Final Project Documentation: The Cube

Matt Steele

22 April 2024

Intro and Objective

The objective of "The Cube of Knowledge" project is to create an interactive, voice-controlled pentagonal box capable of being a basic home assistant that has total control over items such as the smart LEDs in my apartment . The cube utilizes a trained wake word for activation, thereafter capturing a brief audio snippet to be processed on a server and then sent back for action determination. The envisioned outcome is a seamless interaction between the user(myself) and their hyper intelligent cube through the most natural human medium, language.

0.1 Components and Hardware Design

- 2 I2S Microphones for wake word detection and audio processing. Each one corresponds to a separate board Link Here
- 2 ESP32 boards for processing. Link here
- 1 speaker for audio output Link here
- PAM8302A Audio Amplifier Link here

The hardware is put together in the way that the diagram below shows, though it took significantly more wires than arrows to have it working. The ESPs operate as two independent programmatic entities that will interact with each other through 2 one way state indicator wires. This is what allows them to transition state to be able to make the device interactive. All hardware components are soldered together inside the casing, with each ESP board having its own prototoboard. There is also a protoboard for the ground for all items connected to the wake word ESP, and a bunch of wago connectors for that on the command ESP. A diagram of the rough outline of the hardware connections is on the next page.

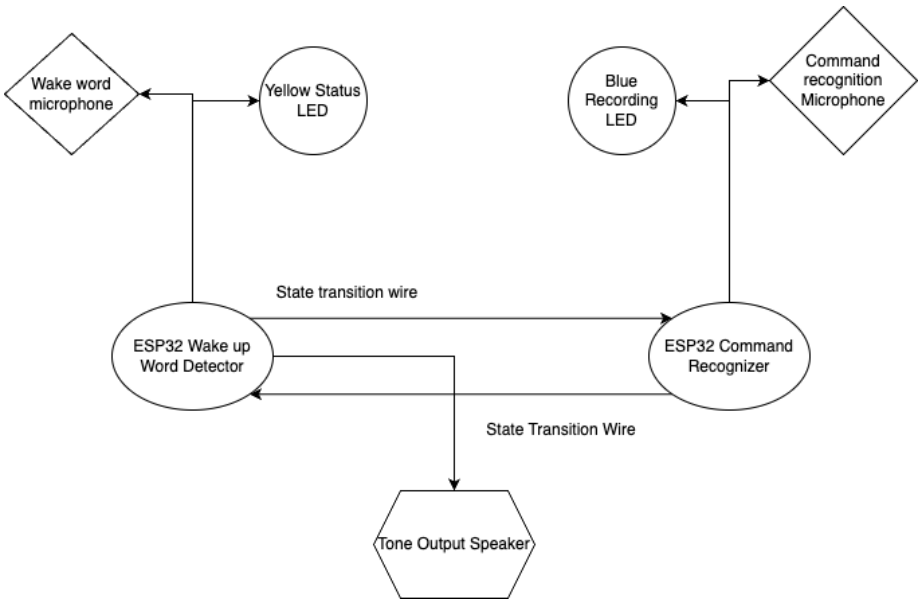

Figure 1: Overview of the Hardware Used

0.2 Software Libraries and Architecture

0.2.1 Software Architecture

The software architecture consisted of two state machines, one on each ESP. These state machines change by interacting on the hardware level as shown earlier. In terms of the software, the wake word is run on the first ESP. When activated, it creates a tone on the speaker, turns on an LED, and sends a signal to the second ESP to record. This will record for 3 seconds, the send to GCP speech to text. Upon the return of that, it will go ahead and process the command, send the proper API call to Govee, and then send the high signal back to the wake word ESP to reset its state to detect mode. A diagram of this is on the next page

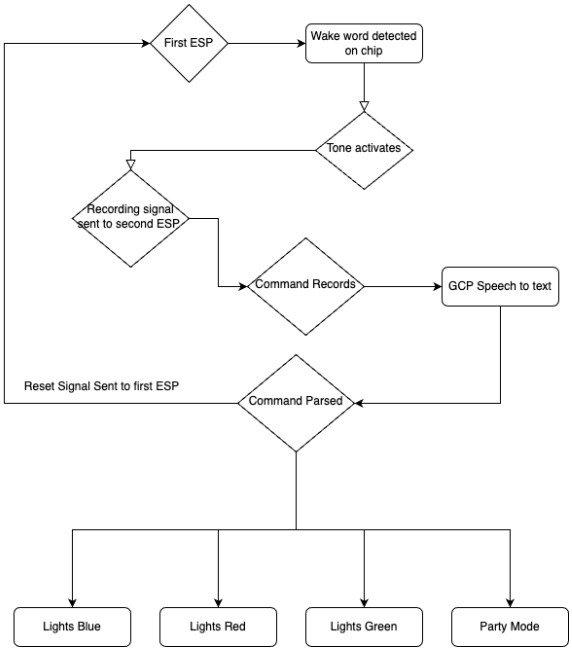

Figure 2: Software Architecture Diagram

0.2.2 Software Libraries

The software is a combination of multiple tutorials that were not fully functional, plus some custom extra code. The voice recognition piece came from the video linked here but the tutorial was outdated and didn't actually work in entirety. Their circuit in the video was non functional, and their code on GitHub had some segmentation faults. After fixing some of the segmentation faults, and changing the way the microphone was wired, I was able to get it to detect a wake word. However, their audio streaming code was entirely non functional, in addition to their MP3 audio output, so this was axed from the project. Instead, I added a secondary ESP32 that handled all tasks outside of that initial wake word detection. This was also partially from a tutorial linked here. This ESP handled sending audio up to GCP to process it into text, but the rest of the tutorial was not functional so I wrote my own custom processing code. From there, the connection to the Govee lights API was not documented well so I wrote that entirely from scratch.

