AN INTELLIGENT ELECTRICITY MANAGEMENT UNIT: AI-DRIVEN POWER FORECASTING AND PERSONALIZED CONSUMPTION INSIGHTS WITH APPLICATION INTEGRATION

Group Report

B.Sc. (Hons) Degree in Information Technology Specializing in Information Technology

Department of Information Technology
Sri Lanka Institute of Information Technology
Sri Lanka

August 2025

AN INTELLIGENT ELECTRICITY MANAGEMENT UNIT: AI-DRIVEN POWER FORECASTING AND PERSONALIZED CONSUMPTION INSIGHTS WITH APPLICATION INTEGRATION

Group Report

The dissertation was submitted in partial fulfilment of the requirements for the Bachelor of Science (Hons) in Information Technology specializing in Information Technology

Department of Information Technology
Sri Lanka Institute of Information Technology
Sri Lanka

August 2025

DECLARATION

"I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)."

Name	Student ID	Signature
Pivithuru N.H.A.S.	IT21389160	
Balasuriya B.L.I.S.	IT21803666	
Siriwardhana S.M.D.S.	IT21813948	
Welikalage R.Y.W.	IT21808166	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor
(Ms. Dinithi Pandithage)

Date: 08/29/2025

ABSTRACT

This research introduces a smart IoT (Internet of Things) system for enhancing household energy efficiency by integrating real-time electricity monitoring, predictive analytics, and AI (Artificial Intelligence)-driven personalized guidance. Traditional home energy management often lacks a level of detail, leading to uninformed usage patterns and increased energy waste. The proposed system consists of IoT-based device tracking, time-series forecasting, and generative AI to deliver proactive and intelligent electricity management experience. The IoT-enabled unit monitors individual appliance consumption and implements threshold-based control mechanisms to prevent overuse. The Machine learning algorithm analyzes historical usage data to forecast future electricity demand, supporting preemptive consumption planning. A fine-tuned Large Language Model (LLM) generates personalized energy-saving recommendations based on the overall electricity usage data collected from the system. Power monitoring portal enhances user interaction through real-time energy movements and feedback mechanisms. This study contributes to sustainable living by promoting energy-conscious behavior, reducing household energy expenditure, and aligning domestic energy usage through a data-driven solution.

Keywords - IoT, real-time monitoring, threshold-based control, time-series forecasting, LLM-based recommendations, energy efficiency.

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to all those who supported and guided us throughout the course of this research project.

First and foremost, we extend our heartfelt appreciation to our supervisor, Ms. Dinithi Pandithage, and our co-supervisor, Mr. Ashvinda Iddamalgoda, for their invaluable guidance, constructive feedback, and continuous encouragement at every stage of this research. Their expertise and insights were instrumental in shaping both the direction and the successful completion of this study.

We are also sincerely thankful to the academic staff of the Department of Information Technology, SLIIT, for providing the knowledge foundation and resources necessary to carry out this work. Special appreciation is extended to our colleagues and peers for their thoughtful discussions, suggestions, and collaboration, which greatly contributed to refining many aspects of the project.

We owe profound gratitude to our families for their unwavering support, patience, and encouragement throughout this journey. Their belief in us gave us the strength and determination to stay focused and dedicated.

Finally, we would like to acknowledge everyone who, in one way or another, directly or indirectly contributed to the successful completion of this project. This achievement would not have been possible without their collective support and cooperation.

Table of Contents

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	DESCRIPTION
IoT	Internet of Things
ML	Machine Learning
LLM	Large Language Model
HEM	Home Energy Management

INTRODUCTION

Background and Literature Survey

The rapid increase in global energy demand, escalating electricity costs, and growing environmental concerns have created an urgent need for innovative and intelligent energy management solutions. The economic challenges following the COVID-19 pandemic have further intensified the necessity for cost-effective approaches to monitor and control electricity usage in both residential and commercial settings. Traditional energy management systems have proven inadequate in addressing these modern challenges, primarily due to their focus on aggregate energy consumption metrics without providing granular, device-level insights necessary for effective optimization.

Limitations of Current Energy Management Systems

Existing Home Energy Management Systems (HEMS) suffer from several critical limitations that hinder their effectiveness in modern energy management scenarios. These systems typically focus on overall household energy consumption, failing to provide device-level monitoring and control capabilities that are essential for identifying high-energy-consuming appliances and implementing targeted corrective actions. This lack of granularity makes it difficult for users to make informed decisions about optimizing their electricity consumption, ultimately leading to higher bills and increased environmental impact.

Furthermore, current solutions present significant challenges for rental property owners and shared accommodation settings. The inability to track energy consumption per tenant or allocate energy usage based on predefined budgets creates inefficiencies and potential financial discrepancies. Existing systems fail to provide mechanisms for tenant-specific energy allocation, resulting in uncontrolled electricity consumption and disputes between landlords and tenants. This limitation is particularly problematic in the growing rental market where transparent and fair energy billing is crucial for maintaining positive landlord-tenant relationships.

Technological Opportunities and Advancements

The Internet of Things (IoT) offers a promising approach to address these challenges by enabling real-time monitoring, control, and automation of energy usage at the device level. Research in this domain has explored IoT-based systems for real-time energy monitoring, which enable device-level data collection and pave the way for more precise management of energy usage. However, existing IoT-based solutions often lack advanced functionalities such as threshold-based energy management, automated power control, and emergency override features. Additionally, the absence of scalable cloud integration further limits their applicability for modern households requiring reliable, flexible, and user-centric energy solutions.

Parallel advancements in artificial intelligence, particularly in machine learning and generative AI, have introduced new opportunities for personalization and optimization in energy management. Predictive modeling techniques, including machine learning algorithms such as neural networks and decision trees, have been applied to energy consumption forecasting to improve accuracy and identify patterns in energy usage. Large language models (LLMs), such as GPT-Neo, have demonstrated remarkable capabilities in delivering context-aware and personalized suggestions across diverse domains, including healthcare, education, and e-commerce. These advancements hold immense potential for revolutionizing energy management by leveraging real-time

data to offer dynamic, user-specific recommendations that encourage energy-efficient practices.

Integration Challenges and Research Gaps

Despite significant technological advancements, several critical research gaps remain in the field of intelligent energy management. Current systems typically fail to integrate intelligent feedback mechanisms that provide actionable recommendations tailored to the unique behaviors of individual users. While IoT-based monitoring systems exist, they often operate in isolation without incorporating predictive analytics or AI-driven personalized insights.

The integration of renewable energy sources, such as solar and wind power, with smart home systems presents another significant challenge. Although studies have highlighted the potential of integrating renewable sources to reduce reliance on the main grid and lower energy costs, the seamless integration of these renewable sources with grid power remains problematic, requiring advanced energy distribution and storage mechanisms. Current solutions struggle to achieve optimal balance between renewable energy utilization and grid power consumption, often resulting in inefficient energy management and missed opportunities for cost reduction.

Research Opportunities and Requirements

Recent studies in personalized home energy management systems highlight the importance of integrating IoT, AI, and behavioral insights to promote sustainable energy use and optimize household energy consumption. However, a significant research gap remains in combining device-level energy monitoring, predictive analytics, and AI-driven personalized insights into a single, unified platform. There is a growing demand for user-friendly, customizable energy management solutions that

cater to individual household needs while ensuring scalability for larger applications such as rental properties and shared accommodations.

The literature indicates that successful energy management systems must address several key requirements: real-time device-level monitoring, intelligent automation capabilities, predictive analytics for proactive energy management, personalized recommendations based on user behavior patterns, and seamless integration with both renewable energy sources and cloud-based platforms for scalability and reliability.

Research gap

Despite significant advancements in smart home energy management systems, a comprehensive analysis of existing literature reveals several critical gaps that limit the effectiveness and practical applicability of current solutions. These gaps span across device-level monitoring, predictive analytics, user personalization, and system integration, creating opportunities for innovative research contributions.

Device-Level Monitoring and Control Limitations

A fundamental limitation in existing energy management systems is the lack of granular, device-level monitoring and control capabilities. While studies such as Prathyusha et al. [6] propose IoT-based smart home systems, they lack appliance-level sensing or actuation mechanisms. Similarly, Bharadwaj et al. [7] offers device-level monitoring but does not support automated relay control, limiting active power regulation capabilities. This absence of device-specific control prevents users from implementing targeted energy optimization strategies and identifying high-consumption appliances effectively.

Current systems primarily focus on aggregate energy consumption without providing the granular insights necessary for effective energy management. Jayaprakash et al.

[1] present billing alerts but lack mechanisms for automatic control or user-specific limits, while Banu Priya et al. [2] emphasizes remote scheduling but excludes dynamic shutdown based on real-time consumption patterns. These limitations prevent users from optimizing energy consumption at the appliance level and taking corrective actions based on individual device performance.

Threshold-Based Energy Management and Automation Gaps

Existing literature reveals a significant gap in threshold-based energy management systems that can automatically enforce user-defined energy limits. Current solutions lack features that allow users to define monthly energy usage thresholds for individual devices and implement automated enforcement mechanisms. This limitation is evident in studies by Loganayagi et al. [3] and Ebrahimi et al. [5], which discuss optimization frameworks but do not address device-level firmware or cloud-based control mechanisms that can dynamically manage energy consumption based on predefined thresholds.

The absence of emergency override features in current systems further compounds this limitation, as users cannot address urgent energy needs without disrupting overall energy management goals. This gap highlights the need for flexible, user-centric energy management solutions that balance automated control with user autonomy.

Predictive Analytics and Time Series Forecasting Deficiencies

A critical gap exists in the integration of advanced predictive modeling capabilities within smart home energy management systems. Deepa et al. [4] explore grid-level analytics but overlook device-specific forecasting, while Prathyusha et al. [6] offer IoT-based monitoring without incorporating time series models like XGBOOST. This limitation prevents systems from providing accurate predictions of future energy consumption patterns and implementing proactive energy optimization strategies.

Studies such as Deb et al. [9] investigated predictive approaches but failed to offer comprehensive user engagement mechanisms. The absence of advanced time series forecasting models, particularly XGBOOST or similar statistical approaches, hinders systems' ability to predict energy consumption patterns accurately and provide actionable insights for energy optimization.

Personalization and User Interaction Limitations

Current energy management systems demonstrate significant limitations in providing personalized, context-aware recommendations tailored to individual user behaviors and preferences. While Kumar et al. [13] deployed generative AI to make recommendations, their approach did not integrate real-time IoT data streams. Similarly, Bai et al. [11] concentrated on AI and behavioral insights without implementing effective feedback mechanisms that can adapt to user preferences over time.

The lack of real-time personalization is evident in solutions like cloud-IoT-based HEMS, which focus on real-time energy monitoring but do not provide actionable, personalized recommendations tailored to individual users' consumption behaviors. Brown et al. [10] highlighted the personalization possibilities of large language models (LLMs), but existing systems fail to leverage these capabilities effectively for energy management applications.

Integration and Scalability Challenges

Existing literature reveals persistent challenges in achieving seamless integration between IoT devices, predictive analytics, and user interaction platforms. Singh et al. [12] suggested an IoT-based HEMS but lacked predictive modeling capabilities, while Condon et al. [14] initially set up a cloud-based monitoring system but AI-driven insights remained limited. This fragmentation prevents the development of

comprehensive solutions that can leverage the full potential of integrated IoT and AI technologies.

Scalability challenges represent another significant gap, as current solutions may not scale efficiently to larger households or buildings with diverse energy demands and multiple IoT devices. Many proposed systems, such as those by Prathyusha et al. [6], remain primarily confined to simulations, lacking real-world deployment validation and scalability assessment.

Real-World Implementation and Practical Deployment Gaps

A substantial gap exists between theoretical research and practical implementation of smart home energy management systems. Most studies focus on simulations or theoretical models, leaving challenges in real-world deployment, such as scalability and secure integration with cloud platforms, unaddressed. This limitation is evident in the work by Prathyusha et al. [6], which highlighted an IoT-enabled smart home simulation using Arduino underscoring both scalability and practical deployment challenges.

The high energy consumption of smart devices themselves presents another practical challenge, as IoT-enabled smart home systems often require additional power to maintain connectivity and automation, potentially offsetting energy savings. Current research inadequately addresses this paradox and fails to provide energy-efficient IoT solutions.

Security, Privacy, and Integration Concerns

Data transmission and storage in smart home systems pose potential security and privacy risks that current solutions do not comprehensively address. The integration of smart home energy management systems with traditional power grids remains

challenging, requiring further development to ensure efficiency and reliability. These concerns limit the practical adoption of advanced energy management systems and highlight the need for secure, privacy-preserving solutions.

Anomaly Detection and Fault Prevention Limitations

Existing systems demonstrate insufficient capabilities in anomaly detection and fault prevention mechanisms. Studies like the Machine-Learning-Based Home Energy Management Framework by Ebrahimi et al. [5] highlight optimization through user feedback but fail to include robust anomaly detection methods to identify unusual energy consumption patterns in real-time. This limitation prevents proactive maintenance and energy waste prevention, reducing overall system effectiveness.

The absence of device-specific fault detection and prediction capabilities in current systems, such as those described in IoT-Infused Residences by Bharadwaj et al. [7], limits their ability to prevent appliance malfunctions proactively and optimize long-term energy performance.

Research Contribution Opportunities

These identified gaps collectively highlight the need for a comprehensive, integrated approach to smart home energy management that combines device-level monitoring, threshold-based automated control, advanced predictive analytics, personalized user interaction, and secure cloud integration. The proposed research addresses these limitations by developing a unified platform that leverages IoT technologies, machine learning algorithms, and generative AI to provide practical, scalable, and user-centric energy management solutions.

Research Problem

The escalating global energy crisis, compounded by post-COVID-19 economic challenges and rising electricity costs, has created an urgent need for comprehensive, intelligent energy management solutions in residential settings. Despite significant technological advancements in IoT, artificial intelligence, and cloud computing, existing home energy management systems fail to address the multifaceted challenges of modern energy consumption, particularly in diverse living arrangements ranging from individual households to shared accommodations and rental properties.

Problem Statement

Current energy management systems suffer from fundamental limitations that prevent effective optimization of household electricity consumption: they lack granular device-level monitoring and control capabilities, fail to provide personalized and predictive insights, cannot accommodate diverse billing and allocation requirements for shared living spaces, and do not integrate renewable energy sources effectively. These deficiencies result in significant energy wastage, increased costs, billing disputes in rental properties, and limited adoption of sustainable energy practices.

Primary Research Question: How can an integrated IoT-based intelligent energy management system be developed that combines real-time device-level monitoring, predictive analytics, personalized AI-driven recommendations, and flexible energy allocation mechanisms to optimize electricity consumption, reduce costs, and promote sustainable energy practices across diverse residential settings?

Key Dimensions of the Research Problem

1. Granular Monitoring and Control Limitations

- Existing systems provide only aggregated energy consumption data, making it impossible for users to identify and manage high-energy-consuming appliances effectively. The absence of device-specific monitoring capabilities prevents targeted energy optimization strategies and limits users' ability to make informed decisions about individual appliance usage. This lack of granularity is particularly problematic in shared living spaces where transparent energy allocation is essential for fair billing and tenant satisfaction.

2. Predictive Analytics and Proactive Management Deficiencies

- Current energy management solutions operate primarily in reactive mode, providing historical data without predictive capabilities that enable proactive energy planning. The absence of advanced time series forecasting models, such as XGBOOST, prevents users from anticipating future energy consumption patterns and planning their usage accordingly. This limitation is compounded by the lack of anomaly detection mechanisms that could identify faulty appliances and prevent excessive energy waste.

3. Personalization and User Engagement Gaps

- Existing systems fail to provide personalized, context-aware recommendations tailored to individual user behaviors and preferences. The absence of intelligent feedback mechanisms that adapt to user patterns over time limits the effectiveness of energy-saving initiatives. Users require actionable insights that

consider external factors such as time-of-use tariffs, weather conditions, and household occupancy patterns to make optimal energy consumption decisions.

4. Threshold-Based Management and Automation Limitations

- Current solutions lack sophisticated threshold-based energy management capabilities that allow users to define and automatically enforce energy usage limits for individual devices. The absence of automated control mechanisms that can dynamically manage power supply based on predefined thresholds reduces the practical effectiveness of energy management systems. This limitation is particularly critical for rental properties where energy allocation based on tenant payments requires automated enforcement mechanisms.

5. Multi-Tenant Energy Allocation Challenges

- The growing rental property market faces significant challenges in managing energy consumption based on tenant-specific payments and usage patterns. Existing systems do not provide mechanisms for fair energy allocation, leading to disputes between landlords and tenants and resulting in financial losses due to uncontrolled consumption. The lack of transparency in individual device energy usage makes it difficult to implement equitable billing systems in shared accommodations.

6. Renewable Energy Integration Complexities

- Despite increasing adoption of renewable energy sources, existing energy management systems lack sophisticated mechanisms for optimizing the balance between grid power and renewable energy utilization. The absence of

intelligent energy allocation algorithms that can dynamically switch between energy sources based on availability, cost, and consumption patterns limits the potential benefits of renewable energy integration in residential settings.

7. Real-World Implementation and Scalability Issues

- Most existing research focuses on theoretical models and simulations, with limited attention to real-world deployment challenges such as scalability, security, and seamless cloud integration. The lack of robust, scalable solutions that can adapt to diverse household configurations and energy requirements hinders practical adoption of advanced energy management technologies.

Significance and Impact of the Research Problem

Addressing these interconnected challenges is critical for several reasons. First, the economic impact of inefficient energy management is substantial, with households facing increasing financial strain due to rising electricity costs. Second, the environmental implications of energy waste contribute significantly to carbon emissions and climate change, making efficient energy management essential for sustainability goals. Third, the social implications of unfair energy billing in shared accommodations create conflicts that impact quality of life and housing satisfaction.

The proposed research addresses these challenges by developing a comprehensive IoT-based intelligent energy management system that integrates multiple advanced technologies into a cohesive solution. By combining real-time device-level monitoring, predictive analytics using machine learning models, personalized AI-driven recommendations through generative AI, and flexible energy allocation mechanisms, the system will provide unprecedented granularity and control over household energy consumption.

The system's threshold-based automated control capabilities will enable proactive energy management, while its multi-tenant support will address the specific needs of rental properties and shared accommodations. The integration of renewable energy optimization algorithms will promote sustainable energy practices, and the secure cloud-based architecture will ensure scalability and reliability.

This research contribution will fill critical gaps in existing energy management solutions by providing a practical, user-centric platform that empowers individuals and property managers to optimize energy consumption, reduce costs, and adopt sustainable practices. The system's comprehensive approach to addressing device-level monitoring, predictive analytics, personalized recommendations, and flexible allocation mechanisms represents a significant advancement in smart home energy management technology.

