POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Wojciech Zdzisław Adamczyk
Nr albumu s32438

Zastosowanie uczenia maszynowego w systemach sterowania oświetleniem domowym do symulacji obecności mieszkańców

Praca magisterska

promotor
dr inż. Mariusz Trzaska

Warszawa, luty, 2026

Streszczenie

Tematem niniejszej pracy było zbadanie możliwości oraz skuteczności działania systemu sterowania oświetleniem domowym, opartego na uczeniu maszynowym. Projekt przewidywał zintegrowanie z narzędziami typu asystent domowy, w celu odwzorowania zachowań mieszkańców i symulowania ich obecności w domu. Celem było odstraszanie potencjalnych intruzów. W ramach prac badawczo-projektowych opracowano prototyp rozwiązania opartego na platformie Home Assistant. Rozszerzony został o implementację modeli uczenia maszynowego oraz dodatkowe komponenty, takie jak dokumentowa baza danych i systemy wspomagające zarządzanie modelami ML. Wykorzystując przygotowany prototyp, w trakcie kilku miesięcy zbierano dane dotyczące rzeczywistej aktywności oświetlenia, które następnie posłużyły do przeprowadzenia analiz działania proponowanego rozwiązania.

Słowa kluczowe:

uczenie maszynowe, asystent domowy, Internet rzeczy, sterowanie oświetleniem, analiza skuteczności

Podziękowania

Autor pracy składa serdeczne podziękowanie Promotorowi, dr inż. Mariuszowi Trzasce, za inspirację przy wyborze tematu oraz cenne i trafne wskazówki udzielane na poszczególnych etapach realizacji pracy. Okazane merytoryczne wsparcie oraz zaangażowanie miały kluczowy wpływ na ostateczny kształt niniejszego opracowania.

Spis treści

1. Wstęp

Celem niniejszej pracy było wykorzystanie uczenia maszynowego (ang. *Machine Learning*) w automatyzacji sterowania oświetleniem domowym umożliwiającym symulowanie pobytu mieszkańców w domu podczas ich nieobecności, tak aby zmylić potencjalnych intruzów i zniechęcić ich do podjęcia próby włamania. W celu zweryfikowania słuszności przyjętej koncepcji, został opracowany prototyp systemu typu PoC (ang. *Proof of Concept*), którego rezultaty działania posłużyły do przeprowadzenia analiz. Autor miał nadzieję, iż wykorzystanie sztucznej inteligencji może zapewnić jak najlepsze odwzorowanie faktycznych praktyk mieszkańców.

Uczenie maszynowe zostało wykorzystane w projekcie do analizy i uczenia się specyficznych praktyk związanych z obsługą oświetlenia, czyli określania, kiedy światło jest włączane, a kiedy wyłączane w pomieszczeniu objętym badaniem. Weryfikacja zachowania systemu odbywa się na podstawie odczytu stanu inteligentnego przełącznika światła komunikującego się z siecią domową poprzez Wi-Fi. Dodatkowo model uwzględnia pomiar natężenia światła zewnętrznego, dokonywany przez odpowiedni czujnik, a także dane związane z datą i czasem.

Wyuczony model, działając w okresie nieobecności mieszkańców decyduje, kiedy światło powinno być włączone, a kiedy wyłączone tak, aby jak najwierniej odzwierciedlać rzeczywiste zwyczaje użytkowników i tworzyć wiarygodną symulację ich obecności.

Zakres prac obejmował:

- wybór i implementację nowoczesnego centralnego systemu zarządzania domem,
- zastosowanie odpowiedniego modelu uczenia maszynowego, który potencjalnie zapewnia najlepsze rezultaty w kontekście postawionego zadania,
- oraz implementację narzędzi wspomagających działanie systemu, takich jak baza danych do przechowywania informacji treningowych, system wersjonowania modeli oraz narzędzie do przechowywania artefaktów modeli,

Realizacja pracy została podzielona na kilka etapów. Po opracowaniu i wdrożeniu prototypu przeprowadzono testy jego ciągłej, nieprzerwanej pracy w warunkach rzeczywistych. Celem była identyfikacja potencjalnych problemów, takich jak awarie oprogramowania, błędy komunikacji sieciowej, wygoda użytkowania czy reakcja systemu na niepożądane sytuacje. Zebrane obserwacje posłużyły do dopracowania i stabilizacji całego rozwiązania.

W trakcie długotrwałego działania prototypu zbierano dane, które zostały następnie wykorzystane w okresie rzeczywistej i zaplanowanej nieobecności mieszkańców. Jednak jeszcze istotniejszym celem pozyskanych danych była możliwość przeprowadzenia analizy skuteczności proponowanego rozwiązania oraz porównania jego działania z istniejącymi systemami dostępnymi na rynku. Porównania obejmowały popularne heurystyki automatyzacji oświetlenia, oparte np. na sztywnych godzinach włączenia i wyłączenia światła lub innych prostych regułach eksperckich.

Podczas planowania tematu pracy pojawił się szereg pytań badawczych, na które praca miała przynajmniej częściowo udzielić odpowiedzi:

- Czy system oparty na technologii uczenia maszynowego jest w stanie skutecznie imitować rzeczywiste zachowania użytkowników w zakresie obsługi systemów inteligentnego domu?
- Czy zaprojektowany prototyp (ang. *Proof of Concept*), wykorzystujący integrację platformy Home Assistant z modelami uczenia maszynowego, cechuje się wystarczającą optymalnością, stabilnością oraz niezawodnością w warunkach praktycznego użytkowania?
- Czy zaproponowana mechanika sterowania oparta na modelach predykcyjnych może w wybranych aspektach przewyższać rozwiązania bazujące na sztywnych regułach eksperckich?

Przyjmując powyższy plan etapów projektowych i badań, niniejsza praca została podzielona na kilka głównych części:

- W pierwszej kolejności przedstawiono motywację wyboru tematu oraz analizę istniejących rozwiązań i ich ograniczeń,
- Kolejna część zawiera opis architektury sprzętowej i programowej, umożliwiający zapoznanie się ze szczegółami implementacji systemu,
- Dalsze rozdziały poświęcono opisowi zastosowanej technologii uczenia maszynowego oraz analizie skuteczności modelu w porównaniu do tradycyjnych metod opartych na regułach,
- Pracę kończą wnioski, w których przedstawiono obserwacje, rezultaty i potencjalne kierunki dalszego rozwoju projektu.

2. Motywacja i założenia proponowanego rozwiązania

Założeniem proponowanego rozwiązania było opracowanie technologii, która w możliwie najbardziej realistyczny sposób potrafiłaby naśladować aktywność mieszkańców w zakresie włączania i wyłączania świateł domowych. Celem takiego działania jest zwiększenie bezpieczeństwa domu poprzez symulację obecności domowników, co może skutecznie zmylić potencjalnych włamywaczy.

Projekt został zrealizowany w oparciu o jeden włącznik światła, obejmujący jedno pomieszczenie, jednak opracowany system został zaprojektowany w taki sposób, aby umożliwiać rozszerzenie na wiele przełączników i źródeł światła w przyszłości. Główną motywacją było stworzenie rozwiązania, które działałoby bardziej inteligentnie i mniej przewidywalnie niż proste, powszechnie stosowane automatyzacje oparte na sztywnych regułach. Przykładami takich praktyk jest włączanie świateł zawsze o godzinie 21:00 i wyłączanie o 23:00 lub sterowanie jedynie w zależności od poziomu natężenia światła zewnętrznego. Tego typu systemy, choć popularne, są łatwe do przewidzenia i mogą zostać szybko rozpoznane przez obserwatora.

Oprócz celu praktycznego, projekt miał także wartość edukacyjną i badawczą. Jednym z istotniejszych motywatorów była wszechstronność technologiczna zastosowanych rozwiązań. Realizacja projektu obejmowała między innymi:

- wykorzystanie mikrokomputera z podłączonym czujnikiem światła i przełącznikiem,
- implementację komunikacji sieciowej pomiędzy urządzeniami,
- budowę działającego PoC (ang. *Proof of Concept*) systemu zintegrowanego z platformą zarządzania inteligentnym domem,
- wykorzystanie nowoczesnych narzędzi inżynierskich, takich jak MLflow (do rejestrowania i wersjonowania modeli), MongoDB (do przechowywania danych w bazie dokumentowej) czy komunikacji po REST API.

Tak szeroki zakres zastosowanych technologii daje ogromne możliwości rozwoju, poszerzania wiedzy i praktycznych umiejętności w wielu dziedzinach, takich jak Internet Rzeczy (IoT), przetwarzanie danych, konteneryzacja czy sztuczna inteligencja. Współczesny świat coraz bardziej docenia wszechstronność kompetencji technicznych, dlatego rozwijanie tego typu interdyscyplinarnych umiejętności stanowiło dodatkową wartość projektu.

Kolejnym kluczowym motywatorem była możliwość zastosowania technologii uczenia maszynowego (ang. *Machine Learning*) w praktycznym scenariuszu. W projekcie wykorzystano bibliotekę Scikit-learn dla języka Python, która umożliwia implementację i testowanie różnych algorytmów uczenia. Tematyka sztucznej inteligencji rozwija się obecnie w niezwykle szybkim tempie, a jej zastosowania obejmują coraz więcej dziedzin życia. Wiele codziennych czynności wykonywanych jest dziś z pomocą inteligentnych narzędzi, od środków transportu po urządzenia gospodarstwa domowego.

Równocześnie rośnie świadomość społeczna w zakresie możliwości, jakie daje sztuczna inteligencja. Coraz więcej osób na świecie korzysta z systemów opartych na modelach językowych (LLM), takich jak ChatGPT, w podejmowaniu decyzji i uzyskiwaniu informacji. Obok takich modeli ogólnego przeznaczenia istnieją także modele wyspecjalizowane, przystosowane do konkretnych zastosowań. W ramach niniejszego projektu zaimplementowano model klasyfikacji binarnej, którego zadaniem było określenie, czy światło w pomieszczeniu powinno zostać włączone, czy wyłączone.

Sztuczna inteligencja coraz częściej nie tylko reaguje na warunki otoczenia, takie jak temperatura czy oświetlenie, ale również uczy się i przewiduje zachowania ludzi. Dotyczy to zarówno powtarzalnych czynności wynikających z codziennej rutyny, jak i pozornie losowych decyzji podejmowanych spontanicznie. Większość osób, poza standardowymi przyzwyczajeniami, wykonuje szereg działań wynikających z chwilowych potrzeb, takich jak przygotowanie posiłku, odpoczynek przy telewizorze czy czytanie książki. Wiele z tych czynności wiąże się z aktywnością oświetlenia w domu.

Z tego względu opracowany model ma na celu uczenie się wzorców zachowań mieszkańców i przewidywanie ich w sposób zbliżony do rzeczywistych ludzkich decyzji, które często wydają się nieprzewidywalne. Modele uczenia maszynowego można w uproszczeniu traktować jako złożone funkcje matematyczne, które na podstawie analizy danych próbują uogólniać zależności i przewidywać przyszłe zdarzenia.

Jednym z głównych celów niniejszego projektu było zbadanie czy modele sztucznej inteligencji, oparte na ścisłych regułach matematycznych, mogą skutecznie przewidywać ludzkie zachowania, które z natury są zmienne i trudne do jednoznacznego opisania. Odpowiedzi na to pytanie dostarczają analizy skuteczności modeli przedstawione w dalszej części pracy.

3. Opis konkurencyjnych technologii

Wraz z dynamicznym rozwojem sztucznej inteligencji oraz coraz szerszym zakresem jej zastosowań, coraz trudniej wskazać dziedzinę, w której technologia ta nie znajduje praktycznego wykorzystania. Nie inaczej jest w przypadku tematu niniejszej pracy. Na rynku dostępne są już rozwiązania, w których producenci deklarują wykorzystanie wyuczonych modeli sztucznej inteligencji w procesie automatyzacji. Przykłady takich rozwiązań, zidentyfikowanych w trakcie analizy rynku, zostały przedstawione w kolejnych podrozdziałach.

3.1. Philips Hue - „Mimic presence”

System Philips Hue [1] oferuje w aplikacji funkcję automatyzacji o nazwie Mimic Presence. Jej działanie polega na automatycznym włączaniu i wyłączaniu lamp w godzinach, w których są one zazwyczaj używane w wybranych pomieszczeniach. Celem tej funkcji jest symulacja obecności domowników podczas ich nieobecności. Istotną wadą tego rozwiązania jest jednak konieczność korzystania wyłącznie z urządzeń ekosystemu Philips Hue, co znacznie zwiększa koszt wdrożenia i ogranicza elastyczność systemu w kontekście integracji z innymi urządzeniami.

3.2. Lampy Luvo od Luke Roberts

Producent lamp Luvo marki Luke Roberts [2] reklamuje swoje urządzenia jako rozwiązania uczące się rutyn użytkownika i automatycznie dobierające najlepsze sceny świetlne w zależności od sposobu korzystania z oświetlenia. Funkcja ta dotyczy jednak głównie dopasowania oświetlenia w trakcie codziennego użytkowania domu, a nie symulacji zachowań podczas nieobecności domowników. Podobnie jak w przypadku Philips Hue, integracja wymaga zastosowania dedykowanych urządzeń producenta, co ogranicza możliwość adaptacji rozwiązania w istniejącym środowisku inteligentnego domu.

3.3. Aplikacja HeyZack

HeyZack [3] to aplikacja reklamowana jako inteligentny asystent domowy wykorzystujący sztuczną inteligencję do nauki nawyków i stylu życia użytkownika. Umożliwia m.in. automatyzację oświetlenia, również na czas nieobecności domowników. Jest to jednak rozwiązanie obejmujące znacznie szerszy zakres funkcjonalności, co przekłada się na wysoki koszt wdrożenia. W momencie pisania niniejszej pracy koszt implementacji systemu wynosił około 3500 euro. Dodatkowym mankamentem jest przesyłanie danych o aktywnościach użytkowników do zdalnych serwerów, co może stanowić potencjalne ryzyko z punktu widzenia prywatności i bezpieczeństwa danych.

3.4. KEVIN® Mitipi

Urządzenie KEVIN® Mitipi [4] służy do symulacji obecności domowników poprzez generowanie efektów świetlnych, cieni i dźwięków. Wykorzystuje elementy sztucznej inteligencji do tworzenia realistycznych scen przedstawiających m.in. światło telewizora, ruch czy odgłosy codziennego życia. W tym przypadku AI odpowiada głównie za generowanie wiarygodnych scenariuszy, a nie za uczenie się rzeczywistych zachowań mieszkańców. Wadą rozwiązania jest brak odwzorowania indywidualnych wzorców aktywności występujących w danym domu oraz ograniczenie

funkcjonalności do jednego urządzenia emitującego światło i dźwięk. W konsekwencji główne oświetlenie pomieszczenia, takie jak żyrandol, pozostaje nieaktywne.

3.5. Thesillyhome-container

Na wczesnym etapie prac nad niniejszym projektem, w trakcie analizy dostępnych rozwiązań open-source, odnaleziono projekt o nazwie Thesillyhome-container [5], opublikowany w 2022 roku na platformie GitHub przez użytkownika lcmchris. Jego założenia były zbliżone do koncepcji realizowanej w ramach niniejszej pracy. Głównym celem również było stworzenie systemu automatyzacji oświetlenia z wykorzystaniem modeli uczenia maszynowego, które na podstawie danych historycznych potrafią przewidywać zachowania użytkowników i sterować światłem w sposób inteligentny.

Pomimo podobnych założeń koncepcyjnych, struktura i implementacja projektu Thesillyhome-container różniły się znacząco od rozwiązania opisanego w niniejszej pracy. Projekt GitHubowy zakładał wykorzystanie jednego, stosunkowo prostego modelu, drzewa decyzyjnego jako podstawowego algorytmu predykcyjnego. W ramach niniejszej pracy natomiast przewidziano badania porównawcze kilku różnych modeli uczenia maszynowego, a następnie wybór najbardziej efektywnego modelu pod względem dokładności i stabilności działania.

Nie można wykluczyć, że autor projektu Thesillyhome-container przeprowadził testy potwierdzające skuteczność drzewa decyzyjnego w tym zastosowaniu. Jednak brak dokumentacji opisującej metodykę i wyniki badań nie pozwala na jednoznaczną ocenę jakości proponowanego rozwiązania.

Warto zaznaczyć, że w późniejszym okresie, już na etapie, gdy prace nad niniejszym systemem były zaawansowane, autor projektu Thesillyhome-container dodał do swojego rozwiązania kolejne rodzaje modeli uczenia maszynowego. Mimo to, ogólna struktura projektu i sposób jego działania pozostały w dużej mierze niezmienione i nadal odbiegały od architektury zastosowanej w niniejszej pracy.

Istotną różnicą jest architektura systemu i sposób zarządzania danymi. W projekcie Thesillyhome-container dane przechowywane są w plikach CSV, a wygenerowane modele zapisywane lokalnie w strukturze katalogów projektu. Podejście to, niesie ze sobą istotne ograniczenia takie jak: brak skalowalności, brak centralnego repozytorium modeli, a także podatność na utratę danych w przypadku awarii systemu lub środowiska. Dane wejściowe do modeli pozyskiwane są z historii czujników systemu Home Assistant, co jak wykazały doświadczenia własne autora niniejszej pracy, może być rozwiązaniem zawodnym, szczególnie przy dłuższych okresach działania systemu.

W ramach niniejszego projektu zaproponowano natomiast bardziej elastyczną i modularną architekturę. Do zarządzania i wersjonowania modeli wykorzystano platformę MLflow, która pozwala przechowywać modele wraz z metadanymi i metrykami jakości. Dane pomiarowe są zapisywane w bazie danych MongoDB, natomiast artefakty modeli w lokalnym magazynie S3. Wszystkie te serwisy uruchomione są w kontenerach Dockera w ramach tego samego urządzenia (mikrokomputera Raspberry Pi), co oznacza, że w obecnej konfiguracji nie zapewniają one pełnej odporności na awarie sprzętowe. Jednocześnie jednak zastosowana struktura kontenerowa oraz rozdzielenie ról poszczególnych usług tworzą solidne podstawy do potencjalnego przekształcenia projektu w rozwiązanie chmurowe. Dzięki temu dane i modele mogłyby być przechowywane w środowisku bardziej odpornym na utratę danych w wyniku awarii systemu. Takie podejście zwiększyłoby niezawodność i skalowalność całego systemu.

Dodatkowo, projekt realizowany w ramach niniejszej pracy zakłada ciągłą aktualizację danych w bazie, synchronizowaną z każdą zmianą stanu czujnika natężenia światła. Takie rozwiązanie pozwala uniknąć ewentualnych problemów związanych z bazowaniem na historii czujników.

W projekcie Thesillyhome-container nie udostępniono także żadnych badań porównawczych ani analiz skuteczności zastosowanego rozwiązania. Brak informacji o jakości predykcji czy efektywności

energetycznej sprawia, że trudno ocenić praktyczną wartość tego systemu. W niniejszej pracy natomiast takie badania zostały przeprowadzone, a ich wyniki przedstawiono w dalszych rozdziałach.

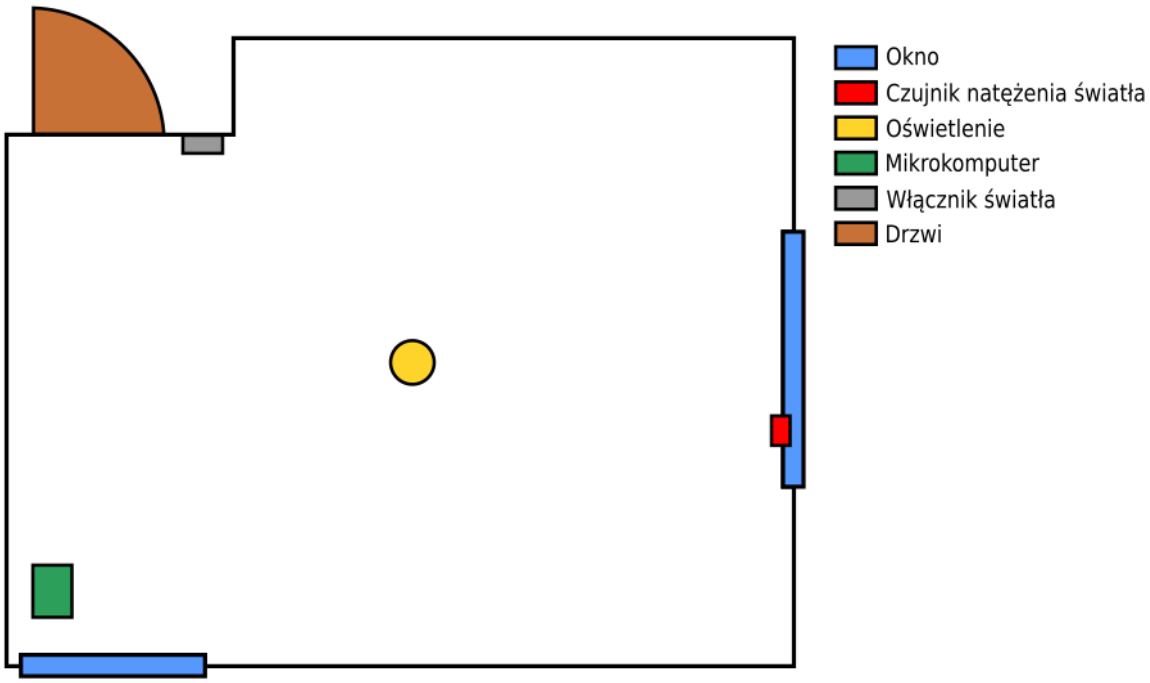
Pomimo tych różnic należy podkreślić, że projekt Thesillyhome-container stanowi interesującą próbę implementacji idei inteligentnego sterowania oświetleniem z wykorzystaniem sztucznej inteligencji. Sam fakt pojawienia się podobnego rozwiązania w przestrzeni open-source potwierdza, że koncepcja realizowana w ramach niniejszej pracy jest aktualna, uzasadniona i potencjalnie użyteczna praktycznie. Warto również zaznaczyć, że oba projekty rozwijane były całkowicie niezależnie.

4. Architektura laboratorium testowego

Projekt inteligentnego domu, w celu uproszczenia implementacji i weryfikacji działania systemu, został zrealizowany w oparciu o jedno pomieszczenie jakim był gabinet pełniący funkcję miejsca codziennej pracy. Wybór tego pomieszczenia podyktowany był jego uniwersalnym charakterem oraz możliwością odzwierciedlenia typowych warunków użytkowych, w jakich system automatyki domowej może być wykorzystywany na co dzień.

4.1. Opis ogólny

W gabinecie zainstalowano podstawowe urządzenia niezbędne do realizacji założeń projektu, w tym włącznik światła z funkcją zdalnego sterowania oraz czujnik natężenia światła zewnętrznego. Głównym elementem infrastruktury technicznej był mikrokomputer pełniący rolę jednostki centralnej, na którym uruchomiono w kontenerach Docker kluczowe serwisy systemu opisane w kolejnym rozdziale. Mikrokomputer odpowiadał za komunikację z urządzeniami peryferyjnymi, zbieranie danych pomiarowych oraz realizację logiki sterowania oświetleniem. Schemat rozmieszczenia poszczególnych elementów w pomieszczeniu przedstawiono na rysunku 1.



Rysunek 1. Schemat pomieszczenia

Głównym źródłem światła dziennego w pomieszczeniu było okno skierowane na wschód, co zapewniało naturalne oświetlenie we wczesnych godzinach dnia. Z tego względu czujnik natężenia światła został umieszczony na wewnętrznej stronie okna, bezpośrednio na szybie, w miejscu umożliwiającym precyzyjny pomiar zmiennych warunków oświetleniowych. Drugie okno, zwrócone w kierunku południowym, przez cały czas pozostawało zasłonięte roletą, dlatego umieszczenie czujnika w tej lokalizacji mogłoby prowadzić do uzyskania nieadekwatnych wyników pomiarowych i zafałszowania danych wejściowych dla systemu sterującego. Wszystkie elementy składowe systemu zestawiono w tabeli 1, która zawiera nazwy producentów, modele oraz funkcjonalne przeznaczenie w ramach projektu.

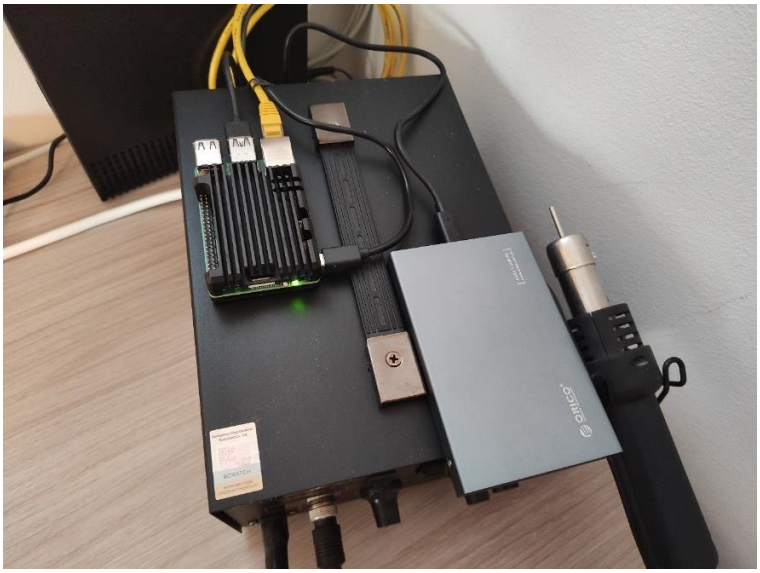
Tabela 1. Wykaz urządzeń

Rodzaj	Producent	Model
Czujnik natężenia światła	BleBox	luxSensor
Zasilacz czujnika światła	Extremestyle	TC30CUGAN
Włącznik światła	Sonoff	TXT1EU
Mikrokomputer	Raspberry Pi	5 8GB
Obudowa mikrokomputera	justPi	JUS-24108
Zasilacz mikrokomputera	Raspberry Pi	27W USB-C Power Supply
Dysk SSD	GoodRam	CX400 Gen.2 512GB
Obudowa dysku	Orico	2518C3-G2-GY

Źródło: Opracowanie własne.

