Distributed AI Fire Detection System

A Spatial-Temporal Edge Computing Approach

Thierry Isaac N'kouka

PhD Candidate, Computer Networks & Distributed AI

March 2026

Abstract

This paper presents the design, implementation, and mathematical foundations of a distributed artificial intelligence system for real-time fire detection and propagation prediction. The system deploys five Raspberry Pi 3B+ nodes as edge computing units, each equipped with a BME680 multi-parameter environmental sensor. Sensor data streams are windowed, preprocessed, and fed into a two-stage AI ensemble: a Long Short-Term Memory network (LSTM) for fire phase classification, and a Multi-Layer Perceptron (MLP) for spatial fire propagation detection. Inference is executed in a synchronized manner: all five nodes are processed at the same temporal index t, enabling spatially aware fire spread detection that would be impossible with independent per-node inference. The controller orchestrates distributed LSTM execution via SSH and SFTP, achieving end-to-end latency of 7–10 seconds per node per sequence, well within the 60-second window production rate.

Contents

1 Introduction

Early fire detection is critical for minimizing structural damage and protecting lives. Traditional smoke detectors provide binary alerts with no spatial context. Modern IoT deployments can do far better: a network of environmental sensors distributed across a physical space can detect fire phases before visible flame appears and, critically, predict whether fire is likely to spread from one zone to another.

The key research contribution of this system is **joint spatial-temporal inference**: rather than processing each sensor node independently, the controller waits until all $N = 5$ nodes have completed preprocessing for time index t, then triggers inference simultaneously. This allows Model B (the propagation detector) to receive the full spatial state vector of the environment, enabling cross-node fire spread prediction.

The remainder of the paper is organized as follows. Section 2 describes the hardware and software architecture. Section 3 formalizes the data acquisition and windowing procedure. Section 5 details the preprocessing pipeline. Sections 6 and 7 present the LSTM and MLP models, respectively. Section 8 formalizes the joint inference mechanism. Section 9 describes the distributed orchestration protocol. Section 10 covers output structure and logging, and Section 12 analyzes system performance.

2 System Architecture

2.1 Hardware Layer

The system consists of five Raspberry Pi 3B+ units (referred to as `rpi3-a` through `rpi3-e`) connected via Ethernet to a central switch on subnet `10.42.0.0/24`. Table 1 summarizes the network configuration.

Table 1: Node Network Configuration

Node	IP Address	Role
rpi3-a	10.42.0.51	Edge sensor + inference worker
rpi3-b	10.42.0.75	Edge sensor + inference worker
rpi3-c	10.42.0.80	Edge sensor + inference worker
rpi3-d	10.42.0.118	Edge sensor + inference worker
rpi3-e	10.42.0.163	Edge sensor + inference worker
Controller	10.42.0.x	SSH orchestrator (office workstation)

Each Pi is equipped with a **BME680** sensor, which provides four simultaneous physical measurements:

- Temperature T (°C)
- Relative humidity H (%)
- Gas resistance G (Ω), a proxy for volatile organic compounds and combustion products
- Atmospheric pressure P (hPa)

2.2 Software Stack

Table 2 summarizes the software components at each layer of the system.

Table 2: Software Component Stack

Layer	Component	Technology
Sensor acquisition	`pi_sensor_publisher.py`	Python 3.13, smbus2
Transport	MQTT pub/sub	Paho MQTT
Shared storage	NAS-mounted NFS	`/mnt/nas/`
Preprocessing	`preprocessor_priority.py`	NumPy, scikit-learn
Orchestration	`data_controller.py`	Paramiko SSH
Edge controller	`edge_controller.py`	Python 3.13
Inference	`infer.py`	NumPy (custom forward)
Distributed runtime	`orchestrator.py`	Paramiko, SFTP
Worker	`worker.py` on each Pi	NumPy

3 Data Acquisition and Windowing

3.1 Sensor Sampling

Each Pi polls the BME680 at sampling frequency $f_s = 1\,\text{Hz}$. The raw observation at time step k on node n is the vector:

$$\mathbf{x}_k^{(n)} = \left[T_k^{(n)},\ H_k^{(n)},\ G_k^{(n)},\ P_k^{(n)}\right]^\top \in \mathbb{R}^4 \tag{1}$$

3.2 Sliding Window Construction

Every $W = 60$ seconds, a non-overlapping window is finalized and written to the NAS as a pickle file. The window matrix for node n at sequence index t is:

$$\mathbf{X}_t^{(n)} = \left[\mathbf{x}_{tW}^{(n)},\ \mathbf{x}_{tW+1}^{(n)},\ \ldots,\ \mathbf{x}_{tW+W-1}^{(n)}\right] \in \mathbb{R}^{W\times 4} \tag{2}$$

The file is saved at path:

```
iot_windowed_data/rpi3-{n}_fire_sensors/
    {timestamp}_seq{t:08d}.pkl
```

Windows are non-overlapping, giving a temporal resolution of one minute per inference cycle. Upon writing, `pi_sensor_publisher.py` publishes an MQTT message to topic `fire_sensors/rpi3-{n}/window_ready`, triggering the data controller.

4 Preprocessing Pipeline

4.1 Overview

When `data_controller.py` receives the MQTT notification for window t on node n, it spawns `preprocessor_priority.py` as a remote SSH process on that Pi. This script

performs three transformations: statistical feature engineering, standardization for LSTM input, and LSTM feature selection.

4.2 Statistical Feature Engineering

From the raw window matrix $\mathbf{X}_t^{(n)} \in \mathbb{R}^{60\times 4}$, a set of statistical descriptors is computed per sensor channel $c \in \{T, H, G, P\}$:

$$\mu_c = \frac{1}{W}\sum_{k=0}^{W-1} x_{k,c} \tag{3}$$

$$\sigma_c = \sqrt{\frac{1}{W}\sum_{k=0}^{W-1}(x_{k,c} - \mu_c)^2} \tag{4}$$

$$x_c^{\min} = \min_{k\in[0,W)} x_{k,c}, \quad x_c^{\max} = \max_{k\in[0,W)} x_{k,c} \tag{5}$$

$$\Delta_c = x_{W-1,c} - x_{0,c} \tag{6}$$

This yields a base feature vector of dimension $4 \times 5 = 20$. Cross-channel derived features are appended (e.g., temperature–humidity product, gas rate of change) to reach the final feature vector:

$$\mathbf{f}_t^{(n)} \in \mathbb{R}^{30} \tag{7}$$

stored in `features.json`.

