Voice Control Car

Cole Strickler, Jackson Yanek

Electrical Engineering and Computer Science
The University of Kansas
Lawrence, KS USA
{cole.strickler, jackson.yanek}@ku.edu

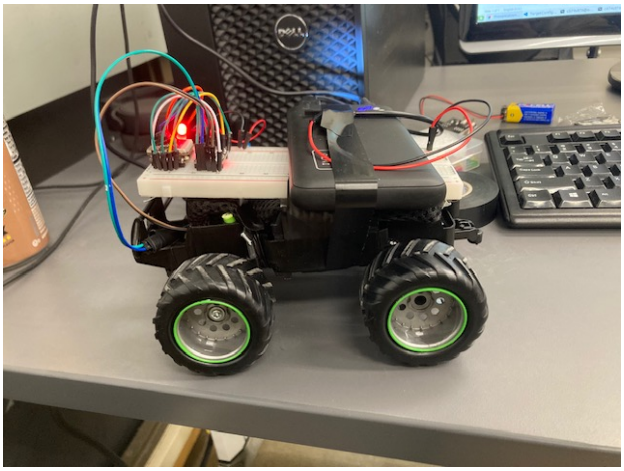

Fig. 1: ESP32 Voice Control Car

I. Introduction

Over the past 50 years, one of the major goals in technology is to create "autonomous systems" capable of removing or limiting human interaction with the system. In 2015, Tesla released their first iteration of "full self driving." This was the first glimpse of a potential realization of an autonomous car system. Since then, research into machine learning models has driven significant advancements in autonomy; however the constraints of power efficiency and physical compute capabilities demanded exploring new approaches. One of the ways to explore computational efficiency is to investigate computing on a small scale, as breakthroughs at a small scale often translate to larger systems. The ESP32 microcontroller is a common test bed for these kinds of explorations. For this project, we used a XIAO ESP32S3, which includes 8MB of ram, 14 IO pins, an additional camera and microphone board, and reduced form factor, making it suitable for a compact low-power application. In this report we will explore how we used a XIAO ESP32S3 microcontroller to control a car using a keyword-spotting machine learning model. Through this exploration, we aim to show the feasibility of autonomous systems on a constrained platform.

II. System Design

A. Hardware

Correct hardware configuration is vital for a successful deployment to any edge device. This is why we carefully selected the components to complement our ESP32. The final system, (As presented in fig. 2) consists of 5 parts. The microphone, the computing platform, the motor driver, the physical car, and a source of power. For our microphone, we chose to use the onboard microphone supplied with the XIAO ESP32S3 add-on board. Predictably, we also chose the only compatible platform for the microphone, the XIAO ESP32S3. In section IV-B, we investigate the digital limits of this platform, and how our project stacks up with the on board constraints. For the motor driver, we chose a DRV8833 [1]. This motor driver supports up to two concurrent input and output channels to control the motors. We use one input to the board for controlling the steering and the other for controlling the throttle. For the physical car, we used a Chevy New Bright 1:24 scale RC car, modified to allow for direct access to the motors. For power, we used an INUI Portable battery. By using a USB to jumper connection, we were able to supply both the ESP32 and the motor driver with 5 volts at 2.4 amps.

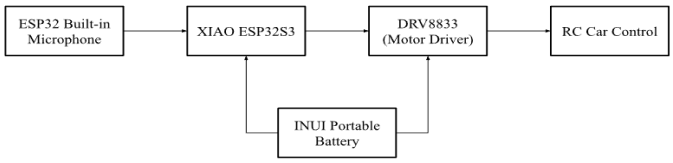

Fig. 2: High Level Hardware Configuration

B. Software

Software design is also a vital part of embedded development. For this project, we chose to write our code in C++ and deploy and debug it through the PlatformIO development suite. When writing our code the main focus was the physical output and responsiveness, which is explored further in section III.

C. Communication

To connect the hardware to the software, we utilized the ESP32's GPIO pins. In our code, we sent HIGH or LOW signals to the digital pins, which were then routed to the motor

driver. The DRV8833 motor driver used a specific combination of HIGH and LOW signals to get the preferred output. For example, to get motor 1 to rotate, one input must be set to HIGH and the other must be set to LOW. When sufficient power is supplied, the motor will run.