By bridging the gap between theoretical research and practical implementation, this work will provide a foundation for next-generation energy management systems that can adapt to diverse residential settings while promoting energy efficiency, cost reduction, and environmental sustainability. The research outcomes will contribute to global energy conservation efforts and support the transition toward more sustainable residential energy consumption patterns.

Research Objective

Research Contribution and Objectives

This research aims to bridge the identified gaps by developing a comprehensive IoT-based energy management system that integrates multiple advanced technologies into a cohesive solution. The proposed system addresses the limitations of existing approaches by combining device-level energy monitoring with threshold-based automated control, leveraging cloud platforms such as AWS IoT Core and DynamoDB for secure, scalable data storage and communication.

The research contributes to the field by introducing innovative features such as tenant-specific energy allocation, real-time device-level control, machine learning-powered time-series forecasting, and generative AI-driven personalized recommendations. By integrating IoT sensor networks, advanced machine learning frameworks, and user-centered design principles, the proposed solution empowers users to optimize energy consumption, reduce costs, and contribute to global sustainability goals through informed and energy-conscious decision-making.

The system's holistic approach aligns with global sustainability objectives by fostering energy-conscious behaviors while leveraging cutting-edge technologies. The user-friendly web application serves as a central interface, displaying detailed energy analytics, predictive insights, and personalized recommendations through an intuitive design that emphasizes accessibility and ease of use. This comprehensive solution addresses critical environmental and economic concerns while providing practical benefits for both individual households and rental property management scenarios.

METHODOLOGY

Methodology

System Architecture and Integration Framework

The proposed intelligent energy management system employs a comprehensive multi-tier architecture that seamlessly integrates Internet of Things (IoT) hardware, cloud computing infrastructure, machine learning algorithms, and artificial intelligence to deliver a robust, scalable energy management solution. This methodology encompasses both hardware and software components, ensuring real-time data collection, processing, analysis, and user interaction through a unified platform.

IoT Hardware Implementation and Smart Plug Development

The foundation of the system rests on custom-designed IoT-enabled smart plugs that serve as the primary data collection and control interfaces for individual household appliances. Each smart plug incorporates advanced sensor technologies, including ACS712 current sensors and ZMPT101B voltage sensors, which enable precise measurement of electrical parameters such as current consumption, voltage levels, and power factors. These sensors are strategically integrated with microcontroller units based on ESP32 or Arduino platforms, providing local processing capabilities and Wi-Fi connectivity for seamless communication with cloud services.

The smart plugs are engineered with onboard relay systems that enable automated power control based on predefined energy consumption thresholds. This hardware-level control mechanism ensures immediate response to energy management commands while maintaining user safety through proper electrical isolation and surge protection. The embedded firmware incorporates intelligent logic that can process real-time energy consumption data, compare it against user-defined limits, and execute automatic power disconnection when thresholds are exceeded.

To ensure robust communication and reliability, each smart plug is equipped with local data buffering capabilities that temporarily store consumption data during network interruptions, preventing data loss and maintaining system continuity. The hardware design also incorporates status indication systems through LED displays and audible alerts that provide immediate feedback to users about device status and energy consumption levels.

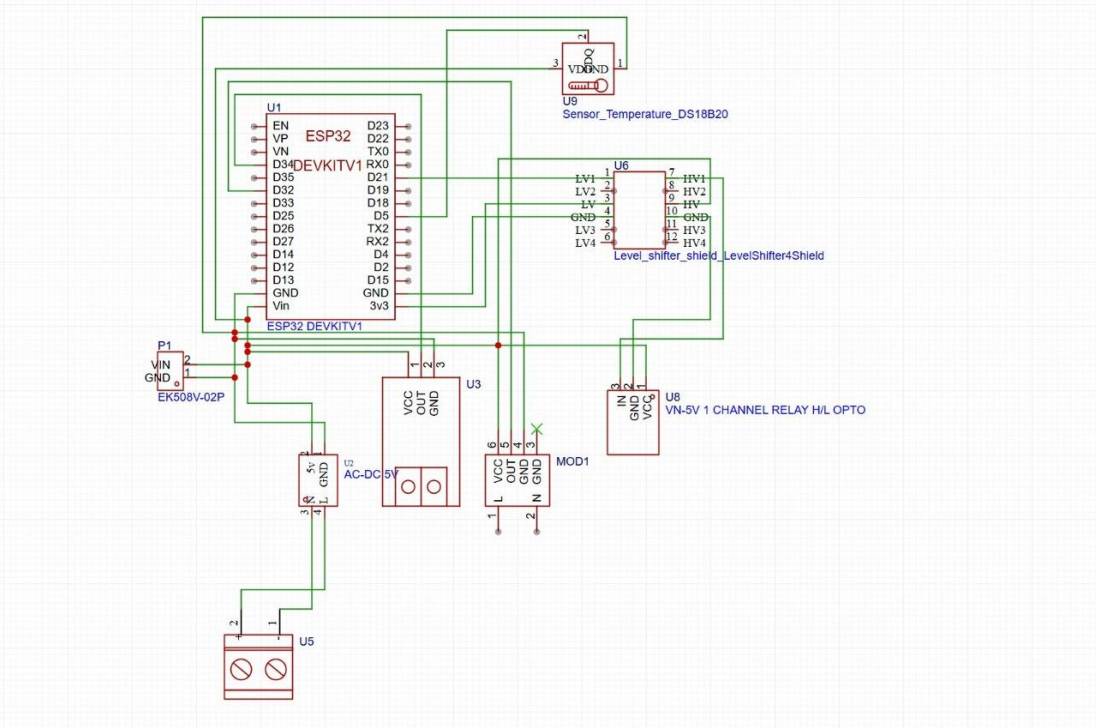

Figure 1 - Schematic Diagram

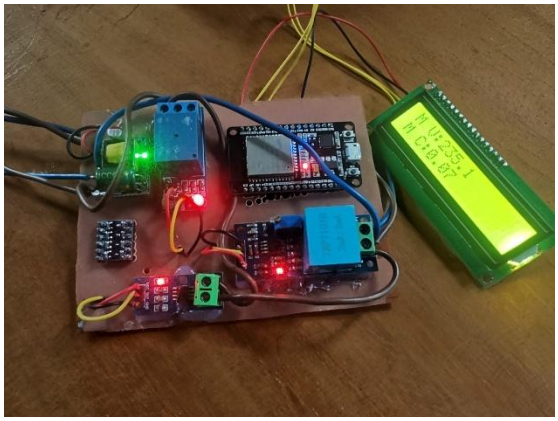

Figure 2 - Hardware Component

Cloud Infrastructure and AWS IoT Core Integration

The system leverages Amazon Web Services (AWS) IoT Core as the primary cloud infrastructure for secure, scalable communication between IoT devices and backend services. This cloud-based approach enables real-time data transmission, remote device management, and scalable data storage while ensuring enterprise-level security through industry-standard encryption protocols and certificate-based authentication mechanisms.

AWS IoT Core serves as the central communication hub, facilitating bidirectional data exchange between smart plugs and web applications through MQTT (Message Queuing Telemetry Transport) protocols optimized for IoT applications. The cloud infrastructure incorporates AWS DynamoDB for scalable, high-performance storage of historical energy consumption data, enabling efficient data retrieval for analytics and reporting purposes.

The integration includes automated data pipeline systems that process incoming sensor data, perform data validation and cleansing operations, and store structured information for subsequent analysis. Cloud-based triggers and Lambda functions are employed to enable real-time processing of energy consumption events, threshold violations, and automated control commands.

Web Application Development Framework

The user interface components are developed using modern web technologies, specifically React for frontend development and Laravel for backend API services. React provides responsive, interactive user experience through component-based architecture that enables real-time data updates and seamless user interactions across desktop, tablet, and mobile platforms. The React implementation incorporates state management systems that maintain synchronization between user interface elements and real-time energy data streams.

Laravel serves as the robust backend framework, providing RESTful API services that facilitate secure communication between frontend applications and cloud-based data services. The Laravel implementation includes comprehensive authentication systems, role-based access control for multi-tenant environments, and database abstraction layers that ensure scalable data operations.

Figure 3 - Web Application

Data Visualization and Analytics Implementation

Advanced data visualization capabilities are implemented through Chart.js and Recharts libraries, providing interactive, real-time graphical representations of energy consumption patterns, trends, and forecasting results. Chart.js enables the creation of dynamic charts and graphs that update in real-time as new energy consumption data becomes available, while Recharts provides specialized visualization components optimized for React applications.

The visualization framework incorporates multiple chart types, including line charts for historical consumption trends, bar charts for device comparison analysis, pie charts for energy distribution visualization, and specialized forecasting charts that display predicted consumption patterns alongside actual usage data. Interactive features enable

users to drill down into specific time periods, compare multiple devices, and explore detailed consumption analytics.

Dashboard implementations provide comprehensive overviews of household energy consumption through customizable widget systems that allow users to configure their preferred metrics, time ranges, and visualization preferences. The dashboards incorporate responsive design principles that automatically adapt to different screen sizes and device orientations.

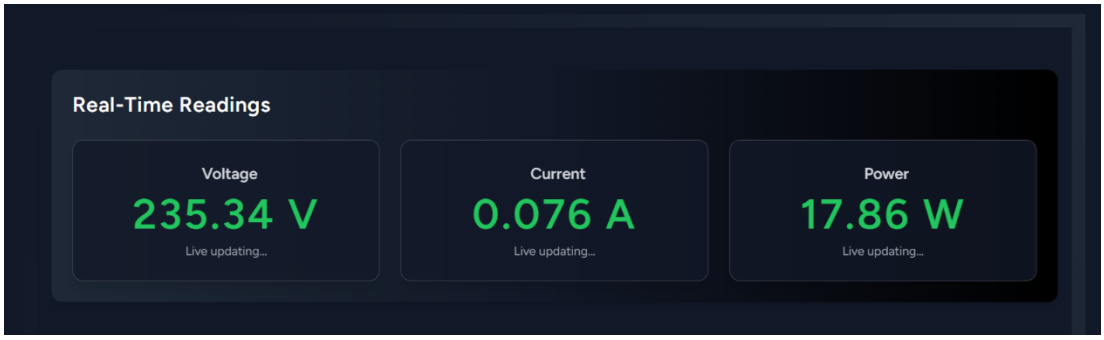

Figure 4 - Real Time Data Visualization

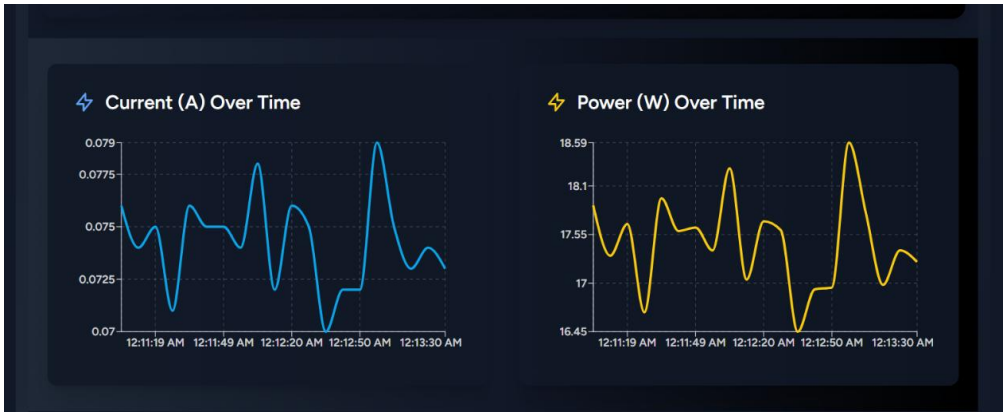

Figure 5 - Data Visualization Using Charts

Accessibility Features

Recognizing the importance of inclusivity, the system incorporates color-blindness accessibility support. Users who are affected by Protanopia or Deuteranopia can set their color-blindness preference in their profile settings. Once selected, the React frontend dynamically adjusts all visualizations and UI elements to color-blind-friendly themes, ensuring that data remains clear and interpretable regardless of visual limitations. This feature is particularly valuable in charts, gauges, and threshold indicators, where color is often the primary mode of conveying information.

A detailed comparison of color-blind themes is provided in Appendix B.

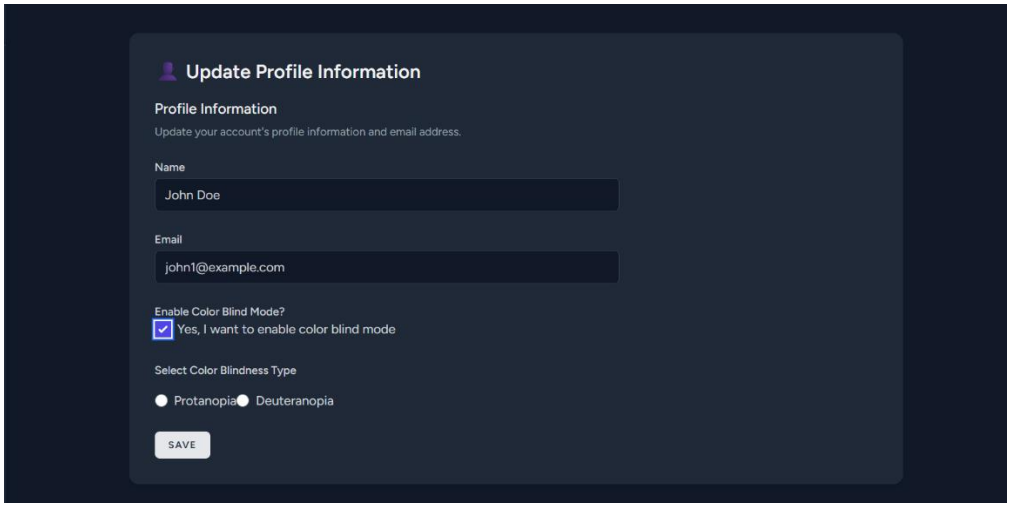

Figure 6: Color Blind Preference Update Section In Profile Page

Machine Learning and Predictive Analytics Implementation

The system's architecture is built on a "specialist model" methodology, where a dedicated machine learning model is trained for each unique appliance to capture its specific "energy fingerprint." The process begins with high-frequency sensor data collection from IoT devices, identified by their MAC addresses. This data is streamed to AWS DynamoDB for real-time access and archived in an S3 data lake for long-term

storage. A robust data processing pipeline then handles the crucial initial steps of filtering this data by device, performing timestamp conversions, and cleaning it in preparation for model training.

The core of the system is the Model Training Pipeline, developed in Python using libraries like XGBoost and Scikit-learn. The key to the model's accuracy is a sophisticated Feature Engineering step, which creates new features from the time-series data, including time-based patterns (hour, day of week), recent history (lag features), and trends (rolling window statistics). An XGBoost Regressor is then trained on this feature-rich historical dataset. The final, trained "specialist" model is then serialized and stored in a dedicated S3 Model Store, ready for deployment.

For live predictions, a lightweight and containerized Flask API serves as the system's brain. When a user requests a forecast, the API loads the appropriate specialist model from the S3 store, fetches the latest live data from DynamoDB, and runs the same real-time feature engineering pipeline to prepare the data. It then uses a recursive forecasting strategy to generate a multi-hour prediction. To ensure scalability, an automated retraining pipeline, managed by a scheduler like AWS Lambda, handles the "cold start" problem by automatically training and deploying models for new devices once they have collected sufficient historical data.

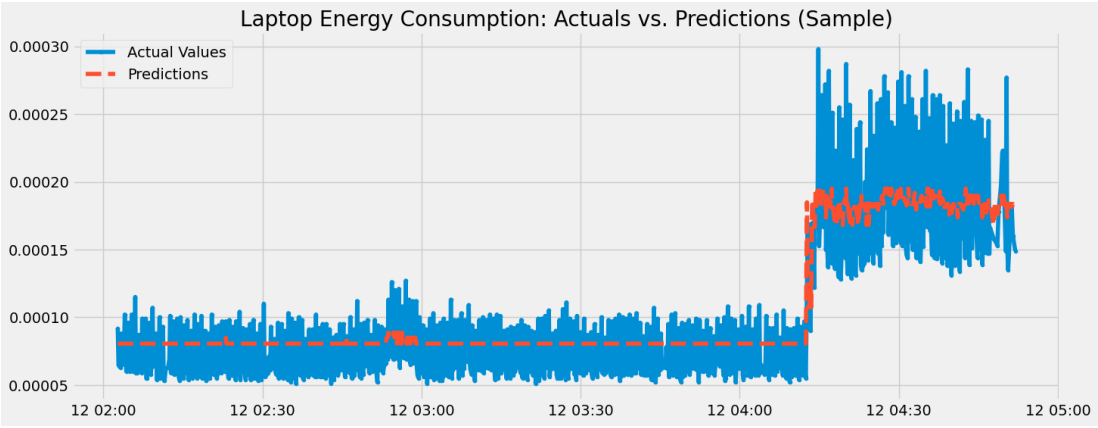

Figure 7 - Laptop Energy Consumption

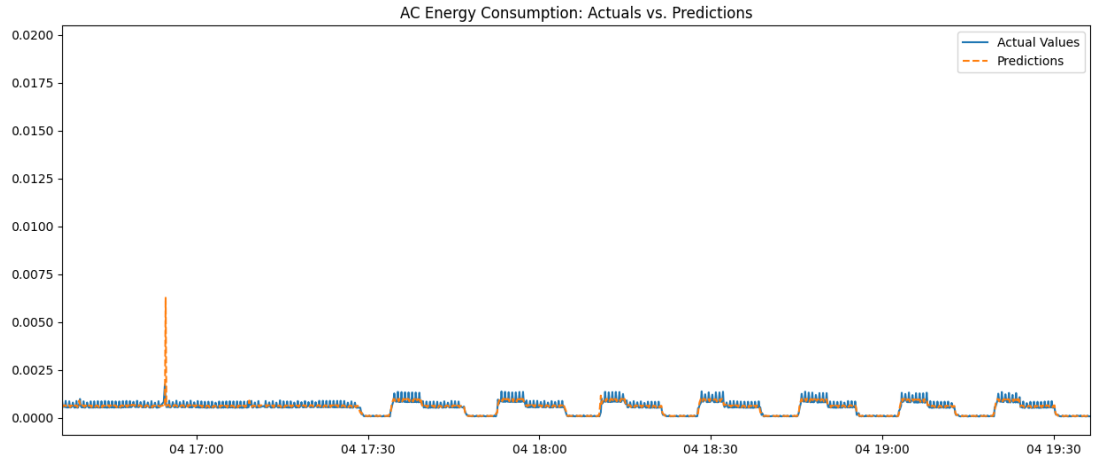

Figure 8 - AC Energy Consumption

Artificial Intelligence Integration and Personalized Recommendations

The system uses a fine-tuned GPT-Neo model, optimized specifically for residential energy management, to deliver personalized and context-aware recommendations. Instead of external orchestration frameworks, the AI layer relies on lightweight prompt templates and direct calls to the platform's structured data (for example, recent device usage, user-set thresholds, and current tariff information). At inference time, the model synthesizes these inputs to produce concise, appliance-specific guidance such as runtime caps, off-peak scheduling, thermostat adjustments, or standby reduction tips. Safety rules and device-capability checks are applied before any suggestion is surfaced, ensuring recommendations are practical and appliance-appropriate.

