National College of Ireland

Browser-Native Machine Learning: WebAssembly and WebGPU-Accelerated Training for Classical Text Classification Models

MSc Research Project
MSc Data Analytics

Shashidhar Yaligar
Student ID: x23426047

School of Computing
National College of Ireland

Supervisor: Sallar Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Shashidhar Yaligar

Student ID: X23426047

Programme: MSc Data Analytics **Year:** 2025 - 2026

Module: MSc Research Project

Supervisor: Sallar Khan

Submission Due Date: 28/01/2026

Project Title: Browser-Native Machine Learning: WebAssembly and WebGPUAccelerated Training for Classical Text Classification Models

Word Count: 10518 **Page Count** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Shashidhar Yaligar

Signature: .....

28/01/2026

Date: .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction

It is a Configuration Manual of the software and environment setup and procedures necessary to execute the GPU-accelerated text-classification system written in Rust, WebAssembly (WASM) and WebGPU and within Safari on macOS. The instructions include preprocessing, model step, and build instructions required to execute Logistic Regression, SVM, and MLP models using the compute shader on the graphics card.

2. System Requirements

The following hardware and software must be available before configuration:

Hardware

- macOS device with **Apple M1/M2/M3 GPU** (tested on M2, macOS 26.0.1)
- Minimum **8 GB RAM**
- 256 GB storage

Software

- Safari 17+ or **Safari Technology Preview**
- Rust (latest stable)
- wasm-pack
- wasm-bindgen
- A local development server (Python/Node)

3. Browser Configuration (Safari WebGPU)

WebGPU must be enabled to run compute shaders inside Safari.