5 Detailed Preprocessing Pipeline and Feature Engineering

5.1 Overview

Raw sensor data collected from the BME680 cannot be fed directly into the AI models for two fundamental reasons. First, the four sensor channels operate on drastically different numerical scales: temperature values are in the range $[20, 85]$ C, while gas resistance values are in the range $[1{,}000, 500{,}000]\,\Omega$. A neural network receiving these raw values would incorrectly weight gas resistance as far more important simply because its numbers are larger. Second, the raw time series does not explicitly encode patterns such as rate of change, cross-channel interactions, or statistical summaries that are known to be discriminative for fire detection. The preprocessing pipeline solves both problems by performing three sequential transformations:

1. **Statistical feature engineering** — Collapsing the 60-second window into a compact descriptor vector
2. **StandardScaler normalization** — Placing all channels on a uniform numerical scale
3. **Per-timestep enrichment** — Augmenting the normalized time series with derived instantaneous features for LSTM input

All transformations are executed by `proprocessor_priority.py`, spawned remotely via SSH on the originating Raspberry Pi, and produce three output files: `features.json`, `transformed.pkl`, and `block5_input.json`.

5.2 Raw Input Data

The raw input to the preprocessor is the windowed matrix collected by `pi_sensor_publisher.py`:

$$\mathbf{X}_t^{(n)} = \begin{bmatrix} T_0 & H_0 & G_0 & P_0 \\ T_1 & H_1 & G_1 & P_1 \\ \vdots & \vdots & \vdots & \vdots \\ T_{59} & H_{59} & G_{59} & P_{59} \end{bmatrix} \in \mathbb{R}^{60\times 4} \tag{8}$$

where each row $k \in \{0, \ldots, 59\}$ is a one-second snapshot from the BME680 sensor at node n and sequence index t:

- T_k — Temperature (C): ambient air temperature
- H_k — Relative Humidity (%): moisture content of the air
- G_k — Gas Resistance (Ω): proxy for VOC and combustion gas concentration; *decreases* in the presence of smoke or fire
- P_k — Atmospheric Pressure (hPa): barometric pressure; fire consumes oxygen locally, creating subtle pressure gradients

5.3 Block 1: Statistical Feature Engineering

For each channel $c \in \{T, H, G, P\}$, five statistical descriptors are computed over the full 60-second window.

5.3.1 Mean

The mean captures the *typical level* of each channel during the window:

$$\mu_c = \frac{1}{W} \sum_{k=0}^{W-1} x_{k,c}, \qquad W = 60 \tag{9}$$

Physical interpretation: A persistently elevated μ_T relative to baseline indicates sustained heating; a persistently depressed μ_G indicates sustained presence of combustion gases.

5.3.2 Standard Deviation

The standard deviation captures *temporal volatility* within the window:

$$\sigma_c = \sqrt{\frac{1}{W} \sum_{k=0}^{W-1} (x_{k,c} - \mu_c)^2} \tag{10}$$

Physical interpretation: During normal conditions, σ_G is low because air quality is stable. During a smoldering or growing fire, σ_G increases sharply as combustion gases

are released in irregular bursts. A high σ_T indicates rapid thermal fluctuations consistent with flame activity.

5.3.3 Minimum and Maximum

$$x_c^{\min} = \min_{k \in [0,W)} x_{k,c}, \qquad x_c^{\max} = \max_{k \in [0,W)} x_{k,c} \tag{11}$$

Physical interpretation: The quantity $x_c^{\max} - x_c^{\min}$ (the *range*) is a key indicator of how dramatically a channel changed within the window. A large $G^{\max} - G^{\min}$ — for example, gas resistance dropping from 125,000 Ω to 85,000 Ω in one minute — indicates a significant air quality event. The minimum $G^{\min}$ is particularly important: it captures the *worst* air quality moment, which may occur only briefly but is highly indicative of fire.

5.3.4 Delta (Trend Direction)

$$\Delta_c = x_{W-1,\,c} - x_{0,\,c} \tag{12}$$

Physical interpretation: Δ_c encodes the *direction* of change over the entire window. A positive Δ_T means temperature is rising; a negative Δ_G means gas resistance is falling (air quality worsening). The sign and magnitude of Δ_G is one of the most discriminative features for distinguishing `smoldering` from `no_fire`.

5.3.5 Base Feature Vector

Applying the five statistics to all four channels yields the base feature vector:

$$\mathbf{f}_{\text{base}} = \begin{bmatrix} \mu_T,\ \sigma_T,\ T^{\min},\ T^{\max},\ \Delta_T, \\ \mu_H,\ \sigma_H,\ H^{\min},\ H^{\max},\ \Delta_H, \\ \mu_G,\ \sigma_G,\ G^{\min},\ G^{\max},\ \Delta_G, \\ \mu_P,\ \sigma_P,\ P^{\min},\ P^{\max},\ \Delta_P \end{bmatrix} \in \mathbb{R}^{20} \tag{13}$$

5.4 Block 2: Cross-Channel Derived Features

Single-channel statistics cannot capture *interactions* between channels. Fire produces a simultaneous signature across multiple channels: temperature rises while humidity falls, gas resistance drops, and pressure may shift slightly. Ten cross-channel features are computed to explicitly encode these joint patterns.

$$f_1 = \mu_T \cdot \mu_H \tag{14}$$

Interpretation: Captures the joint thermal-moisture state. Under normal conditions, temperature and humidity are inversely correlated. A high f_1 value indicates anomalously high temperature coinciding with still-high humidity (early smoldering phase). A low f_1 with high μ_T indicates fire has already dried the air.

$$f_2 = \Delta_T \cdot \Delta_G \tag{15}$$

Interpretation: The product of the temperature trend and the gas trend. During fire: $\Delta_T > 0$ (heating) and $\Delta_G < 0$ (gas worsening), so $f_2 < 0$. Under normal conditions, both trends are near zero, so $f_2 \approx 0$. A strongly negative f_2 is a reliable fire indicator.

$$f_3 = \frac{\sigma_G}{\mu_G + \varepsilon}, \qquad \varepsilon = 10^{-6} \tag{16}$$

Interpretation: Normalised gas volatility — how large the fluctuations are relative to the mean level. A sensor in clean air has low f_3; a sensor near intermittent smoke bursts has high f_3.

$$f_4 = T^{\max} - T^{\min} \tag{17}$$

Interpretation: The temperature range. A wide range within one minute indicates rapid thermal dynamics consistent with growing fire or flashover.

$$f_5 = \frac{G^{\min}}{G^{\max} + \varepsilon} \tag{18}$$

Interpretation: The gas compression ratio. A value near 1 means gas resistance was stable throughout the window (clean air). A value near 0 means gas resistance dropped to near-zero at its worst point — a severe smoke event.

$$f_6 = \mu_T^2 \tag{19}$$

Interpretation: Squaring amplifies high temperatures non-linearly. At 30C, $f_6 = 900$; at 60C, $f_6 = 3{,}600$ — a 4 increase for a 2 temperature increase. This helps the MLP react sharply to extreme heat.

$$f_7 = \mu_G^2 \tag{20}$$

Interpretation: Squaring the mean gas resistance amplifies the difference between clean air (high μ_G, very high f_7) and smoky air (low μ_G, very low f_7), making the contrast easier for the MLP to separate.

$$f_8 = \frac{\Delta_T}{T^{\max} - T^{\min} + \varepsilon} \tag{21}$$

Interpretation: Normalised temperature slope. A value near +1 means temperature rose monotonically throughout the window (sustained heating). A value near 0 means it fluctuated without a clear trend. A monotonic rise is more characteristic of growing fire than random spikes.

$$f_9 = \sigma_T \cdot \sigma_G \tag{22}$$

Interpretation: The volatility product. When both temperature *and* gas resistance are simultaneously unstable, this product is large. This distinguishes fire (both sensors fluctuate) from, for example, cooking (temperature rises but gas resistance may remain stable).

$$f_{10} = \mu_P \cdot \Delta_T \tag{23}$$

Interpretation: Pressure level combined with temperature trend. A confined room fire creates a rising temperature trend at near-normal or slightly reduced pressure; an outdoor fire may show different pressure behaviour. This feature allows the model to distinguish indoor from outdoor fire scenarios.

