest note that belongs in that scale.
3. **Shift the pitch (Harmonizer)**
   After finding the correct target note, the node calculates how far the original pitch needs to move in semitones. That correction amount is sent into the pitch shifter, which adjusts

the pitch of the audio. In short, overlapping delay heads advance at 2^(semitones/12) x write speed.

4. **Blend original and corrected audio**
   Finally, the node mixes the corrected audio with the original audio using the Mix setting. This allows the effect to sound either subtle or stronger depending on how much corrected audio is included

One of the key design decisions was that we did not write a pitch detection or pitch shifting algorithm from scratch. Puke Studio already had both of these as separate built-in nodes (Pitch Detect and Harmonizer), each backed by a DSP library called “puke_dsp”.

# 4. Challenges

As new users, we found Puke to be difficult to learn, especially in the early stages. A significant portion of the first few weeks was spent resolving technical setup issues and understanding basic usage. While the available tutorials were helpful, they were not as comprehensive as needed, and the built-in “help” patches for individual nodes were often incomplete or didn’t explain anything. The custom C++ node template was particularly challenging, as it introduced a level of complexity that was not well supported by documentation. Only someone well-versed in Puke’s infrastructure and node usage (with the ability to code) would be able to effectively utilize custom C++ node creation. Finally, the implementations of autotune and pitch shifting were more complicated than we initially expected, with both our custom nodes and the existing pitch shifter node often producing messy-sounding modified output.

# 5. Resources for Future Groups

GitHub ← Code header files for the 3 custom nodes
Noah Demo ← Autotune with built-in pitch shifter video
Tien Demo ← Autotune with custom autotune node
Custom Node Documentation ← Documentation we wrote for custom nodes

# 6. Future Work

There are several areas where our project work as well as the Puke application itself could be expanded or improved. On the audio side, every version of pitch shifting (existing pitch shifter,

PSOLA pitch shifter, and custom autotune pitch shifter) could be refined to produce more stable and reliable results. Adding easier ways to provide controlled input, which is what the Radio Selection custom node sought to do, would also make development more efficient. From a usability standpoint, the platform would benefit from clearer and more complete help patches, as well as a more beginner-friendly custom C++ node template. Additional interface improvements, like collapsible patch boxes and better support for large patches, could make the system easier to navigate. Finally, adding more built-in audio processing nodes would give users more flexibility when building patches. There are nearly endless possibilities for possible nodes, and the more expansive the node library, the more viable Puke will be.

# 7. Conclusion

Overall, this project gave us hands-on experience working with a visual music programming environment from a beginner's perspective. While we were able to implement several working components, including custom nodes and multiple autotune approaches, the process also highlighted significant usability challenges within the system. Our work contributes both functional examples and supporting documentation that can help future users get started more easily. At the same time, it points to clear areas where the platform could be improved to make it more accessible and effective for new developers. Puke Studio is an application that will be competitive in the field of visual audio modification software tools, provided the proper time and effort is put in to increase its functionality and user experience.