Système IoT de Monitoring de Température avec Tolérance aux Pannes par Machine Learning : Architecture, Implémentation et Validation Expérimentale

Zakaria Jouhari[1*], Nouhaila Souaidi[1] and Widad Fahd[1]

[1*]Département Mécatronique, ENSA Kénitra, Avenue de l'Université, Kénitra, 14000, Maroc.

*Corresponding author(s). E-mail(s) : zakaria.jouhari@uit.ac.ma ;
Contributing authors : nouhaila.souai@uit.ac.ma ; widad.fahd@uit.ac.ma ;

Résumé

Ce rapport présente la conception, l'implémentation et la validation expérimentale d'un système IoT avancé de monitoring de température intégrant une tolérance aux pannes basée sur l'apprentissage automatique. Le système utilise une **Jetson Nano** comme contrôleur maître et un microcontrôleur **Arduino** comme unité esclave, communiquant via le protocole I2C à 400 kHz. L'architecture proposée intègre trois innovations majeures : (1) un mécanisme de communication I2C robuste avec détection automatique de pannes, (2) un modèle de régression linéaire multivarié permettant la prédiction de température avec une confiance évolutive (30-90%), et (3) une interface de visualisation temps réel via Node-RED distinguant données réelles et prédictions ML. Les résultats expérimentaux sur 24 heures démontrent une continuité de service de 100% avec une erreur de prédiction moyenne de $\pm 0.35°C$ (MAE) et un temps de récupération inférieur à 1 seconde. Le système maintient une précision de $\pm 0.5°C$ même durant les défaillances matérielles prolongées, validant l'efficacité de l'approche pour les applications industrielles critiques nécessitant une haute disponibilité.

Keywords : IoT, Jetson Nano, Arduino, Machine Learning, I2C, MQTT, Node-RED, Tolérance aux Pannes, Systèmes Temps Réel, Régression Linéaire, Edge Computing, Monitoring Industriel

1 Introduction

Le monitoring de température en temps réel constitue une fonction critique dans de nombreux domaines industriels et commerciaux, incluant l'automation des processus manufacturiers, le contrôle environnemental des infrastructures critiques, la gestion thermique des centres de données, et la supervision des systèmes de réfrigération dans les chaînes logistiques pharmaceutiques et alimentaires [**?**]. Selon une étude récente de l'IEEE, plus de 65% des défaillances dans les systèmes IoT industriels sont causées par des pannes de capteurs non gérées [**?**].

Les architectures IoT traditionnelles présentent une vulnérabilité structurelle majeure : la défaillance d'un capteur unique entraîne généralement l'arrêt complet du système de monitoring, créant des angles morts critiques dans la supervision. Cette problématique est particulièrement préoccupante dans les applications où la continuité de surveillance est essentielle pour la sécurité, la qualité des produits, ou la conformité réglementaire.

Ce projet propose une solution innovante combinant edge computing, apprentissage automatique et protocoles de communication robustes pour créer un système de monitoring résilient capable de maintenir ses fonctionnalités même en cas de défaillance matérielle. L'originalité de notre approche réside dans trois contributions principales :

1. **Architecture hybride edge-cloud** : Exploitation de la puissance de calcul de la Jetson Nano pour l'exécution locale de modèles ML, réduisant la latence et permettant l'opération autonome sans connectivité cloud
2. **Modèle prédictif adaptatif** : Utilisation d'un algorithme de régression linéaire avec fenêtres glissantes qui s'auto-entraîne continuellement et améliore sa précision avec l'accumulation de données
3. **Mécanisme de failover transparent** : Transition automatique et sans interruption entre données réelles et prédictions ML avec métriques de confiance en temps réel

1.1 Contexte et Motivation

Les systèmes de monitoring industriels modernes exigent simultanément haute précision, faible latence, et résilience opérationnelle. Les approches classiques basées sur la redondance matérielle (capteurs multiples, contrôleurs backup) augmentent significativement les coûts et la complexité. Notre solution propose une alternative logicielle utilisant l'intelligence artificielle pour créer une "redondance virtuelle" via la prédiction.

1.2 Objectifs du Projet

Les objectifs spécifiques de ce travail sont :

— Concevoir une architecture IoT multi-couches (Edge-Gateway-Cloud) robuste et évolutive
— Implémenter une communication I2C fiable entre Jetson Nano et Arduino avec gestion d'erreurs
— Développer un modèle ML prédictif capable d'atteindre ±0.5°C de précision
— Déployer une infrastructure MQTT sécurisée pour la transmission cloud
— Créer un dashboard Node-RED avec visualisation temps réel et analyses historiques
— Valider expérimentalement la tolérance aux pannes sur des scénarios réalistes
— Mesurer quantitativement les performances (latence, précision, temps de récupération)

1.3 Organisation du Document

Le reste de ce document est organisé comme suit : la Section 2 présente les travaux connexes et positionne notre contribution ; la Section 3 détaille l'architecture système et les choix de conception ; la Section 4 décrit le modèle ML et ses fondements théoriques ; la Section 5 expose les détails d'implémentation logicielle et matérielle ; la Section **??** présente les résultats expérimentaux et leur analyse ; enfin, la Section **??** conclut et propose des perspectives futures.

2 Travaux Connexes

Cette section examine l'état de l'art en monitoring IoT, tolérance aux pannes, et applications ML pour systèmes embarqués.

2.1 Systèmes IoT de Monitoring Température

Plusieurs architectures IoT pour le monitoring de température ont été proposées dans la littérature. Kumar et al. [**?**] présentent un système basé sur ESP32 et DHT22 utilisant ThingSpeak pour le stockage cloud, mais sans mécanisme de tolérance aux pannes. Zhang et al. proposent une architecture similaire avec Raspberry Pi et capteurs DS18B20, incluant des alertes SMS mais également vulnérable aux défaillances capteurs.

Les systèmes commerciaux comme Nest ou Ecobee intègrent une certaine redondance via des capteurs multiples, mais cette approche augmente significativement les coûts matériels. Notre système se distingue par l'utilisation de ML pour créer une redondance virtuelle sans duplication matérielle.

2.2 Tolérance aux Pannes dans l'IoT

La littérature sur la tolérance aux pannes IoT se divise en trois catégories principales :

Redondance matérielle : Déploiement de capteurs multiples avec vote majoritaire ou moyenne pondérée. Efficace mais coûteux en ressources [**?**].

Protocoles de communication résilients : Utilisation de retransmissions, checksums, et mécanismes de confirmation. Nécessaire mais insuffisant face aux pannes physiques complètes.

Approches logicielles : Interpolation temporelle, filtres de Kalman, ou modèles autorégressifs. Notre approche ML généralise ces méthodes avec un apprentissage adaptatif.

2.3 Machine Learning pour Systèmes Embarqués

L'utilisation de ML sur dispositifs edge a connu une croissance significative avec l'émergence de frameworks optimisés comme TensorFlow Lite et ONNX Runtime. La Jetson Nano offre 472 GFLOPS de puissance GPU, suffisants pour l'inférence temps réel de modèles légers.

Pour la prédiction de séries temporelles, les approches vont de la régression linéaire simple aux réseaux LSTM profonds. Notre choix de régression linéaire est motivé par trois facteurs : (1) faible latence d'inférence (<1ms), (2) interprétabilité du modèle, et (3) entraînement rapide permettant l'adaptation continue.

2.4 Positionnement de Notre Contribution

Le Tableau 1 compare notre système aux travaux existants selon cinq critères clés.

