Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Inżynieria Systemów (INS)**

PRACA DYPLOMOWA
INŻYNIERSKA

Moduł samouczący do systemu automatyki domowej

Przemysław Grzegorz Barcicki

Opiekun pracy
dr. inż. Patryk Schauer

Słowa kluczowe: automatyzacja domowa, uczenie maszynowe, smart-home, python, tensorflow

WROCŁAW 2023

STRESZCZENIE

Niniejsza praca ma na celu rozwiązanie problemu tworzenia złożonych reguł automatyzacji w systemach *smart-home* poprzez wykorzystanie uczenia maszynowego i metod data-miningu. Moduł jest skierowany do osób, które chcą wypróbować możliwości jakie daje automatyzacja domu, bez konieczności tworzenia reguł dla każdego działania w ich codziennym życiu. Wyodrębniając epizody interakcji użytkownika z zapisanej historii HomeAssistant, moduł tworzy dane dla sieci neuronowych, które próbują przewidzieć następne działanie użytkownika i w przypadku wspólnego zamiaru, wykonuje je. Moduł ma być jak najbardziej rozszerzalny poprzez wdrożenie zorientowanego na obiekty modelu oraz wykorzystanie wolnego i otwartego oprogramowania.

ABSTRACT

This work aims to solve the issue of creating complex automation rules in smart home systems by using machine learning and data mining. The module is addressed to people who are eager to try out the prospects made possible by home automation without the hassle of creating rules for every action in their day-to-day lives. By extracting the episodes of user interaction from HomeAssistant logs, it creates data for neural networks that try to predict the user's next action and, in cases of common intent, execute them. Module aims to be as extendable as possible by implementing object-oriented design and by using free and open source software.

SPIS TREŚCI

1. WSTĘP

1.1. WPROWADZENIE DO PROBLEMATYKI

Automatyzacja to określenie na metody mające na celu ograniczenie ludzkiej interakcji do minimum w różnych procesach. Stosuje się ją w wielu dziedzinach począwszy od przemysłu, kończąc na procesach informatycznych. Pewnym obszarem zyskującym w ostatnim czasie dużo zainteresowania jest pojęcie inteligentnego domu (smart-home). Automatyka domowa to konkretne określenie na zastosowanie automatyzacji wśród urządzeń smart-home skupiających się na obszarze zastosowań gospodarstwa domowego. Celem takiego zastosowania jest kontrola pracy urządzeń znajdujących się w domu do osiągnięcia konkretnego ich stanu w sposób minimalizujący ludzką interakcję. W praktyce takie systemy to bardzo dobre rozwiązanie które niesie ze sobą wiele zalet i uproszczeń w codziennym życiu [27].

Reguła w automatyce, czasem nazywana zasadą, to w dokładny sposób określenie jakie akcje muszą zostać wykonane pod warunkiem pewnego stanu systemu. Warunkiem początkowym może być każdy stan, zmiana stanu dowolnego urządzania lub całkowicie zewnętrzny bodziec. W przypadku takiej automatyzacji często też korzysta się z kombinacyjnego połączenia wielu źródeł i informacji w celu stworzenia stanu początkowego, który jeśli wystąpi, jest przesłanką do wysłania przez system zarządzający urządzeniami polecenia do wykonania pojedynczej lub ciągu akcji. Każde dostępne na rynku dedykowane oprogramowanie obsługujące automatykę domową dostarcza różne narzędzia do szybkiego i intuicyjnego sporządzania takich zasad, ale także dostarcza sposoby na komunikację z tymi urządzeniami. Dostepny jest zatem pełny system, który agreguje wiele pomniejszych mechanizmów, protokołów i sposobów wymiany danych w celu monitorowania i sterowania urządzeniami w ramach np. lokalnej sieci.

1.2. OPIS PROBLEMU

Mimo tego, że analiza i projektowanie systemów inteligentnej automatyki domów zaczęła się kilka dekad temu, istnieje wiele nierozwiązanych problemów, które muszą zostać rozwiązane, aby autonomiczna forma stała się popularna. Ograniczeniami są nieprawidłowe algorytmy oraz niskie celności wynikowe [17].

Mimo wygody w użytkowaniu i łatwości w sporządzaniu zasad automatyki domowej pojawiają się pewne trudności. Jednym z nich jest problem ze zmiennością ludzkich na-

wyków. Sporządzenie zasad sprawia, że dane czynności zostaną wykonywane wtedy i tylko wtedy gdy zostanie spełniony pewien konkretny stan określony przez użytkownika podczas sporządzania danej reguły. Z logicznego punktu widzenia jest to odpowiednia reakcja systemu, natomiast z tego praktycznego już nie, ponieważ dana reguła może zostać wykonana mimo tego, że w pewnych specyficznych okolicznościach użytkownik nie zechce aby się wykonała. Dodaje to pewien wymóg dalszego komplikowania danej zasady poprzez dodatkowe warunki w systemie lub ciągłe jej edytowanie aby odpowiadała naszym zmiennym nawykom.