Enable Developer Menu

Enable WebGPU

- Safari → Develop → Experimental Features → WebGPU
- WebGPU Developer Features

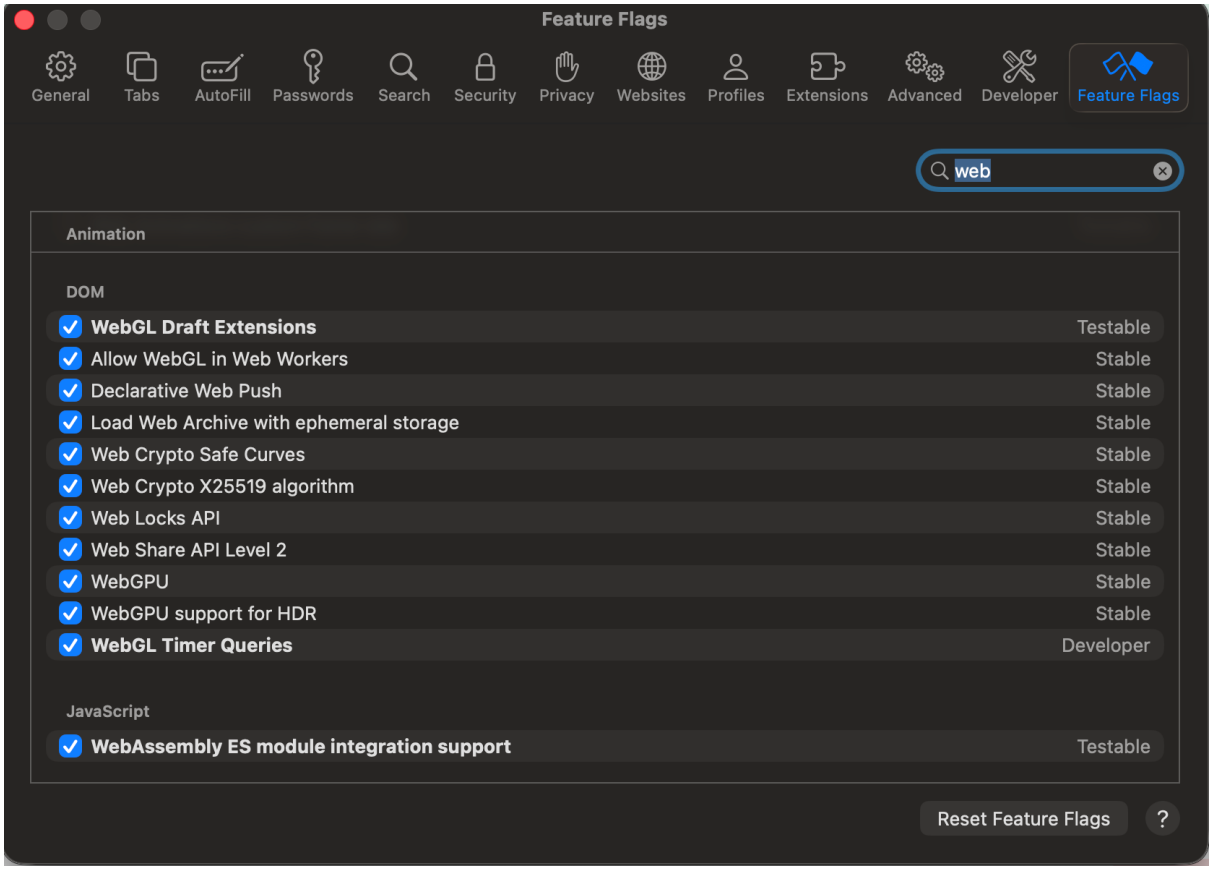

Figure 1. WebGPU Enabled in Safari Experimental Features

4. Software Tools Setup

Rust Toolchain

Install Rust and add the WASM target:

rustup target add wasm32-unknown-unknown

wasm-pack Installation

brew install wasm-pack

Version Checks

rustc --version
wasm-pack --version
wasm-bindgen --version

```
(venv) (base) shashidharyaligar@AA:55:8C:7D:68:2E news_classifier % rustc --version
wasm-pack --version
wasm-bindgen --version
Figure 3. Tool Version

rustc 1.91.0 (f8297e351 2025-10-28)
wasm-pack 0.13.1
wasm-bindgen 0.2.105
```

Figure 2. Tool Version Outputs

5. Project Dependencies

All dependencies are listed in **Cargo.toml**, including:

- wasm-bindgen, wasm-bindgen-futures
- serde & serde_json
- web-sys with WebGPU features
- rand, getrandom (js feature enabled)
- console_error_panic_hook

These libraries support WASM bindings, GPU access, preprocessing, and error reporting.

Figure 4. Cargo.toml Dependencies

```
[dependencies]
wasm-bindgen = "0.2"
wasm-bindgen-futures = "0.4"
js-sys = "0.3"
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
console_error_panic_hook = "0.1"
rand = "0.8"
#Important for WebAssembly (fixes getrandom error)
getrandom = { version = "0.2", features = ["js"] }
[dependencies.web-sys]
version = "0.3"
features = [
   "console",
   "Window",
   "Navigator",
   "Performance",              # <-- for window.performance().now()
   "Gpu",
   "GpuAdapter",
   "GpuDevice",
   "GpuQueue",
   "GpuBuffer",
   "GpuBufferDescriptor",
   "GpuBufferUsage",
   "GpuMapMode",
   "GpuShaderModule",
   "GpuShaderModuleDescriptor",
```

```
    "GpuComputePipeline",
    "GpuComputePipelineDescriptor",
    "GpuProgrammableStage",
    "GpuPipelineLayout",
    "GpuBindGroup",
    "GpuBindGroupDescriptor",
    "GpuBindGroupEntry",
    "GpuBindGroupLayout",
    "GpuBufferBinding",
    "GpuCommandEncoder",
    "GpuCommandEncoderDescriptor",
    "GpuCommandBuffer",
    "GpuComputePassEncoder",
    "GpuComputePassDescriptor",
]
```

6. Dataset Configuration

Dataset Used

BBC News balanced dataset
Stored in:

```
dataset/
  bbc_train.json
  bbc_test.json
```

Expected JSON Structure

```
{
  "texts": [...],
  "labels": [...]
}
```

Dataset Folder Structure
bc_mlp_wasm_copy_tuned/phase6-newsgroups-rust-wasm/bbc_news_sample.json

7. Preprocessing Setup

Text preprocessing and vectorisation occur in **text_processor.rs**.

Steps Performed

- Text normalisation
- Tokenisation
- Vocabulary construction (5000 terms)

- Bag-of-words vectorisation
- L2 normalisation

Figure 7. Preprocessing Code Snippet (Tokenisation & Vocab)

```
// Build vocabulary
        self.vocab.clear();
        for (idx, (word, _count)) in
sorted_words.iter().take(self.max_features).enumerate() {
            self.vocab.insert(word.clone(), idx);
        }

        self.vocab_size = self.vocab.len();

        web_sys::console::log_1(&format!(
            "✅ Vocabulary built: {} words from {} candidates",
            self.vocab_size,
            sorted_words.len()
        ).into());
    }

    pub fn transform(&self, text: &str) -> Vec<f32> {
        // 🔧 FIX: Always use max_features, not vocab_size!
        let mut vector = vec![0.0; self.max_features];
        let words = self.tokenize(text);

        for word in words {
            if let Some(&idx) = self.vocab.get(&word) {
                // Only set if idx is within bounds (safety check)
                if idx < self.max_features {
                    vector[idx] += 1.0;
                }
            }
        }


        vector
    }

    pub fn get_vocab_size(&self) -> usize {
        self.vocab_size
    }

    fn tokenize(&self, text: &str) -> Vec<String> {
        text.to_lowercase()
            .split_whitespace()
            .map(|word| {
                // Remove punctuation
                word.chars()
                    .filter(|c| c.is_alphanumeric())
```

```
                    .collect::<String>()
            })
            .filter(|word| !word.is_empty())
            .collect()
    }
```