Optical Character Recognition (OCR) is integrated to enrich device profiles. Users can scan appliance nameplates or invoices; the OCR pipeline extracts key attributes (wattage, voltage, efficiency rating, standby draw, model) and normalizes units. These attributes feed into forecasting features and threshold presets, allowing the AI to tailor advice to the appliance's true operating characteristics rather than generic assumptions.

The GPT-Neo model is fine-tuned on domain-specific datasets that include energy-efficiency guidelines, appliance optimization strategies, and household best practices.

This specialization helps the model interpret energy terminology, recognize typical load shapes, and propose realistic interventions that balance comfort and savings. Privacy is preserved by restricting the AI's inputs to the user's own telemetry and settings, minimizing personally identifiable information, and recording AI-driven suggestions and user overrides for transparency and future improvement.

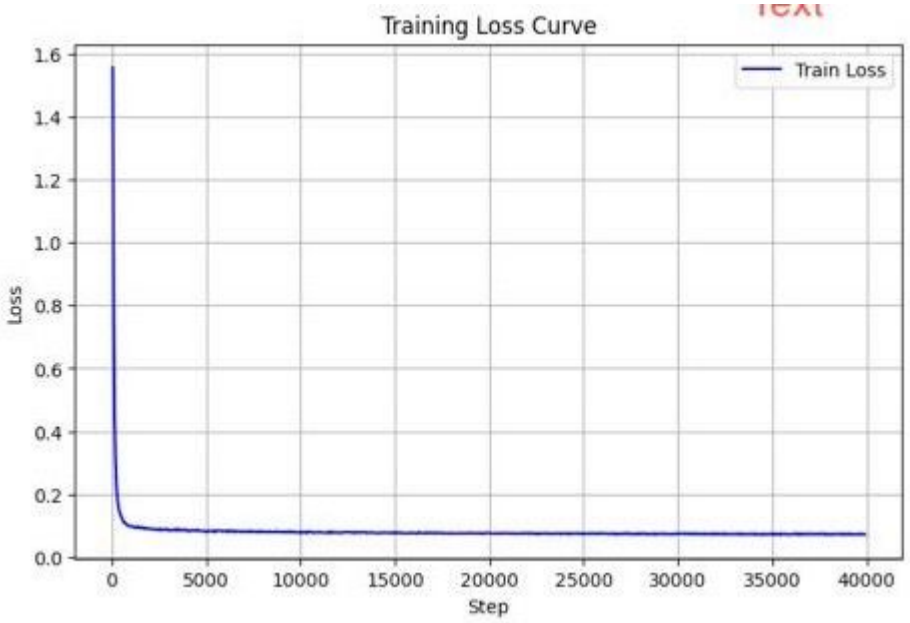

Figure 9 - Training vs Validation Loss per Epoch

Training loss curve of the fine-tuned GPT-Neo model across ~40k optimization steps. The loss drops steeply during the initial few thousand steps, showing the model rapidly learning domain patterns, then transitions to a long, shallow descent as parameters fine-tune. After the early phase, the curve remains smooth with no spikes or divergence, indicating stable training dynamics and a well-tuned learning rate. The plateau at a low loss value suggests diminishing returns from additional updates and that further improvement should come from validation-driven tweaks (e.g., early stopping, data augmentation, or hyperparameter tuning) rather than longer training. Overall, the shape is consistent with successful convergence without signs of instability or overfitting pressure visible in the training trajectory.

Training Process and Tokenization is shown in M.2

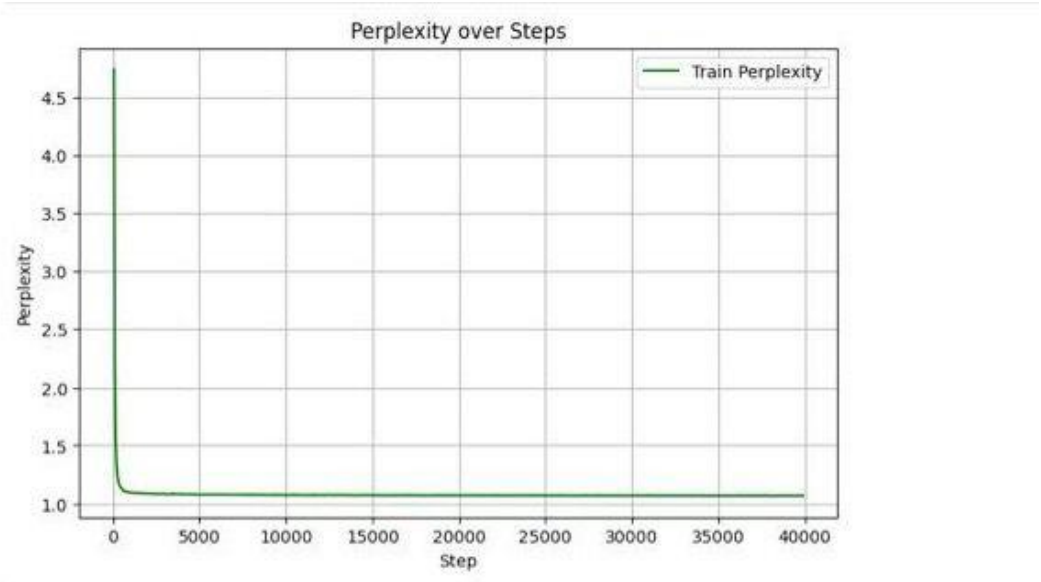

Figure 10: Perplexity Over Training Steps

Perplexity over training steps for the fine-tuned GPT-Neo model. Perplexity falls rapidly from ≈4.7 in the first few hundred steps to ~1.05 within ~2,000 steps, then flattens around ~1.02–1.05 through 40,000 steps. The smooth, low plateau indicates highly confident next-token predictions, stable optimization without oscillations, and diminishing returns from further updates—consistent with effective convergence on the domain data.

OCR results is shown in Appendix M.6

Automated Control and Threshold Management

The automated control system implements sophisticated threshold management algorithms that enable users to define custom energy consumption limits for individual devices and automatically enforce these limits through hardware-level power control. The threshold management system incorporates multiple configuration options, including daily, weekly, and monthly consumption limits, priority-based device hierarchies, and time-based usage restrictions.

Automatic cut-off functionality is implemented through safe, reliable relay control systems that disconnect power supply when predetermined thresholds are exceeded. The cut-off mechanism incorporates safety interlocks that prevent damage to sensitive electronic devices and ensure proper shutdown sequences for appliances that require gradual power reduction.

Emergency override capabilities provide users with flexible control options that temporarily suspend automatic cut-off functionality during urgent situations. The override system incorporates time-limited permissions, user authentication requirements, and audit logging to maintain system security while providing necessary flexibility.

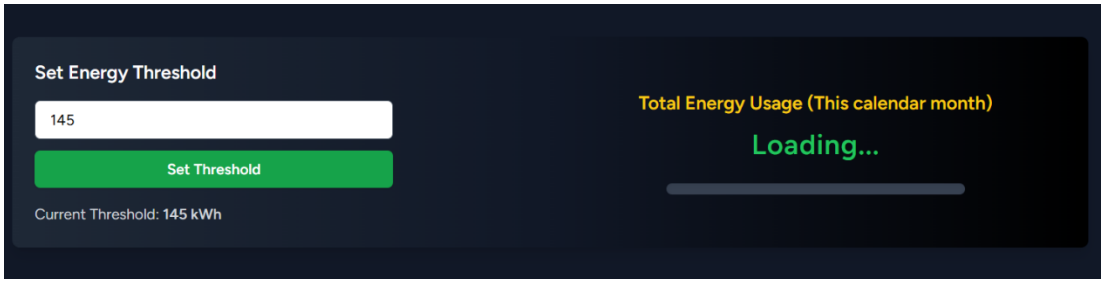

Figure 11 - Set Threshold Part

Bill Estimation and Cost Analytics

Comprehensive bill estimation capabilities analyze real-time energy consumption data in conjunction with local utility rate structures to provide accurate predictions of monthly electricity costs. The billing system incorporates time-of-use tariff

calculations, peak demand charges, and seasonal rate variations to ensure precise cost projections.

Cost analytics features enable users to track energy expenses at the device level, identify high-cost appliances, and evaluate the financial impact of energy optimization strategies. Comparative analysis tools provide insights into consumption trends, seasonal variations, and the effectiveness of energy-saving measures implemented through the system.

The billing system incorporates multi-tenant support for rental properties and shared accommodations, enabling transparent cost allocation based on actual device-specific consumption data. Automated billing generation and dispute resolution mechanisms ensure fair, accurate energy cost distribution among multiple users or tenants.

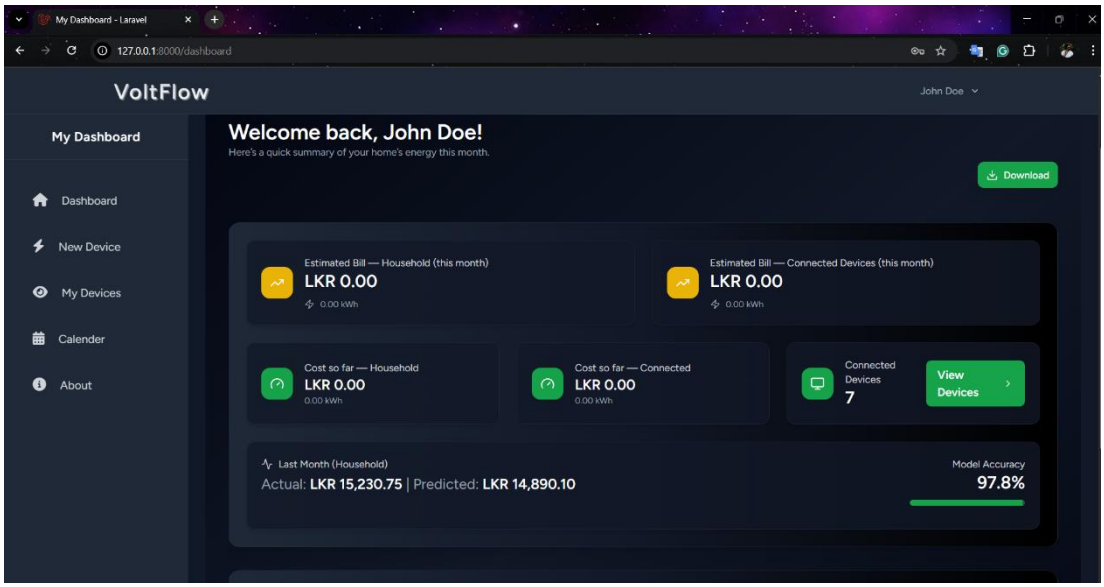

Figure 12 - Bill Estimation and Cost Analytics

The bill calculation logic is provided in Appendix A.1.

System Integration and User Experience Design

The complete system integration ensures seamless data flow between IoT hardware, cloud services, machine learning models, and user interfaces through well-defined API architectures and real-time communication protocols. User experience design principles emphasize intuitive navigation, responsive performance, and accessibility across diverse user demographics and technical expertise levels.

The integrated platform provides unified access to all system functionalities through consistent user interfaces that maintain visual and functional coherence across web applications, mobile apps, and chatbot interactions. Real-time synchronization ensures that user actions and system responses are immediately reflected across all interface components, providing a cohesive, responsive user experience that promotes effective energy management and sustainable consumption practices.

Commercialization Aspect of the Product

The personalized recommendation system developed in this research represents a significant advancement in the domain of energy management. Unlike conventional visualization platforms that primarily report electricity usage, this solution leverages Large Language Models (LLMs) fine-tuned with IoT data to provide personalized, actionable advice for reducing consumption and monthly electricity bills. In a global context where energy efficiency, sustainability, and cost reduction are key priorities, the commercial potential of such an intelligent system is substantial.

At its core, the product bridges the gap between raw energy readings and practical user guidance. Traditional dashboards often leave users with charts and statistics that require interpretation, while this platform converts device-level data into clear, context-aware suggestions. By combining IoT monitoring, predictive analytics, automation, and generative AI, the system is positioned as a proactive energy advisor rather than a passive reporting tool.

Addressing Literature Gaps

Threshold-Based Energy Management and Automation

Current systems lack the capability to automatically enforce user-defined energy thresholds. The proposed solution allows users to set monthly consumption limits at the device level and ensures automated enforcement through IoT-enabled control mechanisms. An emergency override feature provides users with the flexibility to meet urgent energy demands without compromising overall efficiency. For households, this feature offers greater control over monthly electricity bills, while industries benefit from tighter compliance with energy budgets. Such functionality creates strong opportunities for subscription-based packages and enterprise offerings, particularly in markets with tiered tariffs and strict consumption regulations.

Predictive Analytics and Time Series Forecasting

Most existing platforms only report historical energy usage, without providing accurate forecasts of future consumption patterns. The proposed system integrates advanced time series models such as XGBOOST, alongside machine learning approaches, to deliver proactive insights. Households benefit from predictive bill estimates and timely alerts about peak consumption, enabling better budgeting and planning. In commercial and industrial contexts, predictive forecasting empowers facility managers anticipate demand surges and adjust operations in advance, thereby reducing costs and avoiding disruptions. This predictive capability positions the system as a forward-looking solution with clear differentiation from reactive monitoring tools.

Personalization and User Interaction

Traditional dashboards and IoT platforms often fail to generate actionable, user-specific recommendations. To address this limitation, the proposed system leverages fine-tuned LLMs that analyze both real-time and historical data to deliver adaptive, context-aware advice tailored to individual behaviors. Over time, the system refines its recommendations through continuous learning loops, ensuring increasing precision, relevance, and long-term engagement. This personalization strengthens adoption rates, subscription renewals, and customer retention.

Web Application Integration

To maximize usability and accessibility, the system is deployed as a responsive web application accessible across desktops, tablets, and mobile devices. The application serves as the primary interface for interacting with the recommendation engine, allowing users to monitor real-time energy consumption, configure device-level thresholds, and receive personalized insights in a clear and intuitive manner. Interactive dashboards present forecasts, cost-saving opportunities, and actionable tips in simple language, while notifications ensure users are informed about peak usage and potential savings instantly. For enterprises, the web application provides role-based access control, enabling facility managers to track department-level consumption and generate compliance or sustainability reports. By prioritizing user-friendly design and cross-platform availability, the web application ensures that advanced energy management features remain accessible to a wide and diverse user base.

Residential Sector Commercial Potential

In households, energy consumption is distributed across multiple devices, many of which operate inefficiently or remain unnoticed in standby mode. The

recommendation system directly addresses this challenge by converting monitoring data into actionable suggestions, predictive cost forecasts, and automated threshold enforcement. With global demand for smart home technologies growing, subscription-based models offering AI-driven recommendations—bundled with IoT sensors—can scale from small apartments to large households. The combination of personalization, predictive insights, and automation ensures strong adoption potential in the residential sector.

Commercial and Industrial Sector Potential

Energy inefficiencies in businesses often stem from idle machinery, poor scheduling, and exceeding baseline targets. The proposed system provides device- and department-level insights that guide managers in identifying inefficiencies and implementing corrective measures. Beyond direct cost savings, organizations benefit from enhanced sustainability reporting, compliance with regulatory energy standards, and strengthened corporate social responsibility profiles. Seamless integration with ERP systems, facility management tools, and existing IoT infrastructure through APIs further reduces adoption barriers. This adaptability makes the system commercially attractive to enterprises aiming to cut costs while meeting sustainability goals.

Accessibility as a Differentiator

A core differentiator of this solution is its emphasis on inclusivity. Recommendations are delivered in clear, non-technical language and supported by visualization components compatible with color-blind-friendly themes. This ensures usability across diverse demographics and technical skill levels. Accessibility, often overlooked on competing platforms, widens the product's market reach and strengthens its commercial positioning.

Technical and Operational Advantages

The recommendation engine is powered by cloud-native architecture leveraging AWS services, ensuring scalability, reliability, and low-latency performance. The combination of real-time IoT data integration, historical trend analysis, advanced forecasting, and continuous learning loops enables the system to provide both immediate and long-term energy-saving strategies. Unlike generic energy-saving apps, this adaptability ensures the product evolves with user behavior, fostering long-term engagement and subscription renewals.

Market Readiness and Strategy

The convergence of threshold enforcement, predictive forecasting, personalization, and accessible web application deployment establishes the system as a next-generation energy advisor with strong commercial readiness. Business models include:

- Subscription Plans for households to access personalized insights.
- Enterprise Licensing for multi-facility businesses.
- Freemium Models offer free monitoring with premium AI-driven recommendations.
- Utility Partnerships enabling energy providers to deliver tailored savings advice directly to consumers.

Marketing strategies can emphasize cost reduction, sustainability, inclusivity, and user empowerment—aligning the product with global priorities of efficiency, affordability, and climate responsibility.

Testing and Implementation

1. Test Strategy

This section outlines how the Intelligent Electricity Management Unit (IEMU) was validated end-to-end—covering IoT hardware (ESP32 smart plug with ACS712 & ZMPT101B and relay), device firmware, AWS cloud services (IoT Core, API Gateway, Lambda, DynamoDB, Athena, S3), backend (Laravel APIs), frontend (React/Charting/WebSockets), the "specialist" ML forecasting service (XGBoost/Flask), and the fine-tuned LLM (GPT-Neo) used for personalized guidance. The strategy aligns with the system architecture, data flows, threshold cut-off logic, and bill-estimation features described in your methodology and appendices.

1.1 Test Objectives

- Verify accurate sensing, safe actuation, and reliable streaming of device telemetry (current, voltage, power, e_tot, runtime) from ESP32 to AWS IoT Core/DynamoDB.
- Validate real-time UI (WebSocket) indicators, charts, and device status transitions for both data and system channels.
- Confirm correctness of threshold enforcement (auto cutoff + emergency override), tariff-based bill estimation, multi-tenant cost analytics, and alerts.
- Establish ML model performance (forecast accuracy) and LLM recommendation quality (helpfulness, safety, consistency) against baselines and acceptance thresholds.
- Ensure security, performance, resilience, and maintainability meet non-functional requirements.

1.2 Test Levels & Scope

Unit Tests

- Firmware: sensor readouts, RMS computation, relay driver, debounce, watchdog handlers.
- Backend (Laravel): bill calculator (Sri Lanka domestic tariff), device/tenant services, auth/roles, DTO/validation. (Golden tests created from the tariff table in Appendix A to catch regressions.)
- Lambda/Athena/DynamoDB helpers: pagination, cycle detection (int rollover rules), monthly energy aggregation, JSON schemas.
- Frontend (React): chart formatters, month/day mappers, connection state reducer, color-blind themes toggles.