Bez względu na to, jakie narzędzia dostarcza nam dany system, sporządzanie dużej liczby skomplikowanych reguł może być problematyczne. Same środowiska zarządzania takimi systemami mimo wspierania tworzenia na tyle skomplikowanych zasad mogą nie być do tego przystosowane. Często zdarza się, że system automatyki udostępnia dodatkowe, jako wtyczki lub dodatki, moduły do tworzenia reguł za pomocą interpretowanych języków programowania [5], [8], [18]. Tworzy to natomiast zestaw innych problemów, gdzie użytkownik chcący stworzyć takie reguły, musi w przynajmniej podstawowy sposób znać języki programowania, a ponadto mieć wiedzę w jaki sposób sporządzać te skrypty aby mogły być wykonywane przez oprogramowanie nadzorujące.

1.3. CEL PRACY

Celem niniejszej pracy jest opracowanie modułu do systemu automatyki domowej (smart-home) wspierającego tworzenie reguł decyzyjnych w oparciu o mechanizmy uczące się zachowań użytkownika. Opracowane rozwiązanie powinno korzystać z wielu źródeł danych, czujników, stanu aktuatorów, źródeł zewnętrznych (np. prognoza pogody), stanu przełączników wyzwalanych przez użytkownika. Efektem działania modułu powinny być reguły możliwe do zastosowania w systemie automatyki lub bezpośrednio wywoływane w tym systemie akcje.

1.4. ZAKRES PRACY

Zakres pracy obejmuje przegląd literatury w obszarze działania i budowy systemów wspomagania podejmowania decyzji opartych o uczenie maszynowe, a także przegląd tematyki związanej z działaniem i budową urządzeń Internetu Rzeczy z klasy smart-home. W zakresie również znajduje się projekt i implementacja prototypowego modułu samouczącego wraz z jego testami i integracją z istniejącym systemem autmatyki domowej.

2. PRZEGLĄD LITERATURY

W tym rozdziale omówiono umiejscowienie tematu w literaturze. W pierwszej części opisano istniejące algorytmy i rozwiązania problemu. Pod uwagę brano te metody, które korzystają z klasycznych metod uczenia maszynowego oraz te, które zawierają elementy sieci neuronowych. Analizie zostaną poddane użyte algorytmy, jak i podejścia do osiągnięcia celu wspomagania decyzji automatyki domowej. W drugim etapie zawarte zostały opisy budowy i sposobów działania urządzeń Internetu Rzeczy.

2.1. ALGORYTMY I ROZWIĄZANIA

2.1.1. Systemy korzystające z algorytmów wykrywania epizodów

Episode discovery (ang. wykrywanie epizodów) to metoda *data miningu* (ang. kopania danych) wykorzystująca istniejący ciąg występujących po sobie wydarzeń do wykrywania w nich powtarzalnych znaczących epizodów. Wśród epizodów można wyróżnić tak zwane epizody znaczące, które według zależnych od algorytmu charakterystyk, określają dany epizod jako często występujący.

Czesto można spotkać się z pewną interpretacją ciągu zdarzeń w systemie, gdzie każda występująca po sobie w pewnym oknie czasowym akcja jest reprezentowana jako odpowiedni znak ze zbioru możliwych zdarzeń. W przypadku opisu stanu systemu za pomocą reprezentacji znakowej, korzysta się z metodyki, gdzie duża litera oznacza przejście stanu danego urządzenia w załączone, a mała litera oznacza wyłączenie. W przykładowych ciągach zdarzeń, `baDgb`, `abD`, `Dagb`, można zauważyć, że zawsze po wystąpieniu wyłączenia urządzenia `a`, następuje wyłączenie urządzenia `b`. W rzeczywistości algorytmy są bardziej zaawansowane, ponieważ są w stanie wykryć epizody znaczące, które nie występują zawsze a w pewnym odsetku wszystkich wydarzeń i ponadto są w stanie operować na dużo dłuższych łańcuchach historycznych.

Wykorzystywanie algorytmów wykrywania epizodów na zapisanych już strumieniach wydarzeń pozwala na znalezienie pewnych nawyków i zależności z codziennego korzystania z domowych urządzeń. Znalezione i wyodrębnione epizody mogą zostać użyte z innymi algorytmami w celu podniesienia ich celności. Tak przetworzone dane wejściowe z dodatkowym użyciem innego algorytmu dają zdecydowanie lepsze wyniki niż w wypadku użycia samych sieci neuronowych bądź samego wykrywania epizodów [10], [17].