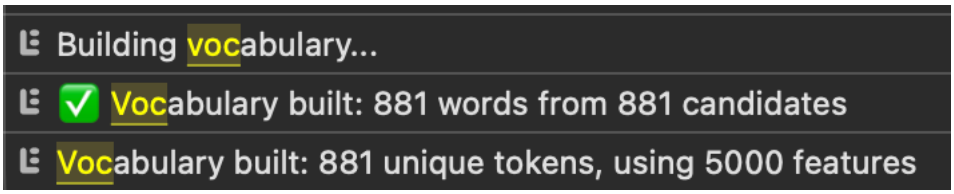

Figure 3. Vocabulary Size Log (Safari Console)

8. GPU Model Configuration

The system includes three models:

8.1 GPU Logistic Regression

- Fully GPU-accelerated forward & backward pass
- Adam optimisation
- Configurable: learning rate, batch size, epochs, label smoothing

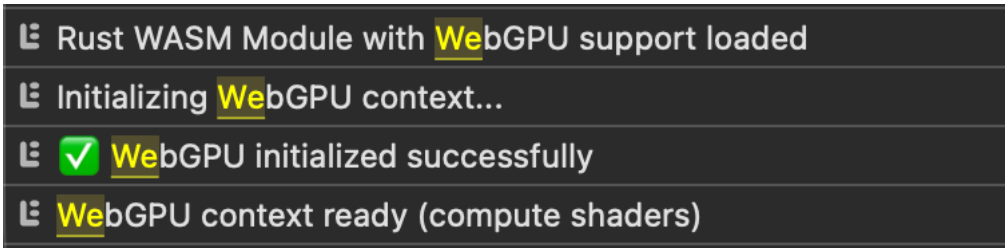

Figure 4. WebGPU Adapter & Device Initialisation

9. Build & Deployment Steps

Step 1 — Build WASM Module

wasm-pack build --target web --release

Produces:

pkg/
  project_bg.wasm
  project.js

Figure 5. wasm-pack Build Output

Step 2 — Move Files to Static Folder

www/
  index.html
  project_bg.wasm
  project.js

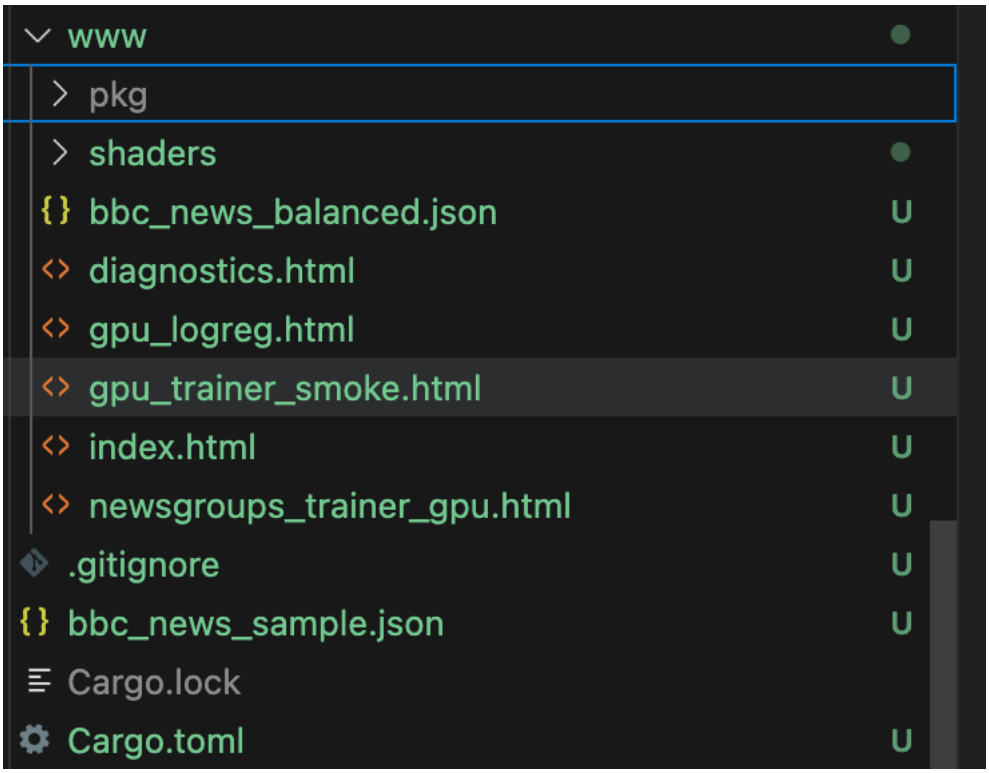

Figure 6. Static Folder Structure

Step 3 — Start Local Server

Python:

python3 -m http.server 8000

Node:

npx http-server ./www

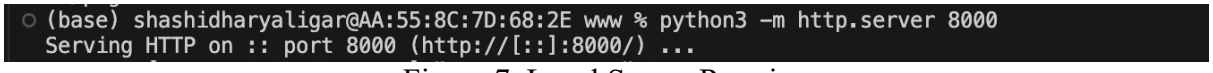

Figure 7. Local Server Running

10. Running the Application in Safari

Open:

http://localhost:8000

Open Web Inspector

Press:

Option + Command + I

Training logs will display:

- Adapter initialisation
- Pipeline creation
- Batch operations
- Epoch accuracy and loss

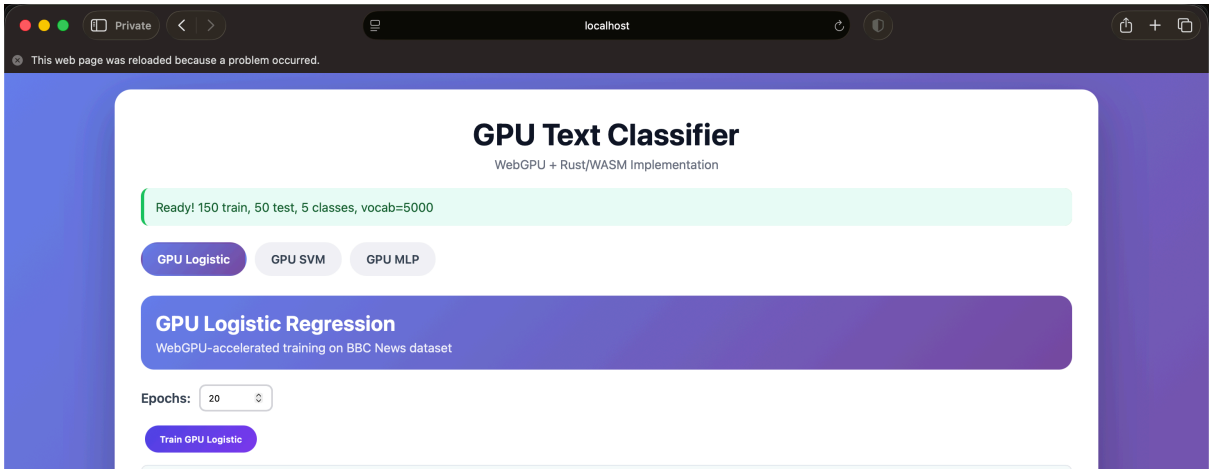

Figure 8. UI Loaded in Safari

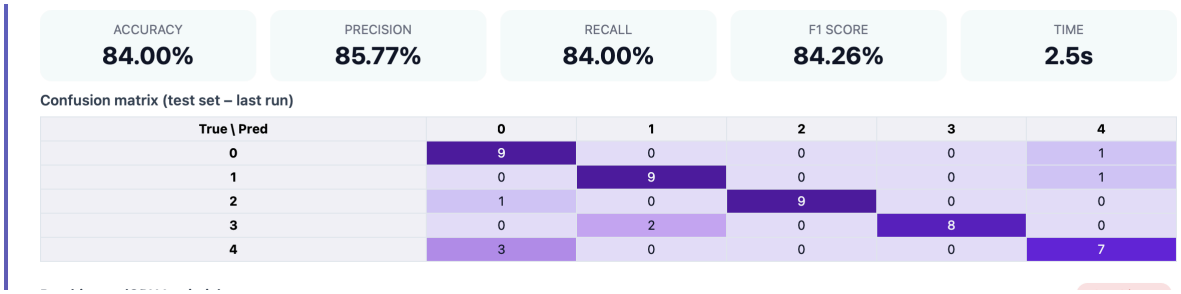

Figure 9. Final Evaluation Metrics (Accuracy, Precision, Recall, F1)

11. Troubleshooting

Issue: WebGPU not detected

Enable:
Safari → Develop → Experimental Features →
✓ WebGPU

Issue: “GPUAdapter is null”

Restart Safari after enabling flags.