5.4.1 Final Statistical Feature Vector

$$\mathbf{f}_t^{(n)} = \left[\mathbf{f}_{\text{base}},\ f_1,\ f_2,\ \ldots,\ f_{10}\right] \in \mathbb{R}^{30} \tag{24}$$

This 30-dimensional vector is saved as `features.json` and constitutes the primary input to **Model B (MLP propagation detector)**.

5.5 Block 3: StandardScaler Normalization

Before the raw time series can be fed to the LSTM, all channels must be placed on a uniform numerical scale. This is achieved using a `StandardScaler` fitted on the training dataset and serialized to `scaler_lstm.pkl`.

For each channel c and timestep k:

$$\tilde{x}_{k,c} = \frac{x_{k,c} - \bar{\mu}_c}{\bar{\sigma}_c} \tag{25}$$

where $\bar{\mu}_c$ and $\bar{\sigma}_c$ are the *training-set* mean and standard deviation for channel c — fixed constants loaded from the scaler file, not recomputed on each window.

Concrete example: Suppose the training set established $\bar{\mu}_T = 22\,C$ and $\bar{\sigma}_T = 3\,C$. A current reading of $T_k = 28\,C$ is transformed to:

$$\tilde{T}_k = \frac{28 - 22}{3} = +2.0 \tag{26}$$

This value of +2.0 means "this temperature is 2 standard deviations above the training average" — a clear anomaly signal. Simultaneously, if gas resistance has $\bar{\mu}_G = 120{,}000\,\Omega$ and $\bar{\sigma}_G = 15{,}000\,\Omega$, then a reading of $G_k = 85{,}000\,\Omega$ becomes:

$$\tilde{G}_k = \frac{85{,}000 - 120{,}000}{15{,}000} = -2.33 \tag{27}$$

Both $\tilde{T}_k = +2.0$ and $\tilde{G}_k = -2.33$ are now on the same numerical scale, allowing the LSTM to weigh them fairly.

After normalization, the matrix becomes:

$$\tilde{\mathbf{X}}_t^{(n)} \in \mathbb{R}^{60\times 4} \tag{28}$$

5.6 Block 4: Per-Timestep Feature Enrichment

The normalized 4-channel time series is further enriched at each timestep k by appending 6 derived features, giving the LSTM explicit information about instantaneous dynamics.

5.6.1 First Differences (Velocity)

$$\Delta\tilde{T}_k = \tilde{T}_k - \tilde{T}_{k-1} \tag{29}$$

$$\Delta\tilde{G}_k = \tilde{G}_k - \tilde{G}_{k-1} \tag{30}$$

with $\Delta\tilde{T}_0 = \Delta\tilde{G}_0 = 0$.

Interpretation: These are the *velocities* of temperature and gas resistance — how much they changed in the last one second. A sudden large negative $\Delta\tilde{G}_k$ at second $k = 35$ (for example) tells the LSTM precisely *when* the air quality deteriorated, not just that it deteriorated overall. This temporal precision is crucial for the LSTM to correctly classify the `growing` vs `smoldering` phase boundary.

5.6.2 Cross-Channel Interaction at Timestep k

$$\phi_k^{TH} = \tilde{T}_k \cdot \tilde{H}_k \tag{31}$$

Interpretation: The instantaneous product of normalized temperature and humidity. In normal conditions both are near 0 (normalized), so $\phi_k^{TH} \approx 0$. As fire develops, $\tilde{T}_k$ rises (becomes positive) and $\tilde{H}_k$ falls (becomes negative), making ϕ_k^{TH} strongly negative — a real-time fire signature at each individual second.

5.6.3 Squared Gas Resistance

$$\tilde{G}_k^2 \tag{32}$$

Interpretation: Squaring the normalized gas resistance amplifies extreme deviations. If $\tilde{G}_k = -2.0$ (already anomalous), then $\tilde{G}_k^2 = 4.0$, exaggerating the signal further. This helps the LSTM activate strongly on severe air quality events.

5.6.4 Rate of Change of Gas (Acceleration)

$$\text{RoC}(G)_k = \frac{\tilde{G}_k - \tilde{G}_{k-2}}{2}, \quad k \geq 2; \quad \text{RoC}(G)_k = 0 \text{ for } k < 2 \tag{33}$$

Interpretation: This is the *acceleration* of gas resistance change — a second-order temporal derivative. A fire in the growing phase causes gas resistance to drop not just consistently, but *increasingly fast* (accelerating drop). $\text{RoC}(G)_k$ captures this acceleration pattern, which is invisible to first-difference features alone.

5.6.5 Squared Pressure

$$\tilde{P}_k^2 \tag{34}$$

Interpretation: Similar to the gas squared term — amplifies extreme pressure deviations. Pressure changes during fire are subtle (small $|\tilde{P}_k|$), so squaring helps the LSTM detect them more clearly by boosting the signal non-linearly.

5.6.6 Final Per-Timestep Feature Vector

At each timestep k, the full enriched feature vector is:

$$\mathbf{z}_k = \big[\underbrace{\tilde{T}_k,\ \tilde{H}_k,\ \tilde{G}_k,\ \tilde{P}_k}_{\text{4 normalized channels}},\ \underbrace{\Delta\tilde{T}_k,\ \Delta\tilde{G}_k}_{\text{velocities}},\ \underbrace{\tilde{T}_k \cdot \tilde{H}_k}_{\text{interaction}},\ \underbrace{\tilde{G}_k^2}_{\text{gas amp.}},\ \underbrace{\text{RoC}(G)_k}_{\text{acceleration}},\ \underbrace{\tilde{P}_k^2}_{\text{pressure amp.}}\ \big]^\top \in \mathbb{R}^{10} \tag{35}$$

The complete LSTM input tensor stacks all 60 such vectors:

$$\mathbf{Z}_t^{(n)} = \big[\mathbf{z}_0,\ \mathbf{z}_1,\ \ldots,\ \mathbf{z}_{59}\big] \in \mathbb{R}^{1\times 60\times 10} \tag{36}$$

saved as `transformed.pkl` and fed directly to **Model A (LSTM fire phase classifier)**.