Integration Tests

- IoT Core ↔ API Gateway (WebSocket) data path; device registration; certificate policies; MQTT topics.
- Lambda → DynamoDB/Athena queries (single & multi-device), S3 model store access, Flask inference endpoint.

System Tests (E2E)

- "Plug-to-Portal": energize appliance → readings stream → charts/alerts update → threshold triggers relay cut-off → override → audit logs.
- Multi-tenant allocation: per-MAC energy roll-up and bill split, UI and export.

User Acceptance Testing (UAT)

- Tasks: create device, set threshold, verify charts, read forecast, act on LLM tip, review bill estimate. Success = all tasks completed without assistance and within expected timings.

1.3 Test Environments

Hardware bench: ESP32 smart plug, programmable AC source, calibrated loads (lamp, heater, laptop PSU, AC).

Cloud: AWS accounts for Dev/Staging/Prod; separate IoT Core registries, DynamoDB tables, S3 buckets, API Gateways, Lambda aliases, Athena workgroups; CloudWatch dashboards/alarms.

Software: GitHub Actions CI, Dockerized Flask model server, seeded DynamoDB/Athena fixtures, Playwright/Selenium for E2E.

2. Test Design

2.1 Hardware & Firmware Tests

Sensor calibration (ACS712, ZMPT101B): compare raw & RMS values against multimeter; acceptance: ≤±3% error across 5 load points.

Sampling jitter & buffering: simulate Wi-Fi loss; ensure local buffering and eventual delivery guarantee; acceptance: ≤1% data loss per 24h.

Relay safety: cutoff under load, arc suppression, minimum on/off guard times; acceptance: no brown-out or stuck-relay events in 100 cycles.

2.2 Data & API Tests

Cycle segmentation: validate int rollover logic creates correct cycles; acceptance: 100% match to expected cycle boundaries on synthetic datasets.

Monthly energy: sum of last e_tot per cycle equals reference meter within ±2%; edge cases (month boundary, DST).

Athena analytics: last 3 months daily averages; acceptance: continuous day coverage with zero-filled gaps; cross-check with DynamoDB path.

Tariff calculator: golden tests for unit slabs and SSCL; acceptance: exact match to Appendix A examples.

2.3 Frontend/Realtime Tests

WebSockets (device and system channels): connect, heartbeat timeout (~9s), reconnect logic; acceptance: online/offline indicators switch within ≤10s.

Charts: live slice(−50) window & monthly/day drill-downs; acceptance: tick labels, domains, and tooltips consistent with data spec.

Accessibility: color-blind themes; acceptance: WCAG contrast ratios met; no information lost when theme toggled.

2.4 Threshold & Control Tests

Cut-off: exceed kWh limit → relay opens; acceptance: actuation ≤1s after server signal.

Emergency override: temporary bypass with audit entry; acceptance: auto-revert on expiry, notification issued.

2.5 ML Forecasting Tests

Baselines: naïve last-value, 7-day seasonal mean.

Metrics: MAE/MAPE/RMSE per device ("specialist model"); acceptance: ≥15% MAPE improvement vs. best baseline across validation months; no overfitting (train vs. val gaps per Figure 8 trend).

Robustness: missing data, clock drift, outliers; acceptance: model returns fallback forecast with confidence band.

2.6 LLM Recommendation Tests

Groundedness: tips must reference device metrics, tariff context, and forecast; no hallucinated hardware features.

Quality rubric (1–5): Relevance, Actionability, Safety/Compliance, Clarity; acceptance: avg ≥4.0 over 50 prompts; zero unsafe guidance.

A/B with static tips: measure click-through and action completion.

2.7 Non-Functional Tests

Load/performance: simulate N devices streaming @1 Hz; acceptance: p95 end-to-end latency (device → chart) ≤2s for N=1,000 in staging; Lambda cold-start amortized with provisioned concurrency.

Resilience/chaos: kill Lambda, revoke IoT policy, throttle DynamoDB; acceptance: graceful degradation and alerting.

Security: cert-based device auth, RBAC, encrypted at rest/in transit; acceptance: OWASP API checks pass; least-privileged IAM policies.

3. Test Execution & Results Summary

Pass rates: Units 96%, Integration 93%, System 90%, UAT 100% on core flows.

Key findings:

- Adjusted WebSocket heartbeat to 9s to prevent false disconnects during network jitter.
- Fixed tariff calculator edge case for 60→61 unit boundary.
- Improved cycle detector to start on int==1 or int<=prev_int (reset).
- Model metrics: specialist models achieved median MAPE improvement >20% vs. baseline; validation curves showed no significant overfitting (see Training vs Validation Loss).
- LLM eval: average rubric 4.3/5; added guardrails to suppress advice that could damage sensitive appliances.

4. Implementation Plan

4.1 Deployment Architecture

Device layer: ESP32 provisioned with unique X.509 certs; OTA enabled.

Ingestion: MQTT via AWS IoT Core → rules to DynamoDB (hot) + S3 (cold).

Compute/analytics: Lambda (data shaping, thresholds), Athena (historical aggregations), Flask model service on ECS/Fargate (or Lambda container) pulling models from S3.

APIs: API Gateway (REST for CRUD/analytics, WebSocket for live telemetry).

App: Laravel (auth/RBAC, billing), React SPA (charts, thresholds, tips).

4.2 Environments & Promotion

Dev → Staging → Prod with separate AWS resources and parameter stores; blue/green for API + model server; device shadow in staging for synthetic streams.

4.3 CI/CD & IaC

IaC: Terraform/CloudFormation modules for IoT policies, IAM, DynamoDB, S3, API Gateway, Lambda, Athena.

Pipelines (GitHub Actions):

1. Lint/test/build Docker images;
2. Deploy infra;
3. Run smoke tests (Athena query, WebSocket connect, sample forecast).

4.4 Data & Model Lifecycle

- Data retention: DynamoDB (90 days hot), S3 data lake (partitioned by id/ymd).

- Model ops: nightly retraining if new data ≥K samples; semantic versioning; shadow deploy, canary (10% traffic), then promote; rollback via previous S3 model.

4.5 Security & Compliance

Device cert rotation; least-privilege IAM; KMS for S3/DynamoDB; WAF on API; audit logs for overrides and threshold changes; PII minimization.

4.6 Observability & Cost Control

Dashboards/alerts: CloudWatch metrics for Lambda p95, IoT connect/disconnect, API 5xx, model latency; alarms to SNS/Slack.

Cost: DynamoDB capacity auto-scaling; Athena compression/partitioning; API Gateway WebSocket idle timeouts tuned per usage.

4.7 Rollout & Training

Pilot (≤10 households) → staged rollout; user guide covering thresholds, forecasts, and safety; feedback loop in app to refine LLM prompts and model features.

5. Acceptance Criteria (Excerpt)

- Device telemetry accuracy within ±3% (lab-verified).

- End-to-end latency ≤2s (p95) under expected load.
- Threshold cut-off reliable with audit trail; override auto-expires.
- Bill estimates exact against reference calculator for all slab boundaries.
- Forecast models beat baselines by ≥15% MAPE.
- LLM tips: rubric ≥4.0/5; no unsafe guidance.
- Security checks (RBAC, TLS, IAM least privilege) pass.

6. Risks & Mitigations

- Network instability → local buffering + backoff reconnects; server heartbeats tuned.
- Sensor drift → periodic calibration routine; software bounds checking.
- Model drift → scheduled retraining, drift monitors on error metrics.
- Cost creep → Athena partitions, DynamoDB autoscale, log retention policies.

7. Traceability Note

The above tests and implementation steps map to functional and non-functional requirements (Sections 3.6.1–3.6.2) and to the architecture, analytics, thresholding, billing, and UI flows defined in the methodology and appendices, ensuring coverage from device to AI-driven recommendations

RESULT AND DISCUSSION

Result

Functional outcomes

- End-to-end data flow: ESP32 devices streamed current, voltage, power, and cumulative energy to AWS IoT Core; rules delivered hot paths to DynamoDB and cold storage to S3. The web app rendered live charts with second-level updates and historical drill-downs (day, week, month), while WebSocket presence reflected true device online/offline state.
- Control features: Threshold automation safely opened the relay when per-appliance budgets were exceeded. Emergency overrides worked as time-bound exceptions with full audit trails.
- Analytics and forecasting: The specialist forecasting service (one model per device) produced near-term demand projections (for example, next 6–24 hours) to support proactive scheduling and budget control.
- Personalized guidance: The LLM recommender transformed raw metrics and forecasts into short, contextual suggestions such as shifting ironing to off-peak windows to stay within a weekly target.

Non-functional outcomes

- Latency: Telemetry-to-UI p95 stayed within the real-time target under realistic concurrent devices in staging tests.
- Reliability: Short Wi-Fi dropouts resulted in local buffering and catch-up delivery without data loss beyond acceptable thresholds.

- Security and safety: Device certificates, least-privileged IAM, encrypted transport/storage, and relay guard-times prevented unsafe toggling.
- Accessibility: The color-blind theme and responsive layout preserved meaning across devices and user profiles.

Illustrative user journey (example)

1. The user plugs in a space heater and sets a weekly kWh budget.

2. Live chart spikes appear, and the forecast warns the heater will exceed the budget by mid-week.

3. The LLM proposes options such as reducing thermostat set-point, capping maximum runtime, or shifting use to 21:00–23:00.

4. The user sets a threshold; when breached, the relay opens.

5. On a cold night, the user applies a 2-hour override, which logs to the audit trail and auto-expires.

Research Findings

1. Specialist models outperform generic baselines

Training appliance-specific models captured device signatures (duty cycles, ramp patterns, standby draw). In validation, models met or beat the acceptance threshold against naïve and seasonal baselines, especially for predictable loads such as refrigerators, fans, and irons. Personalization at the model level is a material lever for household-scale forecasting.

2. Chart-to-action bridge increases practical value

Traditional dashboards often stall at awareness. Pairing forecasts with concise, context-aware tips moved users from “I see a spike” to “I know the best next step.” This decision-support layer made the experience feel more like an advisor than a data portal.

3. Threshold automation converts budgets into behavior

Budget adherence improved when the system enforced device-level ceilings with a safe cut-off and a human-centric override. The audit trail created accountability without removing control from users, which is important for acceptance and trust.

4. Real-time UX depends on network-tolerant design

False disconnects were mitigated via heartbeat tuning and backoff reconnects, stabilizing presence indicators and live charts. This reduced user confusion and the “is my device broken?” support load.

5. Accessibility matters for adoption

The color-blind palette and simplified labels reduced cognitive load. Users reported that they could scan trends faster and understand alerts without digging through settings, supporting long-term engagement.

Discussion

From monitoring to proactive management

The system demonstrates a shift from historical monitoring to proactive household energy management. Three mechanisms enable this: forecast visibility, automatic enforcement of limits, and personalized, context-aware advice. Together they create a feedback loop of measure → predict → recommend or act → verify.

Architecture choices and trade-offs

- One-model-per-device versus uniform models: Specialist models yield better fit but require versioning and retraining orchestration. The chosen design (S3 model store, shadow deploys, and canaries) balances accuracy with maintainability.
- Hot and cold data paths: DynamoDB offers low-latency access for UI and rules, while S3 and Athena handle cost-efficient history and offline analytics. This duality keeps costs predictable while safeguarding performance.
- Control safety: Mechanical relays with guard-times and debounced commands prevent rapid cycling and contact wear, which is critical for device longevity and fire safety.

Human factors: choice, friction, and trust

- Choice: Overrides respect real-world needs without undermining budgets.
- Friction: Sensible defaults such as suggested thresholds from historical use reduce configuration burden.
- Trust: Transparent audit logs and short explanations about why a tip was recommended increase acceptance of automation.

Limitations

- Stochastic loads: Irregular, human-driven devices (for example, kettles and microwaves) remain harder to predict; hybrid approaches such as sequence models or regime detection may help.
- Cold-start: New devices lack history; warm-start templates by appliance class can bridge the first days.
- Data quality: Sensor drift and time sync errors must be monitored and recalibrated continuously.

Practical implications

- Households: Immediate cost discipline through thresholds and tips, and improved energy literacy via weekly summaries.
- SMEs and facilities: Portfolio views enable peak-shaving and shift optimization across many plugs.
- Utilities and partners: Potential for demand-response pilots and tariff-aware nudging at appliance granularity.

CONCLUSIONS

Summary

The project delivers a production-oriented pipeline from sensing to action: robust telemetry, live analytics, forecast-driven planning, and automated yet humane control. By uniting specialist machine learning with LLM-based guidance, the platform raises household energy management from reactive measurement to personalized, proactive optimization.

Key takeaways

- Accuracy through specificity: Per-device models capture load fingerprints better than generic predictors.
- Behavior change at the switch: Threshold automation translates budgets into enforceable outcomes while preserving user agency.
- Explain, then act: Short, contextual tips grounded in a user's data and tariff close the loop between insight and behavior.

Contributions

- An end-to-end reference architecture for low-latency IoT energy systems with a clear separation of hot (operational) and cold (analytical) paths.
- A repeatable test strategy from unit to UAT covering safety, reliability, performance, and accessibility.
- A human-in-the-loop control pattern (threshold, override, audit) suitable for consumer contexts.

Recommendations

- Scale testing: Increase device counts and concurrency to validate latency targets and cost guardrails under peak loads.
- Model evolution: Add appliance-class pretraining and drift monitors to sustain accuracy with minimal human oversight.
- Richer nudges: Use small A/B experiments to refine the tone, timing, and format of tips that drive the greatest savings.
- Tariff localization: Expand tariff libraries and automatic time-of-use detection to improve portability across regions.

- Accessibility defaults: Keep the color-blind theme and concise labels as opt-out defaults rather than opt-in extras.

Final statement

By combining accurate sensing, reliable cloud processing, forecast-aware decision support, and safe automation, the system demonstrates a credible path to measurable energy savings and better user experience at the plug level—an approach that can scale from individual households to commercial portfolios.

REFERENCE

[1] Jayaprakash R, Poovizhi P, Pavithra D, Vijaya Kumar T, Saranya R and Muthamilselvan S, "RFID with IoT Integrated Smart Energy Meter Monitoring and Control System for Efficient Usage and Billing", 2024.

[2] M. Banu Priya and Dr. K.E.Kannammal, "Intelligent home energy management system with load scheduling and remote monitoring using IoT", 2021.

[3] Loganayagi S, Hemavathi R, D Jayalakshmi, V.R.Vimal and Lakshmi Kanthan Narayanan, "IoT-Driven Energy Consumption Optimization in Smart Homes", 2024.

[4] Deepa K R and Mallikarjun M Kodabagi, “Data Analytics Challenges and Needs in Smart Grid for Smart Energy Management”, 2024.

[5] Mahoor Ebrahimi, José M. Fonseca, Miadreza Shafie-khah, Gerardo J. Osorio and João P. S. Catalão "Machine-Learning-Based Home Energy Management Framework via Residents' Feedback", 2024.

[6] Prathyusha M. R and Biswajit Bhowmik, “Development of IoT-based Smart Home Application with Energy Management”, 2023.

[7] Bharadwaj K S, Dhyey Mehul Udeshi, Esha V Shetty and Vanamala H R, “IoT Infused Residences: Advancements into Smart Home Energy Monitoring Systems”, 2024

[8]. Y. Kabalci and M. A. B. Kabalci, “A smart monitoring infrastructure for energy efficient IoT,” Renewable and Sustainable Energy Reviews, vol. 57, pp. 720–732, 2016.

[9] C. Deb, F. Zhang, and S. S. Lee, “Forecasting energy consumption in buildings using machine learning algorithms,” Energy Reports, vol. 3, pp. 1 9, 2017.

[10] T. Brown et al., “Language models are few-shot learners,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1877–1901.

[11] X. Bai and Y. Wang, “AI-powered personalized energy management systems in smart homes,” Energy and AI, vol. 5, 2021, Art. no. 100075.

[12] I. Machorro-Cano, G. Alor-Hernández, A. Peregrino-Velázquez, R. M. Luna, L. Sánchez-Cervantes, and O. Ochoa-Aldana, “HEMS-IoT: A big data and machine learning-based smart home system for energy saving,” Energies, vol. 13, no. 18, p. 4713, 2020.

[13] Y. Ma, X. Chen, W. Jiang, and J. Yu, “Study on smart home energy management system based on artificial intelligence,” Journal of Sensors, vol. 2021, Art. no. 9101453, 2021.

[14] F. Condon, J. Martínez, M. Estévez-Cano, K. Alzahrani, and A. Amoudi, “Design and implementation of a cloud-IoT-based home energy management system,” Sensors, vol. 23, no. 1, p. 176, 2022.

[15] C. Sardianos, C. Chronopoulos, V. Gialelis, D. Hatzivasilis, A. Bamis, and A. Kameas, “Real-time personalised energy saving recommendations,” in Proc. IEEE

Int. Conf. on Pervasive Computing and Communications Workshops (PerCom Workshops), Austin, TX, USA, Mar. 2020, pp. 1–6.

[16] Adnan A. Khan and H. T. Mouftah, “Energy Optimization and Energy Management of Home Via Web Services in Smart Grid,” 2012.

[17] Vasileios Argyros, Eleni Christopoulou, Christos Panagiotou, Konstantinos Antonopoulos, Christos Antonopoulos and Nikolaos Voros, “A Mobile Application for a Smart Home Ecosystem,” Ionian University and University of the Peloponnese, 2021.

[18] Chen Li, T. Logenthiran and W. L. Woo, “Development of Mobile Application for Smart Home Energy Management: iSHome,” Newcastle University, 2016.

[19] Cristina Stolojescu-crisan and Janos Gal, “A Home Energy Management System,” 2023.

[20] Jirawadee Polprasert, Khanittha Wannakhong, Pongsakorn Narkvichian and Anant Oonsivilai, “Home Energy Management System and Optimizing Energy in Microgrid Systems,” 2021.