Issue: Validation Errors

Typically caused by:

- Incorrect buffer sizes
- Mismatch between shader bindings and buffer definitions
- Unsupported workgroup sizes

---

12. Reproducibility Checklist

Requirement	Status
Rust installed	Yes
wasm-pack installed	Yes
WebGPU enabled in Safari	Yes
Dataset placed correctly	Yes
WASM builds successfully	Yes
Training logs visible	Yes
Metrics match expected	Yes

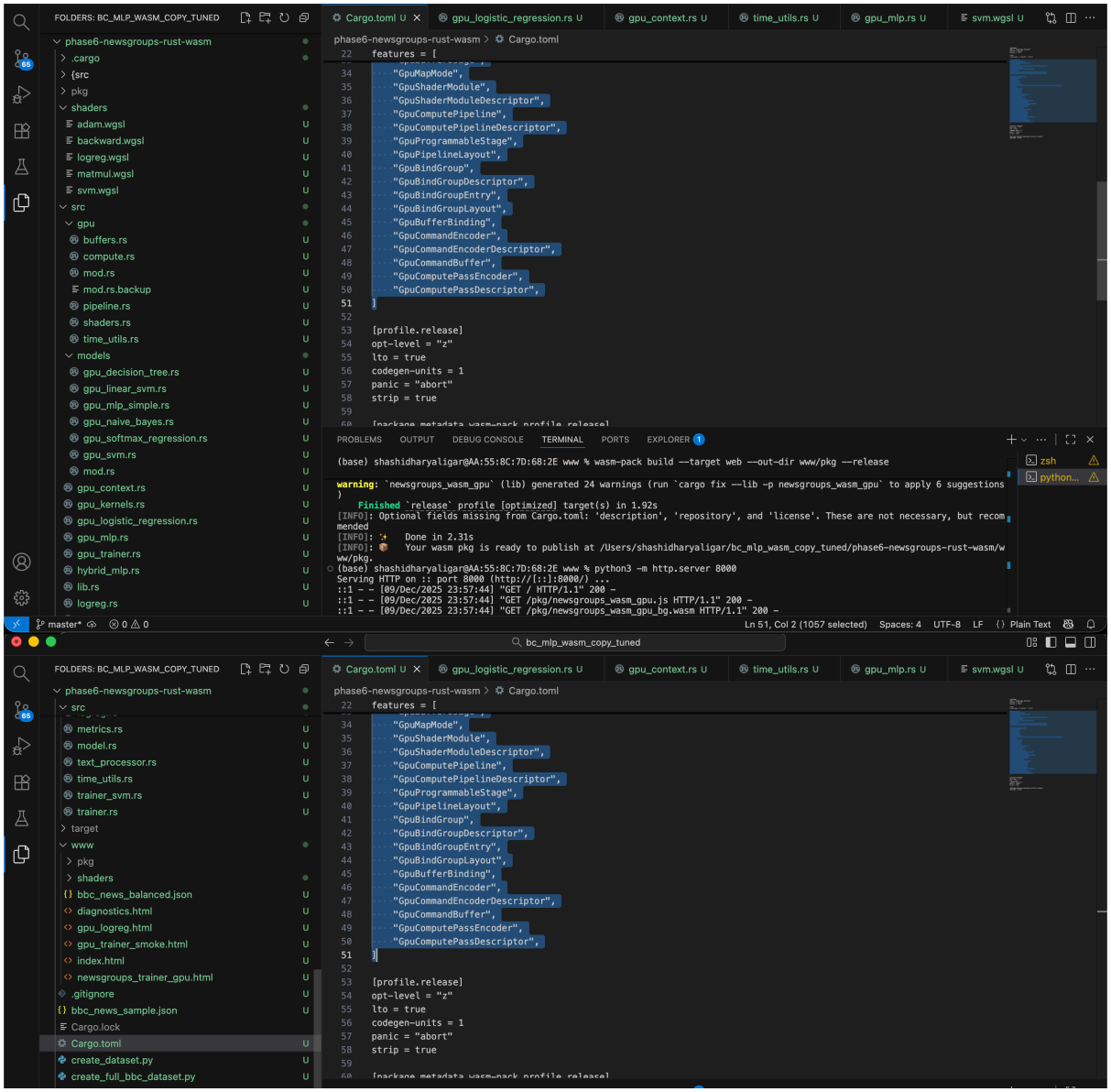

Figure 10. Final Project Folder Layout