5.7 Complete Preprocessing Output Summary

From a single raw window $\mathbf{X}_t^{(n)} \in \mathbb{R}^{60\times4}$ (240 raw numbers), preprocessing produces three structured outputs:

Table 3: Preprocessing Output Files and Their Consumers

File	Contents	Shape	Used by
`features.json`	20 statistical + 10 cross-channel features	$\mathbb{R}^{30}$	Model B (MLP)
`transformed.pkl`	Normalized + enriched time series	$\mathbb{R}^{1\times60\times10}$	Model A (LSTM)
`block5_input.json`	LSTM block-5 intermediate activations	$\mathbb{R}^{128}$	Model B (MLP)

The total number of engineered values produced per window per node is therefore:

$$30\ \text{(statistical)}+60\times10\ \text{(LSTM tensor)}+128\ \text{(block-5 activations)} = 758\ \text{engineered values from}\ 240\ \text{raw} \tag{37}$$

This represents a 3.2× expansion in information density, providing the AI models with a far richer representation of each 60-second environmental snapshot than the raw sensor readings alone could offer.

5.8 Standardization for LSTM Input

The raw time series $\mathbf{X}_t^{(n)}$ is standardized using per-feature statistics computed on the training set:

$$\tilde{x}_{k,c} = \frac{x_{k,c} - \bar{\mu}_c}{\bar{\sigma}_c} \tag{38}$$

where $\bar{\mu}_c$ and $\bar{\sigma}_c$ are the training-set mean and standard deviation for channel c. This produces the normalized matrix:

$$\tilde{\mathbf{X}}_t^{(n)} \in \mathbb{R}^{60\times4} \tag{39}$$

saved as `transformed.pkl`.

5.9 LSTM Feature Tensor Construction

The LSTM model uses $d_{\text{lstm}} = 10$ input features per timestep, formed by augmenting the 4 normalized channels with 6 derived features:

$$\mathbf{z}_k = \left[\tilde{T}_k,\ \tilde{H}_k,\ \tilde{G}_k,\ \tilde{P}_k,\ \Delta\tilde{T}_k,\ \Delta\tilde{G}_k,\ \tilde{T}_k\cdot\tilde{H}_k,\ \tilde{G}_k^2,\ \text{RoC}(G)_k,\ \tilde{P}_k^2\right]^\top \in \mathbb{R}^{10} \tag{40}$$

where $\Delta\tilde{x}_k = \tilde{x}_k - \tilde{x}_{k-1}$ is the first difference (with $\Delta\tilde{x}_0 = 0$), and $\text{RoC}(G)_k$ is the rate of change of gas resistance estimated by finite differences. The final LSTM input tensor is:

$$\mathbf{Z}_t^{(n)} = \left[\mathbf{z}_0,\ \mathbf{z}_1,\ \ldots,\ \mathbf{z}_{59}\right] \in \mathbb{R}^{1\times60\times10} \tag{41}$$

6 Model A: LSTM Fire Phase Classifier

6.1 Architecture

Model A is a stacked LSTM network. Table 4 summarizes its configuration.

Table 4: LSTM Model A Configuration

Parameter	Value
Input shape	(1, 60, 10)
Hidden dimension	$d_h = 128$
Number of layers	$L = 2$
Output classes	4
Model file	`lstm_params.pkl` ($\approx 154\,\text{KB}$)
Config file	`lstm_config.json`

6.2 LSTM Cell Equations

For each layer $l \in \{1, 2\}$ and timestep $k \in \{0, \ldots, 59\}$, the LSTM computes the following gated operations:

Forget gate

$$\mathbf{f}_k^{(l)} = \sigma\Big(\mathbf{W}_f^{(l)}\mathbf{z}_k^{(l)} + \mathbf{U}_f^{(l)}\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_f^{(l)}\Big) \tag{42}$$

Input gate

$$\mathbf{i}_k^{(l)} = \sigma\Big(\mathbf{W}_i^{(l)}\mathbf{z}_k^{(l)} + \mathbf{U}_i^{(l)}\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_i^{(l)}\Big) \tag{43}$$

Candidate cell state

$$\tilde{\mathbf{c}}_k^{(l)} = \tanh\Big(\mathbf{W}_c^{(l)}\mathbf{z}_k^{(l)} + \mathbf{U}_c^{(l)}\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_c^{(l)}\Big) \tag{44}$$

Cell state update

$$\mathbf{c}_k^{(l)} = \mathbf{f}_k^{(l)} \odot \mathbf{c}_{k-1}^{(l)} + \mathbf{i}_k^{(l)} \odot \tilde{\mathbf{c}}_k^{(l)} \tag{45}$$

Output gate

$$\mathbf{o}_k^{(l)} = \sigma\Big(\mathbf{W}_o^{(l)}\mathbf{z}_k^{(l)} + \mathbf{U}_o^{(l)}\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_o^{(l)}\Big) \tag{46}$$

Hidden state

$$\mathbf{h}_k^{(l)} = \mathbf{o}_k^{(l)} \odot \tanh\Big(\mathbf{c}_k^{(l)}\Big) \tag{47}$$

where $\sigma(\cdot)$ denotes the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$, and $\odot$ is the Hadamard (element-wise) product. The input to layer l is:

$$\mathbf{z}_k^{(l)} = \begin{cases} \mathbf{z}_k & \text{if } l = 1, \\ \mathbf{h}_k^{(l-1)} & \text{if } l > 1. \end{cases} \tag{48}$$

Initial states are zero: $\mathbf{h}_0^{(l)} = \mathbf{c}_0^{(l)} = \mathbf{0} \in \mathbb{R}^{128}$.

The weight matrices have dimensions: $\mathbf{W}^{(1)} \in \mathbb{R}^{128\times 10}$, $\mathbf{U}^{(1)} \in \mathbb{R}^{128\times 128}$, $\mathbf{W}^{(2)}, \mathbf{U}^{(2)} \in \mathbb{R}^{128\times 128}$, and $\mathbf{b}^{(l)} \in \mathbb{R}^{128}$ for each gate.

6.3 Classification Head

After processing all $T = 60$ timesteps, the final hidden state $\mathbf{h}_{59}^{(2)} \in \mathbb{R}^{128}$ is passed through a linear projection followed by softmax:

$$\mathbf{s} = \mathbf{W}_{\text{cls}}\mathbf{h}_{59}^{(2)} + \mathbf{b}_{\text{cls}}, \qquad \mathbf{W}_{\text{cls}} \in \mathbb{R}^{4\times 128} \tag{49}$$

$$\hat{p}_c = \frac{e^{s_c}}{\sum_{j=1}^{4} e^{s_j}}, \qquad c \in \{0, 1, 2, 3\} \tag{50}$$

The predicted fire phase is:

$$\hat{y}_t^{(n)} = \arg\max_c \ \hat{p}_c \tag{51}$$

decoded via label encoder `le_phase.pkl` to one of: {`no_fire`, `smoldering`, `growing`, `flashover`}.

