Design

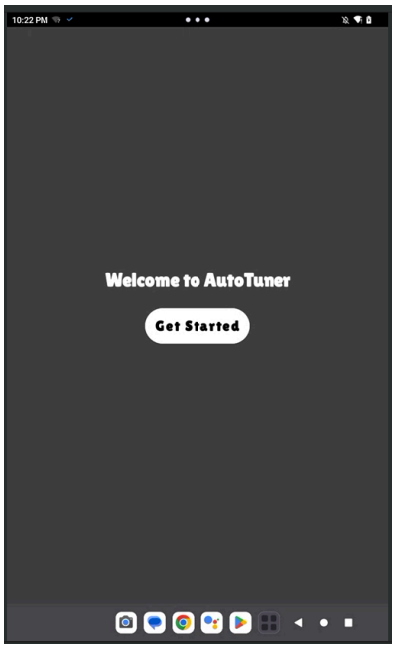
External Interface

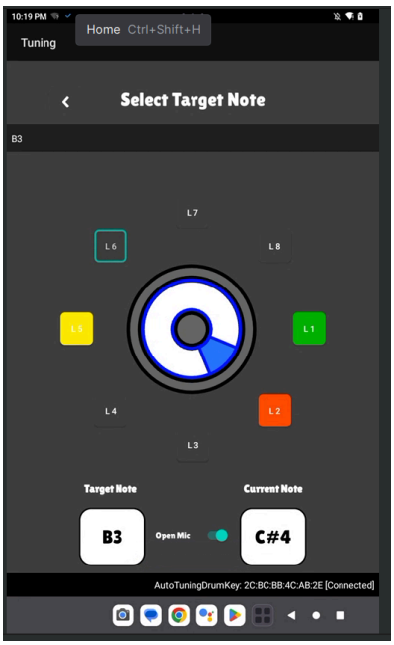
User Interaction

The primary user interaction occurs via the mobile application. The hardware requires minimal user involvement—turning the device on and holding/clamping it during tuning. Users select the drum's lug count, record current notes, and set the desired note. The device then tunes automatically as the user continues hitting the drum. A triangular subsection on the app interface guides the user to strike specific areas in sequence. No haptic feedback from the motor indicates when tuning is complete.

App Interface & Controls

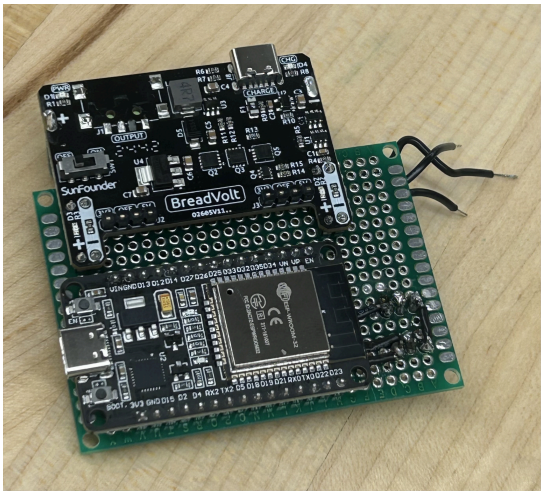
The app prioritizes lug count selection, allowing dynamic modifications; it displays real-time note detection during tuning as a visual aid. The app interface includes a visual indicator where the current lug changes from red to green once the target frequency is reached. The user selects the tuning device from a dropdown menu and is guided through a star pattern tuning process.

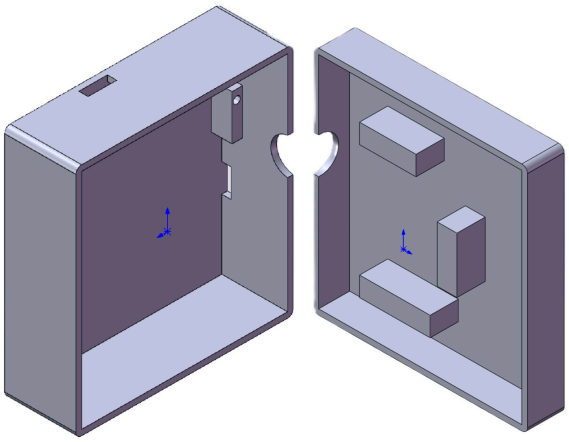




Hardware Interface

The ESP32 microcontroller, microphone, and motor are compactly connected on a perf board. The switch and ports are repositioned for accessibility, ensuring easy charging and operation. The encasing conceals internal components, protects from damage, and features a power switch and a charging port. The motor operates with sufficient torque (13 kg*cm) to ensure effective lug adjustments.