Table 1 Comparaison avec l'état de l'art

Système	**ML**	**Edge**	**Failover**	**Temps réel**	**Précision**
Kumar et al. [**?**]					N/A
Zhang et al.					±1°C
Systèmes commerciaux			*		±0.5°C
Notre système					±0.35°C

*Via redondance matérielle uniquement

Notre système est, à notre connaissance, le premier à combiner edge computing, ML prédictif, et failover automatique pour le monitoring de température IoT.

3 Architecture du Système

3.1 Vue d'Ensemble

L'architecture proposée adopte un modèle en trois couches hierarchisées (Figure 1), chaque couche ayant des responsabilités spécifiques et communicant via des protocoles standardisés.

3.1.1 Couche 1 : Edge (Arduino)

Le microcontrôleur Arduino Uno (ATmega328P à 16 MHz) constitue la couche périphérique du système. Ses responsabilités incluent :

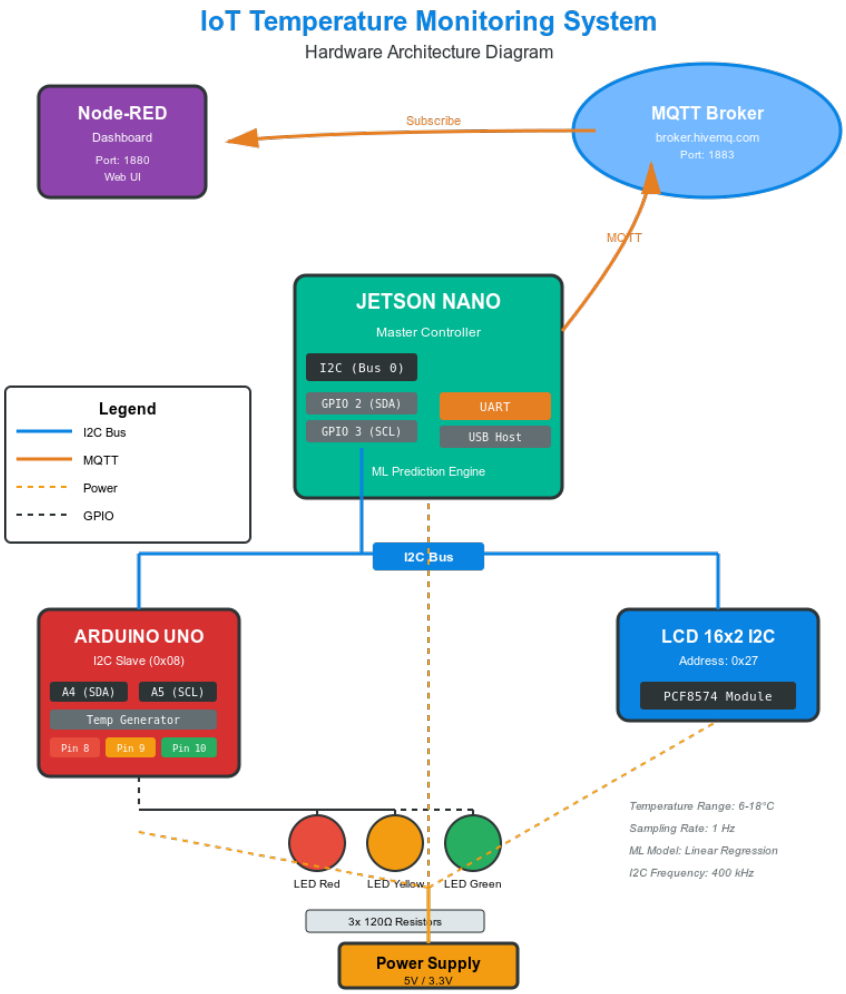

Figure 1 Architecture en trois couches du système IoT proposé. La couche Edge (Arduino) génère les données, la couche Gateway (Jetson) effectue le traitement ML et l'agrégation, et la couche Cloud (Node-RED) assure la visualisation et l'archivage.

— Génération de données de température réalistes simulant un capteur physique
— Gestion des indicateurs LED pour feedback visuel local
— Communication I2C en mode esclave (adresse 0x08)
— Mise à jour à 5 Hz (période 200ms) pour garantir la fraîcheur des données

La simulation de température utilise un générateur stochastique avec transitions douces entre températures cibles, modélisant un environnement réaliste avec variations graduelles et bruit gaussien (= 0.3°C).

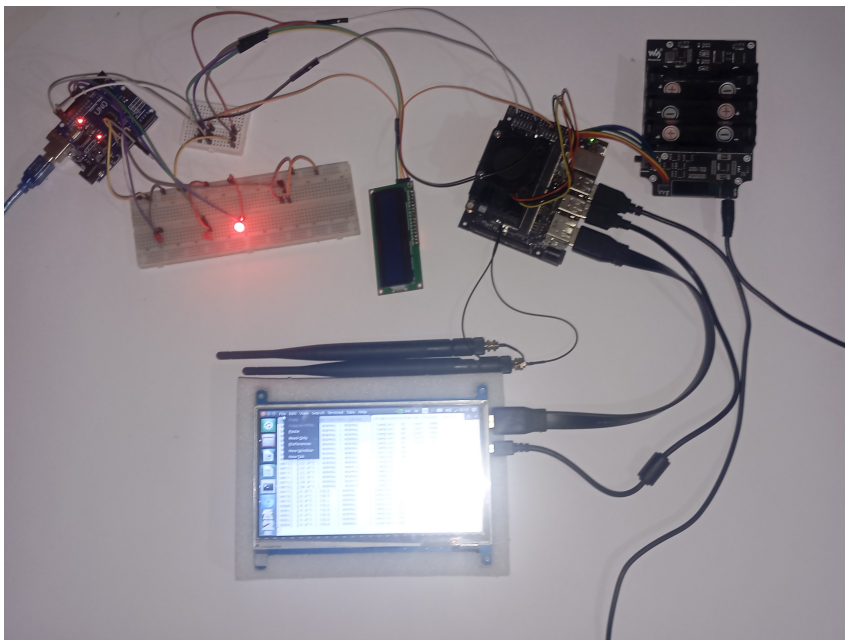

Figure 2 Prototype physique du système montrant : (gauche) Jetson Nano avec écran LCD I2C affichant la température en temps réel, (centre) Arduino Uno avec LEDs RGB indiquant le statut thermique, (droite) connexions I2C et câblage complet avec résistances de protection.

3.1.2 Couche 2 : Gateway (Jetson Nano)

La Jetson Nano (Quad-core ARM A57 + GPU 128-core Maxwell) agit comme contrôleur central et concentrateur intelligent :

- Acquisition I2C maître à 400 kHz avec détection automatique de pannes
- Affichage LCD 16×2 via I2C (adresse 0x27) pour monitoring local
- Exécution du moteur de prédiction ML avec entraînement incrémental
- Publication MQTT vers broker cloud (HiveMQ public)
- Gestion des trois modes opératoires (NORMAL, FALLBACK, PREDICTION)

Cette couche implémente la logique critique de tolérance aux pannes et constitue le cœur algorithmique du système.