The confidence score is:

$$\text{conf}_t^{(n)} = \max_c \ \hat{p}_c \tag{52}$$

6.4 Distributed LSTM Execution

In distributed mode, the 2-layer LSTM is split across the Pi cluster via layer-wise model parallelism. For a node being inferred:

- **Layer 1** executes on Pi n_1: it processes $\mathbf{Z}_t \in \mathbb{R}^{60\times 10}$ and produces intermediate activations $\mathbf{H}^{(1)} \in \mathbb{R}^{60\times 128}$.
- **Layer 2** executes on Pi n_2: it receives $\mathbf{H}^{(1)}$ via SFTP and produces the final hidden state $\mathbf{h}_{59}^{(2)} \in \mathbb{R}^{128}$.

The SFTP transfer cost for the inter-layer activation tensor is:

$$C_{\text{transfer}} = T \times d_h \times 4 \text{ bytes} = 60 \times 128 \times 4 = 30{,}720 \text{ bytes} \approx 30.7 \text{ KB} \tag{53}$$

At observed SFTP bandwidth $B \approx 5\,\text{MB/s}$ on the local 100 Mbps Ethernet link, the transfer latency is approximately 6 ms, which is negligible compared to the compute time.

7 Model B: MLP Propagation Detector

7.1 Architecture

Model B is a fully connected MLP that detects fire propagation events using merged statistical and LSTM-derived features. Table 5 summarizes its configuration.

Table 5: MLP Model B Configuration

Parameter	**Value**
Input dimension	$d_{\text{mlp}} = 30$
Hidden layers	3 layers: [64, 32, 16] units
Hidden activation	ReLU
Output activation	Sigmoid
Output	scalar $\hat{p}_{\text{prop}} \in [0, 1]$
Model file	`mlp_params.pkl` ($\approx 16\,\text{KB}$)

7.2 Input Construction

The MLP input vector merges the statistical features from preprocessing with the LSTM block-5 activations:

$$\mathbf{v}_t^{(n)} = \phi_{\text{select}}\Big(\text{merge}\big(\mathbf{f}_t^{(n)},\ \mathbf{b5}_t^{(n)}\big)\Big) \in \mathbb{R}^{30} \tag{54}$$

where $\mathbf{b5}_t^{(n)}$ is the block-5 LSTM output loaded from `block5_input.json`, and ϕ_{select} selects the 30 most informative features as defined in `mlp_config.json`. This vector is then standardized:

$$\tilde{\mathbf{v}}_t^{(n)} = \frac{\mathbf{v}_t^{(n)} - \boldsymbol{\mu}_{\text{mlp}}}{\boldsymbol{\sigma}_{\text{mlp}}} \tag{55}$$

with $\boldsymbol{\mu}_{\text{mlp}},\ \boldsymbol{\sigma}_{\text{mlp}} \in \mathbb{R}^{30}$ loaded from `scaler_mlp.pkl`.

7.3 Forward Pass

For hidden layer $l \in \{1, 2, 3\}$ with input $\mathbf{a}^{(0)} = \tilde{\mathbf{v}}_t^{(n)}$:

$$\mathbf{a}^{(l)} = \text{ReLU}\big(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\big), \qquad \text{ReLU}(x) = \max(0,\ x) \tag{56}$$

The output layer computes the propagation probability via sigmoid activation:

$$\hat{p}_{\text{prop}}^{(n)} = \sigma\big(\mathbf{w}_{\text{out}}^{\top}\mathbf{a}^{(3)} + b_{\text{out}}\big) = \frac{1}{1 + \exp\big(-\mathbf{w}_{\text{out}}^{\top}\mathbf{a}^{(3)} - b_{\text{out}}\big)} \tag{57}$$

7.4 Event Thresholding

A binary propagation event is declared at node n and time t if:

$$\hat{e}_t^{(n)} = \mathbf{1}\big[\hat{p}_{\text{prop}}^{(n)} \geq \tau\big], \qquad \tau = 0.5 \tag{58}$$

The `spread_from` field is populated from `summary.json`, which records the neighboring node most likely to be the fire source based on phase onset timing.

8 Joint Spatial-Temporal Inference

8.1 Motivation for Joint Inference

Processing nodes independently fails to capture the spatial nature of fire propagation. A `growing` phase at `rpi3-b` is an isolated event; it becomes a *propagation event* only if a neighboring node (`rpi3-a` or `rpi3-c`) was already in a `growing` or `flashover` phase at $t-1$. Independent per-node inference discards this spatial context entirely.

8.2 Synchronization Condition

Let $\mathcal{N} = \{\texttt{rpi3-a}, \ldots, \texttt{rpi3-e}\}$ be the set of all nodes, with $|\mathcal{N}| = N = 5$. Define $\mathcal{R}_t \subseteq \mathcal{N}$ as the set of nodes that have completed preprocessing for sequence t (i.e., have `transformed.pkl`, `features.json`, and `block5_input.json` present). The edge controller triggers joint inference if and only if:

$$\mathcal{R}_t = \mathcal{N} \quad \Longleftrightarrow \quad |\mathcal{R}_t| = N = 5 \tag{59}$$

This condition is checked every $\Delta_{\text{poll}} = 2\,\text{s}$ by polling `priority_results/` with `os.walk`.

8.3 Joint State Vector

At synchronized time t, the joint spatial observation is:

$$\mathcal{S}_t = \left\{ \left(\hat{y}_t^{(n)},\ \hat{p}_t^{(n)},\ \hat{p}_{\text{prop}}^{(n)} \right) \right\}_{n \in \mathcal{N}} \tag{60}$$

This state allows Model B to evaluate propagation probabilities conditioned on the full neighborhood context rather than on a single isolated node.

8.4 Alert Level Function

The final alert level $\mathcal{A}_t^{(n)}$ is determined by the rule-based function `apply_alert_rules`:

$$\mathcal{A}_t^{(n)} = \begin{cases} \texttt{critical} & \text{if } \hat{y}_t^{(n)} = \texttt{flashover} \text{ or } \hat{e}_t^{(n)} = 1 \\ \texttt{warning} & \text{if } \hat{y}_t^{(n)} = \texttt{growing} \\ \texttt{watch} & \text{if } \hat{y}_t^{(n)} = \texttt{smoldering} \\ \texttt{none} & \text{if } \hat{y}_t^{(n)} = \texttt{no\_fire} \end{cases} \tag{61}$$

9 Distributed Orchestration

9.1 SSH Connection Pool

The controller maintains a persistent Paramiko SSH connection pool `_SSH_POOL`. Connections are reused across inference calls to avoid per-task SSH handshake overhead (approximately 200 ms per cold connection). The pool maps each node hostname to an active `SSHClient` instance.

9.2 Task Dispatch Protocol

For each node n at sequence t, the orchestrator executes the following protocol:

1. **Serialize**: Pack LSTM layer parameters and input tensor $\mathbf{Z}_t^{(n)}$ into a local `.pkl` file.
2. **Upload**: Transfer the file by SFTP to `node.remote_dir` on the target Pi.
3. **Execute**: Run the SSH command:

```
{python_bin} {remote_dir}/worker.py \
    --input {r_in} --output {r_out}
```

4. **Download**: Retrieve the result tensor (hidden state or final prediction) via SFTP.
5. **Assemble**: Aggregate results on the controller for decoding.