Ważnym elementem wykorzystania technik wykrywania epizodów jest prawidłowy wybór algorytmu (takiego jak np. SPADE [30], SPEED [3], WINEPI [15], ...) jak i jego własności, ponieważ inne wartości parametrów wybierających epizod znaczący mogą mocno wpływać na końcowy wynik [17]. O ile w przypadku algorytmu, nieprawidłowy wybór może ograniczać się do wydłużonego czasu poszukiwania epizodów, a co za tym idzie, większego zużycia zasobów przez system, tak w przypadku niewłaściwych parametrów, algorytm może proponować za dużo akcji i nawyków, które źle będą wpływały na końcowy wynik, co w końcu sprawi, że komfort korzystania z takiego systemu będzie niedostateczny.

Podejście wykrywania epizodów jest metodą, która skupia się na ciągach zdarzeń i ich wzajemnej kolejności, a nie na konkretnym stanie systemu. Sprawia to, że system uczy się ciągów wydarzeń nie biorąc pod uwagę aktualnego stanu, czyli na przykład pory dnia, temperatury w pomieszczeniu, dnia tygodnia czy też pogody. Połączenie klasycznych algorytmów *episode discovery* z dodatkowymi algorytmami, które biorą pod uwagę inne parametry systemu, rozwiązuje ten problem, ale dodaje dużo skomplikowania w implementacji.

Korzystanie z metody dużych i małych liter do oznaczania wyłączeń i włączeń urządzeń tworzy dodatkowo pewne ograniczenia w postaci braku lub bardzo skomplikowanej obsługi urządzeń o niebinarnych stanach. O ile w przypadku urządzeń gdzie stanów jest kilka, jak na przykład systemy HVAC, można każdy z trybów pracy zinterpretować jako inną czynność, tak w przypadku urządzeń gdzie istnieje teoretycznie nieskończenie wiele stanów pośrednich, tak jak na przykład w ściemniaczach żarówkowych, nie jest możliwe reprezentowanie każdego z nich.

2.1.2. Systemy korzystające z procesów Markowa

Podczas modelowania zmian stanu systemu automatyki domowej można skorzystać z podejścia, gdzie próbujemy opisać ciąg zdarzeń za pomocą prawdopodobieństwa przejść między stanami. Bardzo pomocne w takiej strategii okazuje się korzystanie z modeli Markowa. Istnieje pewne rozszerzenie, które okazuje się bardziej pomocne w wypadku modelowania ludzkiej interakcji i zachowań z powodu uwzględnienia niezależnych i nieznanych przez model stanów, nazywane ukrytym modelem Markowa. Czyste podejście z prawdopodobieństwem odpowiada na pytanie, jaka jest szansa na wykonanie akcji `A` pod warunkiem tego, że poprzednio wykonana akcja to `B`, dodatkowe ukryte informacje pomagają w dokładniejszym określeniu przewidywanego stanu systemu.

Podobnie jak w przypadku wykrywania epizodów (2.1.1) do wytworzenia reprezentacji zmian systemu wykorzystywany jest literowy zapis, co sprawia, że te podejścia mają takie same ograniczenia. Jedna z prac [29], próbuje rozwiązać problem związany z rzeczywistą (niebinarną) naturą pewnych stanów poprzez kwantyzację w konkretne stany. Konkretne wartości temperaturowe zamieniane są na arbitralny identyfikator wskazujący na daną temperaturę, co używane jest w modelu jako jeden z parametrów.

Podejście znalezione w [29] próbuje rozwiązać, za pomocą autorskiej metody *IHMM* (Improved Hidden Markov Models), problem powiązania pewnych konkretnych nawyków i ciągów wydarzeń z pewną temporalną zmienną, co sprawia, że system podpowiada konkretne akcje dokładniej o konkretnych porach dnia, lecz dalej nie rozwiązuje problemu rozpoznania np. dni tygodnia czy pogody.

Inna implementacja [6], korzysta z innego, również autorskiego, rozwinięcia modeli Markowa *TMM* (Task-based Markov Model). Metoda skupia się na zidentyfikowaniu wysokopoziomowych zadań, które to dalej są wykorzystywane do stworzenia pomniejszych modeli reprezentujących ciąg zdarzeń w konkretnym zadaniu. Zadania są identyfikowane na podstawie przerwy pomiędzy kolejnymi wydarzeniami oraz na podstawie dodatkowej informacji o lokalizacji danego urządzenia. Operacje wraz z informacjami o ilości, długości i położenia w przestrzeni używane są przez algorytm k-średnich do pogrupowania zadań i stworzenia zbioru konkretnych ciągów.

