F.A.M: Fault-Aware Management of Energy in Smart Homes using Machine Learning Approaches

Submitted By:
Amna Shah (22K-4579)
Muhammad Raza Khan (22K-4355)
Falah Zainab (22K-4491)

Course: AI
Instructor: Shafique Ur Rehman
Submission Date: May 9, 2025

1 Executive Summary

1.1 Project Overview

This project develops F.A.M (Fault-Aware Management), an AI-driven energy management system for smart homes. Built with a Flask backend, React frontend, and MongoDB database, F.A.M integrates a Bayesian Ridge classifier with MinMaxScaler for energy prediction, a Decision Tree classifier for appliance state optimization, and a Constraint Satisfaction Problem (CSP) solver for fault correction. The system predicts energy usage, optimizes appliance states, and updates configurations in real-time, achieving up to 20% energy savings. It includes robust error handling and is designed for scalability and IoT integration.

2 Introduction

2.1 Background

Smart homes rely on diverse appliances, making static energy management inefficient. F.A.M introduces a dynamic framework that leverages machine learning and constraint-based optimization to manage energy effectively. Unlike traditional systems focusing solely on prediction or scheduling, F.A.M combines real-time prediction, fault detection, and optimization, making it a novel solution for modern smart homes.

2.2 Objectives of the Project

- Develop a hybrid AI model using Bayesian Ridge regression with MinMaxScaler for real-time energy prediction.
- Implement a Decision Tree classifier for appliance state optimization.
- Use a CSP solver to correct appliance states based on user-defined constraints.
- Enable user authentication and real-time state updates via a React frontend and MongoDB.
- Evaluate system performance for accuracy and energy savings.

3 System Description

3.1 Original System

Traditional smart home systems use static schedules or basic automation for appliances like TVs, ACs, fridges, ovens, lights, and fans. These systems lack real-time adaptability and fault detection, leading to energy inefficiency.

3.2 Innovations and Modifications

F.A.M introduces:

- A hybrid AI model combining Bayesian Ridge with MinMaxScaler, Decision Trees, and CSP for prediction, optimization, and fault correction.
- Real-time appliance state vector (e.g., [1, 0, 1, 1, 0, 1] for TV, Fridge, Oven, Fan ON).
- API endpoints: `/predict`, `/optimize`, `/update`, `/api/auth`.
- Environmental data integration (time, temperature, humidity, wind speed) from Open Mateo.
- React-based switch panel for six appliances with real-time kWh tracking.
- Constraint-based optimization with rules such as:
  - AC ON if temperature > 28řC, OFF if 26řC.
  - Oven ON if temperature < 26řC and time between 10 AM and 8 PM.
  - Fan ON if temperature > 25řC and humidity > 60%.
  - Light ON from 6 PM to 6 AM.
  - TV ON from 6 PM to 11 PM.
  - Fridge always ON.

4 AI Approach and Methodology

4.1 AI Techniques Used

- **Bayesian Ridge Regression with MinMaxScaler**: Predicts energy consumption (High/Low) and estimates kWh (e.g., 4.32 kWh). MinMaxScaler normalizes input features (temperature, humidity, hour) to improve model performance.
- **Decision Tree Classifier**: Optimizes appliance states (e.g., "Switch AC to energy-saving mode").
- **CSP Solver**: Resolves conflicting appliance states based on constraints like total energy limits or user preferences.

4.2 Algorithm and Heuristic Design

- **Bayesian Ridge with MinMaxScaler**: Uses probabilistic modeling with normalized features to handle uncertainty in environmental and appliance data. MinMaxScaler transforms inputs to a [0, 1] range for consistent model training.
- **Decision Tree**: Evaluates appliance states against constraints:
  - TV: Max 2 hours daily, energy cap 0.5 kWh.
  - AC: Energy-saving mode if temp < 25řC, max 3 kWh.
  - Fridge: Always ON, max 1.5 kWh.
  - Oven: Max 1 hour, 2 kWh.

  - Light: OFF if no motion detected, 0.2 kWh.
  - Fan: ON if humidity > 60%, 0.3 kWh.
  - Max 3 appliances ON; AC and Oven cannot both be ON (AC prioritized).

  When repetition occurs (e.g., AC re-evaluated), the optimizer aggregates prior decisions, checks constraint satisfaction, and uses a weighted heuristic (energy savings vs. user comfort) to finalize recommendations.

- **CSP Solver**: Models appliance states as variables with domains (ON/OFF, modes) and constraints. It iteratively resolves conflicts, falling back to rule-based defaults if unsolvable.

4.3 AI Performance Evaluation

The Bayesian classifier with MinMaxScaler achieved:

- Accuracy: 92% (52 TP + 40 TN / 100).
- Precision (High): 91.2% (52 / (52 + 5)).
- Recall (High): 94.5% (52 / (52 + 3)).
- F1 Score: 92.8% (2 Œ (91.2 Œ 94.5) / (91.2 + 94.5)).

The Decision Tree and CSP reduced energy use by 20% in test cases, with decisions made in <1 second.

5 System Mechanics and Rules

5.1 Modified System Rules

- Six appliances with binary states (ON/OFF) and modes (e.g., AC energy-saving).
- Environmental inputs (time, temp, humidity, wind speed) affect predictions.
- Constraints limit total kWh and appliance usage (e.g., max 3 appliances ON, AC and Oven mutually exclusive).

5.2 Turn-based Mechanics

- User logs in via `/api/auth`.
- Toggles appliance switches on the dashboard.
- Saves states to MongoDB.
- Views PowerCast graphs (energy vs. environment).
- Runs optimizer for recommendations.

5.3 Winning Conditions

The system aims to minimize energy use while satisfying constraints, achieving optimal appliance configurations.

6 Implementation and Development

6.1 Development Process

- Designed Flask backend with API endpoints.
- Built React frontend with switch panel and PowerCast graphs.
- Integrated MongoDB for user and state storage.
- Trained Bayesian Ridge (with MinMaxScaler) and Decision Tree models using scikit-learn.
- Implemented CSP solver with Pythons `constraint` library.

6.2 Programming Languages and Tools

- **Programming Languages**: Python, JavaScript.
- **Libraries**: Flask, React, scikit-learn, pymongo, constraint, pandas.
- **Tools**: MongoDB Atlas, Open Mateo API, GitHub.

6.3 Challenges Encountered

- **CSP Complexity**: Mitigated by limiting appliances to six and adding rule-based fallbacks.
- **Noisy Data**: Added preprocessing checks and user correction prompts.
- **Model Loading**: Implemented retry mechanisms and logging.

7 Team Contributions

- **Amna Shah**: Developed Bayesian Ridge with MinMaxScaler and Decision Tree models, designed optimizer logic.
- **Muhammad Raza Khan**: Built Flask backend, integrated MongoDB, and implemented API endpoints.
- **Falah Zainab**: Designed React frontend, PowerCast graphs, and conducted performance testing.

8 Results and Discussion

8.1 AI Performance

- Bayesian classifier with MinMaxScaler: 92% accuracy, 0.5-second prediction time.
- Decision Tree + CSP: 20% energy savings, 1-second optimization time.
- System handled edge cases (e.g., all appliances OFF) with clear responses.

9 References

- Smart Energy Systems: `https://www.mdpi.com/1996-1073/11/12/3494`
- Confusion Matrix: `https://www.geeksforgeeks.org/confusion-matrix-machine-learning/`
- Bayesian Ridge: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html`
- MinMaxScaler: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`
- Scikit-learn: `https://scikit-learn.org`
- Flask: `https://flask.palletsprojects.com`
- React: `https://reactjs.org`