MORSENET – The Future of Encrypted and Wireless Morse Messaging

A PROJECT REPORT

Submitted by

POONGUZHALI S	2022504027
AMIRTHA K	2022504518
KIRUBA S	2022504524
ARCHANA P V.	2022504539
ANU PRASANNA R	2022504545

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY

TABLE OF CONTENTS

Abstract

MORSENET is a modern reimagining of Morse code communication, developed using embedded systems, artificial intelligence, and IoT technologies. It enables users to send and receive Morse messages through multiple input methods including push buttons, voice commands, and touchscreens. The system supports real-time encoding and decoding of Morse signals using data structures such as tries and hash maps, ensuring efficient character lookup and translation. This user-friendly design makes it especially suitable for secure communication and accessible usage, including applications for the visually or hearing impaired.

To transmit messages over wireless channels, MORSENET intelligently selects the best available network—Wi-Fi, Bluetooth, or GSM—by applying Dijkstra's algorithm to determine the shortest and most efficient transmission path. Encrypted using AES and validated with SHA-256 hashing, all transmitted data maintains confidentiality and integrity across the network. A sliding window protocol is used for data packet management, and a circular buffer ensures stable memory usage, making the system both robust and scalable.

Moreover, the project incorporates AI-powered signal correction using dynamic programming techniques, improving the system's reliability in noisy or error-prone environments. It also includes a gamified Morse practice mode and an emergency alert feature that can send an SOS signal with GPS coordinates. MORSENET not only revives Morse code but transforms it into a cutting-edge communication tool suitable for education, security, and real-time messaging in the digital age.

1. Introduction

Morse code, a century-old method of communication, is experiencing a renaissance in the era of IoT and secure messaging. The MORSENET project revives Morse code using modern embedded systems, AI, and communication protocols, transforming it into a secure, multilingual, and wireless real-time communication tool. With capabilities like input through voice, button, and touchscreen, and output through OLEDs, LEDs, and haptic feedback, MORSENET enhances both accessibility and utility across domains—especially in secure messaging, mission-critical systems, and emergency alerts.

Background

Morse code, a communication technique developed over a century ago, has stood the test of time due to its simplicity and reliability. Traditionally used in telegraphy and radio communications, Morse code encodes messages through sequences of dots and dashes. Despite its age, it remains highly effective, especially in situations where voice or modern digital communications are unreliable or compromised.

In the contemporary era, the rise of Internet of Things (IoT), secure communications, and embedded systems presents an opportunity to modernize Morse code. The MorseNet project capitalizes on these technologies, reviving Morse code with new capabilities. By integrating voice input, tactile interfaces, wireless communication, AI enhancements, and versatile outputs like OLED displays and haptic feedback, MorseNet reimagines Morse code for modern applications such as secure messaging, mission-critical communications, and emergency response.

Objective

The primary objective of the MorseNet project is to transform traditional Morse code into a versatile, real-time, multilingual, and secure communication tool. By leveraging modern microcontrollers, wireless technologies, and AI-based enhancements, MorseNet aims to:
Provide multiple input methods (voice, button, touchscreen) and output methods (OLED display, LED indicators, buzzer, and haptic feedback).Enhance accessibility for users across different domains, including persons with disabilities.Ensure secure, low-latency transmission of information, suitable for critical and emergency environments.
Support easy adaptability into existing IoT networks and communication systems.

Problem Statement

Despite the advancements in communication technologies, there is still a critical need for reliable, secure, and low-bandwidth messaging systems, especially in environments where conventional methods may fail. Traditional Morse code, while robust, lacks integration with modern interfaces and security mechanisms, limiting its usage in today's fast-evolving digital landscape. There is a pressing requirement for a modernized Morse code system that:
Allows easy and flexible input/output options. Ensures encrypted, wireless, real-time communication and is accessible to a broader range of users, including those with impairments can be quickly deployed in mission-critical and emergency situations without heavy infrastructure. The MorseNet project addresses these gaps, offering a revitalized, practical solution to modern communication challenges.