Input1,Input2 (From ESP32)	Motor Activity
LOW, LOW	No Output
HIGH, LOW	Rotate
LOW, HIGH	Rotate (reverse)
HIGH, HIGH	No Outtput

TABLE I: Evaluation platform specifications

III. Implementation

A. Dataset Formulation

Our base dataset of choice was the google speech commands dataset [2]. This dataset includes 35 example keywords. The dataset information can be summarized in Fig. 3. From this set of keywords, we utilized go, stop, right, and left. We also allocated a category for silence/ambient noise, bringing our total class count to five. This subset mapped directly to basic commands for the car. During our initial prototyping phase, we found that the model performance was not up to expectations, and as an attempt to fix this we decided to collect our own data samples. Part of the issue that was causing low inference accuracy during deployment was the noise that the car produced while running. To combat this, we collected data samples with the car emitting noise in the background. We collected a total of 50 data samples for each category, with half of them being recorded with motor noise in the background. We observed a significant qualitative improvement in deployment performance after adding these data samples. We also opted to augment our data by adding random noise to the generated spectrograms. Doing this caused a slight decrease in the validation accuracy during training, but provided better model performance during deployment and we posit that this is due to the noisy environment that the model was expected to run in.

B. Model Design

We opted to use Edge Impulse [3] to train and design our model due to the automatic integration of audio signal processing and the quick time to deployment that the platform offers. We show an overview of the model design in 4. We tested multiple model designs that were larger than the one shown, but they seemed to be too complex to fit well with the size of the processed inputs. Additionally, we found that the default audio model was not sufficient to capture the full complexity of the input features, and increasing the size of the convolutional layers, along with appending fully connected layers before the classification greatly improved model accuracy. The dropout layers inserted after the convolutional layers were observed to increase the inference accuracy upon the validation set.

Word	Number of Utterances
Backward	1,664
Bed	2,014
Bird	2,064
Cat	2,031
Dog	2,128
Down	3,917
Eight	3,787
Five	4,052
Follow	1,579
Forward	1,557
Four	3,728
Go	3,880
Happy	2,054
House	2,113
Learn	1,575
Left	3,801
Marvin	2,100
Nine	3,934
No	3,941
Off	3,745
On	3,845
One	3,890
Right	3,778
Seven	3,998
Sheila	2,022
Six	3,860
Stop	3,872
Three	3,727
Tree	1,759
Two	3,880
Up	3,723
Visual	1,592
Wow	2,123
Yes	4,044
Zero	4,052

Fig. 3: Google Speech Commands 2018 Dataset

C. Model Deployment

The use of Edge Impulse greatly simplified our model deployment process. We opted to export the model using INT8 quantization, and the effects of this choice will be discussed in IV. We integrated the exported library into our project, and were able to instrument the model and connect it to other components. The main component that the model communicated with was the DRV8833 motor driver as previously discussed in II-C. This driver was used to control steering and speed of the car. Each voice command corresponded directly to a command sent to the motor driver. To prevent erratic behavior from false positives, we added a filter that would prevent a command being sent when P(command) <= 0.5.

IV. Analysis

A. Training Results

We performed a standard 80/20 split of our data into training and validation sets. We utilized a batch size of 32, a learning

