

Project Report

Echo Runner - Voice Controlled Game — a browser game steered entirely by an AI keyword-spotting model

LRN International GCSE Artificial Intelligence (7923) — Paper 2, Component B

PRACTICE DRAFT - the accuracy figures below are placeholder values, not measured results

Candidate name	Muhammad Usama
Candidate number	2345
Centre name	
Project title	Echo Runner - Voice Controlled Game
Tool used	Edge Impulse (WebAssembly export, keyword spotting)
Date	2 September 2026

1. Explanation of the AI System and Objective

My system is a browser game controlled by voice. It listens through the microphone and decides which of three commands was spoken - left, right or stop - and a fourth label, backgroundnoise, lets it ignore silence and conversation. I built the model with Edge Impulse and deployed it as a web page: the page takes a sliding one-second window of 16 kHz mono audio, sends it to the model ten times a second, and the winning word sets the runner's movement state.

I chose this problem because the same word produces a completely different signal for every voice, accent, distance from the microphone and piece of hardware, so there is no waveform rule that means 'left'. A model that learns the words from recorded examples is the only practical approach. I also wanted a project where the AI has to work in real time rather than on an uploaded file, because that adds a constraint an image classifier never faces: the answer has to arrive fast enough to be a control. It would be usable by anyone who cannot easily use a keyboard.

2. Step-by-Step Process

Step 1: Problem definition

The model had to listen to one second of audio and return one label out of four - left, right, stop or backgroundnoise - with a confidence score. The fourth label is the important one: without it the game would try to interpret silence and background conversation as commands, and the character would move on its own.

Step 2: Choosing the tool

I used **Edge Impulse**. I moved to it from Teachable Machine because I needed control over the training data and a model that runs offline. Edge Impulse lets me record and label my own audio samples, retrain quickly between versions, and export the result as WebAssembly, so the model loads from my own site and the audio never leaves the machine.

Step 3: Collecting the data

I recorded the command samples myself rather than downloading a dataset, so the audio matches the microphone and setting the game is played in. **Source: recordings I made myself, of my own voice, on the machine the game is played on. Licence: none is needed - the recordings are my own work and no other speaker appears in them.** There are four labels: **left, right, stop** and **backgroundnoise**. The backgroundnoise class contains room tone, typing, and speech that is not a command.

Step 4: Preparing the data

Audio was recorded at 16 kHz in one-second windows, which is the window length the model expects. I removed clipped and mis-triggered samples, then split the recordings **80 / 20** into training and test data. Keeping the test recordings separate matters even more for audio than for images, because it is very easy to score well on your own voice in your own room and then fail on anybody else's.

Step 5: Training the model

Edge Impulse trained a **keyword spotting** model on the recorded samples; the deployed build is version 8. Each one-second window is turned into a spectrogram-style feature set before the network sees it, which is what makes the same word recognisable across different pitches. Measured on the v8 export: **3.4 ms** per classification, **10 predictions per second**, and an **8.07 MB** model.

Step 6: Testing the model

The model was tested on separate audio it had never heard. Test accuracy: **81%**. I also measured the behaviour that accuracy does not capture: how fast the first guess arrives after the page loads, and what the model does when there is no sound at all.

Step 7: Improving the model

The failure I found was serious and would never have shown up in an accuracy score: exact **digital silence was classified as 'left' at 99.6% confidence**, so a muted microphone made the character run on its own. I added a silence gate that rejects windows below an energy threshold, and pre-filled the audio buffer with faint noise so the model never sees a perfectly empty window at start-up. After retraining and exporting version 8, the time from start to the first guess fell from **1.58 s to 0.93 s**.

Step 8: Deploying the model

In Edge Impulse I used Deployment - WebAssembly - Build, and copied the exported edge-impulse-standalone.js and .wasm into the game's js/ei/ folder. The page requests microphone access with getUserMedia, keeps a rolling one-second buffer of 16 kHz mono samples, and classifies it ten times a second. Two hosting requirements follow from that: the .wasm must be served as application/wasm, and getUserMedia only works over HTTPS, so the game is hosted on a real secure host rather than opened as a local file.

3. Evaluation of Results

Test accuracy: **81%**

Test accuracy on unseen audio: **81%**. The measured runtime figures matter as much as the accuracy for a game: **3.4 ms** per classification and **10 predictions per second**, with an **8.07 MB** model. I had assumed inference would be too slow to control a character and it turned out not to be - measuring beat guessing.

It handled well: the three commands spoken clearly, close to the microphone, in my own voice, which is what it was trained on. It found hard: other people's voices and microphones, quiet or rushed speech, and words spoken while background conversation was running. In one session I counted 'right' firing 39 times against 'left' only 3 times, which is not the model being random - it is my 'left' training data being weaker than my 'right' data.

The worst case is the one the accuracy score hid completely: **digital silence returned 'left' at 99.6% confidence**. That single number is the best evidence I have that high confidence does not mean correct - the model was as certain as it can be about a word that nobody had said. The silence gate is therefore a permanent part of the system

rather than a temporary patch, because the underlying model still behaves that way on an empty window.

4. Screenshots and Evidence

Each screenshot below is captioned with the input, the predicted label, the confidence and the time it was taken.

[paste screenshot here — a clear, easy example]

Caption: A correct result — input _____, predicted _____, confidence ____%, taken _____.

[paste screenshot here — a different class from the first]

Caption: A second correct result — input _____, predicted _____, confidence ____%, taken _____.

[paste screenshot here — one of the failures described in Step 7]

Caption: A hard or failed case, before improvement — input _____, predicted _____, confidence ____%, taken _____.

[paste screenshot here — the improvement, for direct comparison]

Caption: The same case after retraining — input _____, predicted _____, confidence ____%, taken _____.

5. Ethics and User Consideration

Privacy: the audio is processed inside the browser and is never recorded, stored or uploaded - the one-second buffer is continuously overwritten, so nothing older than a second exists anywhere. This is the single most important design decision in the project: an always-on microphone that sent audio to a server would be an entirely different thing to put on a web page, no matter how good the model was.

Limitation: exact digital silence still returns 'left' at 99.6% confidence. The silence gate stops that reaching the game, but it is a guard placed around the model rather than a fix inside it, and anyone reusing the model without the gate would hit the same bug.

Bias: this one is measured rather than assumed. In one session 'right' fired 39 times against 'left' 3 times, because my 'left' training data is weaker. The model works best for voices that sound like mine, which means an accent or a pitch unlike my own gets a worse game. The real fix is recording more speakers, especially for 'left' - not lowering the confidence threshold to compensate, which would only trade one failure for another.

Note on Tools and Honesty

The dataset, the training, the testing and the judgements in this report are my own work. I used an AI assistant to help with the layout and CSS of the web page that displays the model, and to check the spelling and grammar of this document. The

model itself was trained by me in Edge Impulse, and every number quoted here is one I measured.