2.1.3. Sieci neuronowe

Sieci neuronowe to matematyczny model stosowany do celów przetwarzania informacji i podejmowania decyzji. Ich budowa, w pewnym stopniu, jest wzorowana na rzeczywistym funkcjonowaniu biologicznego systemu nerwowego. Składają się one ze sztucznych neuronów. Każdy z nich posiada parametry, które opisują jak jego aktywacje wpływają na wszystkie inne wyjścia w całej sieci. W procesie uczenia, czyli propagacji wstecznej, każdy parametr neuronu jest zmieniany w celu zminimalizowania błędu między oczekiwanym a rzeczywistym wynikiem [19]. Sieć nazywana jest głęboką jeśli zawiera wiele połączonych ze sobą warstw. W rzeczywistości wyróżnia się konkretne architektury w zależności od dziedziny problemu, są to między innymi sieci rekurencyjne i konwolucyjne. W pierwszej z nich odpowiednia struktura sprawia, że sygnał wyjściowy trafia ponownie na wejście sieci generując ciąg nowych próbek. Drugi rodzaj skupia się na matematycznym procesie konwolucji macierzy do przetwarzania i ekstrakcji cech.

Często spotyka się w parze z innym algorytmem sieci neuronowe, ponieważ są w stanie poprawić zdecydowanie celność predykcji niskim kosztem [10]. Istnieją pewne rozwiązania, gdzie sieci neuronowe są głównym sposobem na przeprowadzanie predykcji. Ich szerokie zastosowanie i wiele istniejących rozwiązań sprawia, że są w stanie występować na prawie każdej platformie obliczeniowej. Dodatkowo w ciągu ostatniej dekady widać było zdecydowany rozrost zastosowań i nowych rozwiązań korzystających w pewien sposób z głębokich sieci neuronowych.

Różne istniejące rozwiązania i modele głębokich sieci neuronowych dają duże pole do ich zastosowań. W przypadku pracy autorstwa S. Umamageswari [28] zastosowano jedno z popularnych w ostatnim czasie rozwiązań *LSTM* (*long short-term memory*) będące rozszerzeniem warstw typu rekurencyjnego. Są one w stanie modelować pewne zależności w danych, występujące z dużym odstępem czasowym. Jest to dobre zastosowanie dla sieci

wykorzystujących dużą ilość danych wejściowych zawierające informacje temporalne w odpowiedniej postaci.

Ciekawym podejściem okazuje się wykorzystanie konwolucyjnych sieci neuronowych w przypadku wprowadzania dodatkowych zmiennych do systemu [12]. W przytoczonej pracy system w celu określenia następnej akcji którą ma podjąć bierze pod uwagę wyraz twarzy użytkownika. Obraz twarzy, przekształcany jest przez sieci konwolucyjne do zestawu informacji mówiącym o samopoczuciu użytkownika.

2.1.4. Podsumowanie i wnioski

Na podstawie przeprowadzonej analizy można zauważyć, że wstępne przetwarzanie danych i odpowiednie ich spreparowanie jest kluczowym aspektem w realizacji procesu. Prawidłowa reprezentacja stanu w systemie jest kluczowa, ponieważ to na podstawie tych wartości system będzie przeprowadzał swoje predykcje. W przypadku tradycyjnych algorytmów uczenia maszynowego zauważyć można ich zdecydowane ograniczenia, których rozwiązanie wymaga zdecydowanego komplikowania problemu poprzez wprowadzanie kwantyzowanych informacji czy zmiennych reprezentujących wartości temporalne. Metodą dającą największą elastyczność okazuje się wykorzystanie głębokich sieci neuronowych, ze względu na ich możliwości, przy wcześniejszym odpowiednim przekształceniu wejścia lub wyjścia, wprowadzeniu danych o różnych formatach czy sterowaniu szerokim zakresem różnych urządzeń.

2.2. URZĄDZENIA IOT

Internet jest zbiorem technologii który bardzo zmienił sposób funkcjonowania ludzi. Wraz ze spadającym kosztem podłączenia do Internetu, znajdujemy coraz to nowsze zastosowania dla właśnie takich systemów. Internet Rzeczy (*Internet-of-Things*) to zbiór urządzeń wyposażonych w czujniki oraz odpowiednie oprogramowanie umożliwiające im zbieranie, przesył, interpretację danych oraz reagowanie. Koncepcja od momentu narodzin, na przełomie XX i XXI, została nazwana najbardziej wpływową technologią aktualnej epoki. Ludzkość jest aktualne świadkiem rewolucji w której urządzenia stale *on-line* przenikają do naszego codziennego życia, gdzie poprawiają jakość i komfort ludzkiego życia [24]. Plastyczność rozwiązań systemów IoT gwarantuje im obecność w różnych obszarach, takich jak zdrowie, przemysł, transport czy rolnictwo. Jednym z zastosowań, które cieszy się obecnie dużą popularyzacją wśród urządzeń IoT jest rozwiązanie *smart-home*, ze względu na ich możliwości. W środowisku domowym pełnią rolę różnych czujników i siłowników, które zapewniają reszcie systemu informacji w celu przeprowadzania automatyki. Każde urządzenie, niezależnie od środowiska, składa się z dwóch nierozłącznych elementów. Są to osobno „rzecz” i osobno „internet”.