0.3 Custom Components

There are numerous customization made during this project. Aside from the more trivial soldering and case design/fabrication, lots of custom software and hardware was made. Upon the addition of the second ESP32, I needed to design communication between micro controllers. Rather than have them pass information across wirelessly, I decided to make each have a software state machine that was changed upon activation of a certain pin input going high. This allowed me to have the cube understand when to listen for the wake word and when to not listen for the wake word, and also when to be trying to parse commands. This leads to lower amounts of energy being used by the chips. For the command parsing, the google speech to text connection was pre designed, however, parsing the output was not designed for me very well so I wrote my own parser to extract the JSON exactly. It had some trailing characters that made this nontrivial. From there, I pass them into my custom made command parsing function that then goes and calls custom code I wrote that interacts with the Govee API. This did not work with the Arudino JSON serialization function so I custom wrote a Stringified JSON and tested it until it worked.

0.4 Skills Learned

I got way better at soldering. I now understand ca certificates for web servers. I also now understand state machines a lot better. I also learned how to use tensorflow Lite, and more about how ML models get overfit(this is by default overfit to ignore noisy rooms, will hopefully fix this past the conclusion of the class). I had also never extensively worked with that much C++ code, so I had to learn that. Learning how audio input worked on I2S microphones for processing into other objects such as an API was also something that was well beyond the skill demos.

0.5 Iterative Process

This project, while exceeding its initial minimal requirements, got re-scoped multiple times due to things not working as expected. Initially, none of the tutorials worked at all for the microphone input. It turns out you need to have the L/R channel left sometimes and right other times even with the exact same configuration based on the firmware version of the ESP32. Eventually, once the wake word code got too frustrating to work with, I switched to a multi ESP design to be able to further the project. Additionally, the audio output got scoped out due to a lack of information

on doing it on chip with my given components. Adding the second ESP within a week of the project being due was quite intense.

0.6 Challenges and Solutions

There were many challenges encountered on the course of this project. The first being the fact that nothing worked as intended, so the re-scoping was quite constant. As stated in earlier sections, all of these tutorials exist but don't work in the way that was intended for this project. It became far more difficult than expected and had to go very much into documentation to sort things out. Overall, I was able to design my way around the issues, but would rate this project about 10 levels of magnitude beyond the sum of every skill demo in difficultly.

0.7 Lessons Learned

There are many lessons to be learned through the processes I took on this project. First and foremost, never trust random code on the internet from 4 years ago unless you want to use it one to one. It will be unbelievably painful to work through and is likely better to start from scratch. Had I simply gone through the process of making my own model inference code, starting only with the mic input, it would have likely been able to be a single board project. However, the complexity of working around strange quirks of the code of multiple tutorials ends up making you take long turns.

Secondly, Amazon Alexa type devices are infinitely faster than the thing I have made, and that this project is better suited as a Raspberry Pi project. However, if you do want to make it on a microcontroller, the understanding behind how all the peripherals work is much improved, and provides a more intellectually stimulating experience.

The third most important thing I learned in this project was to always have extra components. I had multiple components break or get fried in the process. This specifically happened to my audio amplifier because I carelessly plugged it in wrong. Also to one of my ESP boards initially.

0.8 Project Photos

Here are some photos over time.

