UNIVERSITY OF PISA

MASTER'S DEGREE IN
ARTIFICIAL INTELLIGENCE AND DATA ENGINEERING

Internet of Things

Energy-Efficient Smart Lighting System for Office Environments

Professors:

Giuseppe Anastasi

Francesca Righetti

Carlo Vallati

Students:

Massimiliano Romani

Academic Year 2024/2025

Index

1 Introduction

In the global effort to reduce energy consumption, office buildings represent a significant area for improvement. A primary source of energy usage in these environments is artificial lighting. Often, workers turn on desk lamps to better see computer screens or documents, even when there is abundant natural light. This behavior is often triggered by weather fluctuations; for example, when clouds temporarily block the sun, the ambient light is reduced, prompting users to turn on their lamps. These lamps may then remain on for the rest of the day, even after the sky has cleared, leading to unnecessary energy waste.

Furthermore, frequent and sharp changes in brightness can negatively affect visual comfort and productivity. To address these issues, this project proposes an IoT-based smart lighting system. The system utilizes dimmable LED lamps and a network of IoT devices to automatically maintain a constant and comfortable brightness level at each workstation. By dynamically adjusting the artificial light based on real-time ambient light measurements, the system enhances both employee well-being and energy efficiency. To achieve an even higher level of energy savings, the lamp also incorporates presence detection, automatically turning off when a worker leaves their desk.

This document details the design, implementation, and evaluation of this smart lighting system, covering its architecture, communication protocols, machine learning model, and data management.

1.1 How to Dim an LED Lamp

There are two primary methods for dimming an LED lamp:

- **Analog Dimming:** The lamp's brightness is reduced by supplying less current to the LED. While this is the simplest method, some bulb models may experience a slight color shift at lower currents. Unlike incandescent bulbs, LEDs have a minimum forward voltage, below which they turn off completely rather than dimming smoothly to zero.
- **Pulse Width Modulation (PWM):** LEDs are switched on and off at a very high frequency. The human eye cannot perceive the individual flashes and instead integrates the light over time. By varying the duration of the "on" time (the pulse width) within each cycle, the perceived brightness can be controlled smoothly. A shorter "on" time results in a dimmer light. This method allows for a wide and precise dimming range but can cause discomfort for individuals who are particularly sensitive to flicker, though this is less of a concern with high-quality bulbs and high-frequency PWM. For this project, the actuator logic is modeled based on the PWM concept.

1.2 Actuator Modeling: Smart Desk Lamp Reference

To ensure the realism of this system and the validity of the generated dataset, the IoT actuator (a dimmable desk lamp) was modeled based on the technical specifications of a commercially available product: the **Xiaomi Mi LED Desk Lamp 1S**. This approach grounds this project in a real-world use case.

A key modeling assumption is a **linear relationship** between the control signal (dimming percentage) and the resulting light output (illuminance). This is a common and reasonable approximation for modern LED-based lighting systems. The core specifications used for the model are as follows:

- **Modeled Maximum Illuminance:** The lamp is modeled to provide a maximum of **1000 lux (lx)** on the desk surface when at 100% brightness. This value represents the maximum light the lamp can add to the ambient environment.

- **Maximum Power Consumption: 9.0 Watts (W)** at 100% brightness.
- **Standby Power Consumption: 0.5 Watts (W)** at 0% brightness (when the lamp is off but its control module is active).

The resulting reference characteristics, used to generate the synthetic dataset for this machine learning model, are summarized in Table 1.

Table 1: Reference characteristics of the modeled smart lamp.

Dimming Level (%)	**Resulting Illuminance (lx)**	**Estimated Power (W)**
0%	0 lx	0.5 W
25%	250 lx	2.63 W
50%	500 lx	4.75 W
75%	750 lx	6.88 W
100%	1000 lx	9.0 W

1.3 How to use the Smart Lamp

To best use the smart lamp, the user should place a written document on the desk and while still looking at the paper, press the button to set the brightness statically or keeping it pressed to slowly increase the brightness until reading becomes comfortable. From that moment on, the smart lamp will adjust its brightness in real time to account for the change in the ambient light.

2 System Implementation

The system is composed of an IoT network of constrained devices, a border router for external connectivity, and a cloud-based backend for data collection and user interaction. Since the nodes are static, which means that every dongle serves always the same purpose, the IPv6 links, being Link-local or not, are directly written in the code of each device and application, therefore there is no need for a configuration file external to the code.

