**Project: TinyML-Based Gesture Recognition on ESP32-CAM**

**Technologies:** PyTorch, Flask, OpenCV, NumPy, scikit-learn, ESP32-CAM, PlatformIO, TinyML, HTML/CSS/JS

**Summary:**  
Designed and deployed a complete edge AI pipeline to recognize specific hand gestures using ESP32-CAM and a custom CNN model trained on self-collected data.

**Overview:**

1. I used **PlatformIO** to set up the development environment for my **ESP32-CAM**, importing essential libraries like WiFi.h, WebServer.h, and camera\_pins.h.
2. I defined the LED pin and WiFi credentials, and created a global variable fb to store captured photos.
3. I built a basic HTML UI via the constructHtml() function, which includes a preview image and a "Take Photo" button. The logic is separated into modular functions:

* capturePhoto(): triggers the camera,
* savePhoto(): sends photo data to a Flask backend,
* submitPhoto() and releasePhoto() handle memory and release.

1. After setting up those functions, I configured the camera, set baud rate, initialized the WiFi and WebServer, and prepared the LED output.

* The loop() function checks WiFi status and continuously handles web requests.

1. On the backend, I built a **Flask** server using Python (my first time using it) to receive photo data sent from ESP32-CAM over LAN and save them as .jpg. Initially, I misunderstood the board was sending raw byte data, not file objects. I overcame networking challenges through trial and error and learned basic socket and HTTP transfer mechanisms.
2. After adding reconnection logic, I successfully collected ~1400 labeled images (772 vs. 601). I wrote a data processing script that used cv2 to:

* grayscale,
* normalize,
* resize, and
* reshape each image into a consistent (120,160,1) shape,  
  then stored them in a NumPy array.

1. I split the dataset using train\_test\_split (80:20) and passed the data into my CNN model for training.
2. I designed a CNN using **PyTorch**, based on torch.nn.Module.

* It consists of 4 convolutional layers (Conv2d) with increasing channels (32→64→128→256),
* ReLU activation after each,
* MaxPooling2d to downsample,
* Dropout to prevent overfitting,
* then Flatten and two Linear layers for final classification.  
  The final output layer has 2 neurons (binary classification).

1. After training, I monitored validation accuracy, saved only the best-performing model, and discarded others. The best model achieved over 90 % accuracy. I then converted .pth → .onnx → .tflite and used TinyML tools to convert the model into C++ arrays for deployment.
2. Finally, I integrated the model with ESP32-CAM using Arduino-C++ to perform on-device hand gesture recognition, lighting up the flash if the specific victory hand gesture is detected. This completed the closed loop from data collection → training → deployment on edge devices.