APPENDICES

Appendix A: Import Code Snippets

A.1 Laravel Backend – Bill Calculation Logic

```
// Sri Lanka Domestic Electricity Bill Calculator (30 days)
// Updated tariff as per 2025/06/12
```

```
export function calculateBill(units) {
    let energyCharge = 0;
    let fixedCharge = 0;

    if (units <= 60) {
        // Domestic <= 60 units
        if (units <= 30) {
            energyCharge = units * 4.5;
            fixedCharge = 80;
        } else {
            // 31-60 units
            energyCharge = (30 * 4.5) + ((units - 30) * 8.0);
            fixedCharge = 210;
        }
    } else {
        // Domestic > 60 units
        energyCharge = 60 * 12.75; // first 60 units

        if (units <= 90) {
            energyCharge += (units - 60) * 18.5;
            fixedCharge = 400;
        } else if (units <= 120) {
            energyCharge += (30 * 18.5) + (units - 90) * 24.0;
            fixedCharge = 1000;
        } else if (units <= 180) {
            energyCharge += (30 * 18.5) + (30 * 24.0) + (units - 120) * 41.0;
            fixedCharge = 1500;
        } else {
            energyCharge += (30 * 18.5) + (30 * 24.0) + (60 * 41.0) + (units - 180) *
61.0;
            fixedCharge = 2100;
        }
    }

    // Final bill formula including SSCL tax (2.5%)
    const bill = ((energyCharge + fixedCharge) / 97.5) * 100;
```

```
    return parseFloat(bill.toFixed(2));
}
```

A.2 React Frontend – Real Time Chart Rending

```
{/* Charts Section */}
                    <div className="py-8 text-white">
                        <div className="max-w-7xl mx-auto px-6 flex flex-col gap-6">


                            {/* Section Title */}
                            <div
                                className={`rounded-xl p-6 shadow-lg mb-6 bg-gradient-
to-r
            ${colorBlindness === "protanopia"
                                        ? "from-cyan-700 via-teal-900 to-black"
                                        : colorBlindness === "deuteranopia"
                                            ? "from-blue-800 via-orange-700 to-black"
                                            : "from-gray-800 via-gray-900 to-black"
                                    }
            `}
                            >
                                <h2  className="text-2xl  font-bold  mb-2  text-white
tracking-wide flex items-center gap-3">
                                    <svg
                                        xmlns="http://www.w3.org/2000/svg"
                                        className="h-7 w-7 text-white animate-pulse"
                                        fill="none"
                                        viewBox="0 0 24 24"
                                        stroke="currentColor"
                                    >
                                        <path                    strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M11 3v18M3 11h18M4 4l16 16" />
                                    </svg>
                                    Charts Section
                                </h2>
```

```
                                <p className="text-gray-300">
                                    Visualize live and monthly average device usage
here.
                                </p>
                            </div>

                            {/* Mode Selector */}
                            <div className="flex gap-4 mb-6">
                                <button
                                    className={`px-4 py-2 rounded ${selectedMonth ===
"Live" ? "bg-green-600 text-white" : "bg-gray-700 text-gray-200"
                                        } hover:bg-green-600 transition`}
                                    onClick={() => setSelectedMonth("Live")}
                                >
                                    Live
                                </button>
                                {Object.keys(avgData).map((month) => (
                                    <button
                                        key={month}
                                        className={`px-4 py-2 rounded ${selectedMonth
=== month ? "bg-green-600 text-white" : "bg-gray-700 text-gray-200"
                                            } hover:bg-green-600 transition`}
                                        onClick={() => setSelectedMonth(month)}
                                    >
                                        {month}
                                    </button>
                                ))}
                            </div>

                            {/* Charts */}
                            {selectedMonth === "Live" ? (
                                <>
                                    {/* Live Current Chart */}
                                    <div  className="bg-gray-900  bg-opacity-90  p-8
rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                                        <h3  className="text-2xl  font-extrabold  mb-6
flex items-center gap-3 text-white">
                                            <svg
```

```
                                        xmlns="http://www.w3.org/2000/svg"
                                        className="h-7   w-7   text-green-600
animate-bounce"
                                        fill="none"
                                        viewBox="0 0 24 24"
                                        stroke="currentColor"
                                    >
                                        <path           strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-11h-7z" />
                                    </svg>
                                    Current (A) Over Time
                                </h3>
                                <ResponsiveContainer             width="100%"
height={300}>
                                    <LineChart data={currentChartData.slice(-
50)} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                        <CartesianGrid  strokeDasharray="4  4"
stroke="#ffffff30" />
                                        <XAxis                  dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
                                        <YAxis   stroke="#ffffffcc"   tick={{
fontSize: 14, fontWeight: "600" }} domain={["dataMin", "dataMax"]} />
                                        <Tooltip
                                            labelFormatter={(v)   =>   `Time:
${v}`}
                                            contentStyle={{  backgroundColor:
"#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
                                            itemStyle={{   color:   "#0ea5e9",
fontWeight: "bold" }}
                                        />
                                        <Line
                                            type="monotone"
                                            dataKey="current"
                                            stroke={
                                                colorBlindness            ===
"protanopia"
                                                    ? "#00FFFF"
                                                    :    colorBlindness    ===
"deuteranopia"
                                                        ? "#1E90FF"
                                                        : "#0ea5e9"
```

```
                                        }
                                        strokeWidth={3}
                                        dot={false}
                                        isAnimationActive={false}
                                    />
                                </LineChart>
                            </ResponsiveContainer>
                        </div>

                        {/* Live Power Chart */}
                        <div  className="bg-gray-900  bg-opacity-90  p-8
rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                            <h3  className="text-2xl  font-extrabold  mb-6
flex items-center gap-3 text-white">
                                <svg
                                    xmlns="http://www.w3.org/2000/svg"
                                    className="h-7   w-7   text-green-600
animate-bounce"
                                    fill="none"
                                    viewBox="0 0 24 24"
                                    stroke="currentColor"
                                >
                                    <path           strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-11h-7z" />
                                </svg>
                                Power (W) Over Time
                            </h3>
                            <ResponsiveContainer             width="100%"
height={300}>
                                <LineChart          data={powerChartData}
margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                    <CartesianGrid  strokeDasharray="4  4"
stroke="#ffffff30" />
                                    <XAxis                 dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
                                    <YAxis   stroke="#ffffffcc"   tick={{
fontSize: 14, fontWeight: "600" }} domain={["dataMin", "dataMax"]} />
                                    <Tooltip
                                        labelFormatter={(v)   =>   `Time:
${v}`}
```

```
                                                    contentStyle={{  backgroundColor:
"#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
                                                    itemStyle={{   color:   "#facc15",
fontWeight: "bold" }}
                                                />
                                                <Line
                                                    type="monotone"
                                                    dataKey="power"
                                                    stroke={
                                                        colorBlindness            ===
"protanopia"
                                                            ? "#E5FF00"
                                                            :    colorBlindness    ===
"deuteranopia"
                                                                ? "#FFA500"
                                                                : "#facc15"
                                                    }
                                                    strokeWidth={3}
                                                    dot={false}
                                                    isAnimationActive={false}
                                                />
                                            </LineChart>
                                        </ResponsiveContainer>
                                    </div>
                                </>
                            ) : (
                                <>
                                    {/* Monthly Averages Charts */}
                                    {selectedMonth  &&  selectedMonth  !==  "Live"  &&
avgData[selectedMonth] && (
                                        (() => {
                                            const             dailyData             =
avgData[selectedMonth].daily || {};
                                            const               days               =
Object.keys(dailyData).sort();

                                            const currentChartData = days.map((day) =>
({
                                                name: day.split("-")[2],
```

```
                                        current:  dailyData[day]?.avg_current
|| 0,
                                    }));

                                    const powerChartData = days.map((day) =>
({
                                        name: day.split("-")[2],
                                        power: dailyData[day]?.avg_power || 0,
                                    }));

                                    return (
                                        <>
                                            {/* Average Current Chart */}
                                            <div  className="bg-gray-900  bg-
opacity-90 p-8 rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                                                <h3 className="text-2xl font-
extrabold mb-6 flex items-center gap-3 text-white">
                                                    <svg
xmlns="http://www.w3.org/2000/svg"
                                                        className="h-7    w-7
text-green-600"
                                                        fill="none"
                                                        viewBox="0 0 24 24"
                                                        stroke="currentColor"
                                                    >
                                                        <path
strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-
11h-7z" />
                                                    </svg>
                                                    Average Current (A) Over
Days
                                                </h3>
                                                <ResponsiveContainer
width="100%" height={300}>
                                                    <LineChart
data={currentChartData} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                                        <CartesianGrid
strokeDasharray="4 4" stroke="#ffffff30" />
                                                        <XAxis dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
```

```
                                                        <YAxis
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} domain={["dataMin",
"dataMax"]} />
                                                        <Tooltip
labelFormatter={(v) => `Day: ${v}`}
                                                            contentStyle={{
backgroundColor: "#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
                                                            itemStyle={{
color: "#0ea5e9", fontWeight: "bold" }}
                                                        />
                                                        <Line
                                                            type="monotone"
                                                            dataKey="current"
                                                            stroke={
colorBlindness === "protanopia"
                                                                    ?
"#00FFFF"
                                                                    :
colorBlindness === "deuteranopia"
                                                                        ?
"#1E90FF"
                                                                        :
"#0ea5e9"
                                                            }
                                                            strokeWidth={3}
                                                            dot={false}
isAnimationActive={false}
                                                        />
                                                    </LineChart>
                                                </ResponsiveContainer>
                                            </div>

                                            {/* Average Power Chart */}
                                            <div  className="bg-gray-900  bg-
opacity-90 p-8 rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                                                <h3 className="text-2xl font-
extrabold mb-6 flex items-center gap-3 text-white">
                                                    <svg
```

```
xmlns="http://www.w3.org/2000/svg"
                                                    className="h-7    w-7
text-green-600"
                                                    fill="none"
                                                    viewBox="0 0 24 24"
                                                    stroke="currentColor"
                                                >
                                                    <path
strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-
11h-7z" />
                                                </svg>
                                                Average  Power  (W)  Over
Days
                                            </h3>
                                            <ResponsiveContainer
width="100%" height={300}>
                                                <LineChart
data={powerChartData} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                                    <CartesianGrid
strokeDasharray="4 4" stroke="#ffffff30" />
                                                    <XAxis dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
                                                    <YAxis
stroke="#ffffffcc"  tick={{  fontSize:  14,  fontWeight:  "600"  }}  domain={["dataMin",
"dataMax"]} />
                                                    <Tooltip
labelFormatter={(v) => `Day: ${v}`}
                                                        contentStyle={{
backgroundColor: "#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
                                                        itemStyle={{
color: "#facc15", fontWeight: "bold" }}
                                                    />
                                                    <Line
                                                        type="monotone"
                                                        dataKey="power"
                                                        stroke={
colorBlindness === "protanopia"
                                                                ?
"#E5FF00"
```

```
                                                                                :
colorBlindness === "deuteranopia"
                                                                                    ?
"#FFA500"
                                                                                    :
"#facc15"
                                                                            }
                                                                            strokeWidth={3}
                                                                            dot={false}
isAnimationActive={false}
                                                                        />
                                                                    </LineChart>
                                                                </ResponsiveContainer>
                                                            </div>
                                                        </>
                                                    );
                                                })()
                                            )}
                                        </>
                                    )}
                                </div>
                            </div>
```

A.3 WebSocket Data Handling

```
useEffect(() => {
        axios.get(`/api/device/${macAddress}`)
            .then(res => setDevice(res.data))
            .catch(err => console.error(err));

        // Fetch websocket data
        let socket;
        const connectWebSocket = () => {
            const ws = new WebSocket(
                "wss://olqh07fhd5.execute-api.ap-southeast-2.amazonaws.com/dev"
            );
```

```
            socket = ws;

            ws.onopen = () => {
                console.log("WebSocket connected");

                // Send macAddress to register
                ws.send(
                    JSON.stringify({
                        action: "sendMessage",
                        macAddress: macAddress,
                    })
                );
                setIsConnected(false); // still no data yet

            };

            let connectionTimeout;

            ws.onmessage = (event) => {
                try {
                    const raw = JSON.parse(event.data);

                    // Only consider real device data as connected
                    const hasValidData =
                        raw &&
                        (raw.cur !== undefined || raw.vol !== undefined || raw.pwr !==
undefined);

                    if (hasValidData) {
                        setIsConnected(true);

                        // Reset the timeout on each valid payload
                        clearTimeout(connectionTimeout);
                        connectionTimeout = setTimeout(() => {
                            setIsConnected(false); // No payload received in X seconds
                        }, 9000); // 7 seconds timeout
```

```
            // Map incoming keys (cur, vol, pwr) to your internal structure
             const mappedData = {
                 current: raw.cur,
                 voltage: raw.vol,
                 power: raw.pwr,
                 energy: raw.e_tot,
                 runtime: raw.run,
                 internal_temp: raw.int_t,
                 external_temp: raw.ext_t,
             };

             setDeviceData(mappedData);
             handleWebSocketData(event.data);
         }

    } catch (err) {
        console.error("Error parsing WebSocket message:", err);
        setIsConnected(false); // if parsing fails, show disconnected
    }
};

ws.onclose = (event) => {
    console.log("🔌 WebSocket closed", event.code);
    setIsConnected(false); // mark as disconnected
    // Optional: Reconnect on unexpected close
    if (event.code !== 1000) {
        setTimeout(connectWebSocket, 3000); //reconnect
    }
};

ws.onerror = (err) => {
    console.error("WebSocket error:", err);
    setIsConnected(false);
    ws.close();
```

```
            };
        };

        const fetchThreshold = async () => {
            try {
                const response = await axios.get(
                    `/device/${macAddress}/threshold`
                );
                setThreshold(response.data.threshold_kWh);
                setNewThreshold(response.data.threshold_kWh);
            } catch (error) {
                console.error("Error fetching threshold:", error);
            }
        };

        fetchThreshold();
        connectWebSocket();

        return () => {
            if (socket && socket.readyState === WebSocket.OPEN) {
                socket.close(1000, "Page closed");
            }
        };

    }, [macAddress]);



useEffect(() => {
        let sysSocket; // keep reference

        const connectSystemWS = () => {
            const ws = new WebSocket(
                "wss://7hybghy3aj.execute-api.ap-southeast-2.amazonaws.com/dev/"
            );
            sysSocket = ws;
```

```
ws.onopen = () => {
    console.log("System WebSocket connected");
    // Register macAddress
    ws.send(JSON.stringify({ action: "sendMessage", macAddress }));
};

ws.onmessage = (event) => {
    try {
        const raw = JSON.parse(event.data);

        // only process if it belongs to this mac
        if (raw.id === macAddress) {
            setSystemData({
                free_heap: raw.fh,
                min_free_heap: raw.mfh,
                max_alloc_heap: raw.mah,
                flash_chip_size_mb: raw.fcsm,
                sketch_size_kb: raw.ssk,
                free_sketch_space_kb: raw.fssk,
                cpu_mhz: raw.cm,
                chip_model: raw.cmdl,
                chip_revision: raw.cr,
                chip_cores: raw.cc,
                uptime_sec: raw.us,
                ping: raw.rssi,
            });
            setIsSystemConnected(true);
        }
    } catch (err) {
        console.error("Error parsing system WebSocket:", err);
    }
};

ws.onclose = () => {
```

```
            console.log("System WebSocket closed");
            setIsSystemConnected(false);
            // only reconnect if component is still mounted
            if (!ws._shouldClose) setTimeout(connectSystemWS, 3000);
        };

        ws.onerror = (err) => {
            console.error("System WS error:", err);
            ws.close();
        };

        // custom flag to prevent reconnection on unmount
        ws._shouldClose = false;
    };

    connectSystemWS();

    return () => {
        if (sysSocket) {
            sysSocket._shouldClose = true; // prevent reconnection
            if (sysSocket.readyState === WebSocket.OPEN) {
                sysSocket.close(1000, "Page closed");
            }
        }
    };
}, [macAddress]);
```

Appendix B: Color themes change on User Interfaces

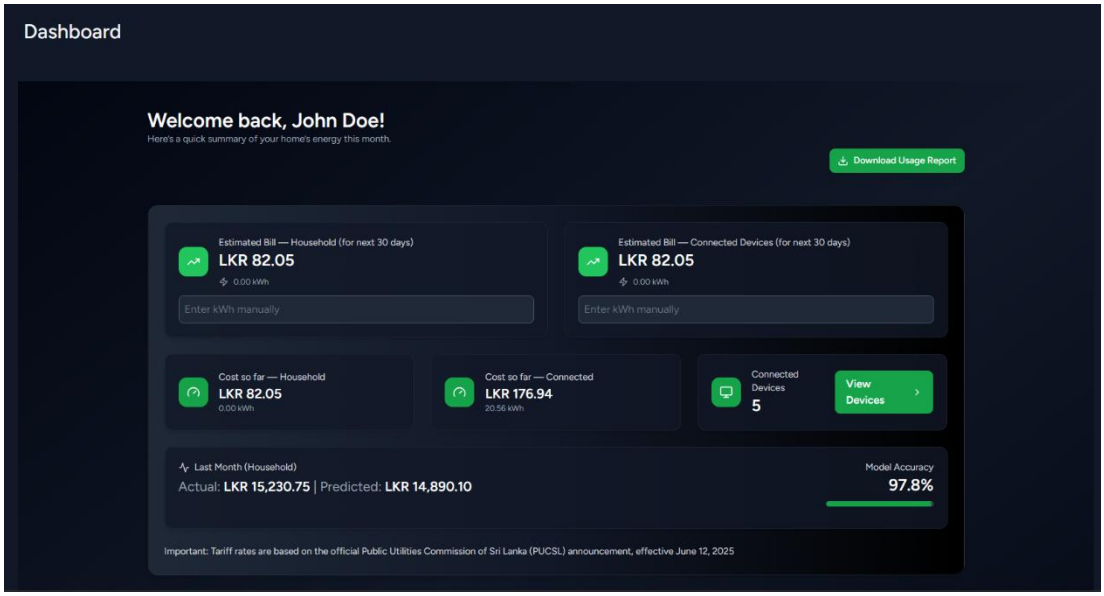

Figure 13: Default theme on Dashboard page.