2.1 System Architecture

The overall architecture is depicted in Figure 1. It consists of three main layers:

1. **Device Layer:** Comprises nrf52840 dongles running the Contiki-NG operating system. These nodes form a 6LoWPAN mesh network and include the "Smart Lamp" device and the "ML Device".
2. **Network Layer:** A Border Router, also running on an nrf52840 dongle, connects the 6LoWPAN network to the host machine's local network. It handles IPv6 packet routing and translation.
3. **Cloud/Application Layer:** Python-based applications (*data-collector.py* and *user-app.py*) run on a host computer. They communicate with the IoT devices via CoAP, store data in a MySQL database, and provide a command-line interface for user control.

2.2 IoT Device Roles

The IoT network consists of two types of devices, each with a specific role.

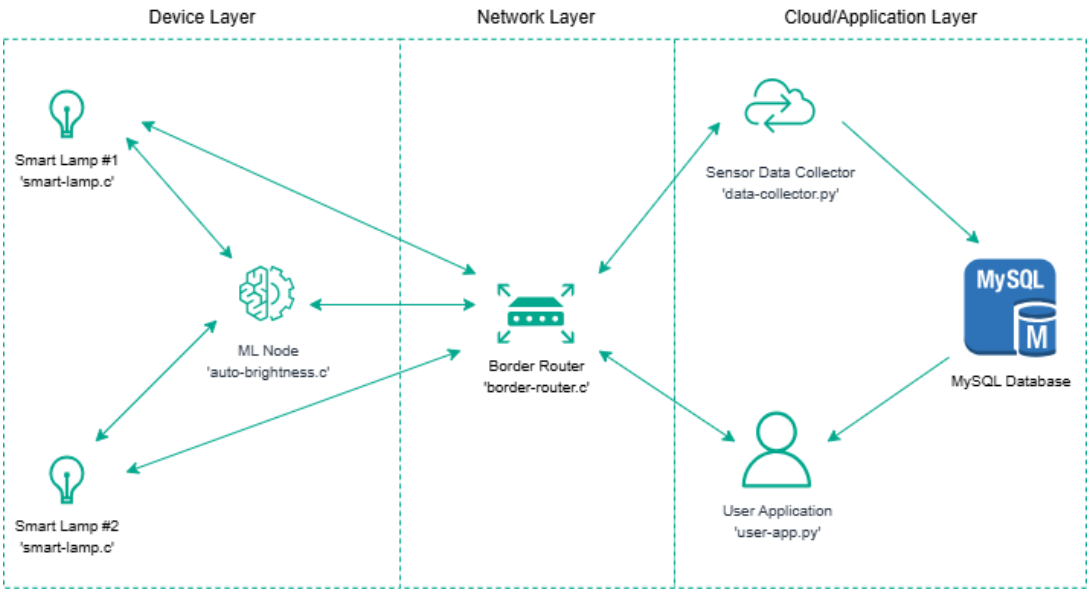

Figure 1: System Architecture Diagram.

2.2.1 The Smart Lamp Device (*smart-lamp.c*)

This device represents the smart lamp standing at each desk and it is the core component of this application. It simulates the sensors and actuators required for automated lighting control, which in the real application should be embedded in the lamp itself. It sends messages to the Machine Learning node to obtain the prediction on the next level of brightness the lamp must have.

- **Functionality:** It acts as both a CoAP client and a CoAP server.

  As a **CoAP client**, it periodically reads its local light sensor. If a user is present, it sends the ambient light value to the ML Device to request a brightness prediction.

  As a **CoAP server**, it exposes several resources that can be queried or updated by the cloud applications or other devices. These resources include:

  /status: Returns the current ambient light, the lux level desired by the user and the actual brightness of the lamp (GET).

  /actuators/brightness: Allows setting the lamp's dimming level (PUT).

- **Sensors (Emulated):**

  Light Sensor: Measures the ambient illumination in lux every second. This sampling rate was chosen for two main reasons:

  Too many samples: sampling at an higher rate would have produced too many samples, which in turn would have increased the network traffic, packet loss rate and the computing resources, while not providing any additional benefit.

  Flickering: More samples involves result in more predictions and therefore more adjustments, which could result in a flickering effect very harmful for the user's eyes.

  Presence Sensor: A simple infrared (IR) sensor model that detects if a user is at the desk. The lamp turns off if no presence is detected for a predefined period.

