Minor in AI

TinyML

Anomaly Detection for IoT Networks

June 16, 2025

1 Introduction

Anomaly detection is an increasingly vital tool in modern IoT systems, enabling devices to autonomously recognize unusual or potentially dangerous behavior. As IoT networks grow more complex, the ability to detect deviations from normal behavior becomes essential for ensuring safety, reliability, and efficient operation.

In this notes, we explored the implementation of an end-to-end anomaly detection system using TinyML. Specifically, a temperature and humidity monitoring setup was used to collect environmental data, develop and train a lightweight model, and deploy the trained model on an ESP32 microcontroller for real-time inference and alerting.

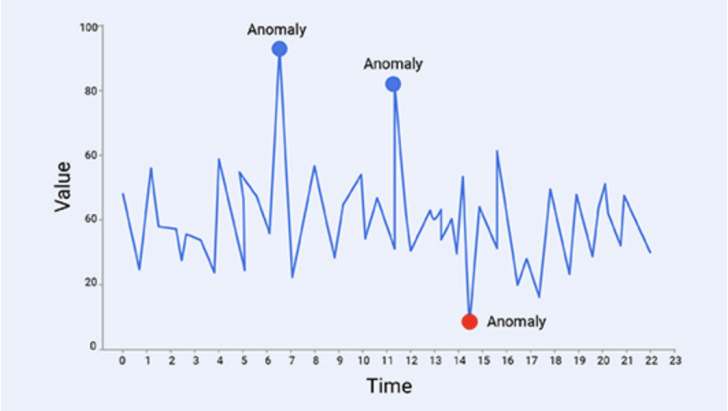


Figure 1: Anomaly Detection

2 Types of Anomalies

Understanding the various types of anomalies is crucial for designing robust models that can distinguish normal variations from true faults. The three principal types are:

1. **Point Anomalies:** These refer to individual data points that differ significantly from the majority. For example, a single sensor reading of 35°C when the expected range is 24–26°C.

2. **Contextual Anomalies:** These occur when a data point is only anomalous within a specific context. For instance, 30°C may be acceptable during the day but considered unusual at night.

3. **Collective Anomalies:** These involve a series of data points that, taken together, indicate abnormal behavior—such as a gradual temperature rise over time even if each individual reading is within normal bounds.

By recognizing these categories, we can better structure the training process and evaluation metrics of our anomaly detection models.

3 Methodology and Implementation

3.1 Data Collection and Preprocessing

The dataset was gathered using environmental sensors capable of measuring temperature and humidity. Under typical conditions, the values remained close to 24°C and 40% humidity. The experiment involved both controlled normal states and induced abnormalities to ensure the model could generalize to unseen conditions.

To support unsupervised learning, K-means clustering was applied. This allowed natural grouping of data without manual labels, effectively identifying clusters of normal vs. abnormal behavior. These labeled clusters were then used to train a supervised classifier.

3.2 Model Training and Optimization

A compact neural network was designed to classify inputs as normal or anomalous. The model architecture was deliberately kept shallow to fit within the memory and compute constraints of the ESP32. The network was trained using standard classification objectives and validated to ensure reliable anomaly detection.

Following training, the model was quantized—converting 32-bit floating point weights to 8-bit integers—to significantly reduce memory usage and increase inference speed, a critical step for deployment on microcontrollers.

3.3 Deployment on ESP32

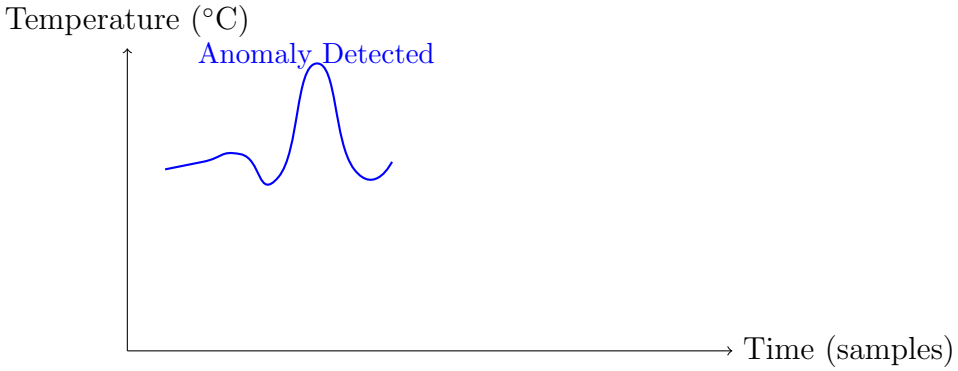
The trained and quantized model was converted to TensorFlow Lite format, and then embedded into an ESP32 board. The Arduino IDE served as the programming interface, where the TFLite model was linked as a C header file. The ESP32 continuously monitored sensor inputs and executed real-time inference to flag potential anomalies.

Upon detecting an anomaly, the microcontroller issued alerts—either through LED signals, buzzer sounds, or UART messages to a connected computer. This simulated a simple yet effective edge-based alerting system for real-world deployments.

Threshold Guidelines

Normal Range: Temperature = 23–25°C, Humidity = 35%–45%
Anomaly Trigger: Temperature ¡20°C or ¿28°C; Humidity ¡30% or ¿50%

4 Visualization: Sensor Anomaly Event



5 Hardware Integration and System Considerations

Deploying a neural network on a microcontroller involves multiple constraints:

- **RAM limitations:** ESP32 has limited memory; hence quantization and model pruning are crucial.
- **Real-time needs:** Models must perform inference quickly enough to detect anomalies without delays.
- **Sensor accuracy:** Sensor drift and noise can affect detection accuracy, so preprocessing may be necessary.
- **Power management:** For battery-operated systems, power-efficient inference is a key design requirement.

The successful deployment in this case demonstrates that with careful optimization, complex anomaly detection pipelines can be compressed and deployed directly at the edge.

Key Takeaways

1. Anomaly detection enhances the robustness of IoT systems by allowing proactive responses to abnormal conditions.
2. Integrating K-means with neural networks provides a hybrid strategy for labeling and classification in resource-limited settings.
3. Model quantization and memory-efficient architectures are essential for microcontroller compatibility.
4. ESP32 serves as a flexible and low-cost platform for real-time ML inference in embedded environments.
5. Practical deployments must account for sensor reliability, latency, and system robustness under different environmental conditions.