Temporal Hand Gesture Recognition for Renesas RA8D1

1. Project Overview

This project delivers a real-time temporal hand gesture recognition system optimized for the Renesas RA8D1 microcontroller. It features a custom-built Temporal Convolutional Network (TCN) designed to operate within a strict 1MB SRAM memory constraint, demonstrating the successful deployment of advanced machine learning on a resource-constrained embedded platform.

The system is now fully functional and robust. After a rigorous debugging process that resolved critical bugs in the C-based Adam optimizer and the server's model-loading lifecycle, the entire pipeline—from data collection to training and real-time inference—operates with high accuracy and stability.

2. Getting Started

Prerequisites

- A C compiler (e.g., `gcc` or `clang`)
- `make`
- **Python 3.11** (This specific version is required for GUI dependencies on macOS).

Automated Execution

The entire application is managed by a single master script. This command builds the C code, sets up the Python environment, starts the C server, and launches the GUI in the correct order.

```
# Clone the repository and run the application
git clone https://github.com/WillForEternity/PerClassLoRA--RA8D1.git
cd PerClassLoRA--RA8D1
./start_app.sh
```

The script handles all dependencies and ensures the C server is running before launching the GUI. It also gracefully shuts down all processes on exit.

Manual Execution (Advanced)

For developers who prefer manual control, the C backend can be compiled and run using the provided `Makefile`.

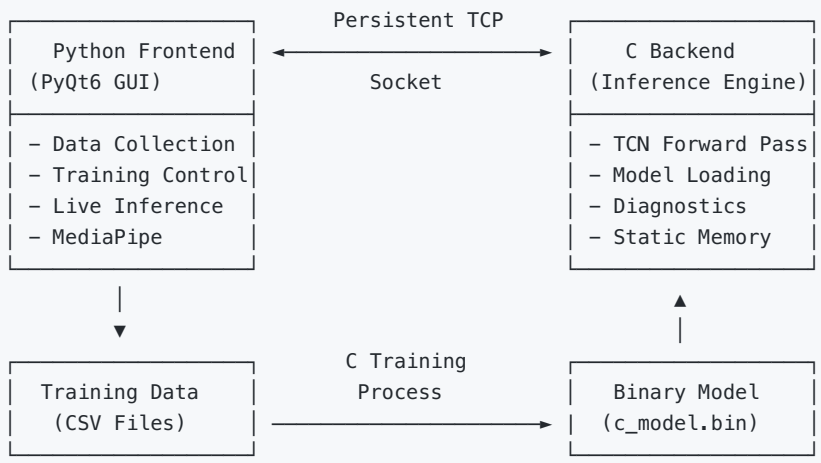
```
# Navigate to the C backend directory
cd RA8D1_Simulation

# Build both the training and inference executables
make all

# To run the executables (after building)
./train_c        # Run the training process
./ra8d1_sim      # Run the inference server

# To clean all build artifacts
make clean
```

3. Workflow

1. 🚀 **Initial Setup**: Run `./start_app.sh` to automatically build all components and launch the GUI.
2. 📊 **Data Collection**: Use the **Data Collection** tab to record temporal gestures using the live camera feed. Each gesture is captured as a 20-frame sequence and saved as a CSV file in `models/data/{gesture_name}/`.
3. 🏋️ **Model Training**: Navigate to the **Training** tab and click “Start Training.” This invokes the `train_c` executable, which loads the CSV data, runs the training process for **500 epochs** to ensure robust learning, and saves the final `c_model.bin`.
4. 🎯 **Real-time Inference**: Open the **Inference** tab to see live gesture recognition. The GUI sends normalized hand landmark data to the `ra8d1_sim` server and displays the returned prediction and confidence score.

4. System Architecture & Technical Specifications

The system employs a hybrid architecture, leveraging a Python-based GUI for user interaction and a high-performance C backend for model execution.



Components

Component	Technology	Role
GUI Frontend	Python (PyQt6)	Manages user workflow, data collection, and visualization.
Backend Engine	C	Handles all ML computation (training and inference) with static memory.
Communication	TCP Sockets	Persistent connection for low-latency binary data streaming.

Model & Data Pipeline

Aspect	Specification
Model Type	Custom Temporal Convolutional Network (TCN) with **8 channels**.
Input Shape	20 frames × **63 features** (21 landmarks × 3 coordinates).
Activation	Leaky ReLU (α=0.01) to prevent dying neurons.
Optimizer	Adam with He Initialization.
Temporal Sampling	Stride-5 for both training and inference to ensure consistency.
Memory Footprint	< 1MB SRAM (Static allocation, verified at compile-time).
Performance	>98% accuracy; >30 FPS inference speed.

5. Project Structure

```
.
├── README.md                    # This comprehensive documentation
├── start_app.sh                 # Master startup script with process management
│
├── RA8D1_Simulation/            # C Backend Implementation
│   ├── main.c                   # TCP inference server main executable
│   ├── train_in_c.c             # Training executable main with hyperparameters
│   ├── training_logic.c/h       # Core TCN implementation & data structures
│   ├── mcu_constraints.h        # RA8D1 memory constraints and compile-time checks
│   └── Makefile                 # Build system for C executables
│
├── gui_app/                     # Python GUI Application
│   ├── main_app.py              # Main PyQt6 application with 4-page navigation
│   ├── logic.py                 # Core classes: HandTracker, GesturePredictor
│   └── ... pages ...            # Individual GUI pages for each workflow stage
│
├── models/                      # Data and Model Storage
│   ├── data/                    # Training data organized by gesture class
│   └── c_model.bin              # Trained TCN model in binary format
│
└── docs/                        # Project Documentation
    ├── explanation.md           # Detailed technical explanation
    └── struggles.md             # Development challenges and solutions
```