Communication Protocol

The system relies on classic Bluetooth for communication. The backend will store user preferences like the lug count and ESP32 mac-address. The tuning process remains unchanged, with real-time note detection and adjustments via microphone I2S communication and motor PWM.

Persistent State

Software Persistent State

The app provides UI feedback, reads Bluetooth frequency data, and sends target frequency commands. Currently, the app supports one target frequency (250 Hz, D3), with expansion plans. The Bluetooth communication model ensures reliable data transfer.

Hardware Persistent State

The ESP32 processes microphone data via I2S and applies FFT for frequency extraction. A threshold-based system ensures accurate data collection. The motor adjusts based on whether the detected frequency falls within ±3 Hz of the

target. As the frequency nears the target, the servo motor makes smaller angular adjustments. The ESP32 manages memory in the heap for FFT sampling. The encasing design ensures portability and ease of use while housing the essential components securely.

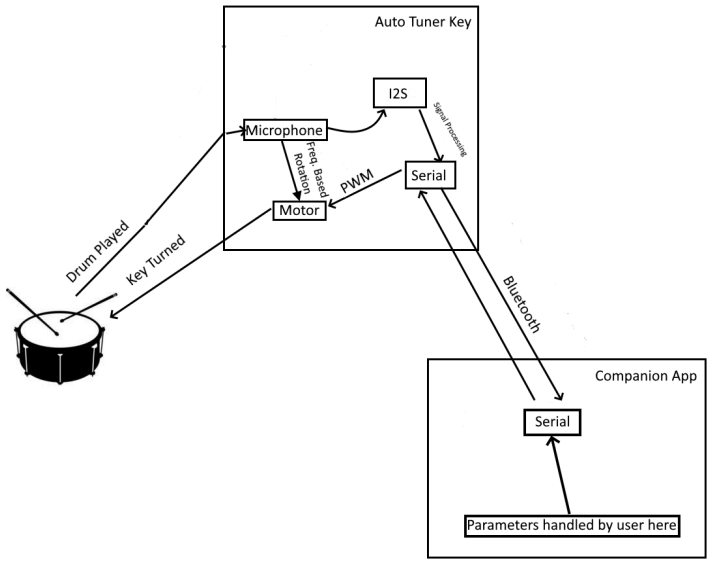
Internal State

Component Architecture

The system consists of:

- **Hardware:** ESP32 microcontroller, I2S microphone, LED indicator, FFT processing, servo motor, and a battery pack.
- **Software:** Android app for user interaction and Bluetooth communication.

The ESP32 receives a target frequency from the app, processes microphone input, and adjusts the motor accordingly. The app displays real-time data to guide the user. The encasing also helps provide a stable environment for the tuning process.



Hardware Components

- **ESP32:** Manages Bluetooth communication, microphone processing, and linear motor control.
- **I2S Microphone:** Captures drum sounds, converting them into frequency data.
- **Servo Motor:** Rotates based on tuning adjustments with sufficient torque.

- **Power Supply:** Battery pack providing 3.3V to the ESP32 and 5V to the motor.

Software Components

- **ESP32 Firmware:** Handles I2S audio processing, FFT analysis, motor control, and Bluetooth communication.
- **Android Application:** Manages Bluetooth connectivity, displays frequency data, and sends tuning commands. Sensory indicators include visual prompts and frequency updates at the bottom of the screen. The app's interface ensures clarity for the user through graphical representations of the tuning process.

Major libraries include `arduinoFFT.h` for frequency analysis, `BluetoothSerial.h` for communication, and Android Bluetooth APIs for app-device interaction.

Risks and Mitigation

The electric tuner is a device that is viewed as one made for beginners to learn as they pick up an instrument. As such, the view of such tools as a crutch, potentially keeping them from learning techniques like tuning by ear could be seen as an ethical issue. The Auto Tune Drum Key could potentially be one such other tool, as not only would this remove the process of learning to tune by ear, but also the process of using a drum key to tighten or loosen a drum. Some might view the development of a product such as this as taking away from a key skill of learning a new instrument and of music: that is, learning to tune by ear without the use of a tuning device.