9.3 Communication Complexity

For $N = 5$ nodes, $L = 2$ LSTM layers, input dimension $d = 10$, hidden dimension $d_h = 128$, and sequence length $T = 60$:

$$C_{\text{upload}}^{(n)} = \underbrace{T \cdot d \cdot 4}_{\text{input tensor}} + \underbrace{\sim 154{,}000}_{\text{LSTM params}} \approx 156.4 \text{ KB} \tag{62}$$

$$C_{\text{download}}^{(n)} = d_h \cdot 4 = 512 \text{ bytes} \tag{63}$$

$$C_{\text{total}} = N \cdot C_{\text{upload}}^{(n)} \approx 782 \text{ KB per inference cycle} \tag{64}$$

At observed SFTP bandwidth $B \approx 3$–$5\,\text{MB/s}$ on the local network, this yields approximately 0.15–0.25 s of transfer per node, consistent with the observed end-to-end latency of 7–10 s, which is dominated by Pi CPU computation.

10 Output Structure and Logging

10.1 Prediction JSON Schema

Each inference produces `prediction.json` at:

```
inference_results/{node_id}/fire_sensors/
    seq{t:08d}/prediction.json
```

An example output is:

```
{
  "node_id": "rpi3-b",
  "seq": 38,
  "window_start": "2026-03-09T13:59:00+00:00",
  "model_a": {
    "predicted_phase": "growing",
    "confidence": 0.454,
    "probabilities": {
```

```
      "flashover": 0.190, "growing": 0.454,
      "no_fire":  0.180, "smoldering": 0.176
    }
  },
  "model_b": {
    "propagation_prob": 0.0,
    "propagation_event": 0,
    "spread_from": null,
    "confidence": "low"
  },
  "inferred_at": "2026-03-09T15:38:03Z",
  "alert_level": "warning"
}
```


10.2 Inference Log

Every inference appends one row to `inference_log.csv` with fields:

`node_id, seq, window_start, predicted_phase, phase_confidence, propagation_prob, propagation_event, alert_level, inferred_at`

This log enables post hoc temporal analysis, model evaluation, and detection of sustained fire activity across nodes.

11 Detailed Data Flow into Model A (LSTM) and Model B (MLP): Training and Inference

11.1 Preliminary: Training vs. Inference

The data pipeline is architecturally identical for both the training and inference phases. The fundamental distinction lies in what happens *after* the forward pass:

- **Training** — the model receives labeled examples, computes a loss function, and updates its internal weight matrices $\{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$ via backpropagation and gradient descent. This process runs *offline*, once, before system deployment.
- **Inference** — the trained model receives new, unseen data, executes only the forward pass (no gradient computation), and produces a prediction. This runs *live*, every 60 seconds, on the deployed Raspberry Pi cluster.

11.2 Data Flow into Model A: LSTM Fire Phase Classifier

11.2.1 Training Dataset Assembly

A labeled dataset is constructed from controlled fire experiments and simulations. Each training sample is a pair:

$$\left(\mathbf{Z}_t^{(n)},\ y_t^{(n)}\right) \tag{65}$$

where $\mathbf{Z}_t^{(n)} \in \mathbb{R}^{1\times 60\times 10}$ is the enriched time series tensor produced by the preprocessing pipeline (Section 5), and $y_t^{(n)} \in \{0,1,2,3\}$ is the ground truth fire phase label:

$$y_t^{(n)} \in \{0 \mapsto \texttt{no\_fire},\ 1 \mapsto \texttt{smoldering},\ 2 \mapsto \texttt{growing},\ 3 \mapsto \texttt{flashover}\} \tag{66}$$

11.2.2 Mini-Batch Construction

Samples are grouped into mini-batches of size B (typically $B = 32$). The batch input tensor is:

$$\mathbf{Z}_{\text{batch}} \in \mathbb{R}^{B\times 60\times 10} \tag{67}$$

This allows the LSTM to process 32 windows simultaneously, improving gradient stability and computational efficiency during training.

11.2.3 LSTM Forward Pass

The LSTM processes the batch sequentially across all $T = 60$ timesteps. For each timestep $k \in \{0,\ldots,59\}$ and layer $l \in \{1,2\}$, the following gate equations are evaluated:

Forget gate

$$\mathbf{f}_k^{(l)} = \sigma\Big(\mathbf{W}_f^{(l)}\,\mathbf{z}_k^{(l)} + \mathbf{U}_f^{(l)}\,\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_f^{(l)}\Big) \tag{68}$$

Input gate

$$\mathbf{i}_k^{(l)} = \sigma\Big(\mathbf{W}_i^{(l)}\,\mathbf{z}_k^{(l)} + \mathbf{U}_i^{(l)}\,\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_i^{(l)}\Big) \tag{69}$$

Candidate cell state

$$\tilde{\mathbf{c}}_k^{(l)} = \tanh\Big(\mathbf{W}_c^{(l)}\,\mathbf{z}_k^{(l)} + \mathbf{U}_c^{(l)}\,\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_c^{(l)}\Big) \tag{70}$$

Cell state update

$$\mathbf{c}_k^{(l)} = \mathbf{f}_k^{(l)} \odot \mathbf{c}_{k-1}^{(l)} + \mathbf{i}_k^{(l)} \odot \tilde{\mathbf{c}}_k^{(l)} \tag{71}$$

Output gate and hidden state

$$\mathbf{o}_k^{(l)} = \sigma\Big(\mathbf{W}_o^{(l)}\,\mathbf{z}_k^{(l)} + \mathbf{U}_o^{(l)}\,\mathbf{h}_{k-1}^{(l)} + \mathbf{b}_o^{(l)}\Big) \tag{72}$$

$$\mathbf{h}_k^{(l)} = \mathbf{o}_k^{(l)} \odot \tanh\Big(\mathbf{c}_k^{(l)}\Big) \in \mathbb{R}^{B\times 128} \tag{73}$$

where the input to each layer is:

$$\mathbf{z}_k^{(l)} = \begin{cases} \mathbf{z}_k & l = 1 \\ \mathbf{h}_k^{(l-1)} & l > 1 \end{cases} \tag{74}$$

All initial states are zero: $\mathbf{h}_0^{(l)} = \mathbf{c}_0^{(l)} = \mathbf{0} \in \mathbb{R}^{128}$. After processing all 60 timesteps, only the final hidden state $\mathbf{h}_{59}^{(2)} \in \mathbb{R}^{B\times 128}$ is retained.

