Morse Code Translator using TinyML

Authors:

Shail Garg, Yerukola Gayatri, Surya Kausthub A, Pooja Gowda
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru, Amrita Vishwa Vidyapeetham

---

🔍 Overview

This project presents a comparative analysis between two approaches for Morse code translation via touch input:

- **Edge Computing** using TinyML on an ESP32 with a TTP223B capacitive touch sensor
- **Cloud-based** translation using an LSTM autoencoder deployed on a remote server

The objective is to evaluate trade-offs between **accuracy, latency, and deployment efficiency** for real-time Morse code decoding.

🚀 Features

- Tap-based Morse code input via capacitive touch sensor
- Real-time decoding using an LSTM model on ESP32 (TinyML)
- High-accuracy cloud decoding using PyTorch-based LSTM model
- REST API-based cloud inference
- Performance comparison with metrics: **Accuracy**, **Latency**, **F1-Score**, **Deployment Complexity**
- Designed for applications in **assistive tech**, **emergency communication**, and **wearables**

🧱 Hardware & Software Requirements

Hardware:

- ESP32 Microcontroller
- TTP223B Capacitive Touch Sensor
- OLED Display (Optional, for TinyML output)

Software:

- Arduino IDE
- PlatformIO (Optional)
- Python 3.8+

- PyTorch
- TensorFlow & TensorFlow Lite
- Flask (for cloud API)
- Excel (for dataset editing: morse_dataset.xlsx)

📁 Folder Structure

```
├── cloud_model/
│   ├── morse_lstm.py
│   ├── translate_api.py
│   └── cloud_transformer.pt
├── tinyml_model/
│   ├── morse_lstm_tinyml.ipynb
│   ├── tflite_model.h
│   └── main_esp32.ino
├── dataset/
│   └── morse_dataset.xlsx
├── README.md
└── requirements.txt
```

📊 Dataset

- Dataset: morse_dataset.xlsx
- Format: Sequences of dots and dashes (0 = dot, 1 = dash) paired with alphanumeric characters
- Total samples: 2000+
- Balanced distribution of letters and digits

🧠 Model Architecture

TinyML Model:

- Framework: TensorFlow Lite
- Architecture: 2-layer LSTM (quantized)
- Deployment: Runs on ESP32 (TFLite Micro)

Cloud Model:

- Framework: PyTorch
- Architecture: 3-layer LSTM with 128 hidden units each
- Deployment: Flask API served over HTTP

⚙️ How to Run

1. Train the Models

Cloud:

```
cd cloud_model

python morse_lstm.py
```

TinyML:

- Run morse_lstm_tinyml.ipynb and export the model as TFLite .h file
- Flash main_esp32.ino to ESP32 using Arduino IDE

2. Deploy the Cloud API

```
cd cloud_model

python translate_api.py
```

3. Send Morse Code via ESP32

- Connect the TTP223B sensor to ESP32 GPIO
- Taps will be processed and either:
  - Decoded locally (TinyML)
  - Or sent via HTTP POST to the cloud server

🧪 Evaluation

Metric	**TinyML (ESP32)**	**Cloud LSTM**
Accuracy	77.0%	89.0%
Precision	75.0%	88.0%
Recall	78.0%	90.0%
F1 Score	76.0%	89.0%
Avg Latency	~0.12 sec	~80.89 sec

Metric	TinyML (ESP32)	Cloud LSTM
Internet Required	❌	✅

🔄 Use Cases

- **Assistive communication** for people with speech/motor disabilities
- **Disaster response** in high-noise or tech-constrained environments
- **Wearable device** integration
- **Offline communication** using edge AI

📚 References

See the References Section in the report for all cited literature.

📬 Contact

For queries or collaborations, reach out to:

- Shail Garg: bl.en.u4cse22254@bl.students.amrita.edu
- Yerukola Gayatri: bl.en.u4cse22267@bl.students.amrita.edu
- Surya Kausthub A: bl.en.u4cse22287@bl.students.amrita.edu
- Pooja Gowda: g_pooja@blr.amrita.edu