Pozostałe urządzenia takie jak żarówka itp. z racji na ich powszechność i dowolność sprawiają, że celowo zostały pominięte w opisie.

4.2. Mikrokomputer



Rysunek 2. Mikrokomputer w obudowie wraz z podłączonym dyskiem

Mikrokomputer zastosowany w projekcie to Raspberry Pi 5 [6] wyposażony w 8 GB pamięci RAM, zasilany za pomocą dedykowanego zasilacza USB-C 27 W od tego samego producenta. Urządzenie to charakteryzuje się wysoką wydajnością, niskim poborem energii oraz szerokimi możliwościami rozbudowy, co czyni je idealnym rozwiązaniem do realizacji projektów typu smart home. Dostępny procesor to 4 rdzeniowy Cortex A76 o taktowaniu 2.4 GHz.

Producent Raspberry Pi przewiduje możliwość wykorzystania karty microSD jako podstawowego nośnika danych przeznaczonego na system operacyjny. W niniejszym projekcie

zdecydowano się jednak na zastosowanie zewnętrznego dysku SSD (standardowy dysk 2,5” w obudowie), podłączonego do mikrokomputera za pomocą interfejsu USB-C. Rozwiązanie to zapewnia znacznie wyższą niezawodność i trwałość w porównaniu z nośnikami microSD, które są mniej odporne na intensywne operacje zapisu i odczytu danych. Dyski SSD, dzięki swojej konstrukcji i szybkości działania, są znacznie lepiej przystosowane do ciągłej pracy w środowiskach serwerowych i systemach automatyki domowej.

Na mikrokomputerze został zainstalowany system operacyjny Ubuntu 24.04.2 LTS w wersji przeznaczonej dla procesorów opartych na architekturze ARM [7]. Następnie zainstalowano środowisko konteneryzacji Docker Engine oraz Docker Compose Plugin, które umożliwiły uruchomienie i izolację poszczególnych usług w odrębnych kontenerach. Takie podejście pozwoliło na zachowanie wysokiej elastyczności, łatwości aktualizacji oraz niezależności pomiędzy komponentami systemu.

W jednym z kontenerów uruchomiono popularną platformę Home Assistant w wersji 2025.6.3 [8], która stanowiła centralny element projektu i pełniła rolę serwera automatyki domowej. System ten odpowiadał za integrację urządzeń, pobieranie danych z czujników, zarządzanie logiką sterowania oświetleniem oraz realizację zdefiniowanych automatyzacji. Wszystkie operacje, takie jak generowanie nowych modeli sterowania, aktualizacja danych w bazie czy aktywacja trybu inteligentnego oświetlenia, były inicjowane i kontrolowane właśnie przez Home Assistant.

Komunikacja z urządzeniami peryferyjnymi, w tym z włącznikiem światła i czujnikiem natężenia oświetlenia odbywała się poprzez interfejs API Home Assistant. To umożliwiało pełną integrację sprzętu z oprogramowaniem oraz centralne zarządzanie wszystkimi elementami systemu z poziomu jednego środowiska.

4.3. Czujnik natężenia światła

Rysunek 3. Czujnik natężenia światła przymocowany do szyby

Czujnikiem odpowiedzialnym za pomiar natężenia oświetlenia w pomieszczeniu było urządzenie luxSensor firmy BleBox [9]. Jest to nowoczesny czujnik światła zasilany przewodowo za pomocą interfejsu USB typu A (5 V), który komunikuje się z systemem przy użyciu sieci Wi-Fi. Wszystkie urządzenia uwzględnione w projekcie zostały połączone z tym samym routerem bezprzewodowym. Takie podejście zapewnia ich integrację w ramach jednej sieci lokalnej oraz ułatwia komunikację z centralnym serwerem, czyli mikrokomputerem Raspberry Pi.

Czujnik luxSensor charakteryzuje się zakresem pomiarowym od 0 do 50 000 luksów przy wysokiej rozdzielczości 0.01 lux, co umożliwia bardzo precyzyjne monitorowanie poziomu oświetlenia zarówno w warunkach dziennych, jak i przy słabym świetle. Urządzenie zostało zamontowane wewnątrz pomieszczenia, na wewnętrznej stronie szyby okna. Warto jednak zaznaczyć, że dzięki obudowie o klasie szczelności IP65, czujnik ten może być bezpiecznie stosowany również na zewnątrz budynku, gdzie narażony byłby na działanie czynników atmosferycznych. Takie rozwiązanie czyni go uniwersalnym elementem systemów inteligentnego domu, umożliwiającym adaptację do różnych warunków środowiskowych.

Przy montażu urządzenia należy zwrócić szczególną uwagę, aby było ono odpowiednio odizolowane od sztucznych źródeł światła, takich jak oświetlenie pokojowe. Jest to o tyle ważne, że mogłoby to zaburzyć wyniki pomiaru natężenia światła.

Istotną zaletą przyjętej architektury systemu jest zastosowanie interfejsu API platformy Home Assistant [10], który pozwala na pełną elastyczność w doborze urządzeń pomiarowych. Oznacza to, że w ramach projektu można wykorzystać dowolny czujnik natężenia światła, bądź inne urządzenie, kompatybilne z Home Assistant, niezależnie od producenta czy modelu. API przy odpytywaniu o stan urządzenia zwraca dane w uniwersalnym formacie JSON, co zapewnia spójność wymiany informacji i umożliwia bezproblemową integrację z pozostałymi elementami systemu.

W kontekście uczenia maszynowego, parametr natężenia światła stanowi kluczową cechę wejściową wykorzystywaną w modelu predykcyjnym. Dzięki zastosowaniu mechanizmu personalizacji modelu ML, a nie wykorzystania uniwersalnego modelu, system jest w stanie automatycznie dostosować skalę danych wejściowych w przypadku użycia innego czujnika o odmiennym zakresie pomiarowym. Należy jednak podkreślić, że model powinien pracować na danych o tej samej skali, na której był trenowany, gwarantuje to spójność i poprawność uzyskiwanych wyników predykcji.

4.4. Włącznik światła

Rysunek 4. Włącznik światła

Kolejnym elementem systemu jest jednokanałowy przełącznik Sonoff T1EU [11], odpowiedzialny za sterowanie oświetleniem w pomieszczeniu. Urządzenie to jest przeznaczone do montażu podtynkowego i umożliwia zarówno manualne, jak i zdalne włączanie oraz wyłączanie źródła światła. Zasilane jest napięciem 230 V AC, a komunikacja odbywa się za pośrednictwem sieci Wi-Fi, co pozwala na integrację z platformą Home Assistant.

Sonoff T1EU współpracuje z chmurową platformą eWeLink [12], która stanowi pośrednika w komunikacji pomiędzy urządzeniem a systemem Home Assistant. To właśnie eWeLink odpowiada za przesyłanie informacji o aktualnym stanie przełącznika (ON/OFF) oraz odbieranie poleceń sterujących wysyłanych z Home Assistant. Takie rozwiązanie, mimo że wymaga połączenia z Internetem, zapewnia stabilność działania i pozwala na zdalne sterowanie oświetleniem z dowolnego miejsca.

Zastosowany przełącznik Sonoff T1EU obsługuje jeden kanał sterowania, co oznacza możliwość zarządzania jednym obwodem elektrycznym. Jego stan logiczny przyjmuje wyłącznie dwie wartości: ON (włączony) lub OFF (wyłączony). Z punktu widzenia systemu automatyki domowej urządzenie działa zatem w sposób binarny, co upraszcza proces przetwarzania danych i implementację logiki sterowania.

Dwustanowy charakter przełącznika ma również znaczenie w kontekście modelu uczenia maszynowego wykorzystanego w projekcie. Zmienna reprezentująca stan oświetlenia przyjmuje charakter kategoryczny, z dwiema możliwymi klasami odpowiadającymi włączeniu i wyłączeniu światła.

Pomimo swojej prostoty, przełącznik Sonoff T1EU w pełni spełnia założenia etapu prototypowego projektu. Zapewnia niezawodną komunikację poprzez platformę eWeLink, stabilne działanie w środowisku sieciowym oraz łatwą integrację z systemem Home Assistant.

5. Architektura serwisów

Mikrokomputer Raspberry Pi 5 posiadał zainstalowany na zewnętrznym dysku SSD system operacyjny Ubuntu 24.04.2 LTS, a także środowisko konteneryzacji Docker w wersji 28.2.2 oraz Docker Compose w wersji v2.36.2. Taki zestaw narzędzi umożliwił uruchomienie szeregu niezbędnych serwisów w kontenerach, zgodnie z konfiguracją zdefiniowaną w przygotowanym pliku `docker-compose.yml`.

5.1. Home Assistant

Najważniejszym elementem systemu jest oprogramowanie typu asystent domowy. W projekcie zastosowano platformę Home Assistant, cieszącą się dużą popularnością wśród entuzjastów automatyki domowej. Wybór ten był podyktowany przede wszystkim rozbudowanym systemem integracji z różnorodnymi urządzeniami. Istotną kwestią była też aktywna społeczność użytkowników i deweloperów, a także otwarty kod źródłowy, który umożliwia pełną swobodę modyfikacji i dostosowania oprogramowania do indywidualnych potrzeb.

Listing 1. Fragment docker-compose.yml zawierający konfigurację do uruchomienia Home Assistant

```
services:
  homeassistant:
    container_name: homeassistant
    build: ./homeassistant
    restart: unless-stopped
    privileged: true
    ports:
      - "8123:8123"
    environment:
      - TZ=Europe/Warsaw
    volumes:
      - ./homeassistant/config:/config
    networks:
      - automation_net
```

Fragment pliku `docker-compose.yml` (listing 1) [13] definiuje usługę o nazwie `homeassistant`, która odpowiada za uruchomienie instancji systemu automatyki domowej Home Assistant w środowisku kontenerowym Docker. W ramach konfiguracji określono szereg parametrów opisujących sposób budowania, uruchamiania i integracji kontenera z systemem gospodarza.

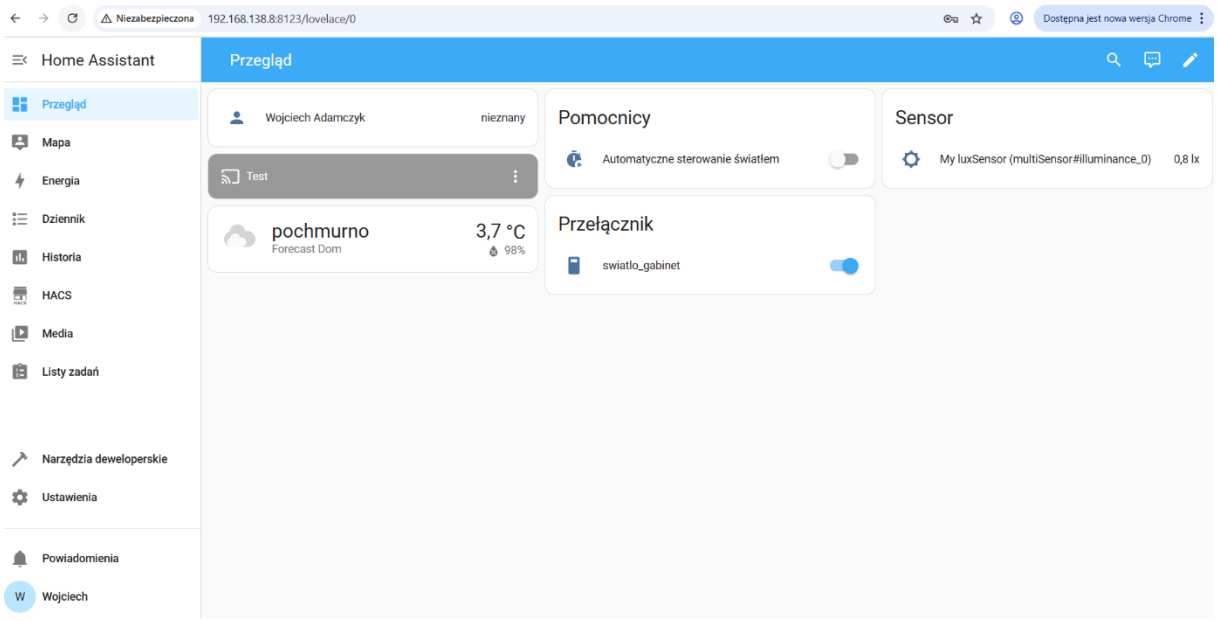
Kontener otrzymuje nazwę `homeassistant`, dzięki czemu można się do niego w prosty sposób odwoływać z poziomu Dockera, zarówno w komendach zarządzających, jak i w odniesieniach pomiędzy usługami. Obraz kontenera jest budowany lokalnie z pliku Dockerfile umieszczonego w katalogu `./homeassistant`. Zastosowano politykę ponownego uruchamiania `unless-stopped`, co oznacza, że kontener będzie automatycznie restartowany po ponownym uruchomieniu systemu lub po wystąpieniu awarii, o ile nie zostanie wcześniej zatrzymany ręcznie.

Kontener został uruchomiony w trybie uprzywilejowanym (`privileged: true`), co zapewnia mu rozszerzony dostęp do zasobów systemu gospodarza. Jest to rozwiązanie często stosowane w przypadku Home Assistanta, gdy konieczna jest komunikacja z urządzeniami fizycznymi poprzez porty USB, takie jak bramki Zigbee czy Z-Wave.

W sekcji mapowania portów wskazano, że port 8123 na hoście jest przekierowany na port 8123 wewnątrz kontenera. Dzięki temu interfejs użytkownika Home Assistanta jest dostępny z poziomu przeglądarki pod adresem http://localhost:8123.

Zmienna środowiskowa TZ została ustawiona na wartość `Europe/Warsaw`, co determinuje strefę czasową używaną w działającym kontenerze. W sekcji `volumes` zdefiniowano trwały wolumen danych. Lokalny katalog `./homeassistant/config` jest mapowany do katalogu `/config` wewnątrz kontenera. Umożliwia to przechowywanie konfiguracji i danych Home Assistanta poza kontenerem, zapewniając ich trwałość nawet po jego usunięciu lub ponownym utworzeniu.

Ostatni element konfiguracji, czyli przypisanie do sieci `automation_net`, powoduje, że kontener jest częścią logicznej sieci Dockera, co pozwala mu komunikować się z innymi usługami w ramach wspólnego środowiska (np. MLflow, MongoDB itp.).



Rysunek 5. Interfejs użytkownika (UI) Home Assistant

Na rysunku 4 przedstawiono interfejs użytkownika systemu Home Assistant zaimplementowanego w ramach projektu. Ekran główny składa się z szeregu kafelków prezentujących najważniejsze elementy systemu sterowania.

Najistotniejsze z nich to:

- Pomocnicy - zawiera przełącznik „Automatyczne sterowanie światłem”, który umożliwia uruchomienie trybu sterowania oświetleniem z wykorzystaniem najnowszego modelu uczenia maszynowego wytrenowanego na zebranych danych. Tryb ten wykorzystywany jest podczas nieobecności mieszkańców w domu.
- Przełącznik - obejmuje element „swiatlo_gabinet” służący do ręcznego włączania i wyłączania oświetlenia w pomieszczeniu objętym badaniem. To właśnie ten przełącznik stanowił obiekt sterowania przez model predykcyjny. Dzięki wizualizacji graficznej użytkownik może w każdej chwili sprawdzić, czy światło jest aktualnie włączone, czy wyłączone.
- Sensor - prezentuje dane z czujnika natężenia światła, które stanowiły kluczową cechę wejściową wykorzystywaną w procesie uczenia oraz w trakcie predykcji wykonywanych przez model.

5.2. MLflow

MLflow [14] jest platformą przeznaczoną do kompleksowego zarządzania cyklem życia projektów uczenia maszynowego (ang. *Machine Learning Lifecycle Management*). Została opracowana w 2018 roku przez firmę Databricks, a obecnie jest rozwijana jako projekt open source pod egidą Linux Foundation (MLflow, 2025). Platforma ta stanowi odpowiedź na rosnącą złożoność procesów związanych z tworzeniem, trenowaniem, oceną, wersjonowaniem i wdrażaniem modeli uczenia maszynowego w środowiskach produkcyjnych.

Głównym celem MLflow jest zapewnienie reproduktywności eksperymentów, transparentności procesu badawczego oraz ułatwienia wdrażania modeli w sposób powtarzalny i niezależny od stosowanej biblioteki czy infrastruktury obliczeniowej. MLflow integruje się z wieloma popularnymi frameworkami, takimi jak Scikit-learn, TensorFlow, PyTorch, czy XGBoost, a także z językami programowania Python, R i Java.

Platforma MLflow składa się z czterech podstawowych komponentów, które wspólnie tworzą spójny ekosystem zarządzania projektami uczenia maszynowego:

- MLflow Tracking - moduł odpowiedzialny za rejestrowanie i przechowywanie metadanych związanych z eksperymentami, takich jak parametry modeli, uzyskane metryki jakości, artefakty (np. pliki wag, wykresy), wersje kodu oraz środowiska uruchomieniowe. Komponent ten wspiera proces porównywania eksperymentów oraz analizę wyników w kontekście replikowalności badań.
- MLflow Projects - komponent służący do definiowania projektów w sposób umożliwiający ich powtarzalne uruchamianie. Wykorzystuje on pliki konfiguracyjne (MLproject), które określają strukturę projektu, zależności środowiskowe oraz sposób uruchomienia eksperymentów.
- MLflow Models - ujednolicony format zapisu modeli, który pozwala na ich standaryzowane przechowywanie i wdrażanie. Modele zapisane w tym formacie mogą być łatwo uruchamiane w różnych środowiskach.
- MLflow Model Registry - centralny rejestr modeli, umożliwiający wersjonowanie, nadawanie statusów oraz zarządzanie cyklem życia modeli. Zapewnia on kontrolę nad procesem publikacji modeli oraz ich aktualizacją w środowisku produkcyjnym.

Główną funkcjonalnością wykorzystywaną w projekcie w ramach systemu MLflow jest moduł MLflow Model Registry, służący do centralnego zarządzania modelami uczenia maszynowego. Każdy zarejestrowany model posiada własną nazwę oraz pełną historię wersji, które są automatycznie tworzone przy każdej nowej publikacji modelu.

Listing 2. Fragment docker-compose.yml zawierający konfigurację do uruchomienia MLflow

```
services:
 mlflow:
    image: ghcr.io/mlflow/mlflow:v3.1.0
    container_name: mlflow
    restart: unless-stopped
    ports:
      - "5000:5000"
    environment:
      - MLFLOW_S3_ENDPOINT_URL=${MLFLOW_S3_ENDPOINT_URL}
      - AWS_ACCESS_KEY_ID=${MINIO_ROOT_USER}
      - AWS_SECRET_ACCESS_KEY=${MINIO_ROOT_PASSWORD}
    volumes:
```

```
    - ./mlflow/db:/data/mlflow/db
  command: >
    mlflow server
    --backend-store-uri sqlite:////data/mlflow/db/mlflow.db
    --serve-artifacts
    --default-artifact-root s3://${MINIO_BUCKET}
    --host 0.0.0.0
  networks:
    - automation_net
```

Fragment pliku `docker-compose.yml` (listing 2) definiuje usługę o nazwie `mlflow`, która uruchamia serwer MLflow w kontenerze Docker. Usługa korzysta z gotowego obrazu kontenera o nazwie `ghcr.io/mlflow/mlflow:v3.1.0`, pobieranego z rejestru GitHub Container Registry. Dzięki temu nie ma potrzeby lokalnego budowania obrazu, wystarczy jego pobranie i uruchomienie. Kontener otrzymuje nazwę `mlflow`, co ułatwia jego identyfikację i odwoływanie się do niego w ramach systemu Docker. Podobnie jak w Home Assistant zastosowano politykę restartu `unless-stopped`.

W sekcji `ports` zdefiniowano mapowanie portów `5000:5000`. Oznacza to, że port `5000` kontenera, na którym działa interfejs serwera MLflow, jest dostępny na tym samym porcie hosta. Użytkownik może więc uzyskać dostęp do interfejsu WWW MLflow, wpisując w przeglądarce adres http://localhost:5000.

Sekcja `environment` określa zestaw zmiennych środowiskowych, które definiują sposób integracji MLflow z zewnętrznym systemem przechowywania artefaktów:

- Zmienna `MLFLOW_S3_ENDPOINT_URL` wskazuje adres punktu końcowego usługi zgodnej z protokołem S3, która służy do przechowywania plików artefaktów (np. w usłudze MinIO).
- Zmienne `AWS_ACCESS_KEY_ID` i `AWS_SECRET_ACCESS_KEY` przekazują dane uwierzytelniające wymagane do komunikacji z tym magazynem. W tym przypadku są one dynamicznie wczytywane z pliku `.env`, gdzie zapisano dane użytkownika (`MINIO_ROOT_USER`) i hasło (`MINIO_ROOT_PASSWORD`).

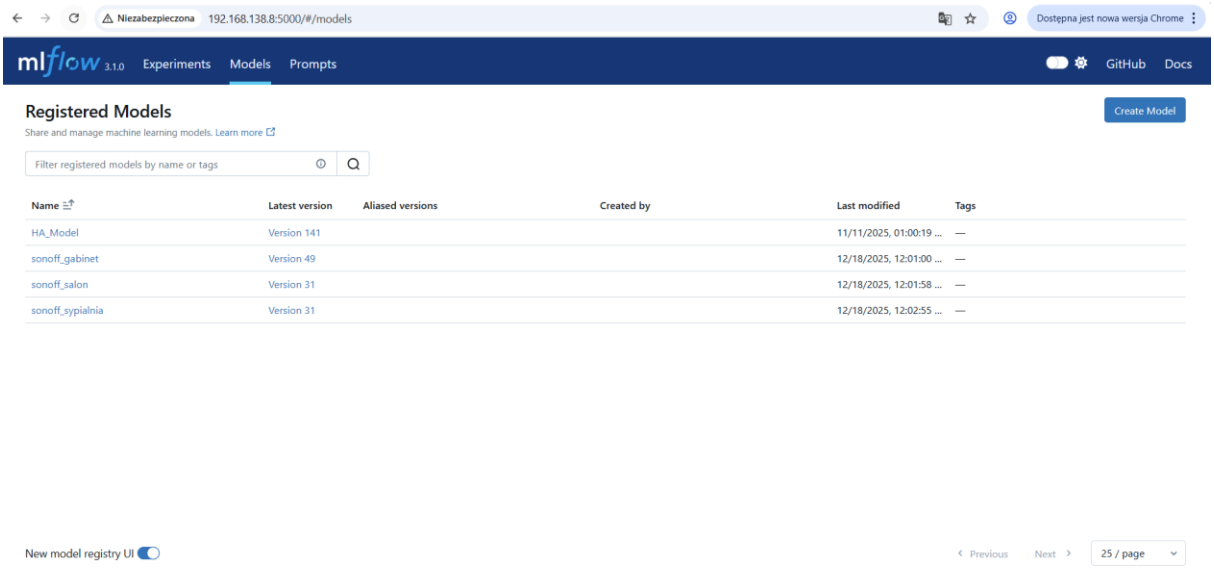
Sekcja `volumes` definiuje wolumen `./mlflow/db:/data/mlflow/db`, który mapuje lokalny katalog projektu na katalog wewnątrz kontenera. Wolumen ten przechowuje plik bazy danych SQLite (`mlflow.db`), w którym zapisywane są metadane dotyczące eksperymentów MLflow (np. nazwy eksperymentów, parametry modeli, wyniki). Takie rozwiązanie zapewnia trwałość danych nawet po usunięciu kontenera.

Instrukcja `command` definiuje polecenie uruchamiane w kontenerze po jego starcie. W tym przypadku jest to uruchomienie serwera MLflow z określonymi parametrami:

- `--backend-store-uri sqlite:////data/mlflow/db/mlflow.db` określa lokalizację bazy danych SQLite, która przechowuje metadane.
- `--serve-artifacts` umożliwia serwerowi MLflow bezpośrednie serwowanie artefaktów zapisanych w magazynie S3.
- `--default-artifact-root s3://${MINIO_BUCKET}` wskazuje domyślną lokalizację w przestrzeni S3 (np. w MinIO), w której zapisywane są artefakty powiązane z eksperymentami.
- `--host 0.0.0.0` umożliwia dostęp do serwera ze wszystkich interfejsów sieciowych kontenera, co jest niezbędne do udostępnienia interfejsu użytkownikom spoza kontenera.