- **Actuator (Emulated):**

Lamp Actuator: Controls the dimming level of the LED lamp from 0% to 100%.

- **User Interaction:**

  Button: Allows for manual lamp brightness control:

  short press: cycles the brightness between off (0%), low (20%), medium (50%), and high (80%) presets (STATIC MODE).

  long press: initiates a gradual increase of 5% brightness every second, allowing fine-tuning. When the threshold hits 100%, it restarts from zero (MANUAL MODE).

  RGB LED: Provides visual feedback by changing its color to match the current dimming level of the lamp:

  Off: 0%

  Green: more than 0% and less or equal to 20%

  Blue: more than 20% and less or equal to 50%

  Violet: more than 50% and less or equal to 80%

  Red: more than 80% and less or equal to 100%

  YELLOW LED: shows the user if the current brightness mode is MANUAL (LED On) or STATIC (LED Off).

2.2.2 The ML Device (*auto-brightness.c*)

This device is responsible for running the machine learning model to provide intelligent, decentralized control decisions. In this project, the model runs on the same type of device as the smart lamp, but in a real application, a finer model could be hosted on a device with more computing capabilites, that is why the machine learning model is not incorporated in the smart lamp.

- **Functionality:** It acts as both a CoAP client and a CoAP server.

  As a **CoAP client**, it sends a command to switch off or on to all the Smart Lamps in its range of communication, when the button is pressed by the user.

  As a **CoAP server**, it exposes a resource for brightness prediction:

  /autobright: This resource accepts a GET request containing the ambient light value from a Smart Lamp device. It then executes the onboard TinyML model (*dimming_forecast.h*) to calculate the optimal dimming level and returns the prediction in the response.

- **User Interaction:**

  Button: A single press acts as a global toggle, broadcasting a command to turn all Smart Lamps in the network on or off.

  RGB LED: By using the same color code as the Smart Lamp Devices, it shows the last brightness predicted.

  YELLOW LED: Indicates the global state of the lamps (on or off), confirming that the button press was registered.

2.3 Border Router (*border-router.c*)

This is device hosts the standard *rpl-border-router* example from Contiki-NG. Its role is to bridge the low-power, IPv6-based 6LoWPAN network with the external IPv4/IPv6 network. This allows the Python applications running on the host computer to communicate directly with the IoT devices using their IPv6 addresses.

2.4 Cloud Applications

The backend logic and user interface are implemented as two separate Python scripts.

- **Data Collector (*data-collector.py*):** This script runs continuously in the background. It sends CoAP GET requests to all known Smart Lamp devices every 10 seconds to poll their sensor data (current lux, desired lux) and actuator state (dimming level). The collected data is then timestamped and stored in the MySQL database, creating a historical log for analysis or future improvements. The rate of 10 seconds was chosen to reduce network traffic and save space on the server, since having this data for every second is not useful. The presence/absence state is not registered for the user's privacy, since for the energy consumption monitoring task, only the lamp brightness (which is naturally set to zero when the user is away) is useful.
- **User Application (*user-app.py*):** This script provides a command-line interface (CLI) for remote system management. The user can perform these actions:

  allOff: Turn all lights off.

  allOn: Turn all lights on at 20% lamp brightness.

  last5: Retrieve the last 5 record from the database.

  lastHour: Retrieve the records of the last hour and estimates the power consumption.

  help: Prints the list of commands with a brief description for each.

  The application queries the MySQL database to retrieve informations and sends CoAP PUT requests to the */actuators/brightness* resource on the target devices to apply changes.

2.5 Communication Protocol: Why CoAP?

The Constrained Application Protocol (CoAP) was chosen as the application-layer protocol for this project. The choice was motivated by the specific requirements of constrained IoT networks.

- **Lightweight and Efficient:** CoAP runs over UDP, which has significantly less overhead than TCP. This reduces packet size and processing requirements, making it ideal for devices with limited memory, processing power, and battery life, such as the nrf52840. Also, given the task at hand, the loss of a packet every once in a while doesn't harm the application as much as high latency would.
- **Request/Response Model:** The interactions in this system fit naturally into CoAP's client-server model. For example, the Smart Lamp acts as a client making a direct request to the ML Device server for a prediction. This is a simple and efficient one-to-one communication pattern.
- **No Central Broker:** Unlike MQTT, which relies on a central broker, CoAP allows for direct device-to-device communication. This is highly advantageous for interactions within the 6LoWPAN network, as it reduces latency and eliminates a single point of failure. Messages between the Smart Lamp and the ML Device do not need to leave the constrained network.
- **Low Latency:** The Smart Lamp and the ML Device need to communicate as fast as possible, because of the adjusting work the Lamp must perform, therefore a direct communication using Link-Local addresses creates a one-to-one single hop communication that is not achievable with MQTT.