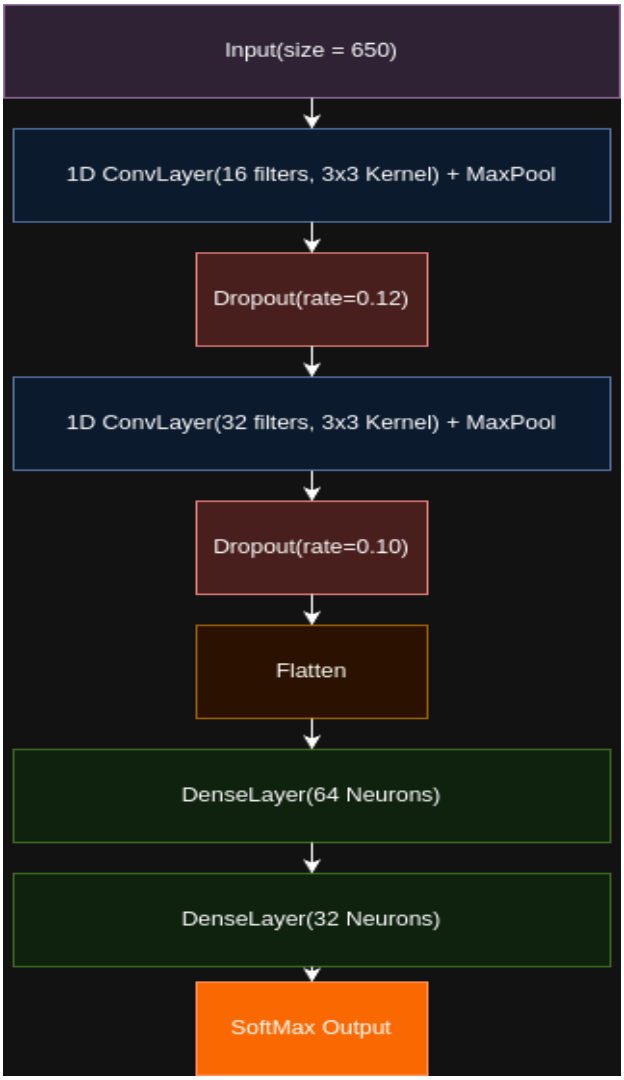

Fig. 4: Neural Net Design

rate of 0.005, and trained over 100 epochs. This setup allowed our model to achieve 93.7% accuracy on the validation set and 95% accuracy on the test set. A visualization of this can be seen in fig. 5.

B. Quantitative Analysis

As can be seen in table II, the size of our model was quite small. There are a total of 27,045 trainable parameters, and this amounts to 26.41 kilobytes. The amount of memory needed for an individual layer never exceeds 1,418 bytes. The audio signal pre-processing module uses a maximum 15.4 kilobytes. Given these statistics, we are well within the means of the ESP32, which sports 8 megabytes of RAM.

C. Latency Analysis

Latency must be taken into account as a critical measure of our model's performance. The nature of the targeted application, control of a car, required our model to process and execute commands in real time. Failure to do so all but

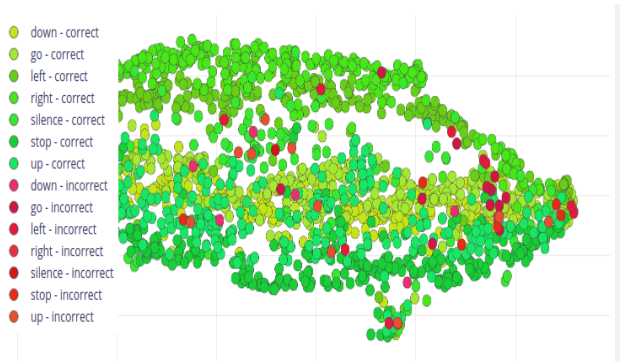

Fig. 5: Training Results Visualized

guarantees that a crash will occur. The latency from sound to classification is dominated by the audio signal processing stage that refines the sensor data into a format suitable for uptake by a CNN. This pre-processing stages takes 375 milliseconds, while the classifier takes 6 milliseconds when the model is quantized, and 77 milliseconds when the model remains in standard float32 format. 375 milliseconds is already a huge amount of time when considered for the application at hand, and was prohibitive towards the types of environments that we could deploy this particular system in. Adding any more latency would make even the most favorable environments infeasible, and therefore we opted to use the INT8 format. It should be noted that there was no inference accuracy tradeoffs for this choice. The bottleneck of audio signal processing will be considered further in IV-C.

V. Challenges

A. Hardware Challenges

The first challenge that we came across was the low sensitivity of the default ESP32 microphone. When we were testing our model's inference on-device, we noticed that many times sounds just seemed to be ignored. We hypothesized that this was due to the relatively low quality of the microphone, and we confirmed this by using the features of EdgeImpulse to deploy to an iPhone and test how the model performed. The model performed much better when using the iPhone's microphone. If we were to attempt to improve our end-to-end design, our first step would be to purchase a higher quality microphone.