Another consideration is the potential for damage of drums due to a failure of the Auto Tuner. If the Auto Tuner is not able to identify a "limit" to the tension of the drum or tries to tune to a frequency impossibly high, the Auto Tuner could potentially snap or damage the drum. Proper caution needs to be taken to ensure that the Auto Tuning Drum Key does not try to tune to a key that is impossible to tune to given the drumhead.

Tools, Design Constraints, and Engineering Standards

Process

The development of our AutoTuner followed structured engineering processes and industry standards across all stages: hardware, firmware, and software. For signal processing, we captured audio data from an INMP441 (I2S) microphone and processed the raw data through an FFT using the arduinoFFT arduinoFFT | Arduino Documentation library, sticking to common digital signal processing practices such as windowing and peak frequency detection possible through an envelope follower. (Envelope following)

The enclosure was designed using CAD tools with dimensions based on the PCB and motor specs, considering mechanical tolerances for 3D printing and heavy reliance on GDT standards for a snug fit to parts and ports. (unit3_how_it_works.pptx)

Protocols

In developing the AutoTuner, we adhered to several engineering protocols and standards to ensure reliable hardware-software integration and user interaction. On the hardware side, we utilized the I2S (Inter-IC Sound) protocol for digital audio communication between the ESP32 microcontroller and the INMP441 microphone (I2S Protocol : Features, Working, Differences and Its Applications). Motor control was achieved through standard ESP32 servo libraries using PWM signaling(ESP32Servo | Arduino Documentation). The PCB was designed following standard practices for soldering etiquette such as soldering paths, and wire connectivity (Using Perfboard | Soldering Basics : 14 Steps (with Pictures) - Instructables)

For embedded programming, we used the Arduino IDE and structured the code using modular practices, separating logic into reusable functions and using header files for organization. Standard Arduino-compatible libraries were cited and used for I2S and motor control. Standard coding heuristics such as segmenting code and including comments for readability were also enforced on all sides.

The companion Android app was made in Kotlin using Android Studio, following the Material Design guidelines for UI/UX design (Guidelines - Material Design). App-to-device communication used Bluetooth/Wi-Fi protocols from both sides to establish a client server connection. (ArduinoBLE | Arduino Documentation) (bluetooth | Jetpack | Android Developers). We also used Git for version control for code collaboration and sharing by using the standard rule of separate branches that can later be merged into the main code along with a back repository for safety.

Design Constraints

Frequency consistency was a key design constraint in our project, specifically in making sure that the same frequency could be reliably captured from a specific section of the drum, regardless of variations in how the user strikes it. While the INMP441 microphone we selected provides accurate frequency data, we are limited in our ability to ensure consistency under realistic use conditions—such as if the mic can pick up the discrepancy in frequency of small motor rotations during the fine tuning stage as the current frequency attempts to meet the target frequency. This limitation is partially inherent to the drums in general since drum tuning is influenced by several factors, including the material, diameter, headtype, and construction style of the drum (The Complete Guide to Drum Tuning: Tips, Tools & Techniques | Equipboard), all of which affect the range of frequencies it can produce. For the purposes of our prototype, and to maintain simplicity and timeliness, we focused our design around the drum's lug count only as a simplifying constraint and honed in on rack toms specifically.

Needs and Impact Analysis

Needs and Impact Analysis
The Drum-Autotuning project simplifies drum tuning by integrating hardware and software, making precise tuning accessible to all skill levels. It bridges musicianship and technology, raising questions about scalability across drum types and real-time acoustic feedback integration.

Cultural, Global, Economic, Environmental, and Social Impacts
Culturally, it enhances musical expression and standardizes sound quality. Globally, its affordability and mobile integration increase accessibility. Economically, it offers a cost-effective alternative to professional tuning services. Environmentally, its 3D-printed design minimizes waste, though sustainable electronics sourcing remains a challenge. Socially, it fosters community engagement and collaboration through its open-source approach.

Limitations and Drawbacks
Challenges include frequency measurement inaccuracies due to drum acoustics, potential over-reliance on automation, and environmental concerns from battery disposal. Addressing these will improve system robustness and sustainability.

Future Work Opportunities
Future developments include expanding frequency ranges for different drums, modular hardware design, and real-time acoustic analysis for other instruments. Multi-device synchronization for ensemble tuning is another potential advancement.

Ethical and Professional Considerations

User safety was ensured by limiting motor torque, and privacy was maintained by storing only temporary tuning data. The project adheres to engineering standards through iterative testing and transparent documentation. It promotes accessibility, affordability, and sustainability while responsibly advancing music technology.