In summary, CoAP provides the necessary functionality for this use case while being optimized for the resource-constrained nature of the hardware, making it a more suitable choice than a broker-based protocol like MQTT for the internal network communications.

3 Data Encoding

For communication between devices and with the cloud applications, a simple and human-readable text-based format was chosen. The selection was based on a trade-off between efficiency and ease of implementation and debugging for a project of this scope.

Payloads are encoded as simple key-value pairs, using the *text/plain* media type. For example:

- When a Smart Lamp requests a prediction from the ML device, it sends a CoAP POST request to */autobright* with a payload like:

```
lux=350&des_lux=800
```

- The ML Device responds with the predicted dimming level:

```
49
```

3.1 Justification

JSON was chosen as the text-based format for packet exchanging because it offers several advantages for this project:

- **Simplicity:** Parsing simple strings is straightforward in both C (on Contiki-NG) and Python, requiring no external libraries.
- **Debuggability:** The payloads are human-readable, which greatly simplifies debugging network traffic using simple print statements.
- **Sufficiency:** Given the low frequency of messages and the small amount of data being transmitted, the overhead of a text format is negligible for this application's performance.

4 Machine Learning Model

To enable autonomous decision-making on the edge, a TinyML model was developed to predict the optimal lamp dimming level. The model runs directly on the "ML Device" dongle.

4.1 Model Objective and Training Data

The goal of the model is to predict the required **Dimming Level (%)** based on two inputs: the current **Ambient Lux** and a user-defined **Desired Lux**.

As collecting a large, real-world dataset was not feasible, a synthetic dataset was generated, as detailed in the Jupyter Notebook (*regression_model_training.ipynb*). The process was as follows:

1. **Base Data:** The 'lighting_lux' column from a public IoT office's environment dataset [1] was used to provide a realistic time series of ambient light conditions.
2. **Feature Generation:** For each ambient lux value, a random *Desired Lux* target was generated within a realistic range (e.g., between 400 and 1500 lux). This simulates different user preferences or task requirements.
3. **Target Calculation:** The target variable, *Dimming Level (%)*, was calculated as the percentage of the lamp's maximum output (1000 lx) needed to bridge the gap between the ambient and desired lux levels. The lamp's physical limits were respected using the formula:

$$\text{Dimming Level} = \frac{\max(0, \min(\text{Desired Lux} - \text{Ambient Lux}, 1000))}{1000} \times 100$$

This process resulted in a dataset of 1000 samples that accurately reflects the physical logic the model needs to learn.

4.2 Model Selection and Training

A **Decision Tree Regressor** from the scikit-learn library was chosen. This model is well-suited for TinyML applications because:

- It is lightweight and has a small memory footprint, especially when the tree depth is limited.
- It does not require data normalization or scaling.
- It is highly interpretable.
- It can be easily converted to C code.

The model was trained on 80% of the synthetic dataset with a *max_depth* of 7 to prevent overfitting and keep the model size small. It was then evaluated on the remaining 20% of the data, achieving excellent performance as shown in the table 2.

Table 2: Metrics and performances

R-squared (R^2)	**Mean Squared Error (MSE)**
3.88	0.995

The results and the figure below 2 indicates that the model learned the underlying mathematical relationship almost perfectly.