B. Software Challenges

It was already mentioned in IV that the latency of audio signal processing presented a challenge for real-time deployment of our system. To overcome this issue, we were forced to make some tweaks in the code that affected the overall functionality of the car. An example of one of these changes is that we chose to operate the car at a fixed slow speed. We chose to do this because the car took 0.4 seconds to respond and execute a command. If the car was travelling at 3m/s and then was instructed to stop, the car would travel another 1.2 meters before executing a stop. By significantly

Layer	Output Shape	Input Buffer Size	Output Buffer Size	RAM Utilization	MACs	Parameters
Input(650 features)	(650,)	650	650	1300	0	0
ReshapeLayer((-1,13), input$_shape = (650,)$)	(50,13)	650	650	1300	0	0
Conv1D(filters=16, kernel_size=3, strides=1)	(48,16)	650	768	1418	29,952	640
MaxPooling1D(1,2)	(24,16)	768	384	1152	0	0
Dropout(0.12)	(24,16)	384	384	768	0	0
Conv1D(filters=32, kernel_size=3, strides=1)	(22,32)	384	704	1088	33792	1,568
MaxPooling1D(1,2)	(11,32)	704	352	1056	0	0
Dropout(0.10)	(11,32)	352	352	704	0	0
Flatten()	(352,)	352	352	704	0	0
Dense(64, activation='relu')	(64,)	352	64	416	22528	22592
Dense(32, activation='relu')	(32,)	64	32	96	2048	2080
Dense(5, activation='softmax')	(5,)	32	5	37	160	165

TABLE II: Model Quantitative Complexity Evalutation

slowing down the speed of the car, we were able to somewhat overcome this challenge. Although we overcame the slow speed of signal processing in software, it is fundamentally a hardware problem. Choosing an embedded device that comes with a dedicated digital signal processor could greatly alleviate the impact of this issue.

C. Environmental Challenges

Deployment of a keyword-spotting model to a model car presents some interesting challenges. The small size of the car necessitates that the microphone is located very proximally to the engine, and therefore the source of a large amount of noise. This noise caused decreased accuracy during deployment, which we attempted to alleviate via data augmentation discussed in section III-A, and resulted in intermittent false positives which we chose to counter with a confidence filter also discussed in II. Not all choices of remote control car would be as noisy as ours, and choosing an option that emits less engine noise would greatly improve the operation of our system.

VI. Discussion/Future Work

The control of a remote control car via voice presents some interesting challenges and begs some further questions about whether this type of system is advisable for domains beyond toy implementation. As was mentioned, latency is a key issue, even in toy systems, and the bottleneck of audio pre-processing must be solved to make such a system real-time feasible. Beyond this consideration, we should ask whether voice control should be put onto other types of vehicles and if so in what situations. We believe that there are limited domains where such a system could prove useful which are natural extensions of the techniques that we explored in this project. An example application of such a system would be calling out to a cruising robo-taxi, which would then alter its current route and proceed to pick you up. There are countless cases where implementing voice command functionality in a robot would increase its interactivity and make them much more useful to human users. For future work we would first of all like to overcome the challenges mentioned in section V, and specifically those that are hardware related. By equipping our system with a better microphone the inference accuracy in deployment would be greatly improved. Instead of the ESP32, we would like to try this system with a device that comes equipped with specialized signal processing hardware. Closing the time taken for the pre-processing step would allow us to eliminate some of the constraints we put on our system such as speed and allow us to target more complex operational environments. Looking even further forward, it would be interesting to start attempting to target some of the potential real world applications that we mentioned in the first paragraph of this section. A first step in this could be implementing the keyword-spotting model as a respective member of a cascading model, that could act as an interrupt to another member model and alter its goal.

References

[1] "Controlling DC Motors with DRV8833 Motor Driver and Arduino," https://lastminuteengineers.com/drv8833-arduino-tutorial/.

[2] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018. [Online]. Available: https://arxiv.org/pdf/1804.03209

[3] "Edge Impulse," https://edgeimpulse.com/.