11.2.4 Classification Head

The final hidden state is projected onto the 4-class output space:

$$\mathbf{s} = \mathbf{h}_{59}^{(2)}\,\mathbf{W}_{\text{cls}}^{\top} + \mathbf{b}_{\text{cls}}, \qquad \mathbf{W}_{\text{cls}} \in \mathbb{R}^{4\times 128} \tag{75}$$

$$\hat{p}_{b,c} = \frac{e^{s_{b,c}}}{\sum_{j=1}^{4} e^{s_{b,j}}}, \quad \hat{\mathbf{P}} \in \mathbb{R}^{B\times 4} \tag{76}$$

11.2.5 Training: Loss Function

The training objective is the categorical **Cross-Entropy** loss:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{B}\sum_{b=1}^{B} \log \hat{p}_{b,\,y^{(b)}} \tag{77}$$

This penalizes the model in proportion to the negative log-probability assigned to the correct class. If the model is highly confident and correct ($\hat{p}_{b,y^{(b)}} \approx 1$), the loss contribution is near zero. If the model is confident but wrong ($\hat{p}_{b,y^{(b)}} \approx 0$), the loss is large.

11.2.6 Training: Backpropagation Through Time (BPTT)

Because the LSTM is a recurrent network unrolled over 60 timesteps, gradients must be propagated *backwards through time*. The gradient of the loss with respect to a weight matrix $\mathbf{W}_f^{(l)}$ is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_f^{(l)}} = \sum_{k=0}^{59} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_k^{(l)}} \cdot \frac{\partial \mathbf{h}_k^{(l)}}{\partial \mathbf{f}_k^{(l)}} \cdot \frac{\partial \mathbf{f}_k^{(l)}}{\partial \mathbf{W}_f^{(l)}} \tag{78}$$

This propagates error signals from the classification output at $k = 59$ all the way back to $k = 0$, allowing the LSTM to learn long-range temporal dependencies — for example, that a sustained gas resistance drop starting at second 10 predicts a `growing` phase declared at second 59.

11.2.7 Training: Adam Optimizer

Weights are updated using the Adam optimizer with learning rate $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$:

$$\mathbf{m}_t = \beta_1\,\mathbf{m}_{t-1} + (1-\beta_1)\,\nabla_\theta \mathcal{L} \tag{79}$$

$$\mathbf{v}_t = \beta_2\,\mathbf{v}_{t-1} + (1-\beta_2)\,(\nabla_\theta \mathcal{L})^2 \tag{80}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \qquad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{81}$$

$$\theta \leftarrow \theta - \alpha\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} \tag{82}$$

This process repeats over all mini-batches across multiple epochs until $\mathcal{L}_{\text{CE}}$ converges. The trained parameters are serialized to `lstm_params.pkl`, and the training-set scaler statistics are saved to `scaler_lstm.pkl`.

11.2.8 Inference: Live Forward Pass

During live operation, the forward pass executes identically but *without gradient computation*:

1. Load `transformed.pkl`: $\mathbf{Z}_t^{(n)} \in \mathbb{R}^{1\times 60\times 10}$
2. Load `lstm_params.pkl`: all weight matrices
3. Initialize $\mathbf{h}_0^{(l)} = \mathbf{c}_0^{(l)} = \mathbf{0} \in \mathbb{R}^{128}$
4. Execute LSTM cell equations for $k = 0$ to 59 (Layers 1 and 2 sequentially)
5. Compute classifier output: $\hat{\mathbf{p}} = \text{softmax}(\mathbf{W}_{\text{cls}}\mathbf{h}_{59}^{(2)} + \mathbf{b}_{\text{cls}}) \in \mathbb{R}^4$
6. Decode prediction: $\hat{y}_t^{(n)} = \arg\max_c \hat{p}_c, \quad \text{conf}_t^{(n)} = \max_c \hat{p}_c$
7. Save $\mathbf{h}_{59}^{(2)}$ as `block5_input.json` for Model B consumption
8. Write `prediction.json` with phase, confidence, and probability vector

11.3 Data Flow into Model B: MLP Propagation Detector

11.3.1 Why Model B Requires Joint Node Context

Model A evaluates each node n in isolation. It cannot determine whether a `growing` phase at node n is an independent fire event or the result of fire spreading from a neighboring node m. Model B solves this by operating on a feature vector that explicitly encodes the spatial state of the full node network at time t.

11.3.2 Spatial Feature Construction

After all $N = 5$ nodes have completed Model A inference at time t, the controller holds the joint state:

$$\mathcal{S}_t = \left\{ \left(\mathbf{f}_t^{(n)},\ \mathbf{b5}_t^{(n)},\ \hat{y}_t^{(n)},\ \text{conf}_t^{(n)} \right) \right\}_{n\in\mathcal{N}} \tag{83}$$

For each node n, its Model B input is built by merging its own features with neighbor context. For each neighbor m adjacent to n, the following spatial features are computed:

Neighbor phase encoding

$$\phi_m = \hat{y}_t^{(m)} \in \{0, 1, 2, 3\} \tag{84}$$

Directly informs Model B of the fire phase at the neighboring node.

Neighbor confidence

$$\psi_m = \text{conf}_t^{(m)} = \max_c \hat{p}_c^{(m)} \tag{85}$$

A high-confidence `growing` at a neighbor is a stronger propagation signal than a low-confidence one.

Phase differential

$$\delta_{nm} = \hat{y}_t^{(n)} - \hat{y}_t^{(m)} \tag{86}$$

Encodes the direction of fire progression. If $\delta_{nm} < 0$, node n is in an earlier fire phase than neighbor m, consistent with fire moving from m toward n.

Inter-node gas resistance differential

$$\Gamma_{nm} = \mu_G^{(n)} - \mu_G^{(m)} \tag{87}$$

A large positive Γ_{nm} means gas resistance is lower (worse air quality) at node n than at neighbor m, suggesting smoke has penetrated node n's zone — consistent with fire spreading from m.

11.3.3 Full Model B Input Vector

The complete input vector for node n is assembled as:

$$\mathbf{v}_t^{(n)} = \phi_{\text{select}}\left(\text{concat}\left(\underbrace{\mathbf{f}_t^{(n)}}_{\mathbb{R}^{30}}, \underbrace{\mathbf{b5}_t^{(n)}}_{\mathbb{R}^{128}}, \underbrace{\phi_m,\ \psi_m,\ \delta_{nm},\ \Gamma_{nm}}_{\text{neighbor context}}\right)\right) \in \mathbb{R}^{30} \tag{88}$$

where ϕ_{select} selects the 30 most informative features from the full concatenation based on variance and mutual information analysis performed at training time. The vector is then standardized:

$$\tilde{\mathbf{v}}_t^{(n)} = \frac{\mathbf{v}_t^{(n)} - \boldsymbol{\mu}_{\text{mlp}}}{\boldsymbol{\sigma}_{\text{mlp}}} \in \mathbb{R}^{30} \tag{89}$$

11.3.4 MLP Forward Pass

With $\mathbf{a}^{(0)} = \tilde{\mathbf{v}}_t^{(n)}$, the forward pass through the three hidden layers is:

$$\mathbf{a}^{(1)} = \text{ReLU}\big(\mathbf{W}^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)}\big) \in \mathbb{R}^{64} \tag{90}$$

$$\mathbf{a}^{(2)} = \text{ReLU}\big(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}\big) \in \mathbb{R}^{32} \tag{91}$$

$$\mathbf{a}^{(3)} = \text{ReLU}\big(\mathbf{W}^{(3)}\mathbf{a}^{(2)} + \mathbf{b}^{(3)}\big) \in \mathbb{R}^{16} \tag{92}$$

$$\hat{p}_{\text{prop}} = \sigma\big(\mathbf{w}_{\text{out}}^{\top}\,\mathbf{a}^{(3)} + b_{\text{out}}\big) \in [0,1] \tag{93}$$

where $\text{ReLU}(x) = \max(0, x)$ and $\sigma(x) = (1 + e^{-x})^{-1}$.