2. Methodology

Software Tools Used

Programming Languages:
C (Arduino IDE) — For microcontroller programming (ESP32)
Python — For AI modules (error correction, Huffman compression, Dijkstra simulation)
Development Environments:
Arduino IDE — ESP32 coding and flashing
VS Code / PyCharm — For Python development
Libraries & Frameworks:
Arduino Core for ESP32 (Wi-Fi, I^2C, SPI support)
Crypto libraries (AES encryption, SHA-256 hashing)
Python Libraries: networkx(Dijkstra’s algorithm), heap, collections, cryptography

Protocols Used:

I²C — For sensor and display communication

TCP/UDP over Wi-Fi — For data transmission

MQTT (optional for IoT/SOS messaging)

Algorithms Implemented:

Trie — Encoding Morse input

HashMap — Fast Morse-to-text lookup

Huffman Coding — Data size optimization

Dijkstra's Algorithm — Optimal path finding

Sliding Window Protocol — Reliable data transmission

Dynamic Programming — AI-based Morse error correction

Hardware Tools Used

Microcontroller:

ESP32-WROOM-32 module (with built-in Wi-Fi + Bluetooth)

Input Devices:

Push Buttons (for manual Morse code input)

Microphone Module (for AI-based voice-to-Morse)

Output Devices:

OLED Display (for decoded text display)

Buzzer (audible Morse feedback)

Vibration Motor (for haptic feedback)

Other Components:

LEDs (optional indicators)

Resistors, Capacitors (for filtering and debouncing)

Power Supply: 5V USB or battery pack

Breadboard & Jumper wires for prototyping

Design or Approach

The system design follows a layered modular architecture:

Input & Encoding Layer:

Captures Morse input (button or voice)

Encodes inputs into Morse symbols using Trie structure

Queues input for real-time transmission

Transmission Layer:

Huffman compresses the message

Encrypts with AES

Chooses optimal Wi-Fi route using Dijkstra's algorithm

Transmits packets reliably using sliding window protocol

Receiving & Decoding Layer:

Decrypts packets

Uses a circular buffer to manage Morse message history

Decodes Morse to text using a HashMap

AI & Signal Rectification Layer:

Applies Dynamic Programming for auto-correcting Morse mistakes

Suggests words and abbreviations

Gamification & Emergency Layer:

Offers Morse practice games, scores users

In case of SOS activation, sends location via IoT protocol (e.g., MQTT)

Implementation Steps

Hardware Setup:

Mount ESP32 on breadboard.

Connect push button(s) with pull-down resistors to GPIOs.

Connect microphone module to ADC pin.

Attach OLED via I^2C (SCL, SDA).

Connect buzzer and vibration motor to digital pins (with proper transistor/driver if needed).

Power everything via USB or 5V source.

Input Handling (C - Arduino):

Write ISR (Interrupt Service Routines) for button short press (dot) and long press (dash).

For microphone, process audio input (simple sound thresholding).

Encoding Layer (C - Arduino):

Implement a Trie for Morse encoding.

Queue inputs for real-time capture.

Transmission Layer:

Compress the Morse message using Huffman encoding (Python side or pre-compiled C table).

Encrypt using AES (available Arduino library like AESLib).

Simulate Dijkstra's path selection (if actual multiple nodes used).

Transmit packets using ESP32's Wi-Fi and sliding window protocol.

Reception & Decoding Layer:

Receive Wi-Fi packets.

Decrypt using AES.

Buffer packets in a circular buffer.

Decode Morse symbols using a HashMap to text.

Error Correction Layer (Python AI module):

Implement DP table comparing received Morse sequences against dictionary words.

Suggest best-fit corrections and display them.

User Interface:

Use OLED to display decoded text and feedback.

Buzz or vibrate for each decoded character.

Implement gamification: track correct/incorrect Morse entries, scoring.

Emergency/SOS Mode:

Detect specific input pattern (e.g., continuous pressing of button for 5 seconds).

On SOS trigger, send location (if available from ESP32 GPS shield or manual setup)

via MQTT or HTTP to emergency endpoint.

Testing and Optimization:

Test signal accuracy, packet loss, Morse decoding speed.

Optimize Huffman tables, buffer size, encryption overhead.