2.2.1. *Rzecz*

Kontekst rzeczy w domenie Internetu Rzeczy jest bardzo rozmyty, ze względu na swoją logiczną strukturę jak i fizyczne zastosowanie, co dalej tworzy podział na *smart* i nie-*smart* urządzenia. Literatura wyróżnia trzy elementy które definiują typ urządzenia. Są to: świadomość kontekstu, autonomia i łączność [26]. Mowa tutaj o bardzo różnych systemach, które mogą agregować w jedność wiele czujników, wiele siłowników, procesorów i sposobów komunikacji. Do pierwszej z tych grup zaliczają się takie obiekty które, poza zbieraniem danych, są w stanie reagować na ich zmianę. Są to na przykład *smart* termostaty, które na podstawie temperatury zadanej i mierzonej decydują o odpowiedniej zmianie w swoim systemie. Mierzona temperatura może pochodzić wprost z urządzenia lub też z innych urządzeń znajdujących się w środowisku. Do drugiej grupy, kontynuując poprzedni przykład, zaliczyć można urządzenia-satelity, które mają dwa główne zadania. Zadaniem pierwszego typu jest zagregowanie danych pochodzących ze środowiska i przesłanie do jednostki zarządczej. W drugim typie zadań, przyrządy reagują na polecenia pochodzące z systemu i sterują mechanizmami w celu osiągnięcia ogrzewania lub chłodzenia, dodatkowo zgłaszając informację o powodzeniu akcji.

2.2.2. *Internet*

Internet w kontekście inteligentnych urządzeń to zbiór technologii i protokołów używanych przez każdy system w celu komunikacji. Bardzo ważnym aspektem w roli inteligentnych mechanizmów jest wymiana informacji, a bez tego byłoby trudno powiedzieć o rzeczywiście mądrym rozwiązaniu. Ważnym aspektem w przesyle danych, jest struktura każdej wiadomości. Systemy IoT będące rozwinięciem istniejących technologii korzystają z gotowych rozwiązań do celów komunikacyjnych. Pozwala im to na użycie istniejącej już infrastruktury do porozumiewania się miedzy sobą [1], a w przypadku braku takowej, za pomocą odpowiednich standardów spontanicznych [21], [7]. Takie użycie protokołów pozwala systemom na integrację w prawie każdym środowisku oraz komunikację z każdym innym podłączonym do sieci obiektem.

3. WYMAGANIA PROJEKTOWE

Celem tego projektu jest wytworzenie systemu zdolnego do wspomagania użytkownika w eksploatacji inteligentnych urządzeń IoT znajdujących się w jego domowym środowisku poprzez wykonywanie pewnych czynności za użytkownika.

Automatyzacja nie powinna wykonywać się wprost, tj. bez ludzkiej interakcji. Użytkownik korzystający z tego modułu będzie spełniał kluczową rolę, ponieważ wymaga się, aby system dokańczał powtarzalne ciągi zadań wykonywanych przez użytkownika.

Konieczne jest aby rozpoczęcie użytkowania tego systemu wymagało od użytkownika minimum konfiguracji. System powinien pobierać dane z istniejącego już programu nadzorczego automatyki domowej i używać ich w celu wytworzenia reguł. Dodatkowo, sam powinien być niezależny od architektury komputera na którym się znajduje. Zaleca się, aby moduł współpracował z gotowymi systemami automatyki domowej. W wypadku utraty przez komputer zasilania, system powinien wznowić swoją pracę bez ponownego procesu tworzenia reguł decyzyjnych.

Konieczne jest, aby implementacja nie wymagała dużej ilości prac w celu dodania nowych typów źródeł i ujść danych, tak aby ewentualne dodawanie obsługi urządzeń dla których nie istnieje gotowe wsparcie było najprostsze. System powinien móc wykorzystywać dodatkowe źródła danych w celach predykcji.

W wypadku innej odpowiedzi systemu niż użytkownik oczekuje, użytkownik powinien mieć łatwy sposób na cofnięcie wykonania błędnej akcji do poprzedniego stanu.

Biorąc powyższe pod uwagę, można określić listę wymagań funkcjonalnych i niefunkcjonalnych tego systemu. Wymagania funkcjonalne:

— wspomaganie codziennej eksploatacji urządzeń w domu,
— cofanie akcji,
— odczyt oraz zapis do pamięci stałej,
— wsparcie istniejącego systemu automatyki,
— dodawanie innych źródeł danych.