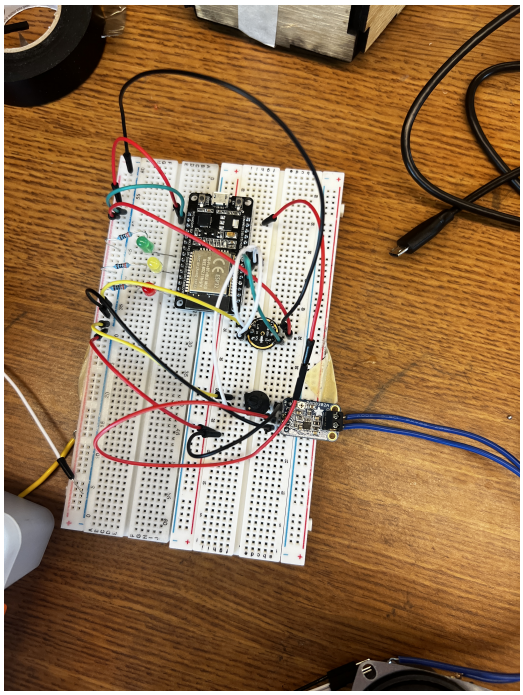

Figure 3: First test of wake word on breadboard

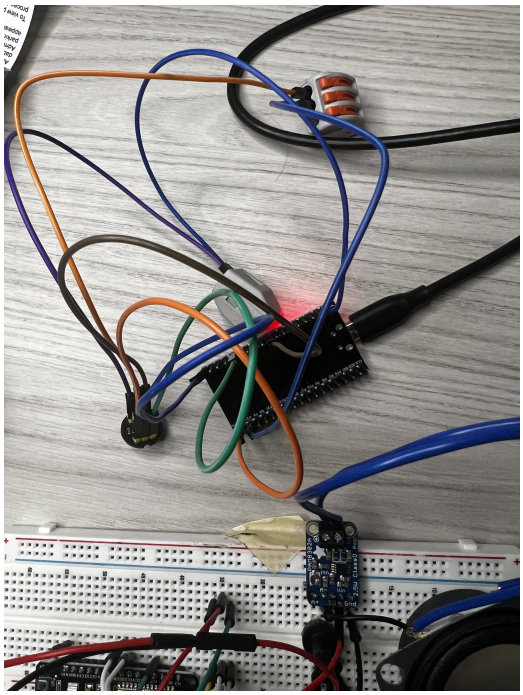

Figure 4: Second ESP connected

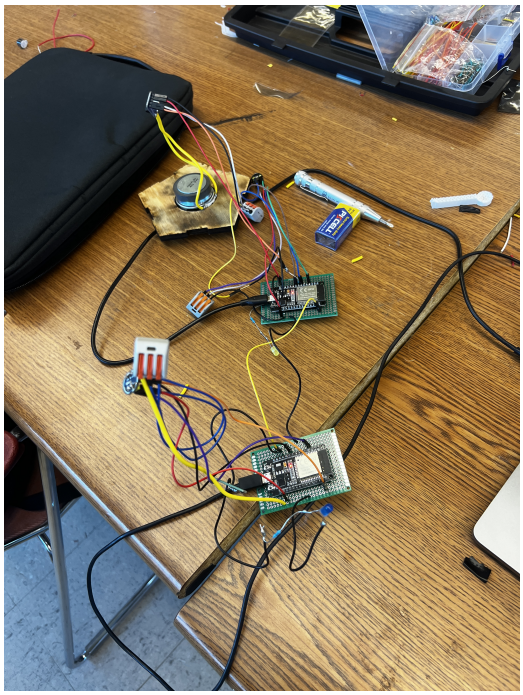

Figure 5: Soldering the circuit together

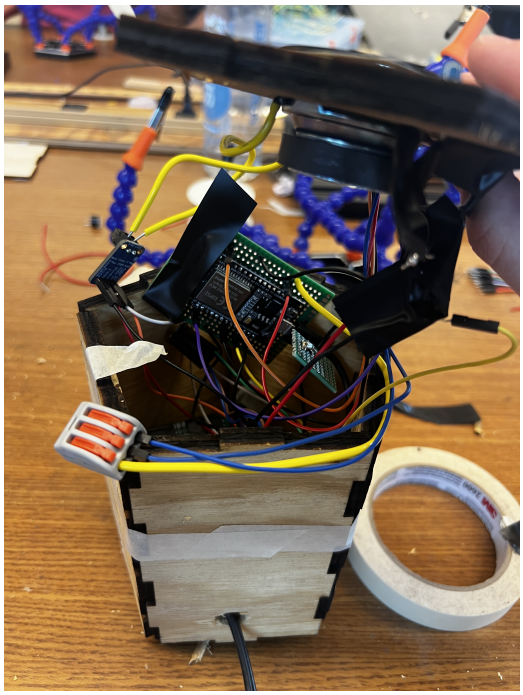

Figure 6: Photo of putting it into case

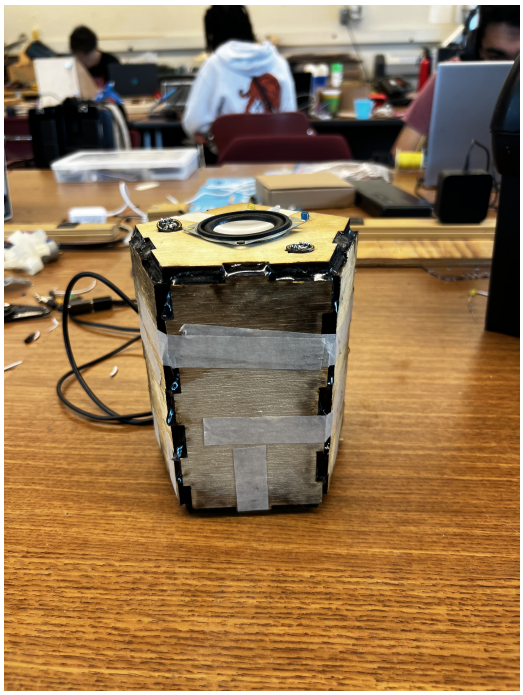

Figure 7: Final build