11.3.5 Training: Loss Function

Model B is a binary classifier; its training objective is the **Binary Cross-Entropy** loss:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{B}\sum_{b=1}^{B}\left[e^{(b)}\log\hat{p}_{\text{prop}}^{(b)} + \big(1 - e^{(b)}\big)\log\big(1 - \hat{p}_{\text{prop}}^{(b)}\big)\right] \tag{94}$$

where $e^{(b)} \in \{0, 1\}$ is the ground truth propagation label. The two terms penalize separately:

- **First term** $e^{(b)} \log \hat{p}^{(b)}_{\text{prop}}$: punishes missed detections — fire was spreading but the model predicted low probability
- **Second term** $(1 - e^{(b)}) \log(1 - \hat{p}^{(b)}_{\text{prop}})$: punishes false alarms — no fire spread but the model predicted high probability

11.3.6 Training: Backpropagation

Since the MLP has no recurrence, standard backpropagation applies. The gradient with respect to layer l weights is:

$$\frac{\partial \mathcal{L}_{\text{BCE}}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}_{\text{BCE}}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{W}^{(l)}} \tag{95}$$

The ReLU derivative is:

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{96}$$

Weights are updated with the same Adam optimizer as Model A (Equation 82). Model B is trained *after* Model A is fully trained, since its input depends on LSTM block-5 activations extracted from the trained Model A.

Trained parameters are serialized to `mlp_params.pkl` and `scaler_mlp.pkl`.

11.3.7 Inference: Live Propagation Detection

During live operation:

1. Wait for synchronization condition $|\mathcal{R}_t| = N = 5$ (all nodes ready at time t)
2. For each node n: load `features.json` and `block5_input.json`
3. Construct neighbor context features $\phi_m, \psi_m, \delta_{nm}, \Gamma_{nm}$ from $\mathcal{S}_t$
4. Assemble and standardize $\tilde{\mathbf{v}}^{(n)}_t \in \mathbb{R}^{30}$
5. Execute MLP forward pass (no gradients): $\hat{p}^{(n)}_{\text{prop}} \in [0, 1]$
6. Apply threshold: $\hat{e}^{(n)}_t = \mathbf{1}\left[\hat{p}^{(n)}_{\text{prop}} \geq 0.5\right]$
7. Determine alert level via `apply_alert_rules`:

$$\mathcal{A}^{(n)}_t = \begin{cases} \texttt{critical} & \hat{e}^{(n)}_t = 1 \text{ or } \hat{y}^{(n)}_t = \texttt{flashover} \\ \texttt{warning} & \hat{y}^{(n)}_t = \texttt{growing} \\ \texttt{watch} & \hat{y}^{(n)}_t = \texttt{smoldering} \\ \texttt{none} & \hat{y}^{(n)}_t = \texttt{no\_fire} \end{cases} \tag{97}$$

8. Write `prediction.json` with all fields

11.4 Complete Training-to-Inference Timeline

Table 6 summarizes the full lifecycle from raw data to live alert for both models.

Table 6: Training and Inference Lifecycle for Model A and Model B

Stage	Model A (LSTM)	Model B (MLP)
Input data	$\mathbf{Z}_t^{(n)} \in \mathbb{R}^{1\times 60\times 10}$	$\tilde{\mathbf{v}}_t^{(n)} \in \mathbb{R}^{30}$
Training label	$y \in \{0,1,2,3\}$ (fire phase)	$e \in \{0,1\}$ (propagation event)
Loss function	Cross-Entropy $\mathcal{L}_{\text{CE}}$	Binary Cross-Entropy $\mathcal{L}_{\text{BCE}}$
Gradient method	BPTT across 60 timesteps	Standard backpropagation
Optimizer	Adam ($\alpha = 0.001$)	Adam ($\alpha = 0.001$)
Saved artefacts	`lstm_params.pkl`	`mlp_params.pkl`
Inference input	`transformed.pkl`	`features.json` + `block5_input.json` + neighbor context
Inference output	Phase label + confidence	Propagation probability
Trigger condition	Per node, per sequence	All 5 nodes ready at time t

12 System Performance Analysis

12.1 Latency Budget

Table 7 breaks down the end-to-end latency per node per sequence.

Table 7: End-to-End Latency Budget per Node

Stage	Duration
Sensor polling (full window)	60 s
MQTT notification	< 100 ms
SSH spawn + preprocessing	5–8 s
SFTP upload to Pi	≈ 0.2 s
LSTM forward pass on Pi CPU	5–8 s
SFTP download result	< 50 ms
MLP forward pass (controller)	< 10 ms
JSON write to disk	< 5 ms
Total per node	**7–10 s**
Joint 5-node cycle	**35–50 s**

12.2 Real-Time Constraint Verification

With $\Delta_t = 60$ s between successive windows and joint inference taking $T_{\text{infer}} \approx 35$–$50$ s, the real-time constraint is satisfied:

$$T_{\text{infer}} \approx 35\text{–}50 \text{ s} \ll \Delta_t = 60 \text{ s} \tag{98}$$

leaving 10–25 s of slack per cycle for network jitter and processing variance.

12.3 Scalability

The architecture scales horizontally: adding a sixth node n_6 requires only:

1. Registering its IP in `nodes.csv`
2. Adding its node ID to the `-all_nodes` CLI argument

No code changes are required. The synchronization condition (Equation 59) generalizes to $|\mathcal{R}_t| = N$ for any N.

13 Conclusion

This paper has presented the full mathematical and implementation details of a distributed AI fire detection system built on commodity Raspberry Pi hardware. The key contributions are:

1. **Joint temporal synchronization**: waiting for all N nodes before triggering inference at time t enables spatially coherent propagation detection (Section 8).
2. **Layer-wise LSTM distribution**: offloading LSTM forward passes to edge nodes reduces controller CPU load and leverages available compute across the cluster (Section 9).
3. **Two-stage ensemble**: separating fire phase classification (LSTM) from propagation detection (MLP) allows each model to specialize with appropriate input features (Sections 6–7).
4. **MQTT-driven reactive pipeline**: event-driven data ingestion minimizes idle computation and ensures low-latency window processing (Section 3).

Future work includes replacing the rule-based alert function with a learned Graph Neural Network over the node topology, incorporating LoRaWAN for wireless deployment, and quantizing the LSTM for sub-second on-device inference.