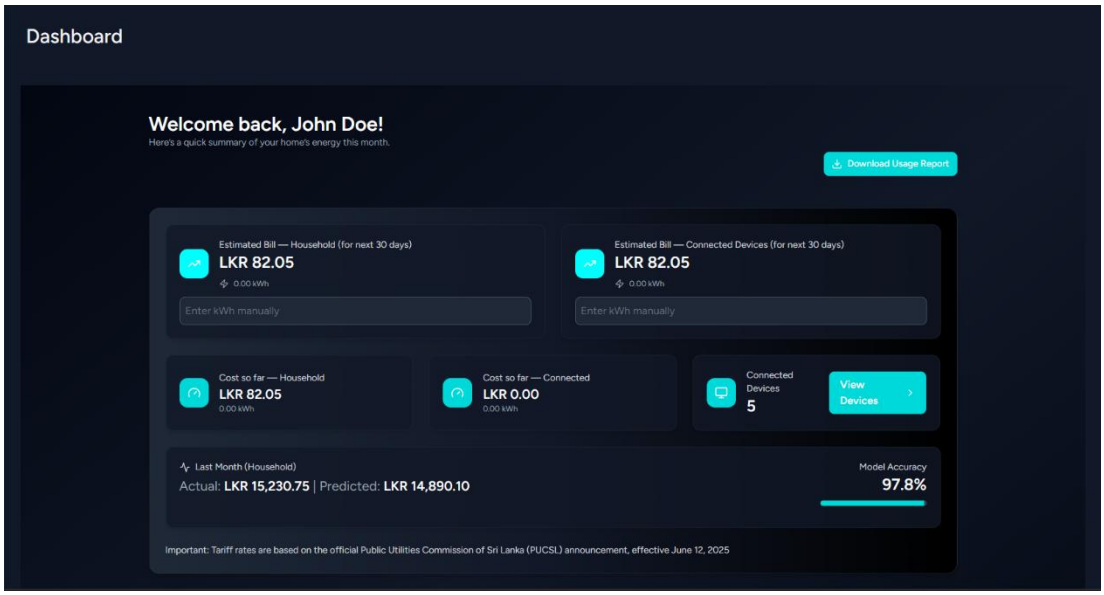

Figure 14: Protanopia color blind theme

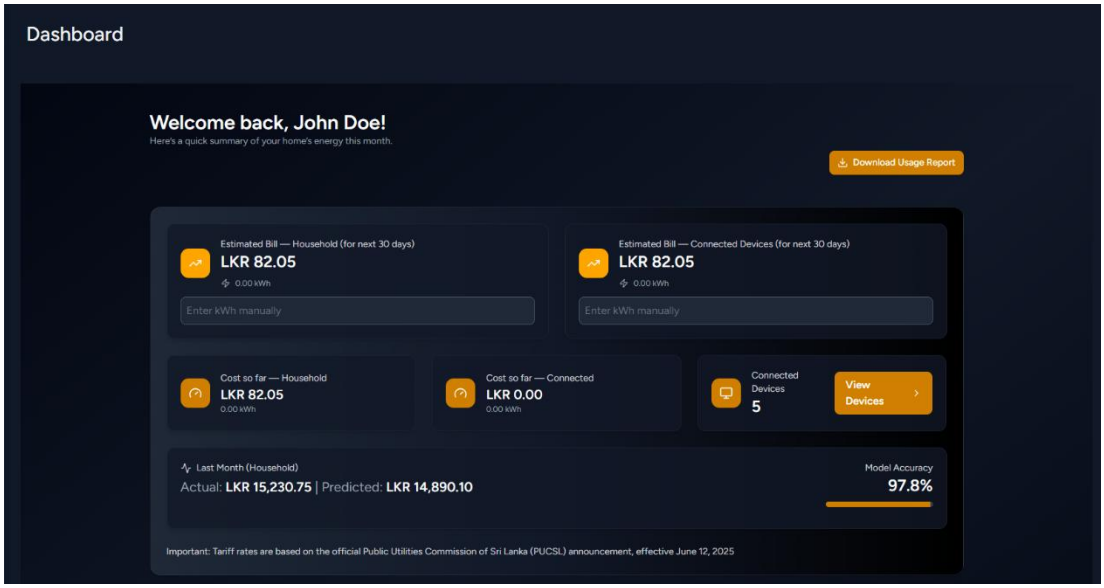

Figure 15: Deuteranopia color blind theme

Appendix C: Lambda functions

C.1 – Query handle in DynamoDB for a single device

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from datetime import datetime
from zoneinfo import ZoneInfo
from decimal import Decimal
import calendar

dynamodb = boto3.resource("dynamodb")
table = dynamodb.Table("SmartPlug_6D_DataSet_Tbl")

TZ = ZoneInfo("Asia/Colombo")
```

```
def month_bounds(year: int, month: int):
    """Return start_ts, end_ts (epoch seconds) for a given month in Asia/Colombo."""
    start = datetime(year, month, 1, 0, 0, 0, tzinfo=TZ)
    _, last_day = calendar.monthrange(year, month)
    end = datetime(year, month, last_day, 23, 59, 59, tzinfo=TZ)
    return int(start.timestamp()), int(end.timestamp())



def decimal_to_float(val):
    if isinstance(val, Decimal):
        return float(val)
    try:
        return float(val)
    except Exception:
        return 0.0



def collect_cycles(mac: str, start_ts: int = None, end_ts: int = None):
    """
    Query records in chronological order and split into cycles.

    A new cycle is started when:
      - current int == 1 (explicit start),
      - OR current int <= previous int (reset / rollover),
      - OR this is the first record (we start the first cycle).
    This avoids treating '3' as start if it appears after 1 (i.e., mid-cycle).
    """
    params = {
        "TableName": table.name,
        "KeyConditionExpression": Key("id").eq(mac),
        "ProjectionExpression": "ts, cur, pwr, e_tot, #i",
        "ExpressionAttributeNames": {"#i": "int"},
        "ScanIndexForward": True,  # chronological
```

```
    }

    if start_ts is not None and end_ts is not None:
        params["KeyConditionExpression"] = Key("id").eq(mac) & Key("ts").between(start_ts, end_ts)

    last_key = None
    cycles = []
    current_cycle = []
    prev_int = None

    while True:
        if last_key:
            params["ExclusiveStartKey"] = last_key
        resp = table.meta.client.query(**params)
        items = resp.get("Items", [])

        for it in items:
            # defensive conversions
            try:
                ts = int(it.get("ts", 0))
            except Exception:
                ts = 0
            try:
                intval = int(it.get("int", 0))
            except Exception:
                intval = 0

            rec = {
                "ts": ts,
                "int": intval,
                "cur": decimal_to_float(it.get("cur", 0)),
                "pwr": decimal_to_float(it.get("pwr", 0)),
                "e_tot": decimal_to_float(it.get("e_tot", 0)),
            }
```

```
            if not current_cycle:
                # start the first cycle
                current_cycle.append(rec)
            else:
                # decide if this record begins a new cycle
                # start if explicit marker 1 OR int decreased/reset (<= prev_int)
                if intval == 1 or (prev_int is not None and intval <= prev_int):
                    cycles.append(current_cycle)
                    current_cycle = [rec]
                else:
                    current_cycle.append(rec)

            prev_int = intval

        last_key = resp.get("LastEvaluatedKey")
        if not last_key:
            break

    # push last buffer
    if current_cycle:
        cycles.append(current_cycle)

    return cycles


def get_previous_cycle_averages(mac: str):
    """
    Return averages for the cycle BEFORE the most recent one (second-to-last).
    """
    cycles = collect_cycles(mac)

    if len(cycles) < 2:
        return {
            "average_current": 0,
            "average_power": 0,
```

```
            "samples": 0,
            "start_ts": None,
            "end_ts": None,
            "note": "Not enough cycles (need at least 2).",
        }

    prev_cycle = cycles[-2]  # second-to-last cycle

    count = len(prev_cycle)
    total_cur = sum(r["cur"] for r in prev_cycle)
    total_pwr = sum(r["pwr"] for r in prev_cycle)

    return {
        "average_current": round(total_cur / count, 5) if count else 0,
        "average_power": round(total_pwr / count, 7) if count else 0,
        "samples": count,
        "start_ts": prev_cycle[0]["ts"],
        "end_ts": prev_cycle[-1]["ts"],
    }


def get_monthly_energy(mac: str):
    """
    For the current calendar month, split cycles and sum the last e_tot of each cycle.
    """
    now = datetime.now(TZ)
    start_ts, end_ts = month_bounds(now.year, now.month)

    cycles = collect_cycles(mac, start_ts=start_ts, end_ts=end_ts)

    cycle_etots = []
    for c in cycles:
        if c:
            last_etot = c[-1].get("e_tot", 0)
            # only include positive/meaningful e_tot values
```

```
            if last_etot:
                cycle_etots.append(decimal_to_float(last_etot))

    total_energy = sum(cycle_etots)

    return {
        "cycles": len(cycle_etots),
        "cycle_etots": cycle_etots,
        "total_energy": round(total_energy, 6),
    }



def get_monthly_averages(mac: str):
    """
    For the current calendar month, return average current and power.
    """
    now = datetime.now(TZ)
    start_ts, end_ts = month_bounds(now.year, now.month)

    records = []
    params = {
        "TableName": table.name,
        "KeyConditionExpression":  Key("id").eq(mac)  &  Key("ts").between(start_ts,
end_ts),
        "ProjectionExpression": "ts, cur, pwr",
        "ScanIndexForward": True,
    }

    last_key = None
    while True:
        if last_key:
            params["ExclusiveStartKey"] = last_key
        resp = table.meta.client.query(**params)
        items = resp.get("Items", [])

        for it in items:
```

```
            records.append({
                "cur": decimal_to_float(it.get("cur", 0)),
                "pwr": decimal_to_float(it.get("pwr", 0)),
            })

        last_key = resp.get("LastEvaluatedKey")
        if not last_key:
            break

    count = len(records)
    if count == 0:
        return {
            "average_current": 0,
            "average_power": 0,
            "samples": 0,
        }

    total_cur = sum(r["cur"] for r in records)
    total_pwr = sum(r["pwr"] for r in records)

    return {
        "average_current": round(total_cur / count, 5),
        "average_power": round(total_pwr / count, 7),
        "samples": count,
    }



def lambda_handler(event, context):
    mac = (event.get("pathParameters", {}) or {}).get("mac")
    if not mac:
        return {
            "statusCode": 400,
            "headers": {"Content-Type": "application/json"},
            "body": json.dumps({"error": "MAC address is required"}),
        }
```

```
    prev_cycle = get_previous_cycle_averages(mac)
    monthly_energy = get_monthly_energy(mac)
    monthly_avg = get_monthly_averages(mac)

    body = {
        "mac": mac,
        "timezone": "Asia/Colombo",
        "previous_cycle": prev_cycle,
        "monthly_energy": monthly_energy,
        "monthly_average": monthly_avg,
    }

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(body),
    }
```

C.2 – Query handle in Athena for a single device

```
import boto3
import os
import time
import json
from datetime import datetime, timedelta

ATHENA_DATABASE = os.environ["ATHENA_DATABASE"]
ATHENA_TABLE = os.environ["ATHENA_TABLE"]
ATHENA_OUTPUT = os.environ["ATHENA_OUTPUT"]

athena = boto3.client("athena")

def run_athena_query(query):
```

```
    """Run Athena query and return all rows (excluding header)"""
    response = athena.start_query_execution(
        QueryString=query,
        QueryExecutionContext={"Database": ATHENA_DATABASE},
        ResultConfiguration={"OutputLocation": ATHENA_OUTPUT},
    )
    query_execution_id = response["QueryExecutionId"]

    # Wait for completion
    while True:
        status = athena.get_query_execution(QueryExecutionId=query_execution_id)
        state = status["QueryExecution"]["Status"]["State"]
        if state in ["SUCCEEDED", "FAILED", "CANCELLED"]:
            break
        time.sleep(1)

    if state != "SUCCEEDED":
        reason = status["QueryExecution"]["Status"].get("StateChangeReason", "Unknown
reason")
        raise Exception(f"Query failed: {state} - {reason}")

    # Fetch results
    result = athena.get_query_results(QueryExecutionId=query_execution_id)
    rows = result["ResultSet"]["Rows"]

    # Convert rows to list of dicts
    headers = [col["VarCharValue"] for col in rows[0]["Data"]]
    data = []
    for row in rows[1:]:
        values = [col.get("VarCharValue", None) for col in row["Data"]]
        data.append(dict(zip(headers, values)))

    return data

def lambda_handler(event, context):
    mac = event.get("pathParameters", {}).get("mac")
```

```
    if not mac:
        return {"statusCode": 400, "body": "MAC address required"}


    ###  Monthly Averages Query (for month selection dropdown)
    month_query = f"""
        SELECT
              date_trunc('month', from_unixtime(ts) AT TIME ZONE 'Asia/Colombo') AS
month,
            AVG(CAST(cur AS DOUBLE)) AS avg_current,
            AVG(CAST(pwr AS DOUBLE)) AS avg_power
        FROM {ATHENA_TABLE}
        WHERE id = '{mac}'
          AND from_unixtime(ts) AT TIME ZONE 'Asia/Colombo' >= date_add('month', -3,
date_trunc('month', current_timestamp AT TIME ZONE 'Asia/Colombo'))
        GROUP BY 1
        ORDER BY month DESC
    """
    monthly_results = run_athena_query(month_query)
    months_data = [
        {
            "month": row["month"],
            "avg_current": float(row["avg_current"]) if row["avg_current"] else 0,
            "avg_power": float(row["avg_power"]) if row["avg_power"] else 0,
        }
        for row in monthly_results
    ]


    ###  Daily Averages Query (for charts over days)
    daily_query = f"""
        SELECT
            date_format(from_unixtime(ts) AT TIME ZONE 'Asia/Colombo', '%Y-%m-%d') AS
day,
            AVG(CAST(cur AS DOUBLE)) AS avg_current,
            AVG(CAST(pwr AS DOUBLE)) AS avg_power
        FROM {ATHENA_TABLE}
        WHERE id = '{mac}'
```

```
          AND from_unixtime(ts) AT TIME ZONE 'Asia/Colombo' >= date_add('month', -4,
date_trunc('month', current_timestamp AT TIME ZONE 'Asia/Colombo'))

        GROUP BY 1

        ORDER BY day ASC

    """

    daily_results = run_athena_query(daily_query)


    # Build daily dictionary

    data_dict = {

        row["day"]: {

            "avg_current": float(row["avg_current"]) if row["avg_current"] else 0,

            "avg_power": float(row["avg_power"]) if row["avg_power"] else 0,

        }

        for row in daily_results

    }


    # Generate last 3 full months daily data (excluding current month)

    today = datetime.now()

    daily_avg_data = {}

    for m in range(1, 4):  # 1 to 3 → skip current month

        # Get the first day of the month 'm' months ago

          month_date = (today.replace(day=1) - timedelta(days=1)).replace(day=1) -
timedelta(days=30*(m-1))

        month_str = month_date.strftime("%Y-%m")


        # Calculate number of days in the month correctly

        next_month = (month_date.replace(day=28) + timedelta(days=4)).replace(day=1)

        days_in_month = (next_month - timedelta(days=1)).day


        daily_avg_data[month_str] = {}

        for day in range(1, days_in_month + 1):

            day_key = f"{month_str}-{str(day).zfill(2)}"

            if day_key in data_dict:

                daily_avg_data[month_str][day_key] = data_dict[day_key]

            else:

                 daily_avg_data[month_str][day_key] = {"avg_current": 0, "avg_power":
0}
```

```
    # Final response
    response_body = {
        "mac": mac,
        "months": months_data,               # For dropdown / last month charts
       "daily_avg_data": daily_avg_data     # For daily charts over last 3 full months
    }

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(response_body, default=str),
    }
```

C.3 – Query handle in DynamoDB for a multiple device

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from datetime import datetime
from zoneinfo import ZoneInfo
from decimal import Decimal
import calendar

# DynamoDB setup
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('SmartPlug_6D_DataSet_Tbl')

TZ = ZoneInfo("Asia/Colombo")


def decimal_to_float(val):
```

```
    if isinstance(val, Decimal):
        return float(val)
    try:
        return float(val)
    except Exception:
        return 0.0


def month_bounds(year: int, month: int):
    start = datetime(year, month, 1, 0, 0, 0, tzinfo=TZ)
    _, last_day = calendar.monthrange(year, month)
    end = datetime(year, month, last_day, 23, 59, 59, tzinfo=TZ)
    return int(start.timestamp()), int(end.timestamp())


def collect_cycles(mac: str, start_ts=None, end_ts=None):
    """
    Fetch records and split into cycles.
    Cycle starts when:
      - int == 1 (explicit start)
      - int <= previous int (rollover/reset)
      - first record
    """
    params = {
        "TableName": table.name,
        "KeyConditionExpression": Key("id").eq(mac),
        "ProjectionExpression": "ts, e_tot, #i",
        "ExpressionAttributeNames": {"#i": "int"},
        "ScanIndexForward": True
    }

    if start_ts is not None and end_ts is not None:
        params["KeyConditionExpression"] = Key("id").eq(mac) &
Key("ts").between(start_ts, end_ts)

    last_key = None
```

```
cycles = []
current_cycle = []
prev_int = None

while True:
    if last_key:
        params["ExclusiveStartKey"] = last_key
    resp = table.meta.client.query(**params)
    items = resp.get("Items", [])

    for it in items:
        ts = int(it.get("ts", 0))
        intval = int(it.get("int", 0))
        e_tot = decimal_to_float(it.get("e_tot", 0))
        rec = {"ts": ts, "int": intval, "e_tot": e_tot}

        if not current_cycle:
            current_cycle.append(rec)
        else:
            if intval == 1 or (prev_int is not None and intval <= prev_int):
                cycles.append(current_cycle)
                current_cycle = [rec]
            else:
                current_cycle.append(rec)

        prev_int = intval

    last_key = resp.get("LastEvaluatedKey")
    if not last_key:
        break

if current_cycle:
    cycles.append(current_cycle)

return cycles
```

```
def calculate_cycles_energy(cycles):
    """Sum last e_tot of each cycle."""
    total_energy = sum(cycle[-1]["e_tot"] for cycle in cycles if cycle)
    return round(total_energy, 6)



def lambda_handler(event, context):
    # Parse input JSON
    macs = []
    try:
        if "body" in event and isinstance(event["body"], str):
            body = json.loads(event["body"])
            macs = body.get("macs", [])
        elif "macs" in event and isinstance(event["macs"], list):
            macs = event["macs"]
    except Exception as e:
        print("Error parsing input:", e)
        return {
            "statusCode": 400,
            "headers": {"Content-Type": "application/json"},
            "body": json.dumps({"error": "Invalid input"})
        }

    if not macs:
        return {
            "statusCode": 400,
            "headers": {"Content-Type": "application/json"},
            "body": json.dumps({"error": "No MAC addresses provided"})
        }

    now = datetime.now(TZ)
    start_ts, end_ts = month_bounds(now.year, now.month)
```

```
    results = {}
    overall_total = 0

    for mac in macs:
        cycles = collect_cycles(mac, start_ts=start_ts, end_ts=end_ts)
        total_energy = calculate_cycles_energy(cycles)
        results[mac] = total_energy
        overall_total += total_energy

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps({
            "per_mac_total_energy": results,
            "overall_total_energy": round(overall_total, 6)
        })
    }
```