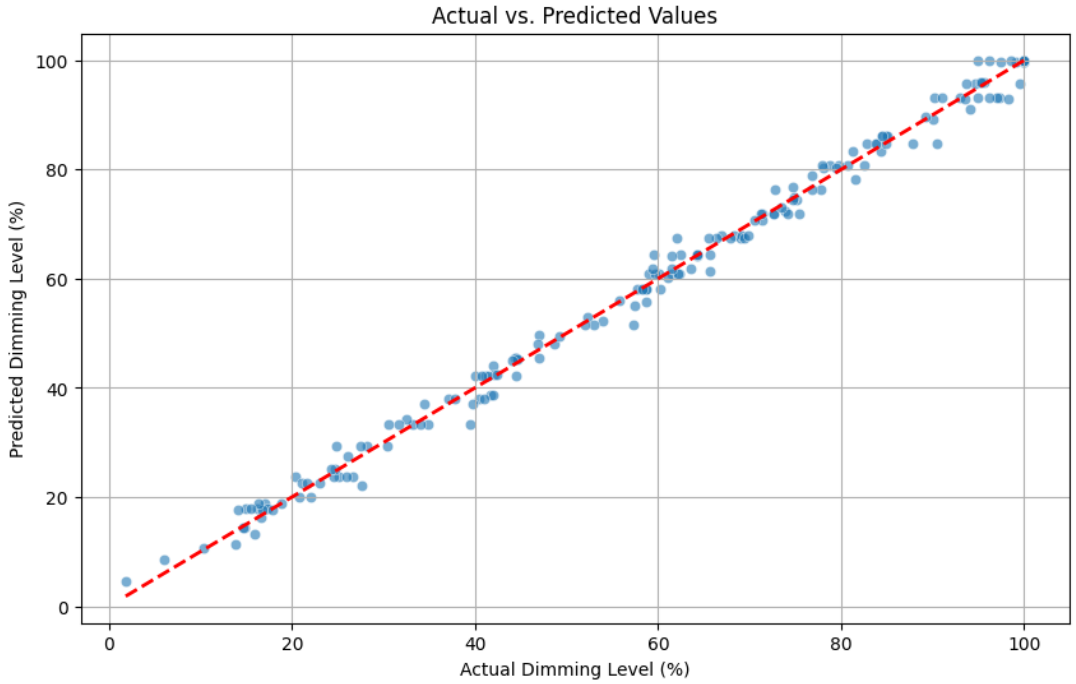

Figure 2: Real relationship and predicted values.

4.3 Deployment with Emlearn

The trained Python model was converted into an embedded-friendly format using the **emlearn** library. This tool automatically exported the decision tree structure into a C header file, *dimming_forecast.h*. This file contains the model as C arrays and provides a simple prediction function, *dimming_forecast_predict()*, which takes the input features (ambient and desired lux) and returns the predicted dimming level. This header was then directly included in the *auto-brightness.c* firmware, enabling on-device inference.

5 MySQL Database Schema

A MySQL database is used to store historical data collected from the IoT devices. This data can be used for monitoring system performance, analyzing energy consumption, or training future machine learning models. The schema consists of two main tables: *devices* and *sensor_readings*.

5.1 Table: *devices*

This table stores information about each registered device in the network. It serves as a central registry for the system.

Table 3: Schema for the *devices* table.

Column Name	Data Type	Description
device_id	*INT* (PK, AI)	Unique identifier for the device.
ipv6_address	*VARCHAR(45)*	The IPv6 address of the device.

5.2 Table: *sensor_readings*

This is the main time-series table, where all data points from the sensors are logged.

Table 4: Schema for the *sensor_readings* table.

Column Name	**Data Type**	**Description**
reading_id	*INT* (PK, AI)	Unique identifier for the reading.
device_id	*INT* (FK)	Foreign key referencing *devices.device_id*.
timestamp	*TIMESTAMP*	The time the sampling was recorded.
lux_perceived	*INT*	The value in lux of the light sampled by the sensor.
lux_desired	*INT*	The value in lux of the light desired by the user at sampling time.
brightness_percent	*INT*	The brightness level of the lamp at sampling time.
power_consumption	*INT*	The power consumption in Watt at sampling time.

The relationship between the tables is one-to-many: one device can have many sensor readings. This normalized structure is efficient for both storage and querying.

6 Conclusions

This project successfully designed and implemented a fully functional, energy-efficient smart lighting system for an office environment. By integrating IoT devices, edge-based machine learning, and a cloud backend, the system demonstrates a practical solution to reduce energy waste while maintaining user comfort.

The use of Contiki-NG on nrf52840 hardware proved to be a robust platform for developing constrained IoT applications. The choice of CoAP for communication was validated by its efficiency and suitability for direct device-to-device interactions. The development of a TinyML model using a Decision Tree and its deployment via emlearn showcased the feasibility of performing intelligent, autonomous control directly on edge devices, reducing reliance on cloud connectivity and improving system responsiveness.

References

[1] Ziya K, *IoT based Environmental Dataset*, Kaggle, 2023. Available: `https://www.kaggle.com/datasets/ziya07/iot-based-environmental-dataset`