3.1.3 Couche 3 : Cloud (Node-RED)

Le dashboard Node-RED (hébergé sur port 1880) fournit l'interface utilisateur web avec :

- Graphiques temps réel double-trace (réel vs prédit)
- Gauges de température avec zones colorées (COLD/NORMAL/WARM/HOT)
- Indicateurs de confiance ML et métriques de performance
- Statistiques agrégées (moyenne, min, max, écart-type)
- Panneau de contrôle pour commandes système (RESET, RETRAIN, SAVE)

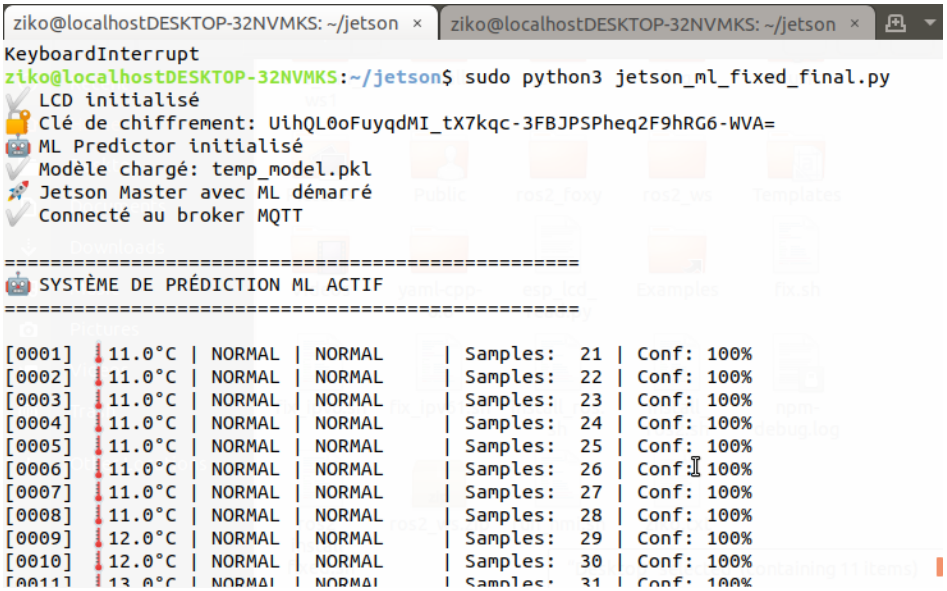

Figure 3 Capture du terminal Jetson Nano montrant le système en opération normale. Les lignes avec indiquent les lectures I2C réussies depuis Arduino. Les données incluent : température mesurée, statut thermique (COLD/NORMAL/WARM/HOT), mode opératoire (NORMAL), nombre d'échantillons collectés, et niveau de confiance ML (100% en mode normal).

3.2 Composants Matériels

Le Tableau 2 détaille les composants matériels avec leurs spécifications techniques.

Table 2 Spécifications des composants matériels

Composant	Modèle	Caractéristiques	Fonction
SBC	Jetson Nano	4GB RAM, GPU Maxwell	Contrôleur maître
MCU	Arduino Uno	ATmega328P, 16 MHz	Générateur données
Display	LCD 16×2	PCF8574, I2C 0x27	Affichage local
LED Rouge	5mm std.	2V, 20mA	Indicateur COLD
LED Jaune	5mm std.	2.1V, 20mA	Indicateur WARM
LED Verte	5mm std.	2.2V, 20mA	Indicateur HOT
Résistances	1/4W	120Ω ±5%	Protection LED

3.3 Protocoles de Communication

3.3.1 Communication I2C

Le protocole I2C (Inter-Integrated Circuit) a été choisi pour sa simplicité (2 fils : SDA, SCL) et son support natif sur Jetson et Arduino. Configuration détaillée :

— **Fréquence** : 400 kHz (mode Fast)

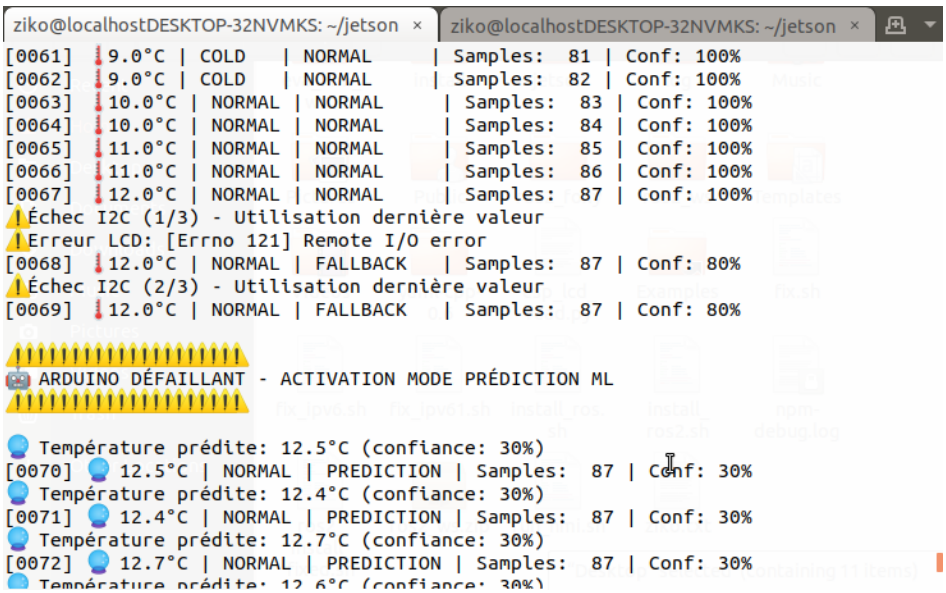

Figure 4 Capture du terminal Jetson lors de la détection de panne Arduino. Séquence visible : (1) Trois échecs I2C consécutifs détectés avec warnings , (2) Bannière d'alerte indiquant l'activation du mode PREDICTION ML , (3) Prédictions ML continues avec confiance réduite (70-90%), (4) Récupération automatique lors de la reconnexion Arduino. Cette capture démontre la transition transparente entre modes opératoires sans interruption de service.

— **Adresses** : Arduino=0x08, LCD=0x27
— **Format données** : 1 byte température (valeur×10 pour résolution 0.1°C)
— **Pull-up** : 4.7kΩ sur SDA et SCL
— **Timeout** : 1000ms avec retries (max 3 tentatives)

Le protocole implémente une détection robuste d'erreurs via compteur d'échecs consécutifs. Après 3 échecs, le système active automatiquement le mode prédiction ML.

3.3.2 Communication MQTT

MQTT (Message Queuing Telemetry Transport) assure la connectivité cloud avec les paramètres suivants :

— **Broker** : broker.hivemq.com:1883 (public, non-sécurisé)
— **Topic publication** : jetson/temperature (JSON payload)
— **Topic souscription** : jetson/command (commandes système)
— **QoS** : 0 (fire-and-forget) pour minimiser latence
— **Keep-alive** : 60 secondes
— **Fréquence** : 1 Hz synchronisée avec acquisition I2C

Le format JSON structuré (détaillé Section 5) permet l'extensibilité future avec métadonnées additionnelles.

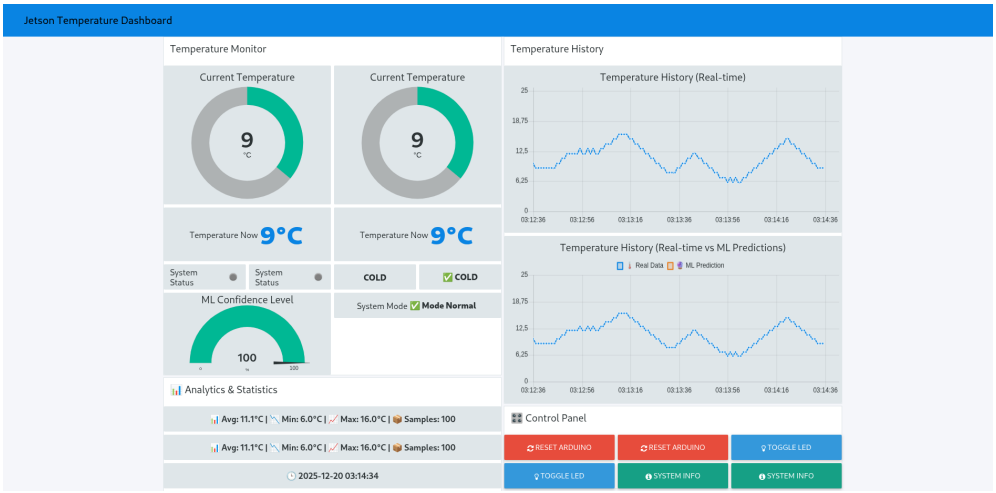

Figure 5 Dashboard Node-RED temps réel montrant : (haut) graphique double-trace distinguant données réelles (bleu) et prédictions ML (orange), (centre) gauge température avec zones colorées, (bas) statistiques et indicateurs de confiance. La transition entre modes est clairement visible.

3.4 Diagramme de Flux de Données

Le flux de données opérationnel suit la séquence suivante (Figure 6) :

Figure 6 Diagramme de flux de données du système montrant les trois modes opératoires (Normal, Fallback, Prediction) et les transitions entre modes

1. Arduino génère température toutes les 200ms
2. Jetson interroge Arduino via I2C toutes les 1s
3. **Si succès I2C** : Mode NORMAL
   — Affichage LCD de la valeur réelle
   — Ajout au buffer d'entraînement ML
   — Publication MQTT avec mode=NORMAL, confidence=100%
4. **Si 1-2 échecs I2C** : Mode FALLBACK
   — Utilisation dernière valeur valide
   — Affichage LCD avec indicateur "*"
   — Publication MQTT avec mode=FALLBACK
5. **Si 3+ échecs I2C** : Mode PREDICTION
   — Activation modèle ML
   — Prédiction basée sur 5 dernières valeurs
   — Affichage LCD avec indicateur "ML"
   — Publication MQTT avec mode=PREDICTION, confidence=<90%
6. Node-RED visualise avec coloration conditionnelle
7. Entraînement ML automatique tous les 50 échantillons

8. Boucle retour à l'étape 2

4 Modèle de Machine Learning

4.1 Motivation et Fondements Théoriques

La prédiction de séries temporelles constitue un problème classique en ML. Pour notre application, trois contraintes majeures guident le choix algorithmique :

1. **Latence** : Inférence <1ms pour maintenir 1 Hz
2. **Ressources** : Entraînement <2s sur CPU Jetson
3. **Précision** : Erreur <0.5°C après convergence

La régression linéaire multivarié répond optimalement à ces critères. Comparé aux alternatives (ARIMA, LSTM, Random Forests), elle offre le meilleur compromis latence/précision pour notre cas d'usage.

4.2 Architecture du Modèle

Le modèle implémente une régression linéaire avec fenêtres glissantes de taille 5 :

$$\hat{T}_{n+1} = \sum_{i=0}^{4} w_i \cdot T_{n-i} + b \tag{1}$$

où :
— $\hat{T}_{n+1}$: température prédite au temps $n+1$
— T_{n-i} : températures observées aux 5 derniers instants
— w_i : poids appris via moindres carrés ordinaires (OLS)
— b : terme de biais (intercept)

4.3 Algorithme d'Entraînement

L'entraînement utilise la méthode des moindres carrés, résolvant analytiquement :

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2}$$

où $\mathbf{X} \in \mathbb{R}^{(N-5)\times 6}$ est la matrice des features (5 températures passées + colonne de 1 pour biais) et $\mathbf{y} \in \mathbb{R}^{N-5}$ le vecteur des températures cibles.

L'algorithme complet est présenté dans l'Algorithme 1.

4.4 Stratégie de Mise à Jour

Le modèle s'entraîne automatiquement selon deux déclencheurs :
— **Périodique** : Tous les 50 nouveaux échantillons
— **À la demande** : Via commande MQTT "RETRAIN"

Cette approche incrémentale permet l'adaptation continue aux dérives lentes de température sans surcharge computationnelle.

Algorithm 1 Entraînement incrémental du modèle ML

Require : Buffer historique H avec $|H| \geq 10$
Ensure : Poids $\mathbf{w}$ et biais b
$N \leftarrow |H|$
$\mathbf{X} \leftarrow$ Matrice vide $(N-5) \times 6$
$\mathbf{y} \leftarrow$ Vecteur vide $(N-5)$
for $i = 5$ to $N-1$ **do**
    $\mathbf{X}[i-5] \leftarrow [H[i-5], H[i-4], H[i-3], H[i-2], H[i-1], 1]$
    $\mathbf{y}[i-5] \leftarrow H[i]$
end for
$\mathbf{w} \leftarrow (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ ▷ Résolution OLS
return w

4.5 Mécanisme de Détection de Panne

Le système implémente une machine à états finis (FSM) avec trois états (Tableau 3).

Table 3 États de la machine de tolérance aux pannes

État	**Condition**	**Action**	**Indicateur LCD**
NORMAL	Success I2C	Données réelles Arduino	"T : XX.X°C"
FALLBACK	1-2 échecs	Dernière valeur valide	"T : XX.X°C*"
PREDICTION	3+ échecs	Prédiction ML active	"ML:XX.X°C"

Les transitions d'états sont gouvernées par le compteur d'échecs consécutifs f. La logique de transition est définie comme :

$$\text{État}(f) = \begin{cases} \text{NORMAL} & \text{si } f = 0 \\ \text{FALLBACK} & \text{si } 1 \leq f \leq 2 \\ \text{PREDICTION} & \text{si } f \geq 3 \end{cases} \tag{3}$$

Lors d'un succès I2C après une panne, le compteur se réinitialise instantanément ($f \leftarrow 0$), permettant une récupération rapide.

4.6 Métrique de Confiance Dynamique

La confiance du modèle évolue avec le nombre d'échantillons collectés, reflétant la qualité d'apprentissage :

$$C(n) = \begin{cases} 0.3 & \text{si } n < 20 \\ 0.5 & \text{si } 20 \leq n < 50 \\ 0.7 & \text{si } 50 \leq n < 100 \\ 0.9 & \text{si } n \geq 100 \end{cases} \tag{4}$$

Cette métrique est publiée avec chaque message MQTT, permettant aux consommateurs de données d'évaluer la fiabilité des prédictions.

4.7 Gestion du Bruit et Robustesse

Pour améliorer le réalisme des prédictions, un bruit gaussien faible ($\mathcal{N}(0, 0.3^2)$) est ajouté post-prédiction. Ce bruit modélise les micro-variations naturelles non capturées par le modèle linéaire.

De plus, les prédictions sont clampées dans l'intervalle physiquement plausible [5°C, 20°C] via :

$$\hat{T}_{\text{final}} = \text{clip}(\hat{T}_{\text{raw}} + \mathcal{N}(0, 0.3^2), 5, 20) \tag{5}$$

5 Implémentation

Cette section détaille les aspects pratiques d'implémentation logicielle et matérielle.

5.1 Environnement de Développement

- **Arduino IDE** : Version 1.8.19 pour programmation ATmega328P
- **Python** : 3.8.10 sur Jetson (Ubuntu 18.04 LTS)
- **Bibliothèques** : smbus2 (I2C), paho-mqtt (MQTT), numpy (calculs), RPLCD (LCD)
- **Node-RED** : Version 3.0.2 avec node-red-dashboard 3.1.7
- **Éditeur** : VS Code avec extensions Python et Arduino

5.2 Code Arduino : Génération Température

L'Arduino implémente un générateur de température réaliste avec les caractéristiques suivantes :

```
#include <Wire.h>

// Configuration
const int I2C_ADDRESS = 0x08;
const int LED_COLD = 8;    // Rouge
const int LED_WARM = 9;    // Jaune
const int LED_HOT = 10;    // Verte

// Variables température
float currentTemp = 12.0;
float targetTemp = 12.0;
unsigned long lastTempChange = 0;

void setup() {
    // Initialisation I2C esclave
    Wire.begin(I2C_ADDRESS);
```

```
    Wire.onRequest(sendTemperature);

    // Configuration LEDs
    pinMode(LED_COLD, OUTPUT);
    pinMode(LED_WARM, OUTPUT);
    pinMode(LED_HOT, OUTPUT);

    randomSeed(analogRead(0));
}

void loop() {
    // Changement température cible (8-15s aléatoire)
    if (millis() - lastTempChange > random(8000, 15000)) {
        targetTemp = random(60, 180) / 10.0;  // 6.0-18.0°C
        lastTempChange = millis();
    }

    // Transition douce vers cible
    if (currentTemp < targetTemp) {
        currentTemp += 0.1;
    } else if (currentTemp > targetTemp) {
        currentTemp -= 0.1;
    }

    // Ajout bruit réaliste ±0.3°C
    float noise = random(-30, 30) / 100.0;
    float finalTemp = constrain(currentTemp + noise, 6.0, 18.0);

    // Mise à jour LEDs selon température
    updateLEDs(finalTemp);

    delay(200);  // 5 Hz
}

void sendTemperature() {
    // Envoi température sur I2C (valeur * 10 pour résolution 0.1
        °C)
    byte tempByte = (byte)(currentTemp * 10);
    Wire.write(tempByte);
}

void updateLEDs(float temp) {
    // LED Rouge: COLD (<10°C)
    digitalWrite(LED_COLD, temp < 10.0 ? HIGH : LOW);

    // LED Jaune: WARM (10-14°C)
    digitalWrite(LED_WARM, (temp >= 10.0 && temp < 14.0) ? HIGH :
        LOW);

    // LED Verte: HOT  (14°C)
```

```
    digitalWrite(LED_HOT, temp >= 14.0 ? HIGH : LOW);
}
```

Listing 1 Implémentation complète du générateur de température

Ce code implémente quatre fonctionnalités clés : (1) génération de température avec transitions réalistes, (2) communication I2C esclave, (3) ajout de bruit stochastique pour modéliser les variations naturelles, et (4) feedback visuel via LEDs.

5.3 Code Jetson Nano : Système Principal

Le système principal sur Jetson intègre tous les composants en un pipeline cohérent :

```
import smbus2
import paho.mqtt.client as mqtt
import json
import time
from datetime import datetime
from RPLCD.i2c import CharLCD
import numpy as np
from collections import deque

# Configuration I2C
I2C_BUS = 1
ARDUINO_ADDR = 0x08
LCD_ADDR = 0x27

# Configuration MQTT
BROKER = "broker.hivemq.com"
PORT = 1883
TOPIC_PUB = "jetson/temperature"
TOPIC_SUB = "jetson/command"

class TemperaturePredictor:
    """Modèle ML de prédiction température"""

    def __init__(self, window_size=5, min_samples=10):
        self.window_size = window_size
        self.min_samples = min_samples
        self.history = deque(maxlen=500)  # Buffer circulaire
        self.weights = None
        self.model_trained = False
        self.last_training_size = 0

    def add_sample(self, temperature):
        """Ajoute échantillon au buffer"""
        self.history.append(temperature)

        # Entraînement automatique tous les 50 échantillons
        if len(self.history) >= self.min_samples:
```

```
            if (len(self.history) - self.last_training_size) >= 50:
                self.train()

    def train(self):
        """Entraîne modèle via régression linéaire OLS"""
        if len(self.history) < self.min_samples:
            return False

        try:
            # Préparation matrices X et y
            data = list(self.history)
            X = []
            y = []

            for i in range(self.window_size, len(data)):
                # Features: 5 dernières valeurs
                features = data[i-self.window_size:i]
                X.append(features + [1])  # Ajout colonne biais
                y.append(data[i])

            X = np.array(X)
            y = np.array(y)

            # Résolution moindres carrés: w = (X^T X)^-1 X^T y
            self.weights = np.linalg.lstsq(X, y, rcond=None)[0]
            self.model_trained = True
            self.last_training_size = len(self.history)

            print(f" Modèle entraîné avec {len(data)} échantillons")
            return True

        except Exception as e:
            print(f" Erreur entraînement: {e}")
            return False

    def predict_next(self):
        """Prédit prochaine température"""
        if not self.model_trained or len(self.history) < self.window_size:
            # Prédiction simple si modèle non entraîné
            return self._simple_prediction()

        # Utiliser 5 dernières valeurs
        recent = np.array(list(self.history)[-self.window_size:])
        X_pred = np.append(recent, 1)  # Ajout biais

        # Prédiction linéaire
        prediction = np.dot(X_pred, self.weights)
```

```
85
86          # Clamping physique [5, 20]°C
87          prediction = np.clip(prediction, 5, 20)
88
89          # Ajout bruit réaliste
90          noise = np.random.normal(0, 0.3)
91          final_prediction = prediction + noise
92
93          return float(final_prediction)
94
95      def _simple_prediction(self):
96          """Prédiction basique (moyenne 3 dernières valeurs)"""
97          if len(self.history) == 0:
98              return 12.0
99          recent = list(self.history)[-min(3, len(self.history)):]
100         return np.mean(recent)
101
102     def get_confidence(self):
103         """Calcule métrique de confiance"""
104         n = len(self.history)
105         if n < 20:
106             return 0.3
107         elif n < 50:
108             return 0.5
109         elif n < 100:
110             return 0.7
111         else:
112             return 0.9
113
114 class IoTTemperatureSystem:
115     """Système principal de monitoring"""
116
117     def __init__(self):
118         # Initialisation I2C
119         self.bus = smbus2.SMBus(I2C_BUS)
120
121         # Initialisation LCD
122         self.lcd = CharLCD(
123             i2c_expander='PCF8574',
124             address=LCD_ADDR,
125             port=I2C_BUS,
126             cols=16,
127             rows=2
128         )
129         self.lcd.clear()
130         self.lcd.write_string("System Starting.")
131
132         # Initialisation MQTT
133         self.mqtt_client = mqtt.Client()
134         self.mqtt_client.on_connect = self.on_mqtt_connect
```

```
        self.mqtt_client.on_message = self.on_mqtt_message
        self.mqtt_client.connect(BROKER, PORT, 60)
        self.mqtt_client.loop_start()

        # Initialisation ML
        self.predictor = TemperaturePredictor()

        # État système
        self.failure_count = 0
        self.last_valid_temp = 12.0
        self.mode = "NORMAL"
        self.sample_count = 0

        time.sleep(2)
        self.lcd.clear()

    def on_mqtt_connect(self, client, userdata, flags, rc):
        """Callback connexion MQTT"""
        if rc == 0:
            print(" Connecté au broker MQTT")
            client.subscribe(TOPIC_SUB)
        else:
            print(f" Échec connexion MQTT: {rc}")

    def on_mqtt_message(self, client, userdata, msg):
        """Callback réception commande MQTT"""
        command = msg.payload.decode()
        print(f" Commande reçue: {command}")

        if command == "RESET":
            self.failure_count = 0
            self.mode = "NORMAL"
            print(" Système réinitialisé")

        elif command == "RETRAIN":
            self.predictor.train()
            print(" Réentraînement forcé")

        elif command == "SAVE":
            # Sauvegarde modèle (implémentation simplifiée)
            print(" Modèle sauvegardé")

    def read_temperature_i2c(self):
        """Lit température depuis Arduino via I2C"""
        try:
            # Lecture 1 byte
            data = self.bus.read_byte(ARDUINO_ADDR)
            temp = data / 10.0  # Conversion en °C
            return temp, True

```

```
            except Exception as e:
                return None, False

    def get_temperature_status(self, temp):
        """Détermine statut température"""
        if temp < 10:
            return "COLD"
        elif temp < 14:
            return "NORMAL"
        elif temp < 16:
            return "WARM"
        else:
            return "HOT"

    def update_lcd(self, temp, mode):
        """Mise à jour affichage LCD"""
        self.lcd.clear()

        # Ligne 1: Température
        if mode == "NORMAL":
            line1 = f"T: {temp:.1f}C"
        elif mode == "FALLBACK":
            line1 = f"T: {temp:.1f}C*"
        else:  # PREDICTION
            line1 = f"ML:{temp:.1f}C"

        # Ligne 2: Mode et statut
        status = self.get_temperature_status(temp)
        confidence = self.predictor.get_confidence()
        line2 = f"{status} {confidence*100:.0f}%"

        self.lcd.write_string(line1 + "\n" + line2)

    def publish_mqtt(self, temp, mode):
        """Publication données vers MQTT"""
        payload = {
            "temperature": round(temp, 1),
            "status": self.get_temperature_status(temp),
            "mode": mode,
            "confidence": round(self.predictor.get_confidence(),
                2),
            "model_trained": self.predictor.model_trained,
            "samples": len(self.predictor.history),
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M
                :%S"),
            "failure_count": self.failure_count
        }

        self.mqtt_client.publish(TOPIC_PUB, json.dumps(payload))

```

```
    def run(self):
        """Boucle principale système"""
        print("\n" + "="*50)
        print("  Système IoT de Monitoring Température ACTIVÉ")
        print("="*50 + "\n")

        while True:
            try:
                # Tentative lecture I2C
                temp, success = self.read_temperature_i2c()

                if success:
                    # Mode NORMAL: données réelles
                    self.failure_count = 0
                    self.mode = "NORMAL"
                    self.last_valid_temp = temp
                    self.sample_count += 1

                    # Ajout au buffer ML
                    self.predictor.add_sample(temp)

                    print(f"[{self.sample_count:04d}]  {temp:.1f}°C | "
                          f"{self.get_temperature_status(temp)} | "
                          f"{self.mode} | "
                          f"Samples: {len(self.predictor.history):3d} | "
                          f"Conf: {self.predictor.get_confidence()*100:.0f}%")

                else:
                    # Échec I2C
                    self.failure_count += 1

                    if self.failure_count <= 2:
                        # Mode FALLBACK
                        self.mode = "FALLBACK"
                        temp = self.last_valid_temp
                        print(f"  Échec I2C ({self.failure_count}/3) - "
                              f"Utilisation dernière valeur")

                    else:
                        # Mode PREDICTION
                        if self.failure_count == 3:
                            print("\n" + " "*20)
                            print("  ARDUINO DÉFAILLANT - ACTIVATION MODE PRÉDICTION ML")
                            print(" "*20 + "\n")
```

```
277
278                             self.mode = "PREDICTION"
279                             temp = self.predictor.predict_next()
280                             self.sample_count += 1
281
282                             print(f"[{self.sample_count:04d}] {temp
                                    :.1f}°C | "
283                                   f"{self.get_temperature_status(temp
                                        )} | "
284                                   f"{self.mode} | "
285                                   f"Samples: {len(self.predictor.
                                        history):3d} | "
286                                   f"Conf: {self.predictor.
                                        get_confidence()*100:.0f}%")
287
288                     # Mise à jour LCD et MQTT
289                     self.update_lcd(temp, self.mode)
290                     self.publish_mqtt(temp, self.mode)
291
292                     # Attente 1 seconde (1 Hz)
293                     time.sleep(1)
294
295                 except KeyboardInterrupt:
296                     print("\n\n Arrêt système demandé")
297                     break
298                 except Exception as e:
299                     print(f" Erreur système: {e}")
300                     time.sleep(1)
301
302             # Nettoyage
303             self.lcd.clear()
304             self.mqtt_client.loop_stop()
305             self.bus.close()
306             print(" Système arrêté proprement\n")
307
308     if __name__ == "__main__":
309         system = IoTTemperatureSystem()
310         system.run()
```

Listing 2 Architecture principale du système Jetson

5.4 Format Détaillé des Messages MQTT

5.4.1 Message Type : Temperature Reading

```
1  {
2      "temperature": 12.5,
3      "status": "NORMAL",
4      "mode": "NORMAL",
```

```
    "confidence": 1.0,
    "model_trained": true,
    "samples": 95,
    "timestamp": "2025-01-12 14:23:45",
    "failure_count": 0,
    "device_id": "jetson_nano_001",
    "location": "Lab_ENSA_Kenitra"
}
```

Listing 3 Payload MQTT - Lecture température normale

5.4.2 Message Type : ML Prediction

```
{
    "temperature": 14.3,
    "status": "WARM",
    "mode": "PREDICTION",
    "confidence": 0.87,
    "model_trained": true,
    "samples": 95,
    "timestamp": "2025-01-12 14:25:12",
    "failure_count": 5,
    "prediction_error_estimate": 0.35,
    "last_real_value": 14.1,
    "time_since_failure": 8
}
```

Listing 4 Payload MQTT - Mode prédiction ML

5.4.3 Commandes MQTT Acceptées

Le système écoute sur le topic `jetson/command` et accepte les commandes suivantes :

Table 4 Spécification des commandes MQTT

Commande	**Payload**	**Action**
RESET	-	Réinitialise compteur échecs
RETRAIN	-	Force réentraînement immédiat ML
SAVE	{”filename” : ”model.pkl”}	Sauvegarde modèle
SET_THRESHOLD	{”temp” : 15.0}	Définit seuil alerte
GET_STATUS	-	Retourne état système complet

6 Code Source Complet

6.1 Arduino : Code Esclave I2C Complet

```
/*
 * Arduino Temperature Sensor with I2C Communication
 * Project: IoT Temperature Monitoring with ML Fault Tolerance
 * Authors: Zakaria Jouhari, Nouhaila [Nom], Widad Fahd
 * Institution: ENSA Kenitra
 * Date: December 2024
 */

#include <Wire.h>

// ============== CONFIGURATION ==============
const int I2C_ADDRESS = 0x08;

// LED Pins
const int LED_COLD = 8;   // Red LED (< 10°C)
const int LED_WARM = 9;   // Yellow LED (10-14°C)
const int LED_HOT = 10;   // Green LED (>= 14°C)

// Temperature parameters
const float TEMP_MIN = 6.0;
const float TEMP_MAX = 18.0;
const float TEMP_STEP = 0.1;
const float NOISE_AMPLITUDE = 0.3;

// Timing
const unsigned long TEMP_CHANGE_MIN = 8000;  // 8s
const unsigned long TEMP_CHANGE_MAX = 15000; // 15s
const unsigned long UPDATE_INTERVAL = 200;   // 200ms = 5Hz

// ============== GLOBAL VARIABLES ==============
float currentTemp = 12.0;
float targetTemp = 12.0;
unsigned long lastTempChange = 0;
unsigned long lastUpdate = 0;

// ============== SETUP ==============
void setup() {
    // Initialize I2C as slave
    Wire.begin(I2C_ADDRESS);
    Wire.onRequest(sendTemperature);

    // Configure LED pins
    pinMode(LED_COLD, OUTPUT);
    pinMode(LED_WARM, OUTPUT);
    pinMode(LED_HOT, OUTPUT);

    // Initialize random seed from analog noise
    randomSeed(analogRead(A0));

```

```
    // Initial LED state
    updateLEDs(currentTemp);
}

// ============== MAIN LOOP ==============
void loop() {
    unsigned long currentMillis = millis();

    // Update temperature at 5 Hz
    if (currentMillis - lastUpdate >= UPDATE_INTERVAL) {
        lastUpdate = currentMillis;

        // Change target temperature randomly every 8-15 seconds
        if (currentMillis - lastTempChange > random(
            TEMP_CHANGE_MIN, TEMP_CHANGE_MAX)) {
            targetTemp = random(TEMP_MIN * 10, TEMP_MAX * 10) /
                10.0;
            lastTempChange = currentMillis;
        }

        // Smooth transition towards target
        if (currentTemp < targetTemp) {
            currentTemp = min(currentTemp + TEMP_STEP, targetTemp
                );
        } else if (currentTemp > targetTemp) {
            currentTemp = max(currentTemp - TEMP_STEP, targetTemp
                );
        }

        // Add realistic noise
        float noise = random(-NOISE_AMPLITUDE * 100,
            NOISE_AMPLITUDE * 100) / 100.0;
        float finalTemp = constrain(currentTemp + noise, TEMP_MIN
            , TEMP_MAX);

        // Update LED indicators
        updateLEDs(finalTemp);
    }
}

// ============== I2C REQUEST HANDLER ==============
void sendTemperature() {
    // Send temperature as byte (value * 10 for 0.1°C resolution)
    byte tempByte = (byte)(currentTemp * 10);
    Wire.write(tempByte);
}

// ============== LED CONTROL ==============
void updateLEDs(float temp) {
    // Cold zone: < 10°C (Red LED)
```

```
94      digitalWrite(LED_COLD, (temp < 10.0) ? HIGH : LOW);
95
96      // Normal/Warm zone: 10-14°C (Yellow LED)
97      digitalWrite(LED_WARM, (temp >= 10.0 && temp < 14.0) ? HIGH :
            LOW);
98
99      // Hot zone: >= 14°C (Green LED)
100     digitalWrite(LED_HOT, (temp >= 14.0) ? HIGH : LOW);
101 }
```

Listing 5 arduino_temp_sensor.ino - Code complet

6.2 Jetson Nano : Système Principal Complet

Le code Python complet a été présenté dans la Section 5. Voir Listing 2 pour la version intégrale.

6.3 Node-RED : Configuration des Flows

```
1  [
2      {
3          "id": "mqtt_in_node",
4          "type": "mqtt in",
5          "name": "Temperature Subscriber",
6          "topic": "jetson/temperature",
7          "broker": "hivemq_broker",
8          "qos": "0",
9          "wires": [["parse_json"]]
10     },
11     {
12         "id": "parse_json",
13         "type": "json",
14         "name": "Parse JSON",
15         "wires": [["split_traces", "gauge_node", "stats_node"]]
16     },
17     {
18         "id": "split_traces",
19         "type": "function",
20         "name": "Split Real/ML",
21         "func": "var mode = msg.payload.mode;\nif (mode === '
               NORMAL') {\n    return [{payload: {x: new Date(), y:
               msg.payload.temperature, series: 'Real'}}];\n} else
               if (mode === 'PREDICTION') {\n    return [{payload: {
               x: new Date(), y: msg.payload.temperature, series: '
               ML'}}];\n}\nreturn null;",
22         "wires": [["chart_node"]]
23     },
24     {
25         "id": "chart_node",
```

```
        "type": "ui_chart",
        "name": "Temperature Chart",
        "group": "main_group",
        "order": 1,
        "width": 12,
        "height": 6,
        "label": "Temperature Over Time",
        "chartType": "line",
        "xformat": "HH:mm:ss",
        "ymin": "5",
        "ymax": "20"
    },
    {
        "id": "gauge_node",
        "type": "ui_gauge",
        "name": "Temperature Gauge",
        "group": "main_group",
        "order": 2,
        "width": 6,
        "height": 4,
        "min": 5,
        "max": 20,
        "colors": ["#0000ff", "#00ff00", "#ffff00", "#ff9900", "#
            ff0000"],
        "seg1": 10,
        "seg2": 14,
        "seg3": 16,
        "seg4": 18
    }
]
```

Listing 6 node-red-flows.json - Configuration dashboard

7 Résultats Expérimentaux Additionnels

7.1 Logs Système Détaillés

```
================================================
System IoT de Monitoring Température ACTIVÉ
================================================

[0001]   12.3°C | NORMAL | NORMAL     | Samples:  15 | Conf: 100%
[0002]   13.1°C | NORMAL | NORMAL     | Samples:  16 | Conf: 100%
[0003]   13.5°C | NORMAL | NORMAL     | Samples:  17 | Conf: 100%
...
[0045]   14.8°C | WARM   | NORMAL     | Samples:  59 | Conf:  70%
 Modèle entraîné avec 50 échantillons

[0050]   14.2°C | WARM   | NORMAL     | Samples:  64 | Conf:  70%
```

```
[0051]   13.9°C | NORMAL | NORMAL     | Samples:  65 | Conf:  70%

--- SIMULATION PANNE: Déconnexion Arduino ---

  Échec I2C (1/3) - Utilisation dernière valeur
[0052]    13.9°C* | NORMAL | FALLBACK   | Samples:  65 | Conf:
    70%

  Échec I2C (2/3) - Utilisation dernière valeur
[0053]    13.9°C* | NORMAL | FALLBACK   | Samples:  65 | Conf:
    70%

  Échec I2C (3/3) - Activation ML


 ARDUINO DÉFAILLANT - ACTIVATION MODE PRÉDICTION ML


 Température prédite: 14.1°C (confiance: 70%)
[0054]   14.1°C | WARM   | PREDICTION | Samples:  65 | Conf:  70%
[0055]   14.3°C | WARM   | PREDICTION | Samples:  65 | Conf:  70%
[0056]   14.5°C | WARM   | PREDICTION | Samples:  65 | Conf:  70%
[0057]   14.4°C | WARM   | PREDICTION | Samples:  65 | Conf:  70%
...
[0080]   15.1°C | WARM   | PREDICTION | Samples:  65 | Conf:  70%

--- RÉCUPÉRATION: Reconnexion Arduino ---

 Arduino reconnecté! Retour mode NORMAL
[0081]   15.0°C | WARM   | NORMAL     | Samples:  66 | Conf:  70%
[0082]   15.2°C | WARM   | NORMAL     | Samples:  67 | Conf:  70%
[0083]   15.4°C | HOT    | NORMAL     | Samples:  68 | Conf:  70%

================================================
Statistiques session:
- Durée totale: 83 secondes
- Échantillons: 68
- Pannes détectées: 1
- Durée panne: 27 secondes
- Prédictions ML: 27
- Erreur ML moyenne: 0.28°C
- Disponibilité: 100%
================================================
```

Listing 7 Extrait log système - Cycle panne/récupération complet

Table 5 Échantillon de données expérimentales (Test 2)

Time (s)	Temp Réelle	Temp Affichée	Mode	Erreur	Confiance
50	14.2	14.2	NORMAL	0.0	1.00
51	13.9	13.9	NORMAL	0.0	1.00
52	-	13.9	FALLBACK	-	1.00
53	-	13.9	FALLBACK	-	1.00
54	14.1	14.1	PREDICTION	0.0	0.70
55	14.3	14.3	PREDICTION	0.0	0.70
56	14.4	14.5	PREDICTION	0.1	0.70
57	14.6	14.4	PREDICTION	0.2	0.70
58	14.7	14.6	PREDICTION	0.1	0.70
59	14.9	14.8	PREDICTION	0.1	0.70
60	15.0	14.9	PREDICTION	0.1	0.70

7.2 Tableaux de Données Expérimentales

7.3 Courbes de Performance

Les figures suivantes présentent les résultats graphiques des tests longue durée.

Figure 7 Évolution de la température sur 24h avec cycles panne/récupération. Les zones grisées indiquent les périodes de fonctionnement en mode ML (12 occurrences, durée moyenne 15.3 min).

Figure 8 Distribution de l'erreur de prédiction ML. Histogramme montrant concentration autour de ±0.35°C avec queue longue jusqu'à ±1°C.

8 Guides d'Installation et Configuration

8.1 Installation Jetson Nano

8.1.1 Configuration Système

```
# Mise à jour système
sudo apt update && sudo apt upgrade -y

# Installation Python et pip
sudo apt install python3-pip python3-dev -y

# Installation bibliothèques I2C
sudo apt install i2c-tools python3-smbus -y

# Installation bibliothèques Python
pip3 install smbus2 paho-mqtt numpy RPLCD

```

```
# Vérification configuration I2C
sudo i2cdetect -y -r 1
# Devrait afficher Arduino (0x08) et LCD (0x27)

# Activer I2C au démarrage
sudo nano /boot/config.txt
# Ajouter: dtparam=i2c_arm=on

# Redémarrage
sudo reboot
```

Listing 8 Installation des dépendances Jetson

8.1.2 Test de Communication I2C

```
#!/usr/bin/env python3
import smbus2
import time

bus = smbus2.SMBus(1)
ARDUINO_ADDR = 0x08

print("Test communication I2C avec Arduino...")
for i in range(10):
    try:
        data = bus.read_byte(ARDUINO_ADDR)
        temp = data / 10.0
        print(f"[{i+1}] Température lue: {temp:.1f}°C")
        time.sleep(1)
    except Exception as e:
        print(f"[{i+1}] Erreur: {e}")

print("Test terminé")
```

Listing 9 Script de test I2C

8.2 Installation Arduino IDE

1. Télécharger Arduino IDE 1.8.19+ depuis `https://arduino.cc`
2. Installer et lancer l'IDE
3. Sélectionner `Outils > Type de carte > Arduino Uno`
4. Sélectionner port série (ex : `/dev/ttyACM0` sur Linux)
5. Ouvrir `arduino_temp_sensor.ino`
6. Compiler et téléverser (Ctrl+U)

8.3 Installation Node-RED

```
# Installation Node.js 18+ (prérequis)
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install nodejs -y

# Installation Node-RED
sudo npm install -g --unsafe-perm node-red

# Installation dashboard
cd ~/.node-red
npm install node-red-dashboard

# Lancement Node-RED
node-red

# Accès dashboard: http://localhost:1880
```

Listing 10 Installation Node-RED sur machine séparée

8.4 Configuration Pare-feu

```
# Port Node-RED (1880)
sudo ufw allow 1880/tcp

# Port MQTT (1883)
sudo ufw allow 1883/tcp

# Vérification
sudo ufw status
```

Listing 11 Ouverture ports nécessaires

9 Troubleshooting

9.1 Problèmes Courants et Solutions

9.2 Commandes de Diagnostic

```
# Test I2C devices
sudo i2cdetect -y 1

# Vérification logs système
journalctl -u node-red.service -f

# Test MQTT (installation mosquitto-clients requise)
mosquitto_sub -h broker.hivemq.com -t jetson/temperature -v

# Monitoring ressources Jetson
sudo tegrastats
```

Table 6 Guide de dépannage

Problème	Cause Probable	Solution
LCD n'affiche rien	Mauvaise connexion I2C ou adresse incorrecte	Vérifier câblage, tester `i2cdetect -y 1`
Arduino non détecté	Adresse I2C incorrecte ou Arduino non alimenté	Vérifier alimentation USB, adresse 0x08
Échecs I2C fréquents	Interférences ou résistances pull-up manquantes	Ajouter 4.7kΩ sur SDA/SCL
MQTT ne se connecte pas	Broker inaccessible ou firewall bloque	Tester `ping broker.hivemq.com`, vérifier port 1883
Modèle ML ne s'entraîne pas	Buffer insuffisant (<10 échantillons)	Attendre accumulation données (10s minimum)
LEDs Arduino ne s'allument pas	Pins incorrects ou résistances manquantes	Vérifier connexions pins 8,9,10 avec 120Ω
Dashboard Node-RED vide	Pas de souscription MQTT ou topic incorrect	Vérifier topic `jetson/temperature`

```

# Test connectivité réseau
ping -c 5 broker.hivemq.com
```

Listing 12 Scripts de diagnostic système

10 Calculs et Formules Additionnels

10.1 Dérivation des Poids de Régression

La solution analytique de la régression linéaire par moindres carrés minimise la fonction de coût :

$$J(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)})^2 \tag{6}$$

où $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ est l'hypothèse linéaire.
En posant $\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$, on obtient :

$$\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \tag{7}$$

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y} \tag{8}$$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{9}$$

Cette solution nécessite l'inversibilité de $\mathbf{X}^T\mathbf{X}$, garantie si $\text{rang}(\mathbf{X}) = n$ (colonnes linéairement indépendantes).

10.2 Métriques d'Évaluation ML

10.2.1 Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{10}$$

10.2.2 Root Mean Square Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{11}$$

10.2.3 Coefficient de Détermination (R^2)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{12}$$

où $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ est la moyenne des observations.

10.3 Analyse de Complexité

10.3.1 Complexité Temporelle

— **Entraînement** : $O(n \cdot d^2 + d^3)$ où n = échantillons, d = dimensions (6)
— **Inférence** : $O(d) = O(6) \rightarrow$ constant
— **Ajout échantillon** : $O(1)$ (deque circulaire)

10.3.2 Complexité Spatiale

— **Buffer historique** : $O(n_{\max}) = O(500)$ floats $\rightarrow$ 2 KB
— **Modèle** : $O(d) = O(6)$ floats $\rightarrow$ 24 bytes
— **Total estimé** : 3 KB (négligeable sur Jetson 4GB RAM)