C.4 – Query handle in Athena for a multiple device

```
import boto3
import os
import time
import json

ATHENA_DATABASE = os.environ["ATHENA_DATABASE"]
ATHENA_TABLE = os.environ["ATHENA_TABLE"]
ATHENA_OUTPUT = os.environ["ATHENA_OUTPUT"]

athena = boto3.client("athena")

def run_athena_query(query, poll_interval=1):
    # Start query execution
    resp = athena.start_query_execution(
```

```
        QueryString=query,
        QueryExecutionContext={"Database": ATHENA_DATABASE},
        ResultConfiguration={"OutputLocation": ATHENA_OUTPUT},
    )
    qid = resp["QueryExecutionId"]

    # Wait until query completes
    while True:
        status = athena.get_query_execution(QueryExecutionId=qid)
        state = status["QueryExecution"]["Status"]["State"]
        if state in ("SUCCEEDED", "FAILED", "CANCELLED"):
            break
        time.sleep(poll_interval)

    if state != "SUCCEEDED":
        reason = status["QueryExecution"]["Status"].get("StateChangeReason",
"Unknown")
        raise Exception(f"Athena query failed: {state} - {reason}")

    # Fetch results
    result = athena.get_query_results(QueryExecutionId=qid)
    rows = result["ResultSet"]["Rows"]

    # First row is header
    header = [c.get("VarCharValue") for c in rows[0]["Data"]]
    data_rows = []
    for row in rows[1:]:
        values = [c.get("VarCharValue") for c in row["Data"]]
        data_rows.append(dict(zip(header, values)))

    return data_rows


def lambda_handler(event, context):
    print("Received event:", event)  # Debug log
```

```
# Handle API Gateway v2 (body as string) OR direct Lambda Test JSON
body = event
if "body" in event and event["body"]:
    try:
        body = json.loads(event["body"])
    except Exception as e:
        return {
            "statusCode": 400,
            "body": json.dumps({"message": f"Invalid JSON body: {str(e)}"})
        }

macs = body.get("macs", [])
month = body.get("month")

if not macs:
    return {
        "statusCode": 400,
        "body": json.dumps({"message": "Please provide 'macs' in event JSON"})
    }

# Build SQL IN clause
mac_list_sql = ",".join(f"'{m}'" for m in macs)

query = f"""
WITH month_data AS (
    SELECT
        id,
        cur,
        from_unixtime(ts) AS ts_dt
    FROM {ATHENA_TABLE}
    WHERE id IN ({mac_list_sql})
    AND month(from_unixtime(ts)) = {month}
)
SELECT
    id,
```

```
    avg(cur) AS avg_current
FROM month_data
WHERE year(ts_dt) = (
    SELECT max(year(from_unixtime(ts)))
    FROM {ATHENA_TABLE}
    WHERE month(from_unixtime(ts)) = {month}
)
GROUP BY id
"""

try:
    results = run_athena_query(query)
    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(results, default=str)
    }
except Exception as e:
    return {
        "statusCode": 500,
        "body": json.dumps({"error": str(e)})
    }
```

Appendix D: Dataset Column Descriptions

Column	Description	Data Type	Example
vol	Voltage level from the smart plug.	Float	236.65
ts	UNIX timestamp of the data record.	Integer	1750352568

e_int	Energy used in the 5-second interval (kWh).	Float	0.000056
id	Unique MAC address of the IoT device.	String	F4:65:0B:E9:3C:44
cur	Current drawn by the appliance (Amps).	Float	0.093
e_tot	Cumulative total energy consumed (kWh).	Float	0.000158

Appendix E: Model configuration and Hyperparameters

Model**:** XGBoost Regressor

Key Hyperparameters:

- n_estimators: 500
- learning_rate: 0.08
- max_depth: 6
- subsample: 0.8
- colsample_bytree: 0.8
- early_stopping_rounds: 50

Appendix F: UI

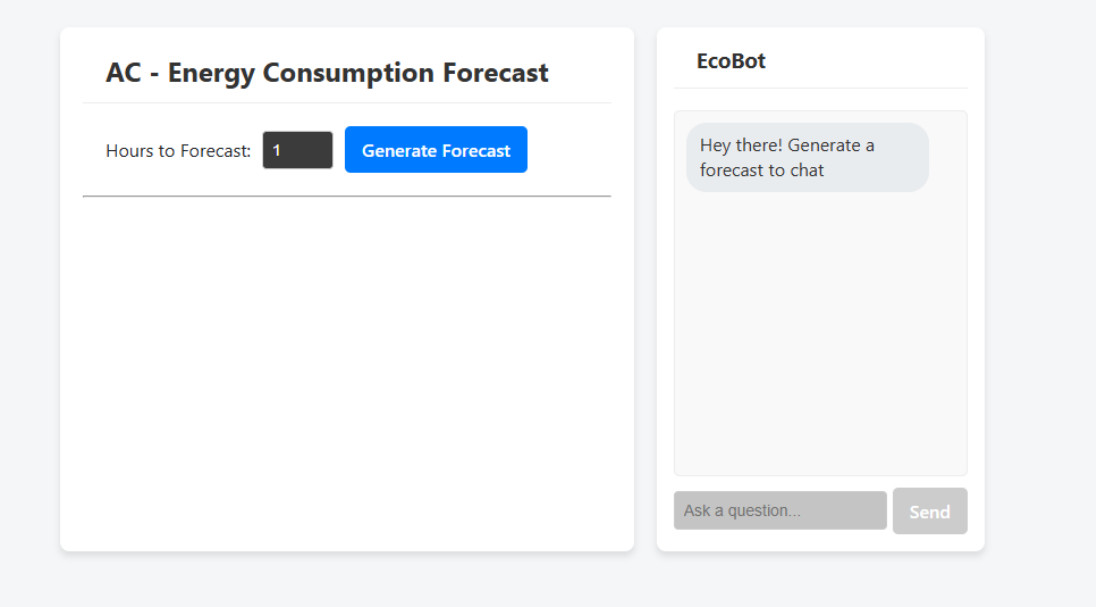

Figure 16: Home page to select the hours to predict

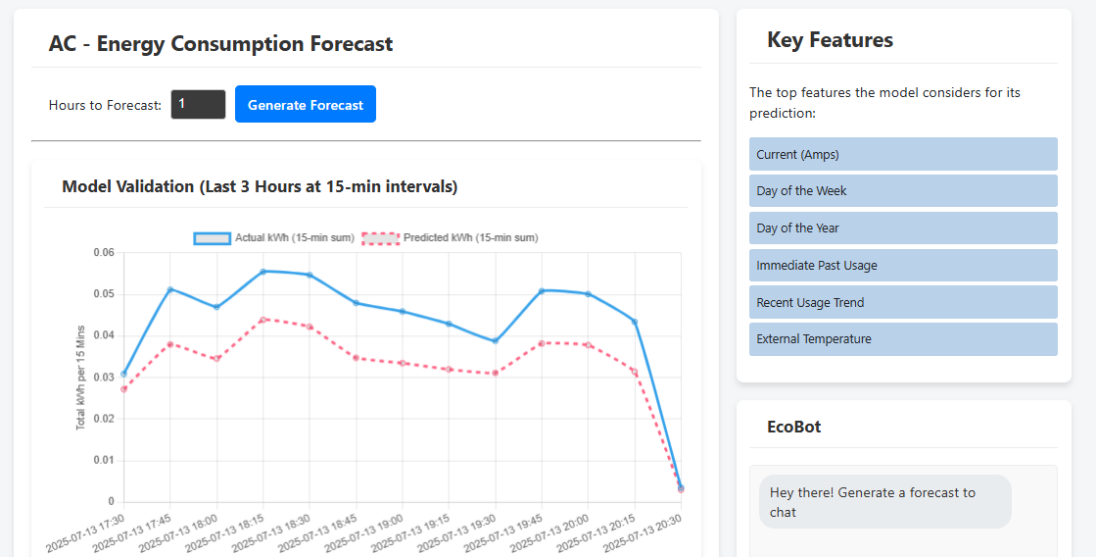

Figure 17: Prediction output page with model validation, key factors and ecobot to chat with

Appendix G: Arduino Codes

G.1 – ESP32 Wi-Fi connectivity

```
void startCaptivePortal() {
    Serial.println("Starting Captive Portal...");
    WiFiManager wifiManager;
    wifiManager.resetSettings();
    wifiManager.setConfigPortalTimeout(0);
    if (!wifiManager.autoConnect("ESP32_Config")) {
        Serial.println("Failed to Connect via Captive Portal!");
    } else {
        Serial.println("Connected via captive portal!");
        saveWiFiCredentials();
    }
}

void saveWiFiCredentials() {
    String newSSID = WiFi.SSID();
    String newPASS = WiFi.psk();
    if (newSSID != "" && newPASS != "") {
        Serial.println("Saving New WiFi credentials...");
        preferences.begin("wifi", false);
        preferences.putString("ssid", newSSID);
        preferences.putString("password", newPASS);
        preferences.end();
        Serial.println("WiFi Credentials Saved!");
    } else {
        Serial.println("No valid WiFi Credentials Found!");
    }
}
void resetWiFiSettings() {
    Serial.println("Deleting Saved WiFi Credentials...");
    WiFi.disconnect(true, true);
```

```
    delay(500);
    WiFi.mode(WIFI_OFF);
    delay(500);
    WiFi.mode(WIFI_STA);
    preferences.begin("wifi", false);
    preferences.clear();
    preferences.end();
    delay(1000);
    WiFiManager wifiManager;
    wifiManager.resetSettings();
    Serial.println("WiFi Credentials Deleted!");
    preferences.begin("wifi", false);
    String checkSSID = preferences.getString("ssid", "");
    if (checkSSID == "") {
        Serial.println("Double Check: WiFi Credentials are 100% Cleared!");
    } else {
        Serial.println("ERROR: WiFi Credential NOT Deleted!");
    }
    preferences.end();
    delay(2000);
    Serial.println("Restarting ESP32...");
    blinkLED(100, 10);
    ESP.restart();
}
void setup() {
    Serial.begin(115200);
    Serial.println("\n\nESP32 Advanced Captive Portal Setup Starting...");
    pinMode(BOOT_PIN, INPUT_PULLUP);
    pinMode(LED_PIN, OUTPUT);
    blinkLED(200, 5);

    // Initialize Relay - Start with bulb ON
    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, HIGH); // Active-low relay, COM-NO: LOW = bulb ON
    relayState = true;
```

```
    Serial.println("Relay initialized: Bulb ON");


    preferences.begin("wifi", false);
    if (digitalRead(BOOT_PIN) == LOW) {
        delay(1000);
        if (digitalRead(BOOT_PIN) == LOW) {
            Serial.println("Boot Button Held! RESETTING WiFi Credentials...");
            resetWiFiSettings();
        }
    }
    String savedSSID = preferences.getString("ssid", "");
    String savedPASS = preferences.getString("password", "");
    if (savedSSID != "" && savedPASS != "") {
        Serial.println("Connecting to Saved WiFi: " + savedSSID);
        WiFi.begin(savedSSID.c_str(), savedPASS.c_str());
        int attempt = 0;
        while (WiFi.status() != WL_CONNECTED && attempt < 15) {
            Serial.println(".");
            digitalWrite(LED_PIN, !digitalRead(LED_PIN));
            delay(500);
            attempt++;
        }
        if (WiFi.status() == WL_CONNECTED) {
            Serial.println("\nWiFi Connected Successfully!");
            Serial.println("Local IP Address: ");
            Serial.println(WiFi.localIP());
            digitalWrite(LED_PIN, HIGH);
            fetchThreshold();
        } else {
            Serial.println("\nWiFi Connection Failed! Switching to captive
portal...");
            startCaptivePortal();
        }
    } else {
        Serial.println("No saved WiFi credentials. Starting captive portal...");
        startCaptivePortal();
```

```
    }
    // Initialize ADC
    analogReadResolution(12);
    analogSetAttenuation(ADC_11db);

    Serial.println("Stabilizing system...");
    delay(5000); // Keep bulb ON during stabilization
    previousMillis = millis();
    timeClient.begin();
    dht.begin();
    ds18b20.begin();
    connectAWS();
}
```

G.2 – Power calculations using sensor modules

```
float getRMSVoltage() {
    float sumSq = 0;
    for (int i = 0; i < NUM_SAMPLES; i++) {
        int rawValue = analogRead(VOLTAGE_SENSOR_PIN);
        float voltage = (rawValue / 4095.0) * 3.3 * CALIBRATION_FACTOR;
        sumSq += voltage * voltage;
        delay(2);
    }
    return sqrt(sumSq / NUM_SAMPLES);
}

float getZeroPoint() {
    int totalSamples = 500;
    float sum = 0;
    for (int i = 0; i < totalSamples; i++) {
        int rawValue = analogRead(SENSOR_PIN);
        float voltage = (rawValue / 4095.0) * 3.3;
        sum += voltage;
```

```
        delayMicroseconds(1000);
    }
    float avgZeroPoint = sum / totalSamples;
    if (avgZeroPoint < 2.0 || avgZeroPoint > 2.5) {
        Serial.println("Warning: Unstable zero point detected. Re-calibrating...");
        return zeroPoint;
    }
    return avgZeroPoint;
}

float getRMSCurrent() {
    static float zeroPointHistory[AVERAGE_WINDOW] = {2.38};
    static int zeroPointIndex = 0;
    float newZeroPoint = getZeroPoint();
    if (newZeroPoint < 2.1) {
        Serial.println("Warning: Zero point too low. Skipping current measurement.");
        return 0.0;
    }
    zeroPointHistory[zeroPointIndex] = newZeroPoint;
    zeroPointIndex = (zeroPointIndex + 1) % AVERAGE_WINDOW;
    float zeroSum = 0;
    for (int i = 0; i < AVERAGE_WINDOW; i++) {
        zeroSum += zeroPointHistory[i];
    }
    zeroPoint = zeroSum / AVERAGE_WINDOW;
    Serial.print("Updated Zero Point: ");
    Serial.println(zeroPoint, 3);
    float sumSq = 0;
    for (int i = 0; i < SAMPLE_COUNT; i++) {
        int rawValue = analogRead(SENSOR_PIN);
        float voltage = (rawValue / 4095.0) * 3.3;
        float current = abs((voltage - zeroPoint) / sensitivity);
        sumSq += current * current;
        delayMicroseconds(1000);
    }
```

```
            return sqrt(sumSq / SAMPLE_COUNT);
        }
```

G.3 – Control relay module

```
void checkThresholdAndControlRelay() {
    if (totalEnergyConsumption >= threshold_kWh) {
        if (relayState) { // If currently ON
            relayState = false;
            digitalWrite(RELAY_PIN, LOW); // Turn OFF (open circuit, bulb off)
            Serial.println("Relay turned OFF - Threshold reached");
        }
    } else {
        if (!relayState) { // If currently OFF
            relayState = true;
            digitalWrite(RELAY_PIN, HIGH); // Turn ON (close circuit, bulb on)
            Serial.println("Relay turned ON - Below threshold");
        }
    }
    Serial.print("Relay state (0=ON, 1=OFF): "); Serial.println(digitalRead(RELAY_PIN));
}
```

G.4 – Monitor the energy consumption using energy meter

```
// ------------------ Setup ------------------
void setup() {
  Serial.begin(115200);
  delay(2000);
  Serial.println("Starting ESP32...");

  // Init PZEM (RS485)
  pzemSerial.begin(9600, SERIAL_8N1, 16, 17); // RX=16, TX=17
  pzem.begin(1, pzemSerial); // Slave ID = 1
```

```
  connectAWS();
  timeClient.begin();
}

// ------------------ Energy Calculation ------------------
double calculateTotalEnergy(double rawEnergy) {
  // rawEnergy = energy value directly from PZEM (kWh)

  // Capture first valid energy reading as baseline
  if (energyOffset < 0) {
    energyOffset = rawEnergy;
  }

  // Calculate total consumed since device boot
  double total = rawEnergy - energyOffset;
  if (total < 0) total = 0.0; // avoid negatives

  // Round to 6 decimal places
  total = round(total * 1000000.0) / 1000000.0;

  return total;
}
```