Wymagania niefunkcjonalne:

— minimum konfiguracji,
— działanie niezależne od architektury komputera,
— łatwość w tworzeniu rozszerzeń.

4. ARCHITEKTURA

4.1. ARCHITEKTURA ŚRODOWISKA I NARZĘDZIA

Na tym stadium pracy bardzo ważne jest dokładne zbadanie wymogów znajdujących się w rozdziale (3) w celu odpowiedniego przygotowania architektury systemu. Błędny wybór może skutkować ograniczeniami co do zastosowań tego systemu.

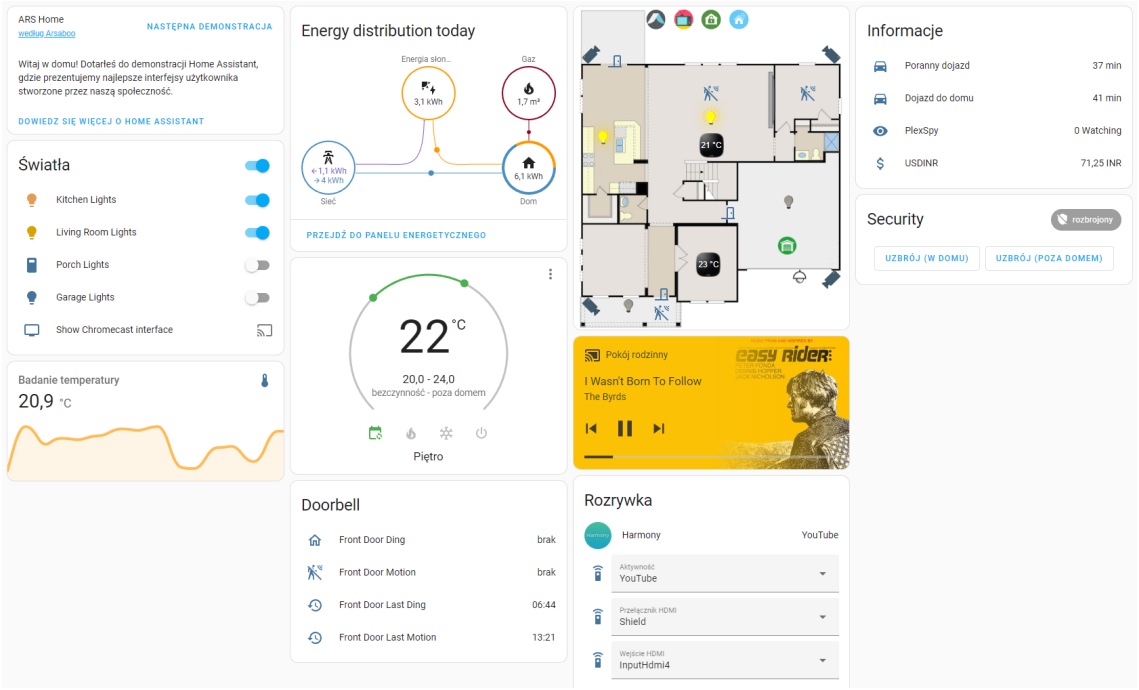
4.1.1. HomeAssistant – HA

Jako system automatyki domowej, kontrolujący wymianę informacji naszego modułu ze światem rzeczywistym, wybrano platformę *HomeAssistant*. Jest to jeden z największych niekomercyjnych systemów tego typu. Jego przewagą w porównaniu do innych dostępnych na rynku systemów są rzesze fanów i majsterkowiczów, którzy oferują świetne wsparcie i pomagają w rozwiązywaniu problemów. HA skupia się także na prywatności. Programiści *open-source*, jak i użytkownicy systemu, zalecają utrzymywanie go na swojej własnej infrastrukturze w domu. Dzięki wiernym fanom i samej architekturze projektu system posiada bardzo bogatą bibliotekę integracji z różnymi systemami, od systemów typu *Google Assistant* w celu dodawania funkcjonalności sterowania głosem, przez własnościowe systemy zarządzania oświetleniem w domu, aż po wsparcie dodatkowych bezprzewodowych protokołów komunikacyjnych przeznaczonych do zastosowań domowych. Bardzo ważnym atutem poza szerokim polem różnych integracji jest także system rozszerzeń, gdzie użytkownik może dodać pewną kompletnie nieistniejącą w systemie funkcjonalność do poprawy działania systemu, czy automatyzacji innych rzeczy niezwiązanych z domem.

Na ilustracji (4.1) znajduje się dostępne na stronie internetowej demo interfejsu mające na celu pokazanie użytkownikom możliwości całego systemu, a także sposoby komponowania przykładowego panelu zarządzania domem.