Ostatnim elementem konfiguracji jest przypisanie, podobnie jak wcześniej, kontenera do sieci `automation_net`, co pozwala mu komunikować się z innymi usługami w ramach tej samej sieci.

Podsumowując, omawiana definicja usługi `mlflow` w pliku `docker-compose.yml` tworzy w pełni funkcjonalny serwer MLflow skonfigurowany do współpracy z lokalnym magazynem S3 (np. MinIO) oraz bazą danych SQLite, umożliwiający trwałe i wygodne zarządzanie cyklem życia modeli uczenia maszynowego.



Rysunek 6. Interfejs użytkownika (UI) MLflow

Na rysunku 6 przedstawiono zrzut ekranu interfejsu użytkownika systemu MLflow opracowanego w ramach projektu. Widoczna jest w nim sekcja `Models`, zawierająca listę modeli zarejestrowanych podczas przeprowadzonych badań.

Pierwotnie przewidziano tylko jeden model o roboczej nazwie HA_Model. Jednak w toku konsultacji, w celu zademonstrowania możliwości obsługi wielu źródeł światła przy wykorzystaniu opracowanego rozwiązania. Podjęto decyzję o dodaniu kolejnych modeli. Zmiana ta wymusiła modyfikację schematu nazewnictwa modeli. W efekcie powstały trzy modele: `sonoff_gabinet`, `sonoff_salon` oraz `sonoff_sypialnia`.

Z uwagi na fakt, że projekt bazował na jednym fizycznym przełączniku i jednym źródle światła, faktycznie użytecznym i uczonym na rzeczywistych danych modelem był `sonoff_gabinet`. Pozostałe modele zostały utworzone równolegle, lecz trenowano je na danych losowych, pełniąc funkcję demonstracyjną w kontekście możliwości skalowania systemu.

Takie podejście pozwala w przyszłości w prosty sposób rozszerzyć system o kolejne pomieszczenia i źródła światła, zapewniając elastyczność i skalowalność rozwiązania w miarę rozwoju infrastruktury inteligentnego domu.

5.3. MinIO

MinIO [15] to otwartoźródłowy system do przechowywania obiektów (ang. *object storage*), kompatybilny z API Amazon S3, co czyni go łatwym do integracji z narzędziami i usługami, które wspierają S3. MinIO bywa wykorzystywane jako tańsza, samodzielna alternatywa dla chmurowych usług (jak AWS S3), zapewniając kontrolę nad danymi, przy jednoczesnej skalowalności i elastyczności.

Integracja MLflow z MinIO [16] polega na skonfigurowaniu MinIO jako `artifact store` dla serwera MLflow. Dzięki temu wszystkie artefakty powiązane z eksperymentami, takie jak zapisane modele, są przechowywane w MinIO w sposób bezpieczny i uporządkowany. W praktyce proces ten polega na uruchomieniu serwera MinIO, utworzeniu odpowiedniego pojemnika (ang. *bucket*) oraz przekazaniu MLflow informacji o lokalizacji tego zasobu poprzez zmienne środowiskowe, takie jak `MLFLOW_S3_ENDPOINT_URL`, `AWS_ACCESS_KEY_ID` i `AWS_SECRET_ACCESS_KEY`. MLflow traktuje wówczas MinIO jako źródło zgodne z S3 i zapisuje artefakty w nim tak, jakby korzystał z Amazon S3.

Listing 3. Fragment docker-compose.yml zawierający konfigurację do uruchomienia MinIO

```
services:
  minio:
    image: minio/minio:latest
    container_name: minio
    restart: unless-stopped
    environment:
      - MINIO_ROOT_USER=${MINIO_ROOT_USER}
      - MINIO_ROOT_PASSWORD=${MINIO_ROOT_PASSWORD}
    volumes:
      - ./minio/data:/data
    ports:
      - "9000:9000"
      - "9001:9001"
    command: server --console-address ":9001" /data
    networks:
      - automation_net
```

Fragment pliku `docker-compose.yml` przedstawiony w listingu 3 definiuje usługę `minio`, która uruchamia serwer obiektowego magazynu danych MinIO w środowisku Docker. Usługa korzysta z oficjalnego obrazu `minio/minio:latest`, który jest pobierany bezpośrednio z publicznego rejestru Docker Hub. Parametr `container_name` nadaje kontenerowi nazwę `minio`. Tu również ustawiono politykę `restart: unless-stopped`, co zapewnia automatyczne ponowne uruchamianie kontenera po restarcie systemu lub po jego awarii.

Sekcja `environment` definiuje zmienne środowiskowe wymagane do skonfigurowania podstawowego uwierzytelniania w serwerze MinIO. [17]

- `MINIO_ROOT_USER` określa nazwę użytkownika,
- `MINIO_ROOT_PASSWORD` zawiera hasło użytkownika,

Wartości tych zmiennych są dynamicznie wczytywane z pliku `.env`, co pozwala oddzielić dane konfiguracyjne i poufne od samego pliku `docker-compose.yml`, zwiększając bezpieczeństwo oraz elastyczność konfiguracji.

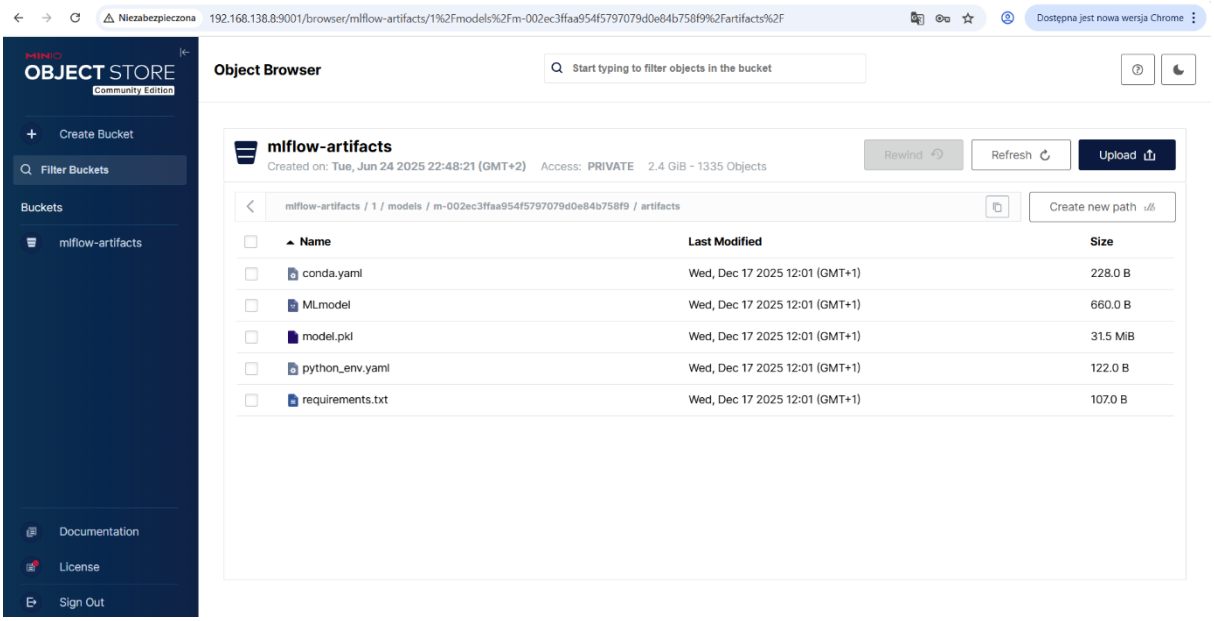
W sekcji `volumes` zdefiniowano mapowanie katalogu `./minio/data` z hosta na katalog `/data` wewnątrz kontenera. Dzięki temu wszystkie dane zapisane w MinIO są przechowywane lokalnie w systemie plików hosta. Wolumen ten stanowi trwały magazyn danych obiektowych, takich jak pliki modeli, logi czy inne zasoby.

W sekcji `ports` określono mapowanie portów:

- `9000:9000` - port, na którym działa główny interfejs API zgodny z S3, wykorzystywany do komunikacji z aplikacjami (np. MLflow),

- `9001:9001` - port interfejsu graficznego MinIO Console, umożliwiającego zarządzanie zasobami poprzez przeglądarkę internetową.

Dzięki temu po uruchomieniu kontenera użytkownik może uzyskać dostęp do konsoli administracyjnej pod adresem http://localhost:9001, a do interfejsu API przez http://localhost:9000.



Rysunek 7. Interfejs użytkownika (UI) MinIO

Na rysunku 7 przedstawiono interfejs użytkownika systemu MinIO wykorzystanego w projekcie. Widoczna jest na nim lista artefaktów (plików) powiązanych z przykładowym modelem uczenia maszynowego.

Kluczowym elementem jest plik `model.pkl`, który stanowi zserializowany, gotowy do użycia model. Kolejnym istotnym plikiem jest `MLmodel`, zawierający metadane opisujące strukturę oraz parametry danego modelu. Pozostałe pliki pełniące funkcje pomocnicze, są wykorzystywane do odtworzenia środowiska uruchomieniowego niezbędnego do poprawnego działania modelu. [18]

5.4. MongoDB

MongoDB [19] jest nierelacyjną bazą danych opartą na modelu dokumentowym. Jedną z jej kluczowych zalet jest elastyczność schematu danych. W przeciwieństwie do relacyjnych baz danych, w których struktura tabel musi zostać zdefiniowana z góry, w MongoDB każdy dokument w obrębie kolekcji może posiadać różny zestaw pól i typów danych.

Takie podejście umożliwia szybki rozwój aplikacji, łatwe modyfikowanie struktury danych oraz iteracyjne dodawanie nowych atrybutów bez konieczności kosztownej migracji schematu. W kontekście projektów badawczych i prototypowych jest to szczególnie korzystne. Pozwala na bardziej eksperymentalne podejście do projektowania danych, co stanowiło główny powód wyboru tej bazy w ramach realizowanego projektu.

Dodatkowym argumentem przemawiającym za rezygnacją z relacyjnej bazy danych był fakt, że projekt zakładał utworzenie jedynie pojedynczej kolekcji danych, co nie wymagało stosowania złożonych relacji charakterystycznych dla tradycyjnych systemów SQL.

Listing 4. Fragment docker-compose.yml zawierający konfigurację do uruchomienia MongoDB

```
services:
  mongodb:
    container_name: mongodb
    image: mongo:6
    restart: unless-stopped
    ports:
      - "27017:27017"
    volumes:
      - ./mongodb/data:/data/db
    environment:
      - MONGO_INITDB_ROOT_USERNAME=${MONGO_ROOT_USER}
      - MONGO_INITDB_ROOT_PASSWORD=${MONGO_ROOT_PASSWORD}
    networks:
      - automation_net
```

Listing 4 przedstawia fragment pliku `docker-compose.yml` definiujący usługę o nazwie `mongodb`, która uruchamia kontener z bazą MongoDB w wersji 6.

Usługa korzysta z oficjalnego obrazu `mongo:6`, dostępnego w publicznym rejestrze Docker Hub. Kontener otrzymuje nazwę `mongodb`. Parametr restart podobnie jak przy poprzednich serwisach został ustawiony na: `unless-stopped`.

W sekcji `ports` zdefiniowano mapowanie portu. Port `27017` na hoście jest przekierowany na port `27017` w kontenerze, który stanowi domyślny port serwera MongoDB. Dzięki temu baza danych jest dostępna z poziomu hosta, np. przez aplikacje klienckie, narzędzia administracyjne czy wiersz poleceń.

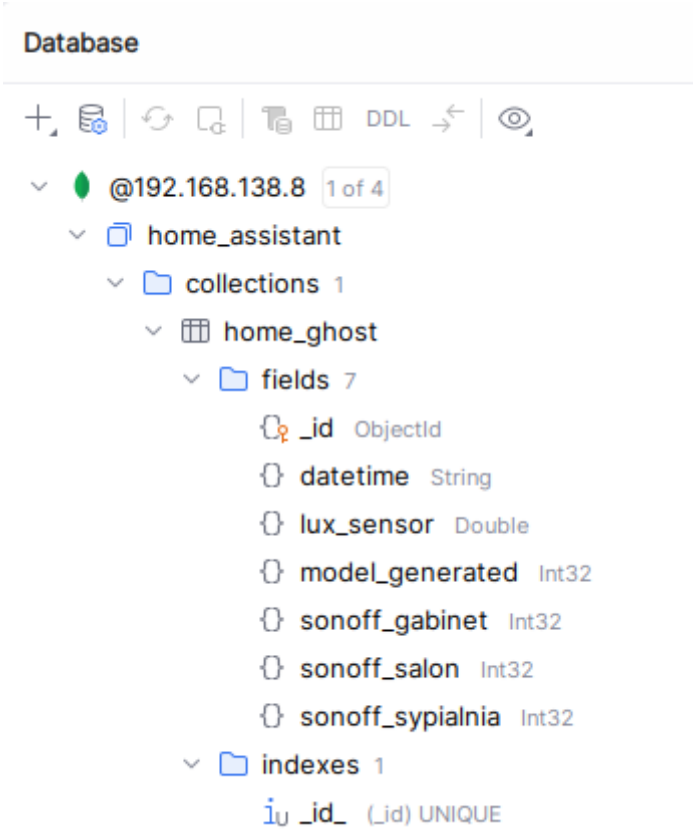
Sekcja `volumes` definiuje trwały wolumen danych. Katalog `./mongodb/data` na hoście jest mapowany do `/data/db` wewnątrz kontenera, czyli domyślnego katalogu przechowywania danych MongoDB. Dzięki temu dane są trwałe i nie zostaną utracone w przypadku ponownego utworzenia kontenera.

W sekcji `environment` określono zmienne środowiskowe służące do inicjalizacji instancji MongoDB z kontem administratora:

- `MONGO_INITDB_ROOT_USERNAME=${MONGO_ROOT_USER}` - ustawia nazwę użytkownika administracyjnego (`root`).
- `MONGO_INITDB_ROOT_PASSWORD=${MONGO_ROOT_PASSWORD}` - ustawia hasło użytkownika administracyjnego.

Wartości tych zmiennych są pobierane z pliku `.env`. Konto administracyjne tworzone jest automatycznie przy pierwszym uruchomieniu kontenera i zapewnia pełny dostęp do bazy danych.

Na końcu konfiguracji podobnie jak w innych serwisach w ramach projektu zdefiniowano przynależność do sieci dzięki `automation_net`.



Rysunek 8. Struktura projektowej bazy MongoDB

Na rysunku 8 przedstawiono strukturę bazy danych MongoDB wykorzystanej w projekcie. W ramach pracy, w bazie o nazwie `home_assistant`, utworzono jedną kolekcję o nazwie `home_ghost`, zawierającą następujące pola:

- `_id` [`ObjectId`] - domyślne pole bazy MongoDB, przechowujące unikalny identyfikator dokumentu,
- `datetime` [`String`] - pole przechowujące datę i godzinę próbki w formacie `YYYY-MM-DDTHH:MM:SS.000000+00:00`; czas zapisywany jest w uniwersalnej strefie czasowej UTC,
- `lux_sensor` [`Double`] - wartość pomiaru z czujnika natężenia światła zewnętrznego, zapisywana z dokładnością do dwóch miejsc po przecinku,
- `model_generated` [`int32`] - pole binarne informujące czy aktualny stan przełącznika oświetlenia został wygenerowany przez model predykcyjny (1), czy wynikał z ręcznej decyzji użytkownika (0),
- `sonoff_gabinet` [`int32`] - stan przełącznika światła (0/1) będącego fizycznym elementem systemu,
- `sonoff_salon i sonoff_sypialnia` [int32] - losowo generowane dane binarne (0/1), wykorzystywane w celach demonstracyjnych do trenowania kolejnych modeli.

6. Architektura oprogramowania

Kolejnym istotnym elementem projektu było opracowanie autorskich skryptów w języku Python, odpowiedzialnych za realizację wszystkich założeń funkcjonalnych stworzonego systemu. Kod ten jest w pełni zintegrowany z platformą Home Assistant, a jego uruchamianie, harmonogramowanie oraz kontrola działania odbywają się bezpośrednio z poziomu tego systemu.

6.1. Narzędzia pomocnicze

Podstawową funkcjonalnością opracowanych skryptów jest komunikacja z bazą danych, serwisem MLflow oraz z platformą Home Assistant. W tym celu utworzono klasy pomocnicze, pełniące rolę konektorów, które są wykorzystywane w kodzie właściwym poszczególnych procedur.

Dodatkowo zastosowano mechanizm dostarczania parametrów konfiguracyjnych niezbędnych do poprawnego działania systemu, w szczególności w postaci zmiennych środowiskowych.

6.1.1 Dostarczanie danych konfiguracyjnych.

W projekcie wykorzystano bibliotekę Python Dynaconf w wersji 3.2.12 [20] przeznaczoną do kompleksowego zarządzania konfiguracją aplikacji. Jej podstawowym celem jest oddzielenie warstwy konfiguracyjnej od kodu właściwego programu. Dzięki temu aplikacja staje się bardziej czytelna, łatwiejsza w utrzymaniu oraz lepiej przystosowana do pracy w różnych środowiskach uruchomieniowych.

Biblioteka umożliwia ładowanie konfiguracji z wielu źródeł jednocześnie, takich jak pliki konfiguracyjne w popularnych formatach (`TOML`, `YAML`, `JSON`, `INI`), w projekcie wykorzystano format `YAML` (listing 5).

Listing 5. Plik konfiguracyjny config.yml stanowiący źródło parametrów w projekcie

```
MongoDB:
  host: "@format {env[MONGO_HOST]}"
  root_user: "@format {env[MONGO_ROOT_USER]}"
  root_password: "@format {env[MONGO_ROOT_PASSWORD]}"
  db: "@format {env[MONGO_DB]}"
  collection: "@format {env[MONGO_COLLECTION]}"
  train_data_days: 365

Mlflow:
  tracking_uri: "@format {env[MLFLOW_TRACKING_URI]}"
  experiment: "@format {env[MLFLOW_EXPERIMENT]}"

HomeAssistant:
  host: "@format {env[HOMEASSISTANT_HOST]}"
  token: "@format {env[HOMEASSISTANT_TOKEN]}"

Sensors:
  sensor: "sensor.my_luxsensor_multisensor_illuminance_0"

Switch:
```

```
  switch: "input_boolean.run_script_loop_switch"

Lights:
  light_1: "switch.sonoff_gabinet"
  light_2: "switch.sonoff_salon"
  light_3: "switch.sonoff_sypialnia"

ModelType:
  model: "rfc"
```

Jak widać w listingu 5 Dynaconf umożliwia uporządkowanie pliku w sekcje co znacznie poprawia czytelność:

- `MongoDB` – sekcja poświęcona parametrom niezbędnym do komunikacji z bazą danych,
- `Mlflow` – parametry wykorzystywane przy połączeniu z MLflow i wskazująca na konkretny eksperyment MLflow w ramach którego agregowane są modele,
- `HomeAssistant` – adres oraz token do połączenia z serwisem Home Assistant,
- `Sensors` – sekcja zawierająca nazwę czujnika natężenia światła na potrzeby identyfikacji w serwisie,
- `Switch` – sekcja z nazwą przełącznika do włączania i wyłączania pracy na modelu w projekcie
- `Lights` – sekcja z listą włączników światła objętych sterowaniem w projekcie (w ramach projektu realnym włącznikiem jest `switch.sonoff_gabinet`,
- `ModelType` – sekcja zawierająca nazwę typu modelu aktualnie uczonego i wykorzystanego w projekcie. Nazwa jest zgodna z utworzoną mapą modeli omówionych w podrozdziale **Porównanie Modeli**,

Niektóre parametry zostały wprowadzone w pliku konfiguracyjnym w formie tekstowej, natomiast część jest dostarczana ze zmiennych środowiskowych (`MONGO_HOST`, `MONGO_ROOT_USER`, `MONGO_ROOT_PASSWORD`, …)

6.1.2 Komunikacja z bazą danych

Listing 6. Klasa wykorzystywana jako konektor do bazy danych

```
class MongoDBConnector:
    def __init__(self, config: MongoDBConnectorConfig) -> None:
        self.logger = logging.getLogger(class_name(self))
        self.config = config
        self.client =
MongoClient(f'mongodb://{config.user}:{config.password}@{config.host}/')
        db = self.client[config.db]
        self.collection = db[config.collection]

    def push_new_df_data(self, df: pd.DataFrame) -> None:
        data = df.to_dict(orient='records')
        if data:
            self.collection.insert_many(data)

    def push_new_record(self, record: dict) -> None:
        if record:
```

```
            self.collection.insert_one(record)


    def download_train_data(self) -> List[dict]:
        start_date = datetime.now(timezone.utc) -
timedelta(days=self.config.train_data_days)
        start_date_iso = start_date.strftime('%Y-%m-%dT%H:%M:%S.%f+00:00')
        cursor = self.collection.find({'datetime': {'$gte': start_date_iso},
'model_generated': 0})
        return list(cursor)


    def close(self) -> None:
        self.client.close()
```

Przedstawiony kod w listingu 6 definiuje klasę `MongoDBConnector`, którego celem jest zapewnienie dostępu do bazy danych MongoDB. Klasa ta pełni rolę warstwy pośredniej pomiędzy logiką aplikacji a systemem bazodanowym, upraszczając operacje zapisu i odczytu danych oraz centralizując konfigurację połączenia.

Konstruktor klasy inicjalizuje obiekt konektora na podstawie przekazanej konfiguracji typu `MongoDBConnectorConfig`. W pierwszym kroku tworzony jest `logger`, którego nazwa jest dynamicznie wyznaczana na podstawie nazwy klasy, co ułatwia diagnostykę i analizę logów. Następnie zapisywana jest konfiguracja, a przy użyciu klasy `MongoClient` nawiązywane jest połączenie z serwerem MongoDB. Parametry połączenia, takie jak użytkownik, hasło i host, są pobierane z obiektu konfiguracyjnego i składane w postaci adresu URI. Po ustanowieniu połączenia wybierana jest konkretna baza danych oraz kolekcja, na której będą wykonywane operacje, co sprawia, że obiekt `MongoDBConnector` jest od razu gotowy do pracy.

Metoda `push_new_df_data` odpowiada za zapis danych pochodzących z obiektu typu `pandas.DataFrame`. Dane te są konwertowane do listy słowników, gdzie każdy słownik reprezentuje pojedynczy dokument MongoDB. Zastosowanie orientacji `records` zapewnia bezpośrednie mapowanie wierszy `DataFrame` na dokumenty w kolekcji. Warunek sprawdzający, czy lista danych nie jest pusta, zapobiega wykonywaniu zbędnych operacji zapisu. Następnie dane są zapisywane do bazy za pomocą metody `insert_many`, co jest wydajnym rozwiązaniem w przypadku większych zbiorów danych.

Metoda `push_new_record` umożliwia zapis pojedynczego rekordu w postaci słownika. Podobnie jak w przypadku zapisu zbiorczego, operacja jest wykonywana tylko wtedy, gdy przekazany obiekt nie jest pusty. Zapis pojedynczego dokumentu realizowany jest poprzez metodę `insert_one`, co czyni tę funkcję odpowiednią do obsługi zdarzeń lub danych napływających w sposób jednostkowy.

Metoda `download_train_data` służy do pobierania danych treningowych z bazy MongoDB na potrzeby dalszego przetwarzania w kontekście uczenia maszynowego. W pierwszym kroku wyznaczana jest data początkowa, obliczana jako bieżący czas UTC pomniejszony o liczbę dni określoną w konfiguracji (`train_data_days`). Następnie data ta jest konwertowana do formatu ISO 8601, co zapewnia zgodność z formatem przechowywanym w bazie. Zapytanie do bazy wyszukuje dokumenty, których pole `datetime` jest większe lub równe wyznaczonej dacie początkowej oraz które spełniają dodatkowy warunek logiczny: `model_generated` równe 0. Gwarantuje on, że w ramach danych treningowych nie pojawią się dane oznaczone jako wygenerowane przez wcześniejszy model, co mogłoby wpłynąć negatywnie na proces uczenia kolejnych modeli. Wynik zapytania, zwrócony w postaci kursora, jest następnie konwertowany do listy słowników, co ułatwia dalsze wykorzystanie danych w aplikacji.

Ostatnia metoda, `close`, odpowiada za poprawne zamknięcie połączenia z bazą danych MongoDB. Jest to istotny element zarządzania zasobami, zapobiegający wyciekom połączeń i zapewniający stabilność aplikacji, zwłaszcza w środowiskach długotrwałego działania lub przy dużej liczbie operacji bazodanowych.

6.1.3 Komunikacja z MLflow

Listing 7. Klasa wykorzystywana jako Manager komunikacji z MLflow

```
class MlflowManager:
    def __init__(self, config: MlflowManagerConfig) -> None:
        self.logger = logging.getLogger(class_name(self))
        mlflow.set_tracking_uri(config.tracking_uri)
        mlflow.set_experiment(config.experiment)
        self.client = MlflowClient()

    @staticmethod
    def register_model(model: Any, model_name: str) -> str:
        with mlflow.start_run() as run:
            mlflow.sklearn.log_model(model, "model")
            model_uri = f"runs:/{run.info.run_id}/model"
            mlflow.register_model(model_uri, model_name)
        return run.info.run_id

    @staticmethod
    def register_params(run_id: str, **params) -> None:
        with mlflow.start_run(run_id=run_id):
            for key, value in params.items():
                mlflow.log_metric(key, value)

    def get_model(self, model_name: str) -> PyFuncModel:
        versions = self.client.get_latest_versions(model_name)
        latest_version = max(int(v.version) for v in versions)
        return mlflow.pyfunc.load_model(f"models:/{model_name}/{latest_version}")
```

Listing 7 zawiera kod definiujący klasę `MlflowManager`, której zadaniem jest zarządzanie cyklem życia modeli uczenia maszynowego z wykorzystaniem MLflow. Klasa ta stanowi warstwę abstrakcji nad interfejsem MLflow, upraszczając proces rejestrowania modeli, zapisywania metryk eksperymentów oraz pobierania najnowszych wersji zarejestrowanych modeli do dalszego wykorzystania w aplikacji.

Konstruktor klasy inicjalizuje menedżera MLflow na podstawie obiektu konfiguracyjnego typu `MlflowManagerConfig`. Podobnie jak wcześniej, tworzony jest `logger`. Następnie ustawiany jest adres serwera śledzenia eksperymentów (`tracking_uri`), co pozwala aplikacji komunikować się z odpowiednią instancją MLflow. Kolejnym krokiem jest ustawienie aktywnego eksperymentu, w ramach którego będą rejestrowane uruchomienia. Na końcu inicjalizowany jest obiekt `MlflowClient`, zapewniający niskopoziomowy dostęp do funkcji zarządzania modelami i ich wersjami.

Metoda statyczna `register_model` odpowiada za rejestrowanie nowego modelu w MLflow. W ramach nowego uruchomienia eksperymentu (`mlflow.start_run`) model jest zapisywany jako

artefakt przy użyciu funkcji `mlflow.sklearn.log_model`, co wskazuje, że kod jest przystosowany do obsługi modeli zgodnych z biblioteką Scikit-learn. Następnie konstruowany jest identyfikator URI modelu, odwołujący się do artefaktu zapisanego w ramach danego uruchomienia. Model ten zostaje zarejestrowany w rejestrze modeli MLflow pod podaną nazwą. Metoda zwraca identyfikator uruchomienia (`run_id`), który może być później wykorzystany do dalszego logowania metryk lub parametrów.

Metoda statyczna `register_params` służy do zapisywania metryk i parametrów powiązanych z istniejącym uruchomieniem MLflow. Przy użyciu identyfikatora `run_id` metoda ponownie otwiera kontekst danego uruchomienia, a następnie iteruje po przekazanych parametrach. Każda para klucz–wartość jest zapisywana jako metryka przy użyciu funkcji `mlflow.log_metric`. Takie podejście umożliwia stopniowe uzupełnianie informacji o eksperymencie, na przykład o wyniki walidacji lub testów modelu, już po jego zarejestrowaniu.

Metoda `get_model` umożliwia pobranie najnowszej wersji zarejestrowanego modelu na podstawie jego nazwy. W pierwszym kroku, za pomocą klienta MLflow, pobierana jest lista najnowszych wersji modelu dostępnych w rejestrze. Następnie wybierana jest wersja o najwyższym numerze, co odpowiada najbardziej aktualnej wersji modelu. Wybrany model jest ładowany przy użyciu interfejsu `mlflow.pyfunc.load_model`. Zwracany obiekt typu `PyFuncModel` może być bezpośrednio używany do wykonywania predykcji.

6.1.4 Komunikacja z Home Assistant

Listing 8. Klasa wykorzystywana jako konektor do Home Assistant

```
class HomeAssistantConnector:
    def __init__(self, config: HomeAssistantConnectorConfig) -> None:
        self.logger = logging.getLogger(class_name(self))
        self.config = config
        self.end_date = datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%SZ')

    @property
    def _headers(self) -> Dict:
        return {"Authorization": f"Bearer {self.config.token}"}

    @staticmethod
    def state_parser(state: str) -> Union[float, int, None]:
        try:
            return float(state)
        except ValueError:
            if isinstance(state, str):
                return 1 if state.lower() == 'on' else 0
        return None

    def get_device_state(self, device_id: str) -> Union[float, int, None]:
        url = f'{self.config.host}api/states/{device_id}'
        response = requests.get(url, headers=self._headers)
        if response.status_code == 200:
            result = response.json()['state']
            result = self.state_parser(result)
            if result is not None:
```

```
            return result
        else:
            raise DeviceException(f'No devices data: {device_id}')

    def get_device_data(self, device_id: str) -> Dict:
        url = f'{self.config.host}api/states/{device_id}'
        response = requests.get(url, headers=self._headers)
        if response.status_code == 200:
            return response.json()
        else:
            raise DeviceException(f'No devices data: {device_id}')

    def set_device_state(self, device_id: str, state: int) -> None:
        switch = 'turn_on' if state else 'turn_off'
        url = f'{self.config.host}api/services/switch/{switch}'
        requests.post(url, headers=self._headers, json={"entity_id": device_id})
```

Przedstawiona w listingu 8 klasa `HomeAssistantConnector`, ma za zadanie zapewnienie uporządkowanego interfejsu komunikacji z systemem Home Assistant za pośrednictwem REST API dającego możliwość zarówno odczytu stanu urządzeń, jak i sterowania ich pracą.

Konstruktor klasy inicjalizuje obiekt konektora na podstawie przekazanej konfiguracji `HomeAssistantConnectorConfig`. Podobnie jak przy konektorze do bazy danych, w pierwszej kolejności tworzony jest `logger` powiązany z nazwą klasy, a następnie zapisywana jest konfiguracja, zawierająca m.in. adres hosta oraz token autoryzacyjny. Dodatkowo inicjalizowane jest pole `end_date`, które przechowuje bieżącą datę i czas w formacie UTC (ISO 8601).

Właściwość `_headers` została zdefiniowana jako metoda typu `@property` i odpowiada za dynamiczne generowanie nagłówków HTTP wymaganych do autoryzacji żądań. Zwracany słownik zawiera nagłówek `Authorization` z tokenem typu `Bearer`, co jest standardowym mechanizmem uwierzytelniania w interfejsie API Home Assistant. Takie rozwiązanie upraszcza kod i eliminuje konieczność ręcznego definiowania nagłówków w każdej metodzie wykonującej zapytanie HTTP.

Metoda statyczna `state_parser` służy do ujednolicenia i normalizacji stanu urządzeń zwracanego przez API Home Assistanta. W pierwszej kolejności próbuje ona przekonwertować stan na wartość typu `float`, co pozwala obsłużyć urządzenia raportujące wartości liczbowe, takie jak czujniki natężenia światła. W przypadku niepowodzenia, a gdy stan jest łańcuchem znaków, metoda interpretuje wartości logiczne, mapując stan 'on' na 1, a wszystkie pozostałe wartości na 0. Jeżeli konwersja nie jest możliwa, metoda zwraca `None`, sygnalizując brak możliwości interpretacji danych.

Metoda `get_device_state` umożliwia pobranie aktualnego stanu konkretnego urządzenia z Home Assistanta. Tworzy ona adres URL odwołujący się do zasobu `/api/states/{device_id}` i wykonuje zapytanie typu GET z odpowiednimi nagłówkami autoryzacyjnymi. W przypadku poprawnej odpowiedzi (kod HTTP 200) metoda odczytuje pole `state` z odpowiedzi JSON, a następnie przetwarza je za pomocą metody `state_parser`. Jeżeli wynik parsowania jest poprawny, zostaje on zwrócony. W przeciwnym przypadku lub gdy żądanie zakończy się błędem, zgłaszany jest wyjątek `DeviceException`, informujący o braku danych dla wskazanego urządzenia.

Metoda `get_device_data` działa analogicznie do poprzedniej, jednak zamiast zwracać jedynie przetworzony stan urządzenia, zwraca pełną odpowiedź JSON otrzymaną z API Home Assistanta. Umożliwia to dostęp do dodatkowych informacji, takich jak atrybuty urządzenia, czas

ostatniej aktualizacji czy metadane, które mogą być istotne w bardziej zaawansowanych scenariuszach analitycznych lub diagnostycznych.

Metoda `set_device_state` odpowiada za zmianę stanu urządzenia w systemie Home Assistant. Na podstawie przekazanej wartości logicznej `state` wybierana jest odpowiednia usługa (`turn_on` lub `turn_off`), a następnie konstruowany jest adres URL odwołujący się do zasobu `/api/services/switch/{switch}`. Żądanie `POST` wysyłane jest z nagłówkami autoryzacyjnymi oraz z treścią JSON zawierającą identyfikator urządzenia (`entity_id`). W ten sposób możliwe jest zdalne sterowanie urządzeniami typu `switch` bezpośrednio z poziomu skryptu.

6.2. Generowanie danych w bazie

Listing 10. Klasa do budowania i zapisywania rekordów w bazie danych

```
class DBBuilder(BuilderMixin):
    def __init__(self, config: DataBuilderConfig, home_assistant: HomeAssistantConnector,
mongo_connector: MongoDBConnector) -> None:
        self.logger = logging.getLogger(class_name(self))
        self.config = config
        self.home_assistant = home_assistant
        self.mongo_connector = mongo_connector

    def run(self) -> None:
        result = dict()

        sensor_data = self.home_assistant.get_device_data(self.config.sensor)
        result['datetime'] = sensor_data['last_changed']
        result['lux_sensor'] = self.home_assistant.state_parser(sensor_data['state'])

        switch_state = self.home_assistant.get_device_state(self.config.switch)
        result['model_generated'] = switch_state

        for light in self.config.lights.values():
            light_name = self.get_light_name(light)
            # MOCK ADDED "if not mock_model(light) else mock_value()" added for mocked
models!!!!!!!!!!
            light_state = self.home_assistant.get_device_state(light) if not
mock_model(light) else mock_value()
            result[light_name] = light_state

        self.mongo_connector.push_new_record(result)

    def close(self) -> None:
        self.mongo_connector.close()
```

Przedstawiony kod (listing 10) definiuje klasę `DBBuilder`, która odpowiada za budowanie i zapisywanie rekordów danych w bazie MongoDB na podstawie aktualnych informacji pobieranych z systemu Home Assistant. Klasa ta pełni rolę komponentu integracyjnego, łączącego warstwę dostarczającą dane z warstwą trwałego przechowywania.

Konstruktor klasy inicjalizuje obiekt `DBBuilder` na podstawie trzech zależności przekazanych wstrzykiwaniem zależności. Pierwszym parametrem jest obiekt konfiguracyjny typu `DataBuilderConfig`, który przechowuje informacje dotyczące identyfikatorów czujników, przełączników oraz świateł. Drugim parametrem jest instancja `HomeAssistantConnector`, odpowiedzialna za komunikację z systemem Home Assistant i pobieranie bieżących stanów urządzeń. Trzecim parametrem jest instancja `MongoDBConnector`, która zapewnia dostęp do bazy danych MongoDB. Dodatkowo w konstruktorze inicjalizowany jest `logger`.

Metoda run stanowi główną logikę klasy i realizuje proces budowy pojedynczego rekordu danych. Na początku tworzony jest pusty słownik `result`, który będzie stopniowo uzupełniany o kolejne pola. Następnie z systemu Home Assistant pobierane są pełne dane dotyczące czujnika określonego w konfiguracji. Z otrzymanej odpowiedzi wyodrębniana jest informacja o czasie ostatniej zmiany stanu (`last_changed`), która zapisywana jest jako znacznik czasowy rekordu, oraz bieżący stan czujnika, który po przetworzeniu przez metodę `state_parser` zostaje zapisany jako wartość natężenia oświetlenia (`lux_sensor`).

W kolejnym kroku pobierany jest stan przełącznika zdefiniowanego w konfiguracji. Wartość ta jest zapisywana w polu `model_generated i` pełni rolę flagi logicznej, informującej czy dany rekord został wygenerowany w kontekście działania modelu (np. automatycznego sterowania), czy w trybie manualnym. Taka informacja może być istotna na etapie dalszej analizy lub trenowania modeli predykcyjnych.

Następnie wykonywana jest iteracja po zbiorze świateł zdefiniowanych w konfiguracji. Dla każdego światła wyznaczana jest jego nazwa logiczna przy użyciu metody `get_light_name`, co zapewnia spójne nazewnictwo pól w zapisywanym rekordzie. Stan każdego światła pobierany jest bezpośrednio z Home Assistanta, chyba że dla danego urządzenia aktywowany jest tryb modelu testowego (`mock`). W takim przypadku zamiast rzeczywistego stanu używana jest wartość generowana przez funkcję `mock_value`. Mechanizm ten umożliwia testowanie logiki aplikacji bez konieczności korzystania z rzeczywistych urządzeń lub aktywnego modelu.

Po zebraniu wszystkich danych, słownik `result` zawiera kompletny rekord, zgodnych z przyjętą strukturą przedstawioną na rysunku 8, opisujący stan środowiska w danym momencie. Rekord ten jest następnie zapisywany do bazy danych MongoDB za pomocą metody `push_new_record`, co kończy proces budowy i utrwalenia danych.

Metoda `close` odpowiada za poprawne zakończenie pracy klasy poprzez zamknięcie połączenia z bazą danych MongoDB. Zapewnia to właściwe zwolnienie zasobów i stabilność aplikacji, zwłaszcza w przypadku długotrwałego działania lub cyklicznego uruchamiania procesu budowania danych.

6.3. Użycie modelu

Listing 11. Klasa do budowania predykcji z wykorzystaniem aktualnego modelu

```
class PredictionBuilder(BuilderMixin):
    def __init__(self, config: DataBuilderConfig, preprocessor: FeaturesPreprocessing,
                 home_assistant: HomeAssistantConnector, mlflow_manager: MlflowManager) ->
None:
        self.logger = logging.getLogger(class_name(self))
        self.config = config
        self.datetime = datetime.now(timezone.utc)
        self.preprocessor = preprocessor
        self.home_assistant = home_assistant
        self.mlflow_manager = mlflow_manager
```

```
    def get_data(self, lux_sensor: float) -> pd.DataFrame:
        if lux_sensor == 'unavailable':
            raise DeviceException(f'No data from the light sensor')
        df = pd.DataFrame()
        df["datetime"] = [self.datetime]
        df["lux_sensor"] = [lux_sensor]
        return self.preprocessor(df)

    def run(self) -> None:
        try:
            lux_sensor = self.home_assistant.get_device_state(self.config.sensor)

            prediction_df = self.get_data(lux_sensor)

            for light in self.config.lights.values():
                model =
self.mlflow_manager.get_model(model_name=self.get_light_name(light))
                prediction = int(model.predict(prediction_df)[0])
                self.home_assistant.set_device_state(device_id=light, state=prediction)
        except DeviceException as error:
            self.logger.error(error)
        except Exception as error:
            self.logger.error(error)
```

Klasa `PredictionBuilder` przedstawiona w listingu 11 ma za zadanie realizację procesu predykcji i sterowania urządzeniami na podstawie aktualnych danych środowiskowych oraz modelu uczenia maszynowego zarejestrowanych w systemie MLflow. Klasa ta stanowi element wykonawczy potoku decyzyjnego, łącząc zbieranie danych, ich wstępne przetwarzanie, predykcję oraz wykonanie akcji w systemie automatyki domowej.

Konstruktor inicjalizuje obiekt `PredictionBuilder` z wykorzystaniem czterech zależności. Pierwszym z nich jest obiekt konfiguracyjny typu `DataBuilderConfig`, który zawiera informacje o identyfikatorach czujników oraz urządzeń sterowanych. Drugim elementem jest instancja klasy `FeaturesPreprocessing`, odpowiedzialna za przygotowanie danych wejściowych w formie zgodnej z oczekiwaniami modeli predykcyjnych. Kolejnym komponentem jest `HomeAssistantConnector`, umożliwiający komunikację z systemem Home Assistant, a ostatnim `MlflowManager`, który zapewnia dostęp do aktualnych wersji modeli zapisanych w rejestrze MLflow. Dodatkowo inicjalizowany jest `logger` oraz bieżący znacznik czasu w formacie UTC, który posłuży do wyznaczenia kompletu cech.

Metoda `get_data` odpowiada za przygotowanie danych wejściowych do predykcji. Przyjmuje ona jako argument wartość odczytu z czujnika natężenia światła (`lux_sensor`). W przypadku, gdy czujnik zwróci wartość `'unavailable'`, zgłaszany jest wyjątek `DeviceException`, sygnalizujący brak danych wejściowych. W przeciwnym przypadku tworzony jest obiekt `pandas.DataFrame`, zawierający pojedynczy rekord z aktualnym znacznikiem czasu oraz wartością natężenia światła. Utworzony zbiór danych jest następnie przekazywany do komponentu `preprocessor`, który wykonuje niezbędne operacje wstępnego przetwarzania, takie jak inżynieria cech czy kodowanie zmiennych czasowych. Wynikiem metody jest ramka danych gotowa do użycia przez model predykcyjny.

Metoda `run` implementuje główną logikę predykcyjną klasy i jest punktem wejścia całego procesu. W pierwszym kroku pobierany jest aktualny stan czujnika światła z systemu Home Assistant. Następnie dane te są przekształcane do postaci wejściowej dla modelu przy użyciu metody `get_data`. Dla każdego urządzenia typu światło zdefiniowanego w konfiguracji wykonywana jest sekwencja operacji polegająca na pobraniu odpowiedniego modelu z rejestru MLflow, wykonaniu predykcji oraz interpretacji wyniku jako wartości binarnej. Otrzymana predykcja jest następnie wykorzystywana do ustawienia stanu urządzenia w Home Assistant poprzez wywołanie metody `set_device_state`.

Cała logika predykcyjna została objęta blokiem `try-except`, co zapewnia odporność systemu na błędy. Wyjątki typu `DeviceException`, związane z brakiem danych z czujników, są obsługiwane oddzielnie i logowane, podobnie jak inne nieoczekiwane wyjątki. Takie podejście zwiększa stabilność systemu i umożliwia jego dalsze działanie nawet w przypadku chwilowych problemów z urządzeniami lub danymi.

6.4. Generowanie modelu

Listing 12. Klasa do budowania predykcji z wykorzystaniem aktualnego modelu

```
class ModelBuilder(BuilderMixin):
    def __init__(self, config: DataBuilderConfig, mongo_connector: MongoDBConnector,
preprocessor: FeaturesPreprocessing,
                 model_manager: ModelManager, mlflow_manager: MlflowManager) -> None:
        self.logger = logging.getLogger(class_name(self))
        self.config = config
        self.mongo_connector = mongo_connector
        self.preprocessor = preprocessor
        self.model_manager = model_manager
        self.mlflow_manager = mlflow_manager

    def run(self) -> None:
        try:
            db_data = self.mongo_connector.download_train_data()

            for light in self.config.lights.values():
                model_data_df = self.preprocessor(pd.DataFrame(db_data),
self.get_light_name(light))
                self.model_manager.prepare_data(df=model_data_df,
label_name=self.get_light_name(light))
                self.model_manager.fit(model_type=self.config.model)
                run_id = self.mlflow_manager.register_model(
                    model=self.model_manager.model,
                    model_name=self.get_light_name(light)
                )
                self.mlflow_manager.register_params(run_id=run_id,
accuracy=self.model_manager.accuracy)
        except Exception as error:
            self.logger.error(error)

    def close(self) -> None:
```

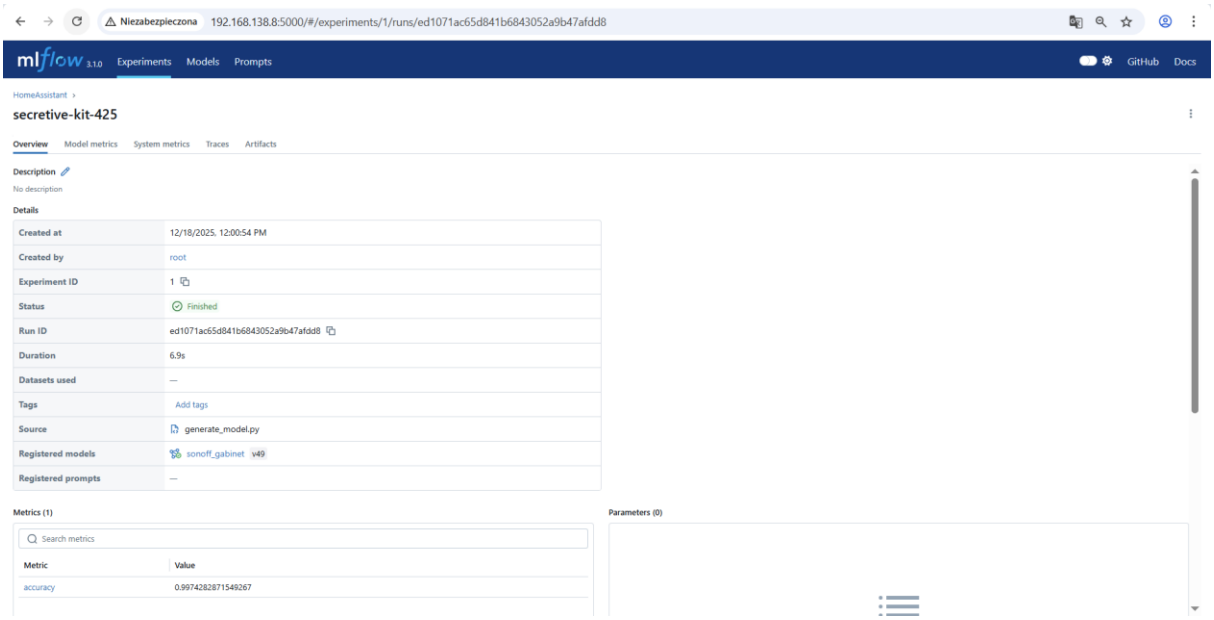
```
        self.mongo_connector.close()
```

Przedstawiony kod w listing 12 definiuje klasę `ModelBuilder`, której zadaniem jest realizacja procesu trenowania modeli uczenia maszynowego oraz ich rejestracji w systemie MLflow na podstawie danych historycznych zgromadzonych w bazie MongoDB. Klasa ta stanowi kluczowy element potoku treningowego, integrując warstwę danych, przetwarzanie cech, uczenie modeli oraz ich wersjonowanie.

Konstruktor klasy inicjalizuje obiekt `ModelBuilder` z wykorzystaniem pięciu zależności. Pierwszym z nich jest obiekt konfiguracyjny typu `DataBuilderConfig`, który zawiera informacje o urządzeniach (np. światłach) oraz typie modelu, jaki ma zostać użyty do uczenia. Drugim komponentem jest `MongoDBConnector`, zapewniający dostęp do danych treningowych zapisanych w bazie MongoDB. Kolejnym elementem jest `FeaturesPreprocessing`, odpowiedzialny za przygotowanie danych wejściowych w postaci odpowiednich cech modelowych. Następnie przekazywana jest instancja `ModelManager`, która zajmuje się przygotowaniem danych, trenowaniem modelu oraz obliczania metryk jakości. Ostatnim komponentem jest `MlflowManager`, służący do rejestrowania wytrenowanych modeli oraz powiązanych z nimi metryk. W konstruktorze inicjalizowany jest również `logger`.

Metoda run implementuje główną logikę budowy modeli i została objęta blokiem `try-except`, co zwiększa odporność systemu na błędy pojawiające się podczas przetwarzania danych lub uczenia modeli. W pierwszym kroku pobierane są dane treningowe z bazy MongoDB przy użyciu metody `download_train_data`. Dane te zawierają historyczne obserwacje środowiska, które stanowią podstawę do uczenia modeli predykcyjnych.

Następnie wykonywana jest iteracja po zbiorze świateł zdefiniowanych w konfiguracji. Dla każdego światła tworzony jest osobny zbiór danych treningowych poprzez zastosowanie komponentu `preprocessor`, który przekształca dane źródłowe do postaci odpowiedniej dla modelu i odpowiedniej etykiety docelowej. Metoda `prepare_data` klasy `ModelManager` odpowiada za rozdzielenie danych na cechy i zmienną docelową. Kolejnym krokiem jest wytrenowanie modelu przy użyciu metody `fit`, z wykorzystaniem typu modelu określonego w konfiguracji.



Rysunek 9. Szczegóły modelu zarejestrowanego w MLflow

Po zakończeniu procesu uczenia wytrenowany model jest rejestrowany w systemie MLflow za pomocą metody `register_model`. Model zapisywany jest pod nazwą odpowiadającą konkretnemu światłu, co umożliwia niezależne wersjonowanie i późniejsze wykorzystanie modeli dla

poszczególnych urządzeń. Metoda ta zwraca identyfikator uruchomienia (`run_id`), który jest następnie wykorzystywany do zapisania metryk jakości modelu. W tym przypadku rejestrowana jest metryka dokładności (`accuracy`) stanowiąca procentowy udział poprawnie ocenionych próbek przez model, przechowywana w systemie MLflow przy użyciu metody `register_params`. Parametry generowane w trakcie procesu można zweryfikować w graficznym interfejsie MLflow co zostało pokazane na rysunku 9.

Metoda `close` odpowiada za poprawne zakończenie pracy klasy poprzez zamknięcie połączenia z bazą danych MongoDB.

6.5. Orkiestracja procesów w Home Assistant

Zadania opisane w poprzednich podrozdziałach zostały zaprojektowane jako trójfazowy proces przetwarzania, obejmujący:

- gromadzenie aktualnych danych pomiarowych w bazie danych,
- okresowe generowanie nowych modeli uczenia maszynowego,
- wykonywanie predykcji oraz sterowanie oświetleniem na podstawie aktualnego modelu.

Każdy z etapów realizowany jest przez oddzielny skrypt w języku Python:

- `update_db.py`,
- `generate_model.py`,
- `use_model.py`,

co pozwala na jednoznaczne rozdzielenie odpowiedzialności oraz ułatwia zarządzanie i utrzymanie systemu.

Mechanizm wyzwalania poszczególnych etapów został skonfigurowany na poziomie platformy Home Assistant, która pełni rolę nadrzędnej warstwy sterującej. W zależności od zdarzeń oraz harmonogramów czasowych system uruchamia odpowiedni skrypt, wykorzystując zdefiniowane automatyzacje. Podstawowa konfiguracja realizowana jest poprzez interfejs użytkownika, natomiast bardziej zaawansowane integracje wymagają modyfikacji pliku `configuration.yaml` [21], a same automatyzacje przechowywane są w pliku `automations.yaml`. [22]

Listing 13. Plik configuration.yaml z konfiguracja Home Assistant projektu

```
# Loads default set of integrations. Do not remove.
default_config:

# Load frontend themes from the themes folder
frontend:
  themes: !include_dir_merge_named themes

automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml

# Credentials for eWeLink App
sonoff:
  username: "username"
  password: "password"
```

```
  mode: local

shell_command:
  run_use_model: "python3 /config/scripts/use_model.py"
  run_update_db: "python3 /config/scripts/update_db.py"
  run_generate_model: "python3 /config/scripts/generate_model_proxy.py"

input_boolean:
  run_script_loop_switch:
    name: "Automatyczne sterowanie światłem"
    icon: mdi:timer-play
```

Sekcja `default_config` w listingu 13 włącza domyślny zestaw integracji Home Assistanta (m.in. podstawowe komponenty interfejsu, obsługę urządzeń, rejestr, historię), co redukuje potrzebę ręcznego definiowania wielu elementów konfiguracji. Następnie `frontend: wraz z themes: !include_dir_merge_named themes` aktywuje obsługę motywów graficznych ładowanych z katalogu `themes` i scala je do jednej przestrzeni nazw, umożliwiając spójne zarządzanie wyglądem interfejsu użytkownika.

Dyrektywy `automation: !include automations.yaml`, `script: !include scripts.yaml` oraz `scene: !include scenes.yaml` przenoszą definicje automatyzacji, skryptów i scen do osobnych plików. Takie rozdzielenie zwiększa czytelność projektu, ułatwia kontrolę wersji i pozwala rozwijać logikę automatyzacji niezależnie od konfiguracji bazowej.

Konfiguracja integracji `sonoff` definiuje poświadczenia dla ekosystemu `eWeLink` oraz tryb `mode: local`. Oznacza to, że komunikacja z urządzeniami `Sonoff` ma odbywać się lokalnie (bez pośrednictwa chmury), co zwykle poprawia opóźnienia, niezawodność i kontrolę nad przepływem danych. W praktyce Home Assistant uzyskuje w ten sposób dostęp do encji reprezentującej przełączniki zarządzany przez Sonoff. Sekcja `username` i `password` celowo została zanonimizowana w niniejszym opracowaniu dla zachowania poufności.

Kluczowym elementem architektury są definicje `shell_command`, które rejestrują trzy komendy systemowe uruchamiane z poziomu usług Home Assistanta. Każda z nich wywołuje skrypt Pythona umieszczony w przestrzeni `/config/scripts/`, czyli w katalogu konfiguracyjnym kontenera Home Assistanta. W konsekwencji automatyzacje mogą wykonywać obliczenia i operacje zewnętrzne (np. inferencję modelu, zapis do bazy, trening) bez implementowania tej logiki bezpośrednio w YAML.

Sekcja `input_boolean` wprowadza przełącznik logiczny `input_boolean.run_script_loop_switch` opisany jako „Automatyczne sterowanie światłem”. Jest to kontrolka UI (i stan encji), która pełni funkcję globalnego warunku wykonania dla automatyzacji sterującej: użytkownik może włączać lub wyłączać automatyczne działanie modelu bez modyfikacji kodu automatyzacji.

Listing 14. Plik automations.yaml z automatyzacją Home Assistant projektu

```
- alias: "Automatic light control"
  description: "Automated lighting control powered by a predictive model"
  trigger:
    - platform: time_pattern
      minutes: "/1"
  condition:
    - condition: state
```

```
    entity_id: input_boolean.run_script_loop_switch
    state: 'on'
  action:
    - service: shell_command.run_use_model

- alias: "Add a record with data to the database after the light sensor updates."
  trigger:
    - platform: state
      entity_id: sensor.my_luxsensor_multisensor_illuminance_0
  action:
    - service: shell_command.run_update_db

- alias: "Generates the current model daily."
  trigger:
    - platform: time
      at: "12:00:00"
  action:
    - service: shell_command.run_generate_model
```

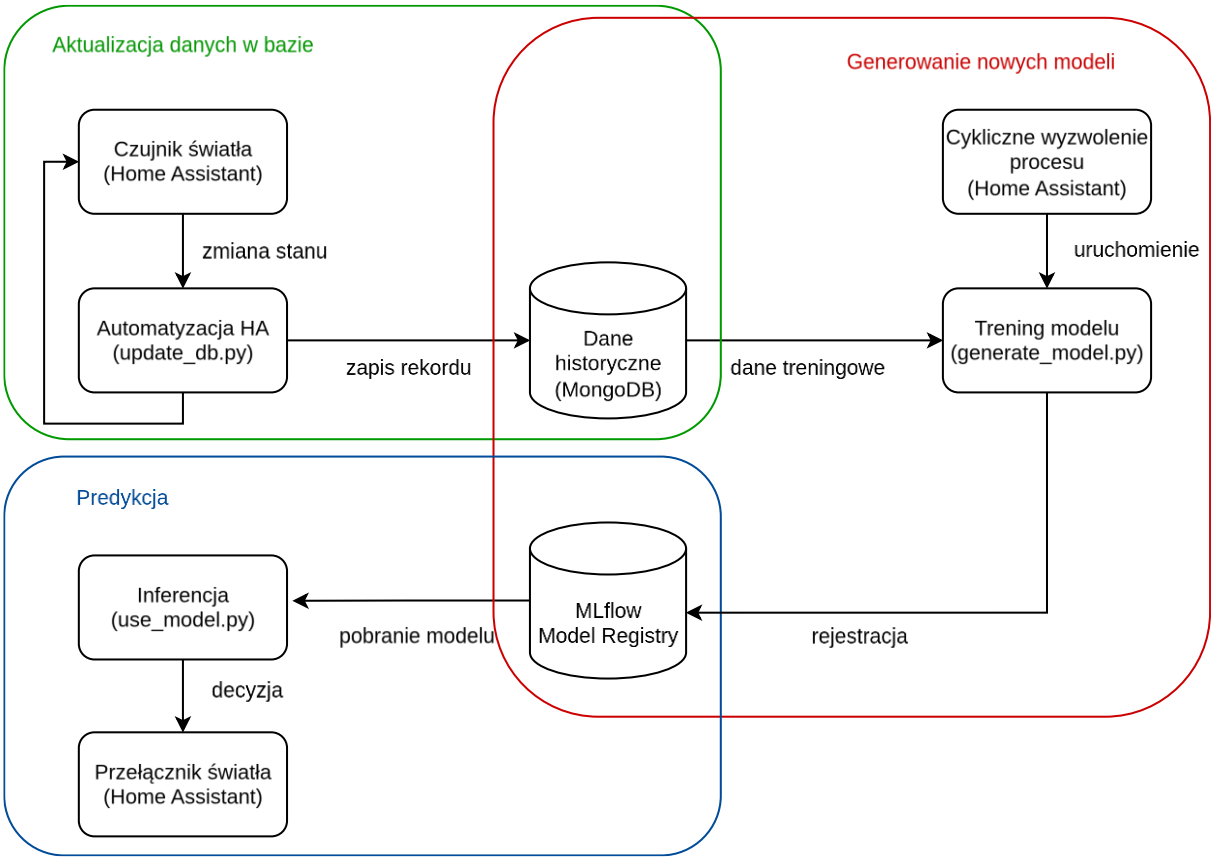
Pierwsza automatyzacja w listingu 14, czyli `Automatic light control` realizuje periodyczne sterowanie oświetleniem oparte na modelu predykcyjnym. Wyzwalacz `time_pattern` z `minutes: "/1"` uruchamia ją co minutę, natomiast warunek `condition: state` wymaga, aby encja `input_boolean.run_script_loop_switch` była w stanie `'on'`. Dopiero spełnienie tego warunku skutkuje wykonaniem akcji, tj. wywołaniem usługi `shell_command.run_use_model`, która uruchamia skrypt `use_model.py`. W ujęciu systemowym jest to mechanizm pętli decyzyjnej: Home Assistant cyklicznie inicjuje inferencję i wykonanie decyzji, ale pod kontrolą użytkownika (przełącznik typu `enable/disable`).

Druga automatyzacja `Add a record with data to the database after the light sensor updates` ma charakter zdarzeniowy i odpowiada za gromadzenie danych pomiarowych. Wyzwalacz `platform: state` reaguje na każdą zmianę stanu encji czujnika `sensor.my_luxsensor_multisensor_illuminance_0`. Po wystąpieniu zdarzenia wykonywana jest akcja `shell_command.run_update_db`, czyli uruchamiany jest skrypt `update_db.py`. Taki wzorzec zapewnia, że baza danych jest aktualizowana wtedy, gdy pojawia się nowa obserwacja a nie wyłącznie w sztywnych interwałach czasowych, co zwykle poprawia zgodność danych z rzeczywistą częstotliwością pomiaru.

Na tym etapie warto zaznaczyć, że pierwotnie aktualizacja bazy danych o nowe informacje z czujnika natężenia światła oraz pozostałych czujników odbywała się cyklicznie raz dziennie, na podstawie historii udostępnianej przez system Home Assistant. Jednak w wyniku długotrwałej obserwacji działania systemu, obejmującej okres kilku miesięcy, zaobserwowano, że rozwiązanie to bywa zawodne. Występowały sytuacje, w których następowała utrata historii danych urządzeń, mimo że w trakcie bieżącej obserwacji system funkcjonował poprawnie. Zjawisko to mogło być związane między innymi z automatycznymi aktualizacjami Home Assistant, które wpływały na sposób przechowywania lub dostępności danych historycznych. W odpowiedzi na te problemy wprowadzono opisane w poprzednim akapicie podejście, polegające na powiązaniu aktualizacji bazy danych bezpośrednio ze zmianą stanu czujnika natężenia światła. Rozwiązanie to okazało się znacznie bardziej niezawodne, zapewniając spójność danych i większą stabilność działania systemu.

Trzecia automatyzacja `Generates the current model daily` realizuje harmonogramowy proces aktualizacji modelu. Wyzwalacz `platform: time` z `at: "12:00:00"`

uruchamia zadanie codziennie o godzinie 12:00, a akcja `shell_command.run_generate_model` wywołuje skrypt `generate_model_proxy.py`. W praktyce odpowiada to cyklicznemu treningowi modelu na podstawie zebranych danych.



Rysunek 10. Graf przedstawiający przepływ danych

Na rysunku 10 przedstawiono schemat ilustrujący uproszczoną architekturę systemu oraz pełny przepływ danych w projekcie. Diagram obrazuje przepływ informacji od momentu ich pozyskania z czujnika natężenia światła, poprzez zapis w bazie danych i proces uczenia modelu, aż do etapu podejmowania decyzji i sterowania urządzeniem wykonawczym.

7. Technologia uczenia maszynowego

Niniejszy rozdział koncentruje się na kluczowym aspekcie projektu, jakim jest wykorzystanie technologii uczenia maszynowego. Początkowy etap prac obejmował dobór odpowiednich modeli oraz cech wejściowych, które mogłyby najlepiej sprawdzić się w istniejących warunkach eksperymentalnych.

W tym celu konieczne było przeanalizowanie, jakie informacje mają istotny wpływ na rzeczywiste wykorzystanie oświetlenia domowego, aby stworzyć możliwie najbardziej sprzyjające warunki do uczenia modelu. Uwzględniono między innymi dane związane z czasem oraz natężeniem światła zewnętrznego, które niewątpliwie odgrywają kluczową rolę w podejmowaniu decyzji dotyczących włączania i wyłączania oświetlenia.

Istotnym ograniczeniem była również relatywnie niewielka liczba dostępnych danych (około 2,9 tyś. rekordów dziennie, czyli nieco ponad 1 milion rocznie), co w znacznym stopniu uniemożliwiało zastosowanie modeli opartych na sieciach neuronowych, wymagających dużych zbiorów uczących. Z tego względu szczególny nacisk położono na modele lepiej przystosowane do pracy z mniejszą ilością danych.

Szczegółowa analiza oraz opis implementacji został przedstawiony w kolejnych podrozdziałach niniejszego rozdziału.

7.1. Implementacja uczenia maszynowego w projekcie

Na wczesnym etapie realizacji projektu przeprowadzono wstępną selekcję algorytmów uczenia maszynowego, które potencjalnie najlepiej odpowiadały charakterowi rozważanego problemu. Kryteriami doboru była w szczególności ograniczona liczba dostępnych danych treningowych oraz relatywnie niewielka liczba cech opisujących obserwacje. W rezultacie wybrano zestaw modeli o zróżnicowanej złożoności i charakterze decyzyjnym, które następnie zaimplementowano w projekcie i poddano dalszym testom porównawczym. Modele te zostały zebrane w postaci mapy, przedstawionej w listingu 15, co umożliwia ich jednolite i przejrzyste wykorzystanie w kolejnych etapach systemu.

Listing 15. Mapa modeli zawarta w projekcie

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier


MODEL = {
    "dtc": DecisionTreeClassifier(
        random_state=42
    ),
    "lr": LogisticRegression(
        random_state=42
    ),
    "rfc": RandomForestClassifier(
        random_state=42
    ),
    "gbc": GradientBoostingClassifier(
        random_state=42
    )}
```

Wszystkie wykorzystane modele pochodzą z biblioteki Scikit-learn [23], zastosowanej w projekcie w wersji 1.7.2. Przedstawiony kod pełni rolę centralnego rejestru modeli, umożliwiającego dynamiczny wybór konkretnego algorytmu na podstawie jego skróconej nazwy, bez potrzeby rozbudowywania logiki warunkowej w pozostałych częściach aplikacji.

Mapa zawiera szereg klasyfikatorów:

- `DecisionTreeClassifier` implementuje model drzewa decyzyjnego, cechujący się prostotą interpretacji i dobrą skutecznością przy niewielkich zbiorach danych,
- `LogisticRegression` reprezentuje liniowy model probabilistyczny, często stosowany jako punkt odniesienia w zadaniach klasyfikacyjnych,
- `RandomForestClassifier` oraz `GradientBoostingClassifier` należą do grupy metod zespołowych, które łączą wiele słabszych modeli w celu uzyskania lepszej jakości predykcji i większej odporności na nadmierne dopasowanie

Zdefiniowany słownik `MODEL` mapuje krótkie identyfikatory tekstowe (`"dtc"`, `"lr"`, `"rfc"`, `"gbc"`) na instancje odpowiadających im klasyfikatorów. Każdy model inicjalizowany jest z parametrem `random_state=42`, co zapewnia powtarzalność wyników eksperymentów i umożliwia rzetelne porównywanie jakości modeli w kolejnych etapach badań.

Listing 16. Implementacja klasy ModelManager

```
class ModelManager:
    def __init__(self) -> None:
        self.logger = logging.getLogger(class_name(self))
        self.model = None
        self.accuracy = None
        self.X = None
        self.y = None

    def prepare_data(self, df: pd.DataFrame, label_name: str) -> None:
        self.X = df.drop(columns=[label_name])
        self.y = df[label_name]

    def fit(self, model_type) -> None:
        X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, test_size=0.2,
random_state=42)
        model = Pipeline(steps=[
            ("scaler", StandardScaler()),
            ("clf", MODEL[model_type])
        ])
        model.fit(X_train, y_train)
        self.model = model
        y_pred = model.predict(X_test)
        self.accuracy = accuracy_score(y_test, y_pred)

    def predict(self, input_df) -> float:
        return self.model.predict(input_df)
```

Dla wprowadzenia porządku w zarządzaniu modelami w projekcie została zdefiniowana klasa `ModelManager` przedstawiona w listingu 16, która pełni rolę warstwy zarządzającej procesem trenowania, oceny oraz wykorzystania modeli uczenia maszynowego.

Konstruktor klasy inicjalizuje podstawowe atrybuty obiektu, w tym `logger` wykorzystywany do rejestrowania zdarzeń, referencję do aktualnie trenowanego modelu, wartość metryki jakości `accuracy` oraz zmienne przechowujące cechy `X` i etykiety `y`.

Metoda `prepare_data` odpowiada za przygotowanie zbioru danych do uczenia modelu. Na podstawie przekazanej ramki danych `DataFrame` oraz nazwy kolumny etykiety `label_name` metoda wydziela macierz cech `X` oraz wektor zmiennej docelowej `y`. Wyraźne oddzielenie tego etapu od samego procesu uczenia zwiększa modularność klasy i pozwala na łatwe modyfikowanie sposobu przygotowania danych bez ingerencji w logikę trenowania.

Metoda `fit` realizuje zasadniczy proces uczenia modelu. Na tym etapie dane wejściowe są dzielone na zbiór treningowy oraz testowy przy użyciu funkcji `train_test_split` [24] z biblioteki Scikit-learn. Podział ten odbywa się losowo, z zachowaniem określonych proporcji, które w zaproponowanym rozwiązaniu wynoszą 80% danych dla zbioru treningowego i 20% dla zbioru testowego (`test_size=0.2`). Zbiór treningowy wykorzystywany jest do dopasowania parametrów modelu, natomiast zbiór testowy służy do jego niezależnej ewaluacji i oceny zdolności generalizacji na danych niewidzianych wcześniej. Zastosowanie parametru `random_state` pozwala na kontrolę losowości podziału, a jego ustalenie na stałą wartość zapewnia powtarzalność wyników, co ma istotne znaczenie w kontekście eksperymentów porównawczych oraz analizy jakości modeli.

W dalszej części metody `fit` tworzony jest obiekt `Pipeline` [26], który łączy w jeden spójny ciąg operacji etap standaryzacji danych (`StandardScaler` [25]) oraz właściwy klasyfikator wybrany dynamicznie na podstawie parametru `model_type`. Takie podejście eliminuje ryzyko niespójności przetwarzania danych pomiędzy etapem treningu i predykcji oraz znacząco upraszcza zarządzanie całym procesem.

Zastosowanie `StandardScaler` w potoku przetwarzania danych ma istotne znaczenie z punktu widzenia jakości i stabilności uczenia modeli. `StandardScaler` dokonuje standaryzacji cech poprzez przekształcenie każdej cechy niezależnie poprzez odjęcie jej średniej oraz podzielenie przez odchylenie standardowe obliczone na zbiorze treningowym. W efekcie każda cecha zostaje przeskalowana do postaci o średniej równej zero oraz odchyleniu standardowym równym jeden. Dzięki temu wszystkie cechy mają porównywalną skalę, co jest szczególnie ważne w przypadku algorytmów wrażliwych na różnice w zakresie wartości, takich jak regresja logistyczna. Standaryzacja zapobiega dominacji cech o dużych wartościach liczbowych nad cechami o mniejszym zakresie, przyczyniając się do poprawy stabilności procesu uczenia. Umieszczenie `StandardScaler` bezpośrednio w obiekcie `Pipeline` gwarantuje, że te same parametry skalowania zostaną konsekwentnie zastosowane zarówno podczas treningu, jak i podczas późniejszej predykcji.

Wzór służący do obliczenia poszczególnych wartości przedstawia się jako:

$$z = \frac{x - u}{s}$$

Gdzie: „z” to wynik standaryzacji, „x” wartość próbki, „u” średnia próbek treningowych, a „s” to odchylenie standardowe.

Po wytrenowaniu modelu metoda `fit` zapisuje wyuczony obiekt w atrybucie `self.model`, a następnie dokonuje ewaluacji jakości predykcji na zbiorze testowym, obliczając miarę dokładności (ang. *accuracy*) [27]. Wartość ta jest przechowywana w atrybucie `self.accuracy`, co umożliwia jej dalsze wykorzystanie, na przykład w procesie porównywania modeli lub rejestrowania metryk w systemie MLflow.

Metoda `predict` udostępnia interfejs do wykonywania predykcji na nowych danych wejściowych. Dzięki wcześniejszemu zastosowaniu potoku przetwarzania danych zawierającego `StandardScaler` i klasyfikator, użytkownik klasy nie musi martwić się o ręczne skalowanie cech. Wszystkie niezbędne operacje są realizowane automatycznie w ramach wytrenowanego modelu.

W ramach porównywania modeli świadomie zastosowano losowy podział danych, przeznaczając 20% całego dostępnego zbioru na zbiór testowy, zamiast wykorzystywania np. danych pochodzących z ostatnich dwóch tygodni obserwacji. Takie podejście wynika z faktu, że losowy dobór próbek do zbioru testowego stanowi klasyczną i powszechnie stosowaną metodę w ocenie modeli uczenia maszynowego. Dodatkowo pozwala ono zminimalizować wpływ czynników czasowych, takich jak stopniowe zmiany warunków zewnętrznych (np. wynikające ze zmiany pory roku) Mogłyby one nieznacznie, lecz istotnie zniekształcić wyniki testów w przypadku wykorzystania wyłącznie najnowszych danych jako testowych.

Dokładność (ang. *accuracy*) jest jedną z podstawowych miar jakości klasyfikacji i określa odsetek poprawnych predykcji modelu w stosunku do wszystkich obserwacji. Innymi słowy, stanowi ona iloraz liczby poprawnie sklasyfikowanych próbek oraz całkowitej liczby próbek w zbiorze danych. Metryka ta umożliwia szybką, ogólną ocenę skuteczności modelu, jednak jej interpretacja jest najbardziej adekwatna w przypadku zbiorów danych o względnie zbalansowanym rozkładzie klas. W analizowanym przypadku warunek ten nie jest w pełni spełniony, ponieważ dane charakteryzują się niezbalansowaniem, to znaczy, że w ciągu doby oświetlenie pozostaje znacznie częściej wyłączone niż włączone. W konsekwencji wysoka wartość dokładności może nie odzwierciedlać rzeczywistej jakości modelu, gdyż model faworyzujący klasę dominującą może osiągać pozornie dobre wyniki. Z tego względu w dalszej części pracy do dokładnej analizy skuteczności modelu wykorzystano również inne metryki.

W ramach projektu, wraz z każdym zarejestrowanym modelem zapisywana jest metryka dokładności (ang. *accuracy*) obliczana na niezbalansowanym zbiorze danych. Metryka ta pełni jednak przede wszystkim funkcję walidacyjną, informując o poprawności procesu uczenia modelu, natomiast w mniejszym stopniu odzwierciedla jego rzeczywistą jakość predykcyjną.

7.2. Wybór cech

Baza danych MongoDB przechowuje kluczowe dane historyczne niezbędne do poprawnego działania systemu (rysunek 8). W tabeli 2 przedstawiono przykładowe wartości rejestrowane w bazie danych.

Tabela 2. Przykładowe rekordy z bazy zawierającej historyczne dane

_id	datetime	lux_sensor	model_generated	sonoff_gabinet	sonoff_salon	sonoff_sypialnia
6917641b485d1f3d5a6ecdb2	2025-11-11T11:59:24.458845+00:00	795.45	0	1	0	0
6917641b485d1f3d5a6ecdb3	2025-11-11T11:59:54.456479+00:00	795.43	1	0	1	1

Źródło: Opracowanie własne.

Baza nie zawiera jednak pełnego zestawu cech wykorzystywanych w procesie generowania nowych modeli oraz wykonywania predykcji. Wynika to z faktu, że znaczna część tych cech opiera się na aspektach czasowych i może być w prosty oraz szybki sposób wyliczana dynamicznie na podstawie pola `datetime`. W związku z tym przechowywanie kompletnego zbioru cech w bazie danych byłoby

działaniem nadmiarowym, prowadzącym do niepotrzebnego skomplikowania struktury dokumentów oraz zwiększenia objętości i obciążenia bazy danych.

Listing 17. Lista FEATURES_ORDER oraz słownik DAY_NAMES_MAP pomocnicze przy generowaniu dany na potrzeby modelu

```
FEATURES_ORDER = ['minute_of_day', 'week_number', 'holiday', 'Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday', 'lux_sensor']


DAY_NAMES_MAP = {
    0: 'Monday',
    1: 'Tuesday',
    2: 'Wednesday',
    3: 'Thursday',
    4: 'Friday',
    5: 'Saturday',
    6: 'Sunday'
}
```

Listing 18. Klasa FeaturesPreprocessing przygotowująca kompletne dane wejściowe przeznaczone do uczenia i predykcji modelu.

```
class FeaturesPreprocessing:
    def __init__(self):
        self.logger = logging.getLogger(class_name(self))

    def __call__(self, df: pd.DataFrame, label_name: str = None) -> pd.DataFrame:
        order = FEATURES_ORDER.copy()

        df["datetime"] = pd.to_datetime(df["datetime"], utc=True, format='mixed')

        df['minute_of_day'] = df['datetime'].dt.hour * 60 + df['datetime'].dt.minute
        df['week_day'] = df['datetime'].dt.weekday
        df['week_number'] = df['datetime'].dt.isocalendar().week

        years = df['datetime'].dt.year.unique()
        polish_holidays = holidays.Poland(years=years)

        def is_holiday(row):
            date = row['datetime'].date()
            if date in polish_holidays or row['week_day'] >= 5:
                return 1
            else:
                return 0

        df['holiday'] = df.apply(is_holiday, axis=1)

        df['week_day'] = df['week_day'].astype(int)
```

```
        for day in DAY_NAMES_MAP.values():
            df[day] = 0

        for idx, row in df.iterrows():
            day_name = DAY_NAMES_MAP[row['week_day']]
            if day_name in df.columns:
                df.at[idx, day_name] = 1

        if label_name and label_name in df.columns:
            order.append(label_name)

        return df[order]
```

Listingi 17 i 18 zawiera kod przedstawiający klasę `FeaturesPreprocessing` oraz listę `FEATURES_ORDER` i słownik `DAY_NAMES_MAP` pomocniczy. W całości one implementują wszystkie założenia dotyczące przetworzenia danych wyjściowych z bazy dostarczanych w formie `dataframe` na potrzeby modelu uczenia maszynowego, tak aby ostatecznie otrzymać obiekt `dataframe` z kompletnymi cechami w odpowiedniej kolejności. Klasa pełni rolę wywoływalnego komponentu przetwarzania danych.

Lista `FEATURES_ORDER` określa docelową kolejność cech w ramce danych wyjściowych. Jawne ustalenie kolejności kolumn ma istotne znaczenie dla spójności danych wejściowych, zwłaszcza w kontekście trenowania i późniejszej inferencji modeli, które oczekują niezmiennej struktury wejścia. Dodatkowo zdefiniowano słownik `DAY_NAMES_MAP`, mapujący numeryczne reprezentacje dni tygodnia na ich nazwy tekstowe, wykorzystywane później do kodowania zmiennych kategorycznych.

Klasa `FeaturesPreprocessing` została zaprojektowana jako obiekt wywoływalny dzięki implementacji metody `__call__`. Umożliwia to jej użycie w sposób funkcyjny, np. jako element potoku przetwarzania danych. Metoda przyjmuje ramkę danych `DataFrame` oraz opcjonalną nazwę etykiety `label_name`, co pozwala wykorzystywać ten sam mechanizm zarówno w procesie trenowania modeli, jak i podczas predykcji.

Pierwszym etapem przetwarzania jest konwersja kolumny `datetime` do typu `datetime` z jednoznacznym ustawieniem strefy czasowej UTC jednolitej dla całego projektu. Następnie z obiektu czasu wyodrębniane są cechy opisujące cykliczność i strukturę kalendarzową:

- `minute_of_day` - liczba minut od początku dnia,
- `week_day` - numer dnia tygodnia,
- `week_number` - oraz numer tygodnia w roku.

Cechy te pozwalają modelowi uchwycić regularności dobowe i tygodniowe, typowe dla zachowań użytkowników i warunków oświetleniowych.

Kolejnym etapem jest identyfikacja dni wolnych od pracy. Na podstawie lat występujących w danych tworzony jest zbiór polskich dni świątecznych z wykorzystaniem biblioteki Holidays w wersji 0.70. Następnie, dla każdej obserwacji, sprawdzane jest, czy dana data przypada na święto lub weekend. Wynik tej analizy zapisywany jest w binarnej cesze `holiday`, która przyjmuje wartość 1 dla dni wolnych oraz 0 dla dni roboczych. Taka cecha wprowadza do modelu kontekst społeczno-kalendarzowy, który często istotnie wpływa na schematy użytkowania oświetlenia.

Aby umożliwić algorytmom uczenia maszynowego efektywne wykorzystanie informacji o dniu tygodnia, zastosowano kodowanie One-Hot. Dla każdego dnia tygodnia tworzona jest osobna kolumna binarna `Monday–Sunday`) inicjalnie wypełniona zerami. Następnie, dla każdej obserwacji, ustawiana

jest wartość 1 w kolumnie odpowiadającej rzeczywistemu dniowi tygodnia. Takie podejście eliminuje sztuczny porządek liczbowy dni tygodnia (np. 1-7) i pozwala modelom poprawnie interpretować je jako zmienne kategoryczne.

Na końcu metody, jeżeli przekazano nazwę etykiety i występuje ona w danych, jest ona dołączana do listy order. Dzięki temu ta sama funkcja może zwracać zarówno zbiór cech wejściowych, jak i pełny zbiór treningowy zawierający zmienną docelową. Ostatecznie metoda zwraca ramkę danych zawierającą wyłącznie wybrane kolumny, uporządkowane zgodnie z wcześniej zdefiniowaną listą `FEATURES_ORDER`.

Ostatnia cecha, czyli `lux_sensor` reprezentująca natężenie światła zewnętrznego przekazywana jest w niezmienionej formie na wyjście wraz z pozostałymi cechami.

Tabela 3. Struktura danych po przetworzeniu

Nazwa	**Przedział wartości**
minute_of_day	[1, 1440]
week_number	[1, 52]
holiday	[0, 1]
Monday	[0, 1]
Tuesday	[0, 1]
Wednesday	[0, 1]
Thursday	[0, 1]
Friday	[0, 1]
Saturday	[0, 1]
Sunday	[0, 1]
lux_sensor	[0.0, 5000.0]

Źródło: Opracowanie własne.

W tabeli 3 przedstawiono kompletny zestaw cech wejściowych wykorzystywanych w projekcie, wraz z zakresami wartości, jakie mogą one przyjmować.

Tabela 4. Etykiety przewidziane w projekcie

Nazwa	**Przedział wartości**
sonoff_gabinet	[0, 1]
sonoff_salon	[0, 1]
sonoff_sypialnia	[0, 1]

Źródło: Opracowanie własne.

W tabeli 4 przedstawiono zestaw dostępnych etykiet wyjściowych, stanowiących wartości przewidywane przez wygenerowane modele uczenia maszynowego. Etykiety te odpowiadają

bezpośrednio stanom poszczególnych przełączników oświetlenia i są wykorzystywane w procesie uczenia oraz predykcji.

Należy przypomnieć, że rzeczywiste dane pomiarowe gromadzone w bazie danych dotyczą wyłącznie etykiety `sonoff_gabinet`, odpowiadającej fizycznie istniejącemu przełącznikowi światła. Pozostałe etykiety, tj. `sonoff_salon` oraz `sonoff_sypialnia`, zawierają losowo generowane wartości binarne. Zostały one wprowadzone wyłącznie w celu demonstracji możliwości skalowania i obsługi wielu źródeł światła w ramach proponowanego rozwiązania.

Takie podejście pozwala na zaprezentowanie uniwersalności systemu oraz jego potencjału rozwojowego, przy jednoczesnym zachowaniu prostoty infrastruktury sprzętowej wykorzystanej w projekcie.

Należy podkreślić, że dobór cech na potrzeby projektu został dokonany w sposób ekspercki, w oparciu o analizę charakterystyki problemu oraz obserwacje rzeczywistego wykorzystania oświetlenia domowego. W procesie tym nie opierano się bezpośrednio na zewnętrznych źródłach informacji ani gotowych zestawach cech, lecz na autorskiej koncepcji, dostosowanej do specyfiki realizowanego rozwiązania.

7.3. Porównanie modeli

Podstawową metodą porównania jakości różnych typów modeli uczenia maszynowego jest ich trenowanie oraz testowanie na identycznych danych wejściowych. Takie podejście pozwala na obiektywną ocenę skuteczności poszczególnych algorytmów, niezależnie od czynników zewnętrznych. Aby wyeliminować wpływ wcześniej wspomnianych zmiennych czasowych, które mogą pojawiać się w danych w trakcie ich gromadzenia, zbiór testowy został utworzony poprzez losowy wybór 20% wszystkich dostępnych próbek. Co istotne, ten sam zbiór testowy został wykorzystany dla wszystkich porównywanych modeli.

Na podstawie wytrenowanych modeli oraz wspólnego zbioru testowego wyznaczono wcześniej omówioną metrykę dokładności, która posłużyła jako podstawowy wskaźnik porównawczy. Pozwoliło to na ocenę względnej skuteczności poszczególnych algorytmów w identycznych warunkach eksperymentalnych.

Jednocześnie należy ponownie podkreślić, że analizowane dane cechowały się znacznym niezbalansowaniem klas, co mogło prowadzić do zawyżonych wyników metryki dokładności i nie oddawać w pełni rzeczywistej jakości modeli. W związku z tym zaistniała potrzeba rozszerzenia badań o dodatkowy etap, obejmujący uczenie modeli na zbiorach zbalansowanych.

W tym celu wykorzystano klasę `RandomUnderSampler` [28] z biblioteki `Imbalanced-learn` w wersji 0.14.1, która umożliwia zrównoważenie liczności klas poprzez losowe próbkowanie klasy dominującej. Modele wytrenowane na tak przygotowanych danych zostały następnie ponownie przetestowane na tym samym, niezmienionym zbiorze testowym, co pozwoliło na bezpośrednie porównanie wyników z modelami uczonymi na danych niezbalansowanych. Dla każdego przypadku ponownie obliczono wartość metryki dokładności.

Dane wykorzystane do przeprowadzenia badań porównawczych modeli zostały zgromadzone w okresie od **16 kwietnia 2025 roku, godz. 21:46 (czas UTC)** do **11 listopada 2025 roku, godz. 11:59 (czas UTC)**. Obejmują one wielomiesięczny przedział czasowy, co pozwoliło na uwzględnienie zróżnicowanych warunków użytkowych oraz zmiennych środowiskowych.

W analizowanym okresie zarejestrowano łącznie **520 269 rekordów** danych. Należy jednak zaznaczyć, że część z nich pochodziła z okresów, w których stan oświetlenia był generowany automatycznie przez model predykcyjny (pole `model_generated` o wartości 1). Rekordy te nie

odzwierciedlały rzeczywistych decyzji użytkowników, dlatego zostały wyłączone z procesu uczenia i testowania modeli.

Po odfiltrowaniu danych generowanych przez model, do dalszych analiz wykorzystano **504 339 rekordów**, które stanowiły zbiór reprezentujący rzeczywiste zachowania mieszkańców. Z tej puli wyodrębniono zbiór testowy, liczący **100 868 rekordów**, przeznaczony do oceny skuteczności wytrenowanych modeli. Pozostała część danych (**403 471 rekordów**) została wykorzystana jako zbiór uczący.

Rezultaty przeprowadzonych eksperymentów, obejmujące porównanie modeli trenowanych na danych niezbalansowanych oraz zbalansowanych, zostały zestawione i przedstawione w tabeli 5.

Tabela 5. Tabela porównawcza zawierająca wyniki porównania dokładności (accuracy) dla poszczególnych modeli z i bez Random Under Samplingu (RUS)

Nazwa modelu	**ACC bez RUS**	**ACC z RUS**
Drzewo Decyzyjne (ang. *Decision Tree Classifier*)	0.9978	0.9860
Regresja Logistyczna (ang. *Logistic Regression*)	0.9202	0.7714
Las Losowy (ang. *Random Forest Classifier*)	**0.9985**	**0.9919**
Wzmacniacz Gradientowy (ang. *Gradient Boosting Classifier*)	0.9672	0.9203

Źródło: Opracowanie własne.

Analizując wyniki przedstawione w tabeli 5, w pierwszej kolejności można zauważyć, że uzyskane rezultaty charakteryzują się bardzo wysoką skutecznością. Spośród wszystkich porównywanych modeli jedynie regresja logistyczna trenowana na zbiorze zbalansowanym osiągnęła wartość dokładności niższą niż 0,9.

Najlepsze wyniki uzyskały modele drzewa decyzyjnego oraz lasu losowego, przy czym las losowy okazał się nieznacznie skuteczniejszy. Warto również zwrócić uwagę, że dla obu tych modeli różnice w jakości predykcji pomiędzy trenowaniem na zbiorze niezbalansowanym a zbalansowanym (z wykorzystaniem `RandomUnderSampler`) były niewielkie. Świadczy to o ich wysokiej odporności na problem niezbalansowania klas.

Podsumowując wyniki badań, najlepszym modelem okazał się las losowy (ang. *Random Forest Classifier*). Osiągnął on dokładność na poziomie **0,9985** w przypadku trenowania na pełnym, niezbalansowanym zbiorze danych oraz **0,9919** przy trenowaniu na zbiorze zbalansowanym.

Uwzględniając czas gromadzenia danych oraz fakt, że próbkowanie odbywało się średnio co 30 sekund, można stwierdzić, że model uczony na danych odpowiadających około **140 dniom** obserwacji poprawnie sklasyfikował próbki równe okresowi około **35 dni**. Natomiast błędy predykcji dotyczyły czasu porównywalnego do **1 godziny i 15 minut**. Taki wynik należy uznać za bardzo dobry, zwłaszcza w kontekście symulacji rzeczywistych zachowań użytkowników.

Należy jednak podkreślić, że w przeprowadzonym badaniu zbiór testowy był losowo wybierany z całego dostępnego zbioru danych. Oznacza to, że próbki testowe mogły pochodzić z bezpośredniego sąsiedztwa próbek treningowych o tej samej klasie, co przy założeniu ciągłości stanu oświetlenia przez dłuższy czas ułatwiało proces predykcji. Aspekt ten powinien być brany pod uwagę przy interpretacji wyników i stanowi istotny punkt odniesienia dla dalszych badań.

7.4. Opis teoretyczny wybranej technologii uczenia maszynowego

Na podstawie przeprowadzonych badań, klasyfikator lasu losowego (ang. *Random Forest Classifier*) został uznany za najbardziej odpowiedni model do wykorzystania w realizacji niniejszego projektu.

Las losowy [29] jest algorytmem uczenia maszynowego należącym do grupy metod uczenia zespołowego (ang. *ensemble learning*). Jego główną ideą jest budowa wielu niezależnych modeli bazowych w postaci drzew decyzyjnych, a następnie agregacja ich wyników w celu uzyskania stabilniejszej i dokładniejszej predykcji. W praktyce algorytm ten łączy zalety drzew decyzyjnych z mechanizmami losowości, które ograniczają przeuczenie i poprawiają zdolność generalizacji modelu. [30]

Podstawowym elementem składowym lasu losowego jest drzewo decyzyjne. Drzewo decyzyjne jest strukturą hierarchiczną, w której dane są dzielone na kolejne podzbiory na podstawie wartości cech wejściowych. Każdy węzeł wewnętrzny odpowiada testowi logicznemu (np. czy wartość danej cechy jest mniejsza od ustalonego progu), natomiast liście drzewa zawierają ostateczną decyzję klasyfikacyjną. W przypadku klasyfikacji podziały w drzewie są wybierane w taki sposób, aby maksymalnie zmniejszyć nieczystość klas w węzłach potomnych.

Jedną z najczęściej stosowanych miar nieczystości w drzewach decyzyjnych (domyślna w RandomForestClassifier) jest indeks Giniego, zdefiniowany wzorem: [31]

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

gdzie „p_{mk}" oznacza udział obserwacji należących do klasy „k" w danym węźle. Im mniejsza wartość indeksu Giniego, tym bardziej jednorodny jest węzeł.

Las losowy opiera się na zbiorze wielu drzew budowanych w sposób losowy. Dla każdego drzewa tworzona jest osobna próba ucząca poprzez losowanie ze zwracaniem (ang. *bootstrap sampling*) z oryginalnego zbioru danych. Oznacza to, że niektóre obserwacje mogą pojawić się w danej próbie wielokrotnie, a inne mogą zostać pominięte. Mechanizm ten prowadzi do powstania zróżnicowanych zbiorów uczących dla poszczególnych drzew, co zwiększa różnorodność modeli w lesie.

Dodatkowym źródłem losowości jest losowy wybór cech podczas budowy każdego węzła drzewa. Zamiast rozważać wszystkie dostępne cechy, algorytm analizuje jedynie ich losowy podzbiór. Spośród tych cech wybierana jest ta, która zapewnia najlepszy podział według przyjętego kryterium (np. indeksu Giniego). Dzięki temu poszczególne drzewa różnią się nie tylko zestawami danych uczących, ale również strukturą, co znacząco redukuje korelację pomiędzy nimi.

Po zakończeniu procesu uczenia każde drzewo w lesie jest w stanie samodzielnie dokonać predykcji (`predict`). W przypadku klasyfikacji, końcowa decyzja lasu losowego wyznaczana jest poprzez głosowanie większościowe.

Las losowy umożliwia również estymację prawdopodobieństw przynależności do klas (`predict_proba`). W tym przypadku prawdopodobieństwo danej klasy jest obliczane jako średnia arytmetyczna prawdopodobieństw zwróconych przez wszystkie drzewa w lesie. Prawdopodobieństwo pojedynczego drzewa wynika z proporcji obserwacji poszczególnych klas w liściu, do którego trafia dana obserwacja.

8. Analiza skuteczności

Niezwykle istotnym aspektem tworzenia innowacyjnych rozwiązań jest rzetelna weryfikacja ich skuteczności. Nie ma bowiem uzasadnienia dla wdrażania, a tym bardziej komercjalizacji systemów, które w praktyce nie działają lub których działanie nie spełnia założonych oczekiwań jakościowych. Problem ten w szczególności dotyczy rozwiązań opartych na technologiach sztucznej inteligencji, które z natury charakteryzują się pewnym stopniem nieprzewidywalności. Już na etapie projektowania systemu często trudno jednoznacznie określić, czy zastosowane podejście okaże się wystarczająco skuteczne w warunkach rzeczywistych.

Choć wiedza ekspercka oraz doświadczenie projektowe znacząco ułatwiają wybór właściwego kierunku rozwoju i zwiększają prawdopodobieństwo powodzenia, nie gwarantują one pełnej pewności sukcesu. Z tego względu kluczowe jest, aby na możliwie najwcześniejszym etapie prac przeprowadzić analizę skuteczności projektowanego rozwiązania. Najlepiej w momencie, gdy zaimplementowane zostały podstawowe funkcjonalności oraz gdy dostępne są dane umożliwiające ocenę działania systemu. Przynajmniej w warunkach laboratoryjnych.

Jednocześnie samo posiadanie działającego prototypu oraz zebranie danych testowych nie są wystarczające do stwierdzenia, że system po komercjalizacji okaże się produktem konkurencyjnym i atrakcyjnym rynkowo. Aby tak się stało, proponowane rozwiązanie powinno być jakościowo porównywalne z istniejącymi systemami, a w to miejsce oferować lepszą cenę. Inną możliwością jest przewyższanie pod względem skuteczności działania, zwłaszcza w sytuacji, gdy rynek jest nasycony darmowymi lub łatwo dostępnymi alternatywami.

W związku z powyższym ocena jakości opracowanego systemu powinna zostać przeprowadzona poprzez jego porównanie z istniejącymi rozwiązaniami lub powszechnie stosowanymi metodami działania. Analiza ta stanowi punkt wyjścia do dalszych rozważań, które zostały przedstawione w kolejnym podrozdziale.

8.1. Szczegółowy opis alternatywnych rozwiązań

Na rynku dostępnych jest wiele narzędzi typu asystent domowy, które zarówno bezpośrednio, jak i za pośrednictwem dedykowanych wtyczek oferują możliwość automatyzacji sterowania oświetleniem podczas nieobecności mieszkańców. Pomimo dużej różnorodności dostępnych rozwiązań, na podstawie często pojawiających się opinii użytkowników można wyróżnić kilka podstawowych mechanizmów, które są wykorzystywane w większości tego typu systemów.

Do najczęściej stosowanych mechanizmów należą:

- mechanizmy oparte na czasie,
- mechanizmy oparte na pomiarze natężenia światła,
- mechanizmy losowe.

W praktyce parametry tych mechanizmów są zazwyczaj konfigurowane ręcznie przez użytkowników. Ze względu na wygodę oraz ograniczoną dostępność długoterminowych danych, użytkownicy rzadko przeprowadzają szczegółowe, wielomiesięczne analizy własnych zachowań. Zamiast tego ustawienia systemów są często określane intuicyjnie, na podstawie aktualnych preferencji i założeń przyjętych w momencie konfiguracji.

Dodatkowo, w większości przypadków konfiguracja opiera się na pojedynczych warunkach decyzyjnych, takich jak konkretna godzina lub określony próg natężenia światła. Takie podejście, choć proste w implementacji, może prowadzić do schematycznych i łatwych do przewidzenia zachowań systemu, co ogranicza jego skuteczność w realistycznej symulacji obecności domowników.

W kolejnych podrozdziałach przedstawione zostaną przykładowe mechanizmy automatyzacji, które stanowić będą punkt odniesienia dla dalszej analizy i porównań z rozwiązaniem zaproponowanym w ramach niniejszej pracy.

8.1.1 Mechanizm oparty na czasie

Mechanizmy oparte na czasie zakładają włączanie i wyłączanie oświetlenia w danym pomieszczeniu lub jednocześnie we wszystkich pomieszczeniach o ściśle określonych godzinach. Najczęściej wybierane są godziny wieczorne, mające na celu symulację obecności domowników po zmroku.

Należy jednak zauważyć, że pora zapadania zmierzchu różni się znacząco w zależności od pory roku, w okresie zimowym zmrok następuje już około 16:00 UTC, natomiast latem dopiero około 20:00 UTC. W celu uproszczenia konfiguracji tego typu mechanizmów często przyjmuje się jedną, stałą godzinę obowiązującą przez cały rok, niezależnie od zmiennych warunków oświetleniowych.

Czas działania oświetlenia ustalany jest zazwyczaj arbitralnie i w analizowanym przypadku przyjęto jego długość na poziomie czterech godzin. W związku z powyższym rozpatrywana heurystyka opiera się na następujących założeniach:

- włączenie światła o 19:00 UTC (20:00 czas zimowy, 21:00 czas letni),
- wyłączenie światła o 23:00 UTC (24:00 czas zimowy, 1:00 czas letni).

8.1.2 Mechanizm oparty na pomiarze natężenia światła

Często istotnym czynnikiem determinującym moment włączenia oświetlenia jest faktyczny zmierzch, określany na podstawie danych astronomicznych lub prognoz pogodowych dla danego dnia. Takie podejście nie zawsze jednak uwzględnia rzeczywiste warunki atmosferyczne, takie jak zachmurzenie, opady deszczu czy mgła, które mogą znacząco wpływać na poziom oświetlenia zewnętrznego.

Bardziej precyzyjnym rozwiązaniem jest wykorzystanie bezpośredniego pomiaru natężenia światła zewnętrznego, pod warunkiem, że system został wyposażony w odpowiedni czujnik. W praktyce oświetlenie najczęściej włączane jest w momencie, gdy na zewnątrz zapada rzeczywista ciemność, niezależnie od teoretycznej godziny zmierzchu. Z kolei czas wyłączenia światła bywa ograniczany godzinowo, aby uniknąć jego niepotrzebnego działania przez całą noc.

Na potrzeby przeprowadzonego badania przyjęto następujące założenia heurystyczne:

- włączenie światła w momencie, gdy natężenie światła zewnętrznego spadnie poniżej 1 lux,
- nie wcześniej niż 12:00 UTC (13:00 czas zimowy, 14:00 czas letni),
- wyłączenie światła o 23:00 UTC (24:00 czas zimowy, 1:00 czas letni).

8.1.3 Mechanizm losowy

Rzadziej spotykanym, choć możliwym wariantem automatyzacji jest losowe włączanie oświetlenia w pomieszczeniach, zarówno jednocześnie we wszystkich, jak i niezależnie dla każdego z nich. Takie podejście ma na celu zwiększenie nieprzewidywalności działania systemu i tym samym lepszą symulację obecności domowników.

Również w tym przypadku konieczne jest jednak wprowadzenie ograniczeń czasowych, aby zapobiec włączaniu oświetlenia w nieadekwatnych porach, takich jak np. godziny południowe w okresie letnim. W związku z tym na potrzeby analizy przyjęto następujące założenia:

- światło zapalane jest losowo,
- nie wcześniej niż o 16:00 UTC (17:00 czas zimowy, 18:00 czas letni),
- nie później niż o 23:00 UTC (24:00 czas zimowy, 1:00 czas letni).

Zastosowanie takiego rozwiązania wiąże się jednak z pewnymi wątpliwościami natury praktycznej. Częste włączanie i wyłączanie oświetlenia mogą negatywnie wpływać na żywotność tradycyjnych źródeł światła, w szczególności żarówek żarowych lub halogenowych, które są bardziej podatne na zużycie w wyniku częstych cykli pracy. [32]

Należy jednak zauważyć, że we współczesnych instalacjach domowych coraz powszechniej stosowane są źródła światła typu LED, które charakteryzują się znacznie większą trwałością oraz odpornością na częste przełączanie. W praktyce oznacza to, że omawiana wada ma obecnie ograniczone znaczenie, choć nadal powinna być brana pod uwagę przy projektowaniu systemów automatyki domowej, zwłaszcza w przypadku starszych instalacji oświetleniowych.

8.2. Wybór metodyk oceniających skuteczność

Podczas porównywania różnych modeli uczenia maszynowego zastosowano miarę dokładności obliczaną na zbiorze testowym losowo wyselekcjonowanym z całego dostępnego zbioru danych. Takie podejście można uznać za właściwe w kontekście porównawczej oceny jakości modeli, ponieważ eliminuje wpływ losowych czynników i umożliwia ocenę algorytmów w identycznych warunkach eksperymentalnych.

Jednocześnie należy zauważyć, że metoda ta nie odzwierciedla rzeczywistego scenariusza użytkowania, w którym system inferencji uruchamiany jest na dłuższy, nieprzerwany okres (np. na kilka lub kilkanaście dni podczas nieobecności mieszkańców). W takim przypadku kluczowe znaczenie ma zdolność modelu do poprawnego przewidywania ciągłych sekwencji decyzji w czasie, a nie jedynie pojedynczych, losowo rozproszonych próbek.

Aby możliwie wiernie odtworzyć sytuację praktyczną, w której użytkownik uruchamia system na czas wyjazdu, a sterowanie oświetleniem odbywa się wyłącznie na podstawie najświeższego dostępnego modelu, przyjęto alternatywne podejście do podziału danych. Zgodnie z nim zbiór testowy stanowią dane z ostatnich dwóch pełnych tygodni, natomiast zbiór treningowy obejmuje całość danych zgromadzonych przed tym okresem. Taki sposób podziału danych został przedstawiony w tabeli 6.

Tabela 6. Tabela przedstawiająca granice zbiorów

Typ	**Początek**	**Koniec**	**Ilość próbek**
Treningowy	2025-04-16T21:46:00.000000+00:00	2025-10-27T23:59:15.420915+00:00	466,416
Testowy	2025-10-28T00:00:15.422163+00:00	2025-11-10T23:59:50.965646+00:00	36,586

Źródło: Opracowanie własne.

Zastosowane podejście umożliwia bardziej realistyczną ocenę skuteczności modelu w warunkach zbliżonych do rzeczywistego użytkowania systemu, a tym samym pozwala na pełniejszą analizę jego przydatności praktycznej.

Po wydzieleniu zbioru treningowego, na którym model był uczony, oraz zbioru testowego, służącego do weryfikacji skuteczności modelu i porównania go z alternatywnymi rozwiązaniami, kolejnym etapem jest dobór odpowiednich metryk oceny.

W prezentowanej analizie, w dalszym ciągu uwzględniona zostanie dokładność (ang. *accuracy*), jednak zostanie ona uzupełniona o inne formy prezentacji wyników, w szczególności macierz pomyłek (ang. *confusion matrix*) [33], która pozwala na bardziej szczegółową ocenę charakteru popełnianych błędów. Dodatkowo w analizach wykorzystana zostanie również metryka F1-score [34], będąca kompromisem pomiędzy precyzją a czułością modelu, co jest szczególnie istotne w przypadku niezbalansowanych zbiorów danych.

Macierz pomyłek (ang. *confusion matrix*) pokazuje porównanie rzeczywistych etykiet z przewidywanymi przez model, dzięki temu można zobaczyć, ile przypadków model sklasyfikował poprawnie, a ile błędnie. W przypadku klasyfikacji binarnej macierz ma wymiar 2×2.

F1-score jest średnią harmoniczną precyzji i czułości i ma wartość od 0 do 1 (1 to najlepszy wynik). W przeciwieństwie do samej dokładności, F1 bierze pod uwagę zarówno fałszywe pozytywy, jak i fałszywe negatywy. Wzór na wyznaczenie F1:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN}$$

Gdzie „TP" oznacza ilość prawdziwie pozytywnych próbek, „FP" oznacza fałszywie pozytywne próbki, natomiast „FN" to ilość fałszywie negatywnych próbek.

W procesie oceny skuteczności systemu konieczne jest także uwzględnienie sumarycznego czasu świecenia oświetlenia, który ma bezpośredni wpływ na zużycie energii elektrycznej. Celem jest uniknięcie sytuacji, w której automatyczny system sterowania generowałby większe zużycie energii niż typowe zachowanie użytkowników, co mogłoby prowadzić do nieoczekiwanego wzrostu kosztów eksploatacyjnych. Uwzględnienie tego aspektu pozwala na ocenę rozwiązania nie tylko pod kątem skuteczności predykcyjnej, lecz również jego praktycznej opłacalności.

8.3. Standardowe miary skuteczności

Do standardowych miar skuteczności których wyniki zostaną przedstawione należy dokładność (ang. *accuracy*), macierz pomyłek (ang. *confusion matrix*) oraz ocena F1 (ang. *F1-score*)

8.3.1 Dokładność

Tabela 7. Wyniki dokładności dla różnych scenariuszy

Scenariusz	**Dokładność**
Mechanizm oparty na czasie	0.7915
Mechanizm oparty na pomiarze natężenia światła	0.7177
Mechanizm losowy	0.7948
Predykcja modelu	**0.8137**
Predykcja zbalansowanego modelu	0.8134

Źródło: Opracowanie własne.

Jak przedstawiono w tabeli 7, wartości dokładności uzyskane dla modelu predykcyjnego, zarówno uczonego na pełnym jak i zbalansowanym zbiorze danych, istotnie odbiegają od wyników uzyskanych przy losowym podziale danych. Tam dokładność osiągała poziom około 0,99. Taki spadek skuteczności był jednak wynikiem oczekiwanym, ze względu na znacznie bardziej wymagające warunki testowe, polegające na ocenie modelu na ciągłym, wielodniowym zbiorze danych, odzwierciedlającym rzeczywisty scenariusz użytkowy.

Pomimo tych trudniejszych warunków testowych można zauważyć, że modele oparte na uczeniu maszynowym nadal osiągały lepsze wyniki dokładności niż mechanizmy bazujące na sztywnych

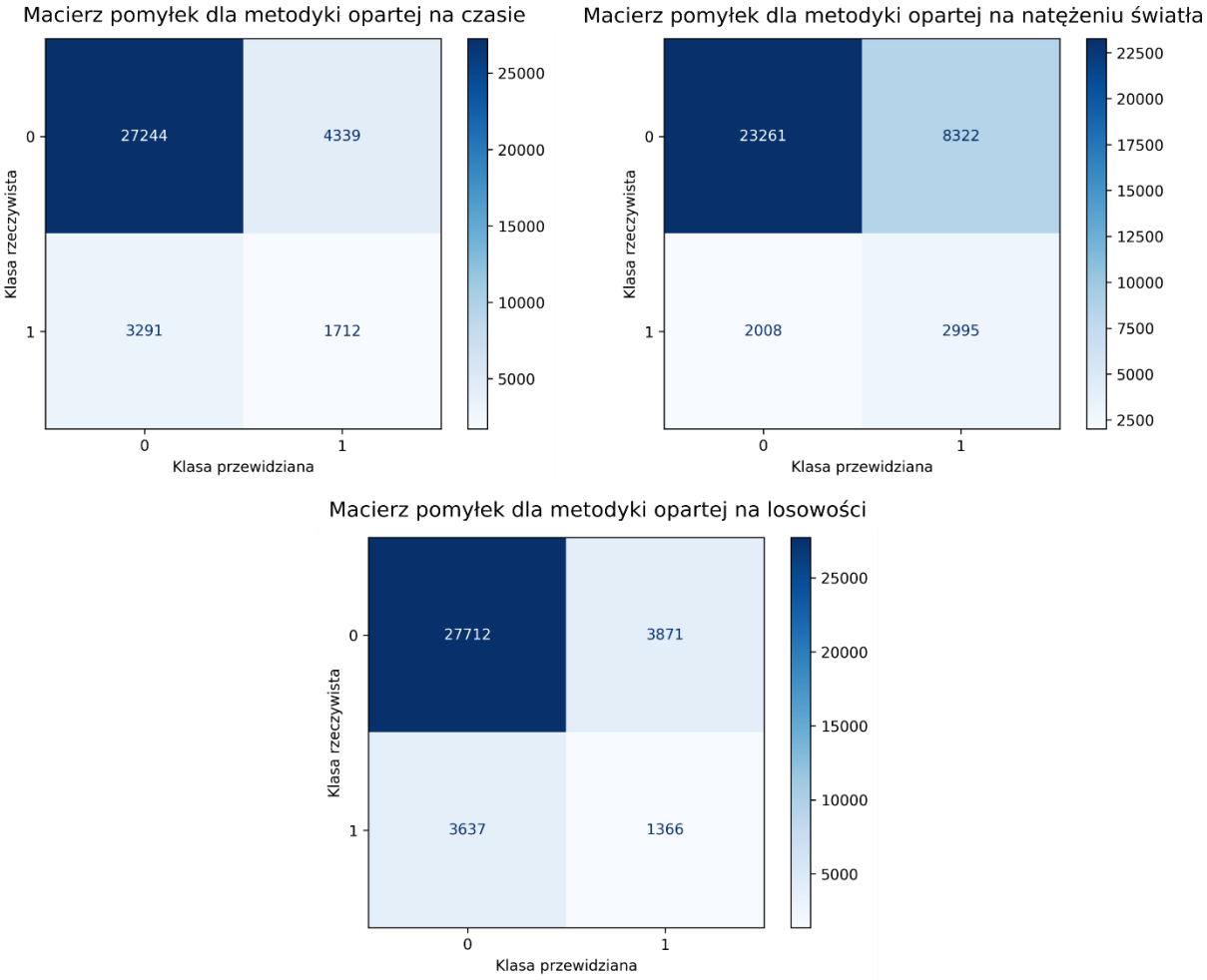
regułach. Różnica ta była szczególnie widoczna w porównaniu z mechanizmem opartym wyłącznie na pomiarze natężenia światła, który okazał się najmniej skuteczny spośród analizowanych podejść.

8.3.2 Macierz pomyłek

Macierz pomyłek jest narzędziem służącym do wizualnej oceny jakości klasyfikatora i składa się z czterech pól, odpowiadających kombinacjom klas rzeczywistych oraz przewidywanych. Najbardziej pożądane są przypadki, w których klasa rzeczywista i klasa przewidywana są zgodne. W szczególności sytuacje, gdy zarówno wartość rzeczywista, jak i przewidywana wynoszą 0 (TN lewy górny róg macierzy) oraz gdy obie przyjmują wartość 1 (TP prawy dolny róg macierzy).

Z punktu widzenia jakości modelu niekorzystne są natomiast przypadki błędnej klasyfikacji, w których przewidywana klasa nie odpowiada klasie rzeczywistej. Obejmują one pola odpowiadające fałszywie pozytywnym (FP prawy górny róg) oraz fałszywie negatywnym (FN lewy dolny róg) predykcjom. Analiza rozmieszczenia tych błędów pozwala na lepsze zrozumienie charakteru pomyłek popełnianych przez model.

Wykres 1. Macierz pomyłek dla metodyk opartych na regułach



Źródło: Opracowanie własne.

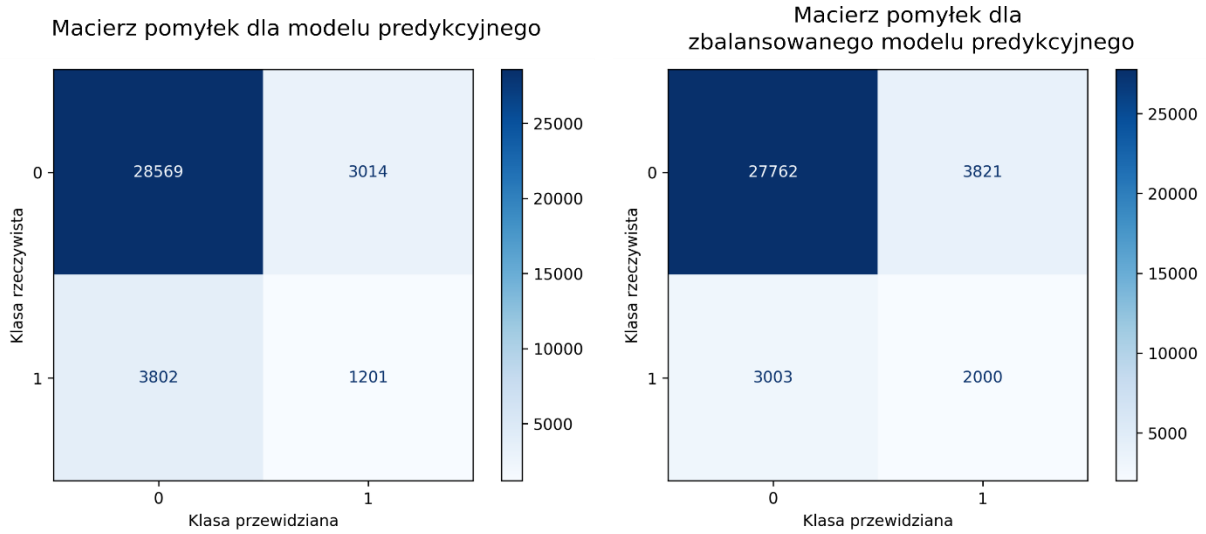
Wykres 1 przedstawia macierze pomyłek dla porównawczych scenariuszy sterowania oświetleniem. Na ich podstawie można zauważyć, że metodyki oparte na regułach czasowych oraz losowych generują zbliżone rezultaty, przy czym niewielką przewagę wykazuje metodyka oparta na

czasie. Przejawia się ona większą liczbą poprawnie wykrytych przypadków włączenia światła (1712 do 1366).

Wyraźnie odmiennie prezentują się natomiast wyniki uzyskane dla metodyki bazującej na pomiarze natężenia światła zewnętrznego. W tym przypadku odnotowano aż 8322 próbki błędnie sklasyfikowane jako stan włączonego oświetlenia (klasa 1). Stanowi to około dwukrotnie większą liczbę błędów tego typu w porównaniu do pozostałych dwóch heurystyk.

Zaobserwowane wyniki są spójne z wcześniejszą analizą, w której metodyka oparta na pomiarze natężenia światła osiągnęła najniższą wartość dokładności, potwierdzając jej ograniczoną skuteczność w realistycznej symulacji obecności mieszkańców.

Wykres 2. Macierze pomyłek dla modeli predykcyjnych



Źródło: Opracowanie własne.

Porównując rezultaty uzyskane przez oba modele (wykres 2) zarówno uczony na pełnym jak i zbalansowanym zbiorze danych, można stwierdzić, że ich ogólna skuteczność jest zbliżona, co potwierdzają również porównywalne wartości metryki dokładności. Jednocześnie modele te wykazują odmienną charakterystykę predykcji, wynikającą ze sposobu przygotowania danych uczących.

Model trenowany na zbiorze niezbalansowanym lepiej radził sobie z poprawną klasyfikacją sytuacji, w których światło było wyłączone (klasa 0), osiągając 28 569 poprawnych predykcji, w porównaniu do 27 762 w przypadku modelu trenowanego na zbiorze zbalansowanym. Odwrotna zależność występowała natomiast przy ocenie sytuacji, gdy światło było rzeczywiście włączone (klasa 1) model zbalansowany poprawnie sklasyfikował 2000 przypadków, podczas gdy model niezbalansowany jedynie 1201. Analogiczne różnice widoczne są również w rozkładzie błędnych predykcji.

Uzyskane wyniki potwierdzają znaną właściwość modeli uczonych na mocno niezbalansowanych zbiorach danych, które mają tendencję do faworyzowania klasy większościowej. Zastosowanie balansowania zbioru uczącego poprawia zdolność wykrywania klasy mniejszościowej, jednak odbywa się to kosztem nieco gorszej skuteczności w klasyfikacji klasy dominującej.

Porównując macierze pomyłek dla scenariuszy opartych na regułach heurystycznych oraz na modelach uczenia maszynowego, można zauważyć, że modele predykcyjne osiągają co najmniej porównywalne rezultaty względem najlepiej wypadających metodyk regułowych, a w niektórych aspektach nawet je przewyższają. Przykładem jest najlepsza skuteczność wykrywania klasy 0 dla modelu niezbalansowanego (28 569 poprawnych predykcji), a także niższa łączna liczba błędów (3014 oraz 3802) w porównaniu do podejść opartych wyłącznie na regułach.

8.3.3 Ocena F1

Tabela 8. Wyniki F1 dla różnych scenariuszy

Scenariusz	**Dokładność**
Mechanizm oparty na czasie	0.3098
Mechanizm oparty na pomiarze natężenia światła	0.3670
Mechanizm losowy	0.2668
Predykcja modelu	0.2606
Predykcja zbalansowanego modelu	**0.3695**

Źródło: Opracowanie własne.

Tabela 8 przedstawia wartości miary F1-score, która w warunkach silnie niezbalansowanego zbioru danych stanowi kluczowy wskaźnik jakości, ponieważ jednocześnie uwzględnia precyzję i czułość klasy mniejszościowej. W analizowanym problemie istotne znaczenie ma nie tylko liczba poprawnych predykcji, lecz również charakter popełnianych błędów oraz ich wpływ na rzeczywiste działanie systemu sterowania.

Mechanizm losowy osiągnął najniższą wartość F1-score (0,2668), co może świadczyć o jego ograniczonej użyteczności. Wysoka liczba poprawnych decyzji dla klasy dominującej nie przekłada się na skuteczne wykrywanie rzadkich, lecz istotnych zdarzeń, co czyni to podejście niepraktycznym w rzeczywistym scenariuszu sterowania.

Mechanizm oparty na czasie uzyskał F1-score równy 0,3098, co wskazuje na umiarkowaną poprawę względem podejścia losowego. Uwzględnienie regularności czasowych pozwala częściowo lepiej przewidywać momenty wymagające włączenia oświetlenia, jednak mechanizm ten nadal nie radzi sobie dobrze z równoważeniem liczby fałszywych i prawdziwych aktywacji.

Na pierwszy rzut oka relatywnie wysoki wynik F1-score dla mechanizmu opartego na pomiarze natężenia światła (0,3670) mógłby sugerować jego dobrą skuteczność. Jednak w kontekście silnego niezbalansowania klas wynik ten wymaga ostrożnej interpretacji. Mechanizm ten wyraźnie faworyzuje klasę 1, odpowiadającą stanowi włączonego światła, co prowadzi do wysokiej czułości, lecz niskiej precyzji. W praktyce oznacza to częste fałszywe aktywacje i znaczne wydłużenie czasu, przez jaki światło pozostaje włączone, w porównaniu do rzeczywistych potrzeb użytkownika. Taki profil błędów, mimo pozornie korzystnej wartości F1-score, skutkuje obniżeniem efektywności energetycznej systemu i czyni to podejście mało użytecznym w realnym zastosowaniu.

Predykcja modelu uczonego na niezbalansowanych danych osiągnęła F1-score na poziomie 0,2606, co jest wynikiem porównywalnym z mechanizmem losowym. Rezultat ten wskazuje, że model ten w znacznym stopniu nauczył się dominującej klasy i minimalizował globalną liczbę błędów kosztem znacznego pomijania klasy mniejszościowej. Jest to typowy efekt uczenia modeli na silnie niezbalansowanych danych bez zastosowania odpowiednich technik kompensacyjnych.

Najbardziej zrównoważone wyniki uzyskała predykcja zbalansowanego modelu (0,3695). Choć wartość ta jest zbliżona do wyniku mechanizmu opartego na natężeniu światła, istotna różnica polega na strukturze popełnianych błędów. Zbalansowany model nie osiąga wysokiego F1-score poprzez nadmierne faworyzowanie klasy 1, lecz poprzez bardziej kontrolowany kompromis pomiędzy precyzją i czułością. W efekcie prowadzi to do decyzji bliższych rzeczywistym wzorcom użytkowania oświetlenia, bez sztucznego wydłużania czasu jego aktywności.

8.4. Niestandardowe oceny skuteczności

Standardowe miary i metody oceny modeli uczenia maszynowego dobrze sprawdzają się w ogólnej ocenie jakości. Zwłaszcza w sytuacjach, gdy te same miary mogą zostać wykorzystane do porównania z innymi sposobami wyznaczania klas, takimi jak opisane wcześniej metody regułowe. Takie podejście umożliwia bezpośrednie i obiektywne zestawienie skuteczności różnych rozwiązań w jednolitych warunkach eksperymentalnych.

Jednocześnie jednak, poza klasycznymi metrykami, zasadne jest zastosowanie niestandardowych metod porównawczych, które w lepszy sposób oddają specyfikę analizowanego problemu oraz jego praktyczny kontekst użytkowy. W szczególności dotyczy to zagadnień, w których istotne są aspekty czasowe, ciągłość działania systemu czy konsekwencje wynikające z długotrwałego użytkowania rozwiązania.

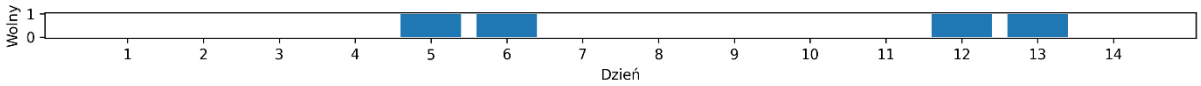
W związku z powyższym, w kolejnych podrozdziałach zaprezentowane zostaną autorskie metody porównawcze, opracowane na potrzeby niniejszej pracy, wraz z przybliżeniem charakterystyki zbioru testowego, na którym przeprowadzono analizy.

8.4.1 Charakterystyka danych testowych

Zbiór testowy obejmuje próbki z dwóch pełnych tygodni, od 28 października 2025 roku do 10 listopada 2025 roku, czyli okres przypadający na środkową część jesieni. W analizowanym przedziale czasowym zachody słońca występowały stosunkowo wcześnie, co miało istotny wpływ na warunki oświetleniowe oraz sposób korzystania z oświetlenia domowego.

Dla porównania, zbiór treningowy obejmował znacznie dłuższy i bardziej zróżnicowany okres, począwszy od wiosny, poprzez lato, aż do początków jesieni. Tak duża rozpiętość sezonowa mogła stanowić dodatkowe wyzwanie dla modelu, który musiał uogólnić wzorce zachowań użytkowników w warunkach istotnie różniących się pod względem długości dnia i poziomu naturalnego oświetlenia.

Wykres 3. Dni wolne



Źródło: Opracowanie własne.

Wykres 3 przedstawia dni wolne od pracy, co ma istotne znaczenie z punktu widzenia analizy wyników, ponieważ charakter codziennych aktywności użytkowników może różnić się pomiędzy dniami roboczymi a weekendami. W analizowanym okresie dni wolne przypadały na 5. i 6. dzień oraz 12. i 13. dzień badanego przedziału czasowego.

Uwzględnienie tego aspektu umożliwia pełniejszą i bardziej trafną interpretację zachowania systemu, szczególnie w kontekście zmiennych wzorców aktywności mieszkańców w zależności od dnia tygodnia.