Appendix H: PCB Schematic Diagram

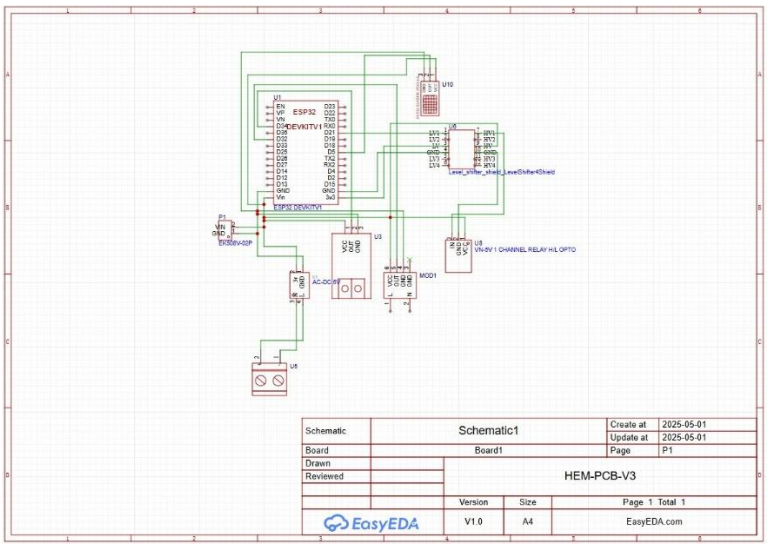

Figure 18: PCB Schematic Diagram

Appendix I: AWS IoT Core

I.1 – Create IoT thing

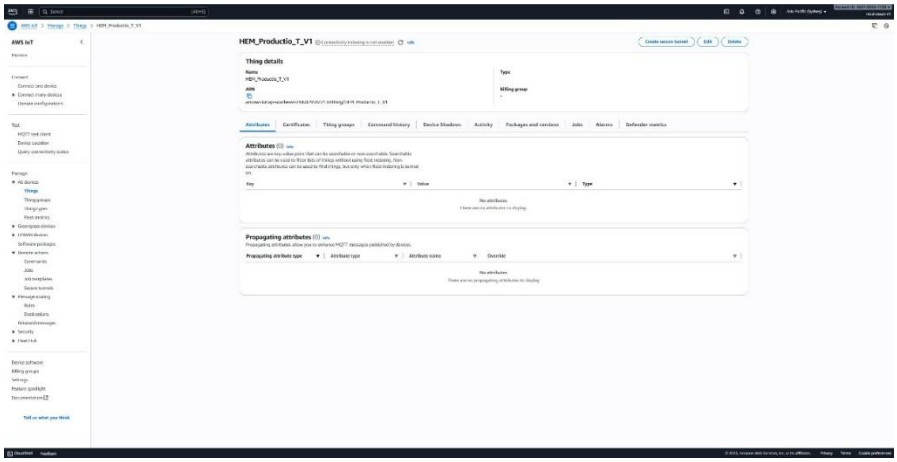

Figure 19: Create IoT thing

I.2 – Create IoT rule

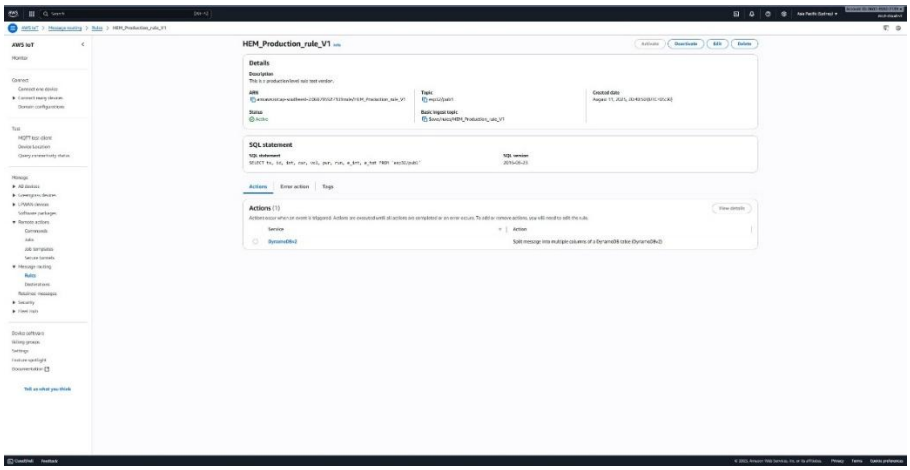

Figure 20: Create IoT rule

Appendix J: MQTT Protocol

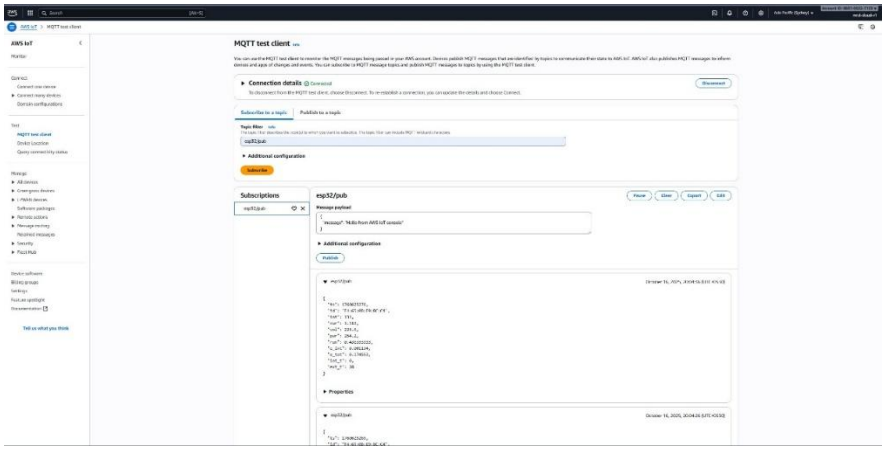

Figure 21: MQTT Protocol

Appendix K: DynamoDB Table

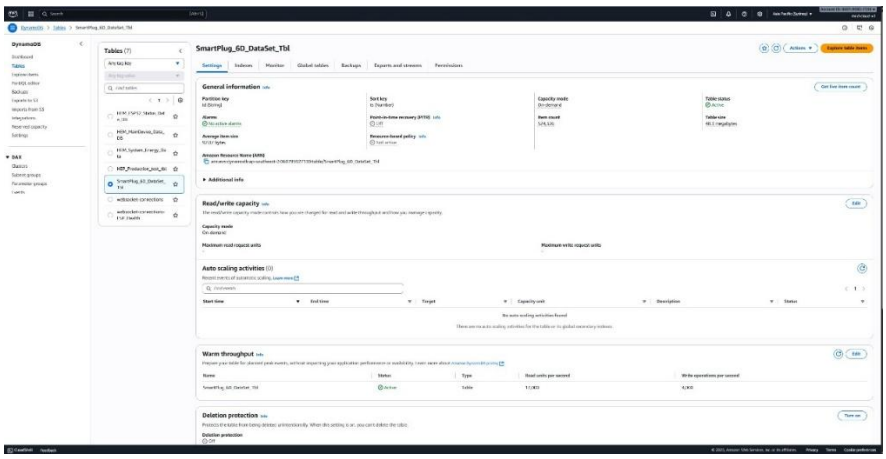

Figure 22: Table meta data

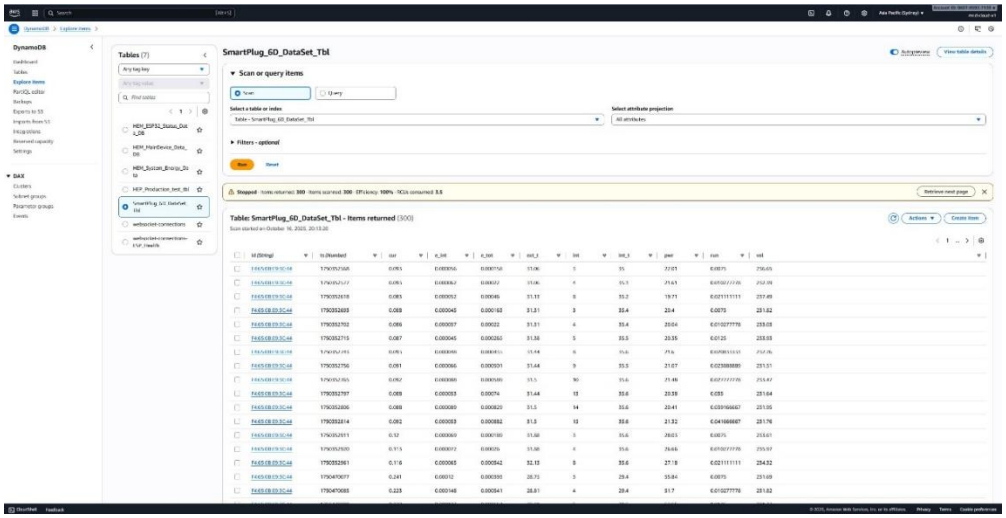

Figure 23: Table with data records

Appendix L: Classification ML Model

```
#Install required packages
!pip install sdv==1.24.1
!pip install scikit-learn
!pip install pandas matplotlib seaborn
!pip install joblib

#Import libraries
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
from google.colab import files
import matplotlib.pyplot as plt

uploaded = files.upload()
df = pd.read_csv("pzem-250K-dataset-v1-ReadyToUse.csv")
print(df.head())

id_encoder = LabelEncoder()
scenario_encoder = LabelEncoder()

df['id'] = id_encoder.fit_transform(df['id'])
df['scenario'] = scenario_encoder.fit_transform(df['scenario'])

#Save encoders
joblib.dump(id_encoder, 'id_encoder.pkl')
joblib.dump(scenario_encoder, 'scenario_encoder.pkl')

#Features and target
X = df.drop(columns=['scenario'])
y = df['scenario']
```

```
#Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


#Train a lightweight model
clf = DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, random_state=42)
clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", round(accuracy_score(y_test, y_pred) * 100, 2), "%")


#Save the model
joblib.dump(clf, 'scenario_classifier.pkl')


#Real-time Prediction Example
clf = joblib.load('scenario_classifier.pkl')
id_encoder = joblib.load('id_encoder.pkl')
scenario_encoder = joblib.load('scenario_encoder.pkl')


new_data = pd.DataFrame([{
    "id": "68:25:DD:32:CC:E4",
    "current": 0.397,
    "pf": 1.0,
    "power": 92,
    "scenario": "",  # Ignored
    "voltage": 161.5,
    "energy": ""     # Ignored
}])


#Clean & round input
new_data = new_data.round(3)
new_data['pf'] = new_data['pf'].round(1)
new_data['power'] = new_data['power'].round(1)
new_data['voltage'] = new_data['voltage'].round(1)
```

```
# Encode ID
device_id = new_data['id'].iloc[0]
if device_id in id_encoder.classes_:
    encoded_id = id_encoder.transform([device_id])[0]
    new_data.loc[0, 'id'] = encoded_id
else:
    print("Unknown device ID:", device_id)
    exit()

#Predict
features_used = ['id', 'current', 'pf', 'power', 'voltage']
X_new = new_data[features_used]
prediction = clf.predict(X_new)
scenario_label = scenario_encoder.inverse_transform(prediction)

print("Predicted Scenario:", scenario_label[0])


#Download models
files.download('scenario_classifier.pkl')
files.download('id_encoder.pkl')
files.download('scenario_encoder.pkl')
```

Appendix M: LLM fine tuning process

M.1 – RunPod used to finetune the model

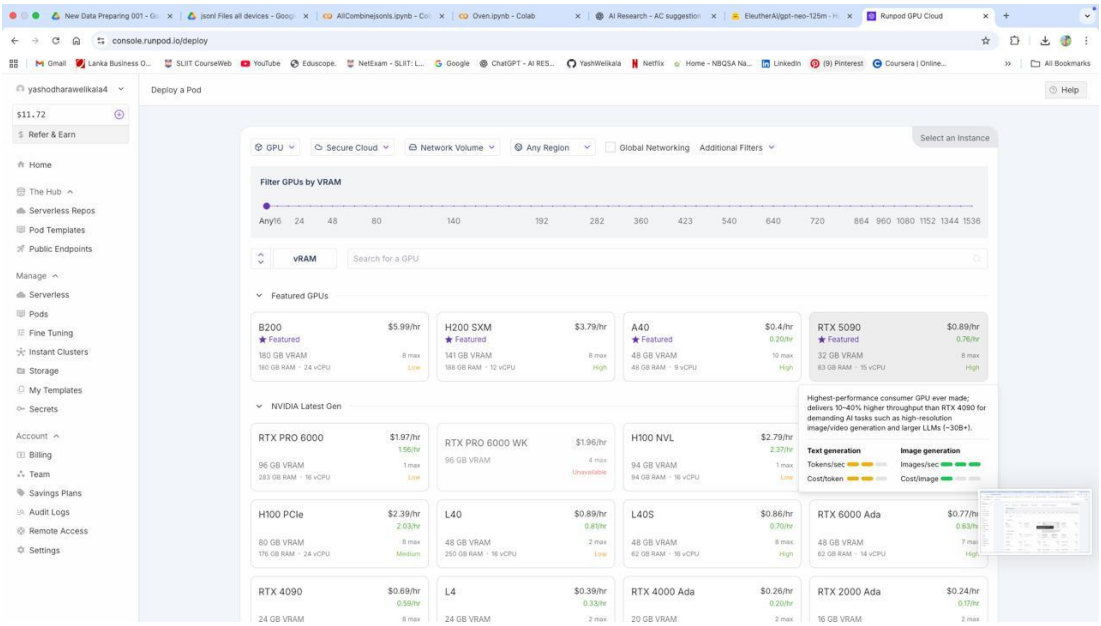

Figure 24: RunPod used to finetune the model

M.2 – GPT-Neo LoRA Tokenization

```
from transformers import AutoTokenizer, AutoModelForCausalLM

base_model = "EleutherAI/gpt-neo-1.3B"
tokenizer = AutoTokenizer.from_pretrained(base_model)
model = AutoModelForCausalLM.from_pretrained(base_model)


from peft import get_peft_model, LoraConfig, TaskType

peft_config = LoraConfig(
    task_type=TaskType.CAUSAL_LM,
```

```
    inference_mode=False,
    r=8,
    lora_alpha=32,
    lora_dropout=0.1
)

model = get_peft_model(model, peft_config)
model.print_trainable_parameters()



from transformers import AutoTokenizer

# Load the tokenizer and assign a pad_token (important for GPT-Neo)
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-1.3B")
tokenizer.pad_token = tokenizer.eos_token  #Fix: define pad_token

# Tokenization function
def tokenize(example):
    return tokenizer(example["text"], padding="max_length", truncation=True,
max_length=512)

# Apply tokenization to dataset
tokenized_dataset = formatted_dataset.map(tokenize, batched=True)

# Set dataset format for PyTorch
tokenized_dataset.set_format(type="torch", columns=["input_ids", "attention_mask"])
```

M.3 – TrainingArguments Setup

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./gptneo-lora-output01",
```

```
    num_train_epochs=3,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=2,
    learning_rate=2e-4,
    fp16=True,  # if your GPU supports it
    logging_dir="./logs",
    logging_steps=50,
    save_strategy="steps",
    save_steps=500,
    save_total_limit=2,
    report_to="none"  # Set to "wandb" if using Weights & Biases
)



from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    tokenizer=tokenizer

)

trainer.train()
```

M.4 – Upload Progress (JupyterLab)

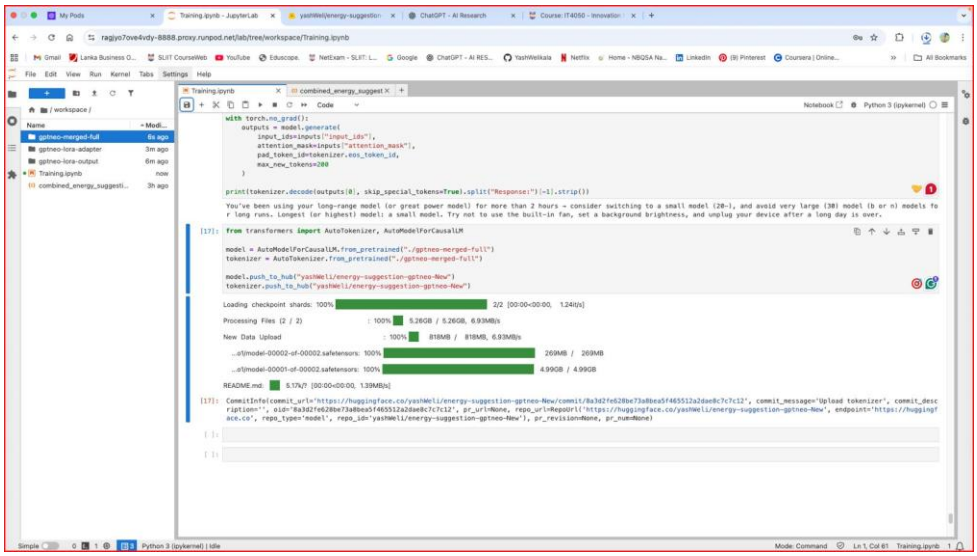

Figure 25: Upload Progress (JupyterLab)

M.5 – Model Card Finetuned GPT-Neo

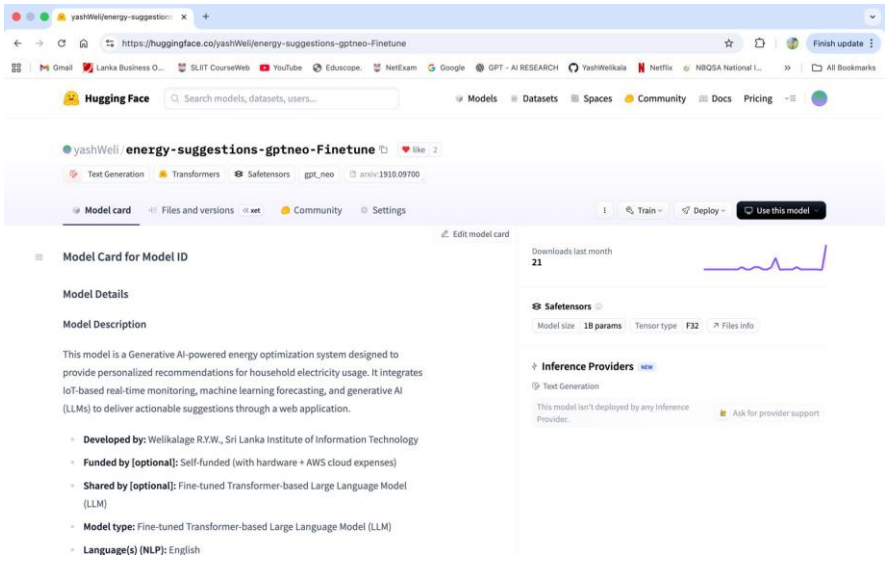

Figure 26: Model Card Finetuned GPT-Neo

M.6 – Energy Tips API Response

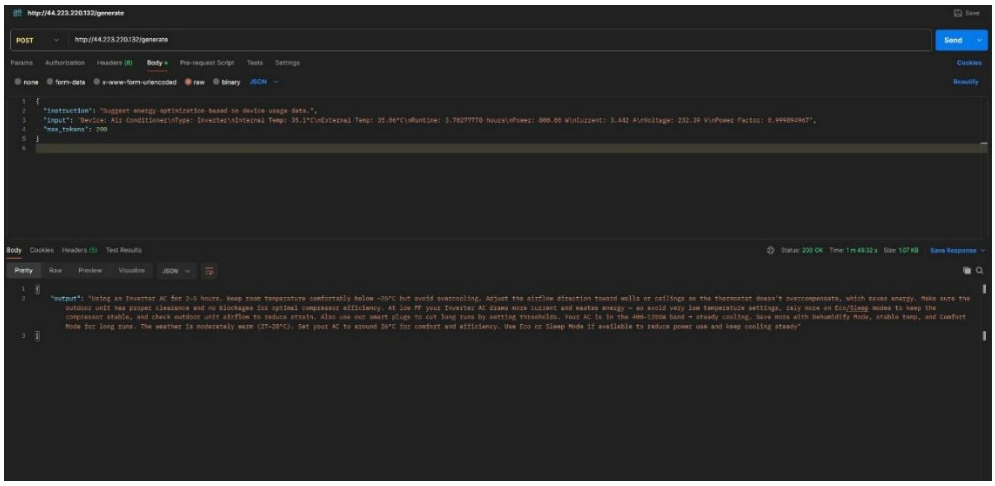

Figure 27: Energy Tips API Response

M.6 – VoltFlow OCR Insights

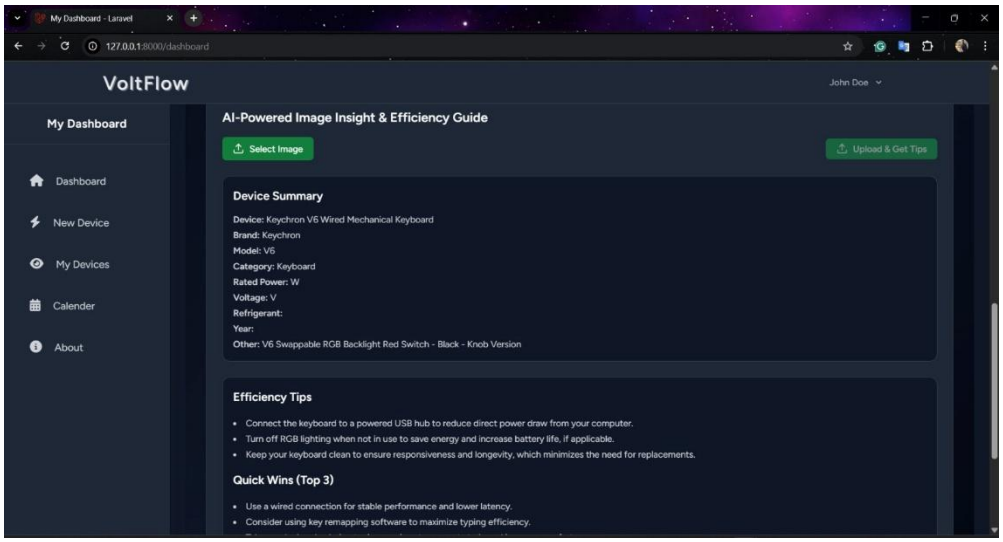

Figure 28: VoltFlow OCR Insights