4.1.2. AppDaemon – AD

System pozwalający stworzenie modułu samouczącego powinien dawać maksimum możliwości i niezależności, zatem wybrano *AppDaemon*. Jest to środowisko wykonawcze języka programowania Python, w wielowątkowej architekturze piaskownicy, służące do pisania aplikacji automatyzacji dla projektów automatyki domowej (i nie tylko), których wymogiem jest solidna architektura sterowana zdarzeniami. AD zaraz po uruchomieniu jest gotowe do współpracy z systemem *HomeAssistant*, co sprawia, że integracja systemów HA/AD jest bezproblemowa. System pozwala natychmiastowo reagować na zmiany stanów



Rys. 4.1. Przykładowy panel sterowania domem dostarczony przez HomeAssistant

urządzeń znajdujących się w domowym środowisku, poprzez korzystanie z asynchronicznej architektury callback.

4.1.3. Python i Tensorflow – tf

Wykorzystanie struktury HA/AD ogranicza projekt do wyboru jezyka Python jako przewodniego języka programowania w tym przedsięwzięciu. Taki wybór nie jest jednak ograniczeniem w tym projekcie, ponieważ posiada on bardzo wielką rzeszę fanów tworzących i udoskonalających paczki kodu dodające nowe możliwości w celu powiększenia pola zastosowań tego języka.

Konkretnym zastosowaniem, który w ostatnim czasie budzi wiele uwagi, jest użycie języka Python do celów uczenia maszynowego. Jedną z bibliotek dostarczających rozwiązania związane z tworzeniem, „uczeniem" i eksploatacją modeli różnego rodzaju sieci neuronowych jest paczka *Tensorflow*. Tf implementuje wiele gotowych modeli i algorytmów przyspieszających tworzenie modeli sieci neuronowych. Dodatkowo, w celu przyspieszenia procesu liczenia błędu propagacji, biblioteka korzysta z różnego rodzaju akceleratorów obliczeń, w tym kart graficznych.

4.1.4. Docker

Ze względu na wymóg niezależności od platformy na jakiej będzie znajdować się ten system zdecydowano o wyborze, jako jednej z głównych technologii, systemu *Docker* w celu zapewnienia aplikacjom pracującym w jego środowisku odpowiednich warunków niezależnie od systemu operacyjnego na jakim się znajdują. Dodatkowym powodem, który

sprawił że wybrano tę technologię jest gotowe wsparcie systemów HA i AD do pracy w konteneryzowanym środowisku bez dużej ilości konfiguracji. Gotowe obrazy aplikacji znajdują się w sieci, a ich instalacja pod warunkiem posiadania środowiska *Docker* jest bardzo prosta i szybka.

Dodatkowo narzędzia które dostarcza *Docker*, pozwalają na tworzenie własnych kontenerów i udostępnianie ich innym co sprawi, że nie będzie to rozwiązanie przygotowywane specjalnie pod konkretne środowisko automatyki.

4.1.5. Pozostałe narzędzia i biblioteki

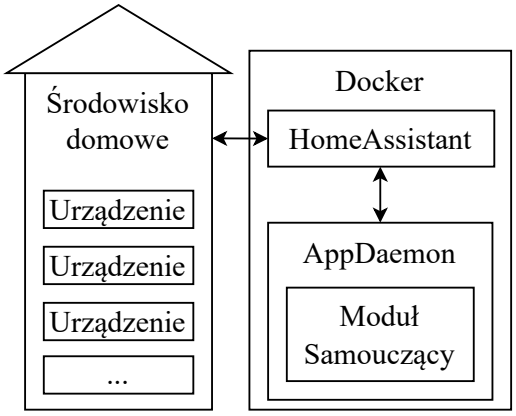
W celu zarządzania środowiskiem programistycznym wykorzystano platformę *Pipenv*, która to ułatwia zarządzanie wirtualnymi środowiskami języka Python oraz instalację wszystkich bibliotek potrzebnych w danej bazie kodu. Aby wytworzyć gotową paczkę kodu która zostanie zainstalowana w AD wykorzystano narzędzie *setuptools*. Do wszelakich numerycznych obliczeń oraz interfejsu z *Tensorflow* została wykorzystana biblioteka *numpy*. Automatyzowanie środowiska programistycznego zostało wykonane za pośrednictwem *Makefile*.

W celu poprawy jakości możliwego w przyszłości rozszerzenia projektu, zdecydowano o skorzystaniu z kilku programistycznych udogodnień. Cała praca podczas tworzenia będzie zapisywana w systemie kontroli wersji *git*, gdzie będzie można cofać lub wprowadzać zmiany w kodzie na różnych gałęziach niezależnie od siebie. Dodatkowo skorzystano z narzędzia *pre-commit*, dzięki któremu przed każdym zapisem konkretnej wersji modułu do systemu kontroli wersji na kodzie będzie przeprowadzana statyczna analiza oraz pewne porządki mające na celu poprawę jakości pracy nad kodem.