8.4.2 Analiza czasu działania oświetlenia

Istotnym wskaźnikiem, który należy uwzględnić w analizie skuteczności systemu, jest sumaryczny czas działania oświetlenia. Ma on znaczenie zarówno z punktu widzenia wiernego odwzorowania rzeczywistych praktyk domowników, jak i w kontekście zużycia energii elektrycznej.

Rozwiązaniem niekorzystnym byłby system, który generowałby większe zużycie energii niż wynika to z typowych zachowań użytkowników, a w szczególności taki, którego działanie prowadziłoby do nieuzasadnionych lub nadmiernych poborów energii. Uwzględnienie tego wskaźnika pozwala ocenić system nie tylko pod kątem skuteczności predykcyjnej, lecz również jego efektywności energetycznej i praktycznej użyteczności.

Tabela 9. Suma próbek odpowiadających za włączone oświetlenie przez cały okres testowy

Scenariusz	Ilość próbek
Realne dane z przełącznika	5 003
Mechanizm oparty na czasie	6 051
Mechanizm oparty na pomiarze natężenia światła	11 317
Mechanizm losowy	5 237
Predykcja modelu	4 215
Predykcja zbalansowanego modelu	5 821

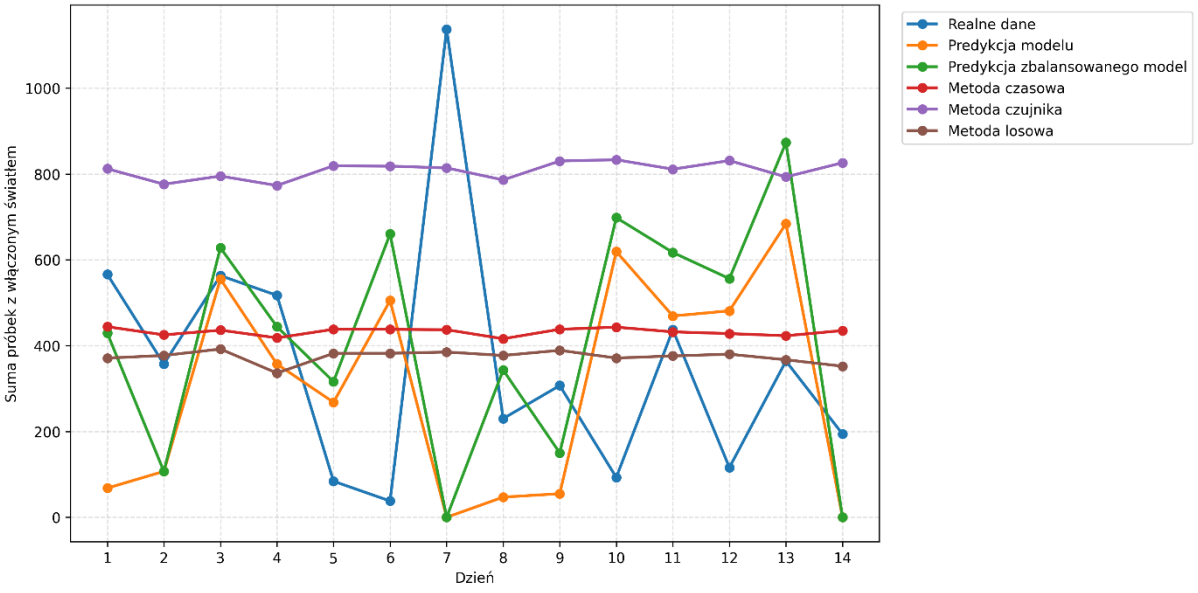
Źródło: Opracowanie własne.

W tabeli 9 przedstawiono rzeczywistą liczbę próbek, w których oświetlenie było włączone (5003 próbki) w trakcie całego okresu testowego, obejmującego dwa tygodnie. Porównując tę wartość z wynikami uzyskanymi dla poszczególnych wariantów sterowania, można zauważyć, że w przypadku większości modeli oraz metodyk różnica nie przekracza 20% względem wartości rzeczywistej.

Wyraźnym wyjątkiem jest jednak mechanizm oparty na pomiarze natężenia światła zewnętrznego, dla którego liczba próbek z włączonym oświetleniem wyniosła 11 317, co oznacza ponad dwukrotnie dłuższy czas aktywności oświetlenia w porównaniu do rzeczywistych danych. Taki wynik może bezpośrednio przekładać się na istotne zwiększenie zużycia energii elektrycznej, a tym samym na wzrost kosztów eksploatacyjnych.

Z tego względu praktyczna użyteczność tej metody w kontekście symulacji obecności domowników może być uznana za wątpliwą, szczególnie w porównaniu z rozwiązaniami, które lepiej odwzorowują rzeczywiste wzorce czasu użytkowania oświetlenia.

Wykres 4. Sumaryczny czas aktywności oświetlenia dla poszczególnych dni



Źródło: Opracowanie własne.

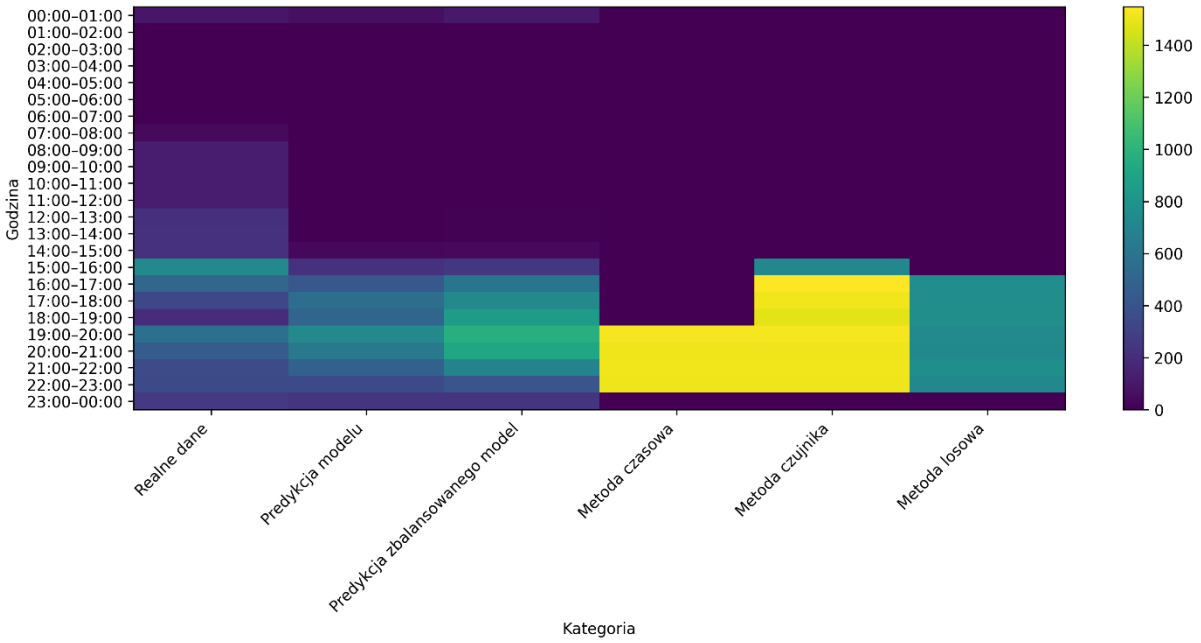
Zestawiając sumaryczny czas aktywności oświetlenia, wyrażony liczbą próbek z włączonym oświetleniem, dla poszczególnych dni analizowanego zbioru danych, można zauważyć, że metody oparte na czasie, czujniku natężenia światła oraz mechanizmie losowym charakteryzują się dużą stabilnością w czasie. Oznacza to, że długość świecenia oświetlenia w kolejnych dniach pozostaje w ich przypadku na zbliżonym poziomie.

Wynika to z odmiennych, lecz przewidywalnych cech każdej z tych metodyk. Metoda czasowa opiera się na stałych godzinach włączenia i wyłączenia oświetlenia, co sprawia, że łączny czas świecenia jest w praktyce identyczny każdego dnia. Metoda losowa z kolei generuje stan oświetlenia poprzez losowanie jednej z dwóch wartości (0 lub 1) z jednakowym prawdopodobieństwem dla każdej próbki. Zgodnie z prawami rachunku prawdopodobieństwa, przy odpowiednio dużej liczbie próbek w ciągu dnia, liczba stanów włączonych i wyłączonych w zadanym przedziale czasowym dąży do wyrównania. W efekcie również w tym przypadku można przyjąć, że czas aktywności oświetlenia w kolejnych dniach będzie zbliżony, co znajduje odzwierciedlenie na wykresie.

Niewielka zmienność obserwowana dla metody czujnikowej może natomiast wynikać z małej dynamiki zmian pory zachodu słońca w analizowanym okresie, co przekłada się na relatywnie płaską charakterystykę czasu świecenia. Dodatkowo warto zwrócić uwagę na dwukrotnie większe czasy świecenia w porównaniu do poprzednich metod.

Odmienny obraz wyłania się w przypadku danych rzeczywistych oraz modeli predykcyjnych, dla których widoczna jest znacznie większa zmienność długości świecenia pomiędzy poszczególnymi dniami. Choć stopień dopasowania charakterystyki modeli do faktycznego wykorzystania oświetlenia jest niewielki to, sama obecność wyraźnej zmienności, analogicznej do tej obserwowanej w danych rzeczywistych warto ocenić na korzyść zastosowania modeli predykcyjnych. Świadczy to o ich potencjale do lepszego odwzorowywania nieregularnych i zmiennych zachowań użytkowników w porównaniu z metodami opartymi na sztywnych regułach.

Wykres 5. Sumaryczny czas aktywności oświetlenia dla poszczególnych godzin



Źródło: Opracowanie własne.

Wykres 5 przedstawia sumaryczną intensywność wykorzystania oświetlenia w całym okresie testowym, z podziałem na poszczególne godziny doby. Celem tej analizy było określenie, w których godzinach występuje największa aktywność oświetlenia dla poszczególnych scenariuszy sterowania

oraz porównanie ich z rzeczywistym wykorzystaniem oświetlenia, wyznaczonym na podstawie danych pochodzących z fizycznego przełącznika.

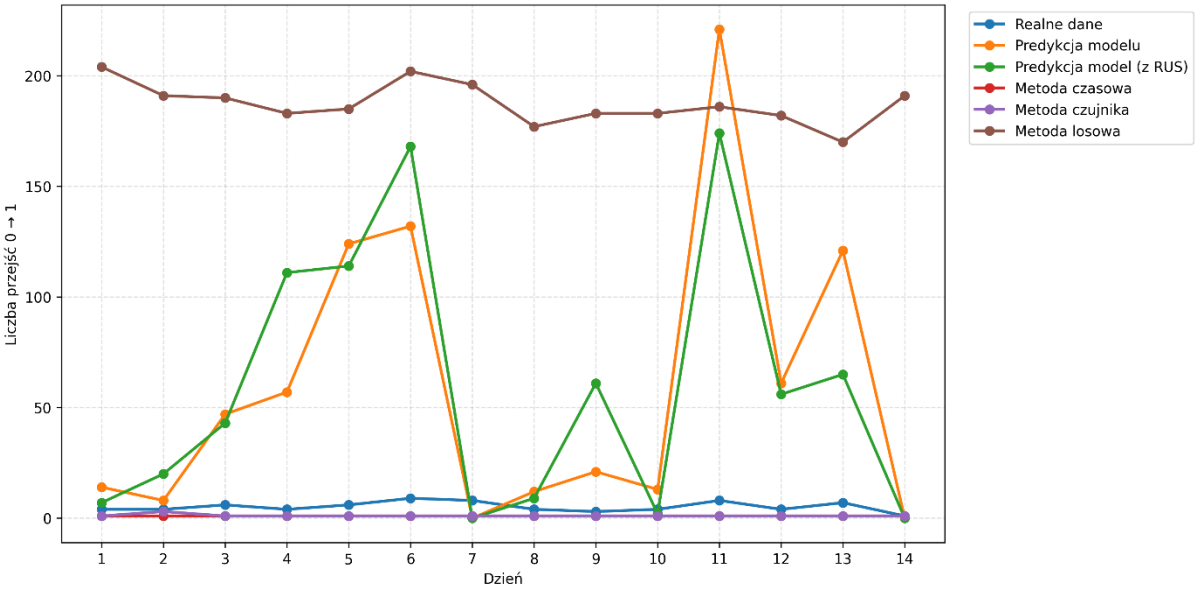
Z analizy wynika, że metody oparte na czasie oraz na pomiarze natężenia światła charakteryzują się bardzo wysoką gęstością próbek z włączonym oświetleniem przez cały okres ich aktywności. Oznacza to, że po jednorazowym włączeniu światła pozostawało aktywne niemal nieprzerwanie aż do momentu zakończenia działania mechanizmu, wynikającego z ograniczeń regułowych (tj. do godziny 23:00 UTC).

Metoda losowa wykazuje podobnie równomierny rozkład aktywności w obrębie przewidzianego okna czasowego, jednak intensywność wykorzystania oświetlenia jest w tym przypadku wyraźnie (około dwukrotnie) niższa. Wynika to z faktu, że światło było w krótkich odstępach czasu naprzemiennie włączane i wyłączane, często nawet co około 30 sekund, co prowadziło do rozproszonego, lecz mniej intensywnego wykorzystania.

Analiza danych rzeczywistych ujawnia natomiast zróżnicowaną intensywność wykorzystania oświetlenia w zależności od godziny, a także szerszy zakres godzin aktywności w porównaniu do metod regułowych. W tym kontekście modele predykcyjne wykazują zbliżoną charakterystykę czasową, choć można w ich przypadku zaobserwować pewien stopień symetrii zagęszczenia godzin aktywności. Symetria ta jest jednak znacznie mniej wyraźna niż w przypadku metod opartych na sztywnych regułach, co może świadczyć o większej elastyczności i lepszym dopasowaniu do rzeczywistych wzorców zachowań użytkowników.

8.4.3 Analiza ilościowa zmiany stanu przełącznika

Wykres 6. Sumaryczna aktywność wirtualnego włącznika oświetlenia



Źródło: Opracowanie własne.

Ostatnim analizowanym aspektem była dzienna liczba włączeń oświetlenia, rozumiana jako liczba przełączeń stanu z 0 na 1, odzwierciedlająca aktywność samego przełącznika światła. Wskaźnik ten pozwala ocenić, jak często w ciągu dnia dochodzi do inicjowania pracy oświetlenia w poszczególnych scenariuszach sterowania.

Na wykresie przedstawiającym wyniki (wykres 6) dla metody czasowej oraz czujnikowej widoczna jest bardzo niewielka liczba zmian stanu przełącznika. Jest to zbieżne z charakterystyką rzeczywistych zachowań użytkowników, gdzie światło najczęściej włączane jest sporadycznie, lecz pozostaje aktywne przez dłuższy czas.

Odmienna sytuacja występuje w przypadku metody losowej, dla której liczba przełączeń ze stanu wyłączonego na włączony jest skrajnie wysoka i w niektórych dniach sięga nawet 200 aktywacji na dobę. Taka charakterystyka znacząco odbiega od realnych praktyk.

W przypadku modeli predykcyjnych obserwowana jest stosunkowo duża zmienność liczby przełączeń pomiędzy poszczególnymi dniami. Choć nie osiąga ona poziomu charakterystycznego dla metody losowej, to jednak nadal wyraźnie odbiega od wzorca obserwowanego w danych rzeczywistych, co w tym aspekcie należy ocenić na niekorzyść zastosowanych modeli.

9. Podsumowanie

Przed rozpoczęciem pracy postawiono szereg pytań, na które niniejsza praca miała przynajmniej częściowo udzielić odpowiedzi. Jedno z nich dotyczyło tego, czy zaprojektowany prototyp, wykorzystujący integrację platformy Home Assistant z modelami uczenia maszynowego, sprawdzi się pod względem niezawodności. Na podstawie kilku miesięcy obserwacji działania systemu można stwierdzić, że taka kombinacja jest jak najbardziej możliwa do zastosowania w warunkach praktycznych. W trakcie działania systemu pojawiły się co prawda nieliczne problemy, takie jak m.in. utrata historii części urządzeń w Home Assistant, jednak udało się na nie szybko zareagować poprzez niewielką modyfikację metody pozyskiwania danych. Warto jednak podkreślić, że zaproponowane rozwiązanie miało charakter dowodowy i stanowiło element szerzej zakrojonego zagadnienia, dotyczącego możliwości symulowania zachowań użytkowników z wykorzystaniem uczenia maszynowego. Aby przekształcić projekt w pełnoprawny produkt, należałoby uporządkować koncepcję i opracować ją w formie rozszerzenia (np. w postaci plug-inu dla Home Assistant lub innego asystenta domowego). Biorąc pod uwagę podobieństwo podstawowych funkcji różnych platform inteligentnego domu, można przypuszczać, że wdrożenie w alternatywnych środowiskach zachowałoby porównywalną skuteczność.

Kolejne pytanie dotyczyło tego, czy mechanika sterowania oparta na modelach predykcyjnych może w wybranych aspektach przewyższać rozwiązania bazujące na sztywnych regułach eksperckich. Przeprowadzone analizy porównawcze, obejmujące zarówno modele predykcyjne jak i alternatywne popularne metody sterowania, wskazują, że w pewnych obszarach techniki oparte na uczeniu maszynowym mogą dorównywać, a nawet przewyższać metody regułowe. W wielu sytuacjach rozwiązania oparte na sztywnych regułach gorzej radziły sobie z odzwierciedlaniem momentów, w których użytkownik faktycznie zdecydowałby się na włączenie światła w warunkach domowych. Przyczyną jest m.in. to, że użytkownicy implementujący takie rozwiązania rzadko prowadzą długotrwałą obserwację własnych nawyków, ograniczając się jedynie do wprowadzenia najprostszych konfiguracji. Dodatkowo metody regułowe cechują się dużą przewidywalnością, co w pewnych kontekstach może stanowić zagrożenie, np. ułatwiać potencjalnemu intruzowi rozpoznanie czy mieszkańcy są faktycznie obecni w domu przez powtarzalność zachowania oświetlenia. Modele predykcyjne wykazały natomiast większą zmienność i mniejszą przewidywalność zachowań, co należy zaliczyć na ich korzyść. Nie oznacza to jednak, że rozwiązanie to jest wolne od wad. Analiza liczby przełączeń stanu oświetlenia wykazała zauważalnie większą aktywność przełącznika w porównaniu z rzeczywistym użytkowaniem lub nawet wybranymi metodami regułowymi. Problem ten można jednak w przyszłości częściowo zniwelować poprzez wprowadzenie dodatkowych mechanizmów ograniczających. Przykładem takich rozwiązań może być zmniejszenie częstotliwości wywoływania modelu (np. z interwału 30 sekund do 1 minuty jak w zaproponowanej konfiguracji projektu lub nawet 5 minut), co mogłoby korzystnie wpływać na ograniczenie nadmiernych zmian stanu oświetlenia.

Ostatnie pytanie dotyczyło tego, czy system wykorzystujący uczenie maszynowe jest w stanie wiarygodnie udawać rzeczywiste zachowania użytkowników. Odpowiedź na to pytanie nie jest jednoznaczna. Jeżeli przez realistyczną symulację rozumie się wierne przewidywanie momentów, w których człowiek faktycznie zapaliłby światło, to osiągnięcie takiej dokładności może okazać się bardzo trudne. Nawet uzyskana dokładność na poziomie około 80% oznacza, że w pozostałych 20% model się myli. Należy jednak zauważyć, że podejścia oparte na regułach radziły sobie w tym zakresie równie słabo, a w pewnych przypadkach nawet gorzej. Duży wpływ na jakość predykcji ma charakterystyka danych wejściowych oraz indywidualne nawyki mieszkańców. Użytkownicy prowadzący bardziej ustrukturyzowany tryb życia (np. posiadający stałe rutyny o określonych godzinach) prawdopodobnie wygenerowaliby dane łatwiejsze do modelowania. Z kolei osoby o bardziej nieregularnym rytmie dnia mogą stanowić trudniejsze wyzwanie dla algorytmu. Jeżeli natomiast przez „udawanie rzeczywistych zachowań” rozumie się samą zmienność oraz brak pełnej przewidywalności w kolejnych dniach, bez jednoczesnego popadania w skrajność charakterystyczną dla

losowego sterowania światłem, to modele predykcyjne wykazały, że potrafią generować zachowania zbliżone do ludzkich pod względem dynamiki.

Podsumowując całość pracy, zarówno w warstwie projektowej jak i badawczej, warto zauważyć, że szeroki zakres poruszanych zagadnień był niewątpliwie wartością dodaną. Udało się dotknąć obszarów związanych z MLOps, mikrokomputerami, uczeniem maszynowym, analizą skuteczności, konteneryzacją oraz integracją systemów inteligentnego domu. Jednocześnie szerokość tematu wiąże się z tym, że omówienie poszczególnych zagadnień miało charakter raczej szeroki niż pogłębiony, co rodzi pewien niedosyt i pozostawia przestrzeń do dalszego rozwoju. W kolejnych krokach warto rozważyć przekształcenie projektu do postaci minimalnego gotowego produktu (MVP), przykładowo poprzez przygotowanie działającego plug-inu. Dobrą praktyką byłoby również wydłużenie okresu zbierania danych do pełnego roku. Dałoby to możliwość oceny jakości predykcji w warunkach obejmujących pełny cykl sezonowy z uwzględnieniem różnych warunków pogodowych, okresów urlopowych, świąt i zmian godzin zachodu słońca. Kolejną możliwością rozwoju jest odejście od domyślnych konfiguracji modeli i przeprowadzenie strojenia hiperparametrów, co mogłoby poprawić skuteczność modeli i ograniczyć zidentyfikowane problemy, takie jak nadmierne przełączanie oświetlenia. Osobną kwestią jest wybór pomiędzy zbalansowanym a niezbalansowanym zbiorem treningowym. Tu warto zastanowić się, którą kategorię (włączone/wyłączone światło) lepiej faworyzować, lub czy pozostawić wybór użytkownikom, dając możliwość konfiguracji czy chcą lepszego odwzorowania włączonego światła, kosztem minimalnie zwiększonego zużycia prądu. Na koniec warto też podkreślić, że wraz z wydłużeniem pracy na modelu (np. kilka miesięcy nieprzerwanej pracy) może zaistnieć zjawisko starzenia zbioru treningowego na których model się uczył, co może wpłynąć na pogarszającą się jakość modelu. Pomimo wskazanych obszarów wymagających dopracowania, niniejszy projekt spełnił zakładane cele i pokazał, że w przedstawionym podejściu drzemie istotny potencjał.

Dodatki

Dodatek A: Bibliografia

[1]. Home automation with smart lighting | Philips Hue NZ https://www.philips-hue.com/en-nz/explore-hue/blog/home-automation Dostęp: 10.10.2025

[2]. Luvo Matrix LED Lamp - Luke Roberts Lighting https://www.luke-roberts.com/ Dostęp: 10.10.2025

[3]. Smart Home Automation | AI-Powered Living Spaces | Heyzack https://heyzack.ai/smarthome Dostęp: 10.10.2025

[4]. KEVIN® - Home Security | Smart.Secure.Swiss | Mitipi https://kevinswiss.com/ Dostęp: 10.10.2025

[5]. GitHub - lcmchris/thesillyhome-container: containerized version https://github.com/lcmchris/thesillyhome-container Dostęp: 11.10.2025

[6]. Raspberry Pi 5 8 GB Sklep Botland https://botland.com.pl/moduly-i-zestawy-raspberry-pi-5/23905-raspberry-pi-5-8gb-5056561803326.html Dostęp: 15.10.2025

[7]. Install Ubuntu on a Raspberry Pi | Ubuntu https://ubuntu.com/download/raspberry-pi Dostęp: 15.10.2025

[8]. Documentation - Home Assistant https://www.home-assistant.io/docs/ Dostęp: 15.10.2025

[9]. BleBox luxSensor - czujnik natężenia światła WiFi - aplikacja Android/iOS Sklep Botland https://botland.com.pl/blebox-automatyka-domowa/25954-blebox-luxsensor-czujnik-natezenia-swiatla-wifi-aplikacja-androidios-5900168580671.html Dostęp: 24.10.2025

[10]. Home Assistant API - Home Assistant https://www.home-assistant.io/integrations/api/ Dostęp: 24.10.2025

[11]. Sonoff Dotykowy Włącznik T1 EU TX (WiFi+RF433 1-kanałowy) - Inteligentne włączniki światła - Sklep komputerowy - x-kom.pl https://www.x-kom.pl/p/524637-inteligentny-wlacznik-swiatla-sonoff-dotykowy-wlacznik-t1-eu-tx-wifirf433-1-kanalowy.html Dostęp: 26.10.2025

[12]. Home - eWeLink https://ewelink.cc/ Dostęp: 26.10.2025

[13]. Compose file reference | Docker Docs https://docs.docker.com/reference/compose-file/ Dostęp: 15.11.2025

[14]. MLflow Tracking Quickstart | MLflow https://mlflow.org/docs/latest/ml/getting-started/quickstart/ Dostęp: 23.11.2025

[15]. AWS S3 Compatible Object Storage | MinIO https://www.min.io/product/aistor/s3-compatibility Dostęp: 25.11.2025

[16]. Artifact Stores | MLflow https://mlflow.org/docs/latest/self-hosting/architecture/artifact-store/#amazon-s3-and-s3-compatible-storage Dostęp: 25.11.2025

[17]. Settings and Configurations | AIStor Object Store Documentation https://docs.min.io/enterprise/aistor-object-store/reference/aistor-server/settings/ Dostęp: 25.11.2025

[18]. MLflow Models | MLflow https://mlflow.org/docs/latest/ml/model/ Dostęp: 25.11.2025

[19]. MongoDB Documentation - Homepage https://www.mongodb.com/docs/ Dostęp: 28.11.2025

[20]. Dynaconf - 3.2.12 https://www.dynaconf.com/ Dostęp: 3.12.2025

[21]. Configuration.yaml - Home Assistant https://www.home-assistant.io/docs/configuration/ Dostęp: 5.12.2025

[22]. Automation YAML - Home Assistant https://www.home-assistant.io/docs/automation/yaml/ Dostęp: 5.12.2025

[23]. scikit-learn: machine learning in Python - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/ Dostęp: 8.12.2025

[24]. train_test_split - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html Dostęp: 8.12.2025

[25]. StandardScaler - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html Dostęp: 8.12.2025

[26]. Pipeline - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html Dostęp: 8.12.2025

[27]. accuracy_score - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html Dostęp: 8.12.2025

[28]. RandomUnderSampler - Version 0.14.1 https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html Dostęp: 11.12.2025

[29]. RandomForestClassifier - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html Dostęp: 15.12.2025

[30]. 1.11. Ensembles: Gradient boosting, random forests, bagging, voting, stacking - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/ensemble.html#random-forests-and-other-randomized-tree-ensembles Dostęp: 15.12.2025

[31]. 1.10. Decision Trees - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/tree.html#classification-criteria Dostęp: 15.12.2025

[32]. Czy żarówki LED można często włączać? Wpływ na pobór prądu https://spidersweb.pl/2022/08/czy-zarowki-led-mozna-czesto-wlaczac-pobor-pradu.html Dostęp 16.12.2025

[33]. confusion_matrix - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html Dostęp: 16.12.2025

[34]. f1_score - scikit-learn 1.7.2 documentation https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html Dostęp: 16.12.2025

Dodatek B: Wykaz tabel

Dodatek C: Wykaz wykresów

Dodatek D: Wykaz rysunków

Dodatek E: Wykaz listingów