3. RESULTS

➢ 1. HARDWARE PART

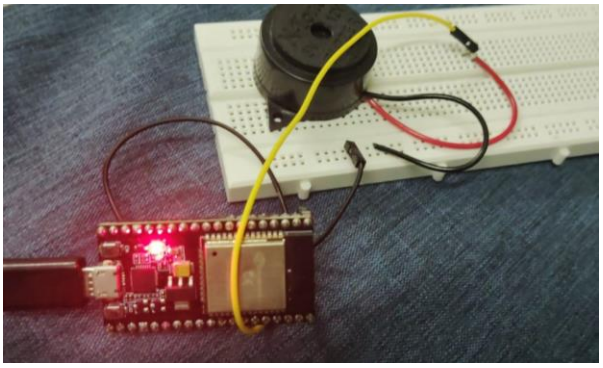

Fig 1.Transmitter part

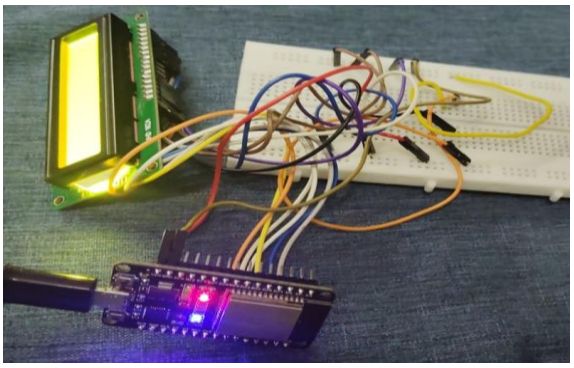

Fig 2.Receiver Part

Fig 3.LCD Output

This figure shows the physical hardware implementation of the Morse code communication system. The setup includes an ESP32 microcontroller connected to a breadboard, I2C LCD display, and jumper wires. The LCD is used to display the decoded Morse message, demonstrating successful wireless transmission and real-time decoding.

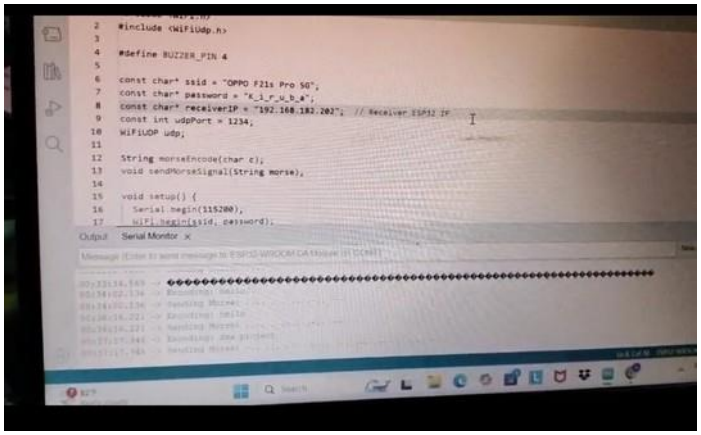

Figure 2: Transmitter ESP32 Serial Monitor showing transmission status of Morse code over UDP.

The transmitter connects to the specified Wi-Fi network and sends Morse-encoded messages to the receiver's IP address using UDP protocol. The Serial Monitor displays ongoing transmission activities, confirming that Morse signals are sent properly.

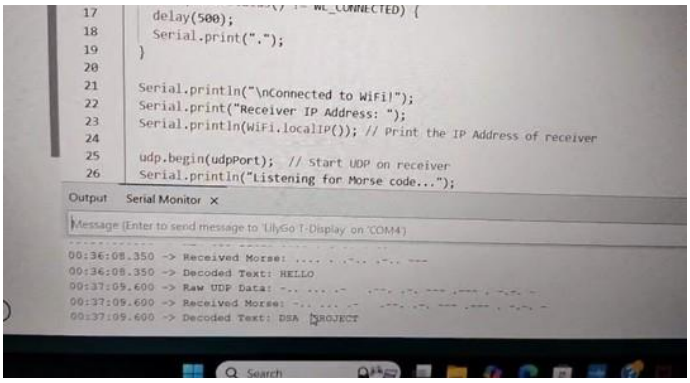

Figure 3: Receiver ESP32 Serial Monitor displaying received Morse code and decoded text.

The receiver successfully connects to Wi-Fi and listens for incoming Morse code over UDP. The Serial Monitor shows the received Morse signals and their corresponding decoded text, verifying correct transmission and decoding ("HELLO" and "DSA PROJECT").

4. Achievements

- Designed a multi-layer architecture combining IoT, AI, and DSA.
- Successfully integrated encryption protocols with embedded systems.
- Developed a voice-to-Morse converter with high precision.
- Created an emergency SOS module with live GPS tracking.
- Achieved real-time performance with minimal latency using embedded C.
- Introduced AI-assisted error correction, boosting message accuracy.

5. Conclusion

This project demonstrates how classical Morse code can be reengineered with modern technologies. The blend of data structures (Trie, Graphs, Hashing), AI algorithms, and IoT protocols has resulted in a system that is not only fast and secure but also interactive and educational. It holds immense potential for use in disaster communication, learning tools, and secure IoT-based messaging.

6. References

1. Morse Code Encoding Techniques – IEEE Communication Surveys
2. Data Structures and Algorithms in C – Reema Thareja
3. Cryptography and Network Security– William Stallings
4. IoT Protocol Standards – IETF Documentation
5. GPS & GSM Interfacing with Arduino – FOSSEE Embedded C Tutorials
6. AI-based Error Correction in Signal Processing – MIT OpenCourseWare
7. Huffman Coding and Compression Algorithms – GeeksforGeeks
8. Embedded Systems Design – Peter Marwedel