4.1.6. Podsumowanie

Na rysunku (4.2) znajduje się zaproponowana architektura wysokopoziomowa środowiska. Urządzenia automatyki znajdujące się w domowym środowisku będą komunikowały się z *HomeAssistant* w celu aktualizowania swojego stanu, ten będzie przesyłany dalej do *AppDaemon*, gdzie będzie znajdował się moduł obsługujący wybrane zdarzenia. Korzystając z informacji o zdarzeniu, system będzie przewidywał następną akcję i wykonywał pozostałą przewidzianą w danym epizodzie zmianę stanu. Informację o poleceniu zmiany stanu będzie obsługiwał AD, który to dalej bedzie przesyłał tę informację do HA, który zdecyduje jak z danym urządzeniem się skomunikować i jaką wiadomość wysłać.

Wewnątrz bloku z modułem samouczącym znajdzie się kod napisany w języku Python, który będzie reagował na zmiany stanu i w odpowiedni sposób go preparował do przekazania modelowi sieci neuronowych, a następnie przekształcany do formy umożliwiającej sterowanie urządzeniami w domu.



Rys. 4.2. Proponowane środowisko rozwiązania problemu.

4.2. ARCHITEKTURA MODUŁU

Ze względu na wymóg obsługi różnych typów urządzeń oraz źródeł danych, bardzo ważnym aspektem do zaprojektowania jest struktura systemu tłumaczącego dane na takie, które zrozumie biblioteka *Tensorflow*, zaprojektowanie samych modeli głębokich sieci neuronowych, ale także tłumaczenie wygenerowanej przez model odpowiedzi do takiej, którą zinterpretuje reszta modułu.

4.2.1. Dane wejściowe i wyjściowe

System zarządzania automatyką domową podczas swojej pracy zbiera informacje na temat zmian stanów urządzeń którymi steruje i zapisuje je w lokalnej bazie danych. Nie jest ona jednak dostępna w łatwy sposób dla programisty. *AppDaemon* w swoim interfejsie programistycznym udostępnia możliwość pobrania historii zmian stanu wybranego urządzenia z ostatniego czasu. Dane przekazane z zapytania do aplikacji są w postaci listy słowników języka Python o strukturze zawartej w listingu (4.1) z której usunięto wszystkie nieważne dla modułu pola. Słownik to pewna struktura danych w której informacja jest zawarta w wartościach dla konkretnego klucza [14].

Warto zauważyć, że informacja zawarta w polu `state`, ma inne znaczenie w zależności od typu urządzenia czy źródła. W przypadku urządzenia typu przerzutnik stabilny o binarnych pozycjach, dane w kluczach `state` oraz `last_changed` wskazuje na czas, kiedy zaszła zmiana do jakiego stanu, a w przepadu zwykłego przycisku wystarcza samo pole ze stanem, ponieważ znajduje się tam czas ostatniej zmiany. Dodatkowo, wartości pola stanu nie ograniczają się do prostych wartości. System *HomeAssistant* dodaje kilka dodatkowych wirtualnych urządzeń, które zmieniają swój stan raz dziennie, a ich wartość stanu przy każdej zmianie przyjmuje czas, np. następnego wschodu czy zachodu słońca.

W celu zaspokojenia wymagania integracji z wieloma urządzeniami system powinien na podstawie dostarczonej mu informacji o typie urządzenia tłumaczyć ciąg historycznych zmian na dane treningowe do modelu sieci. Aby rozszerzenie tego rozwiązania było możliwe

```
[
    {
        "entity_id": "input_boolean.test_switch_1",
        "state": "off",
        "last_changed": "2023-09-11T13:07:52.856406+00:00",
        "last_updated": "2023-09-11T13:07:52.856406+00:00"
    },
    ...
]
```

Listing 4.1: Historyczne informacje na temat stanu urządzenia pochodzące z systemu *AppDaemon*.

przez użytkownika proponuje się, aby system korzystał z klas abstrakcyjnych. Pozwoli to na stworzenie bazowego obiektu i określeniu funkcji, jakie powinno obsługiwać dane urządzenie w celu współpracy z modułem [16]. Dodatkowo wymusi to na użytkowniku tworzącym dodatkowe elementy stworzenie wszystkich wymaganych funkcji, a nie jej pewnej części, co pomoże w redukcji błędów programistycznych.

W celu odpowiedniego przygotowania danych wyjściowych pochodzących z sieci neuronowej można skorzystać z zaproponowanego wyżej kodu obsługującego dane, wejściowe poprzez stworzenie dodatkowych funkcji obsługujących przemiane danych w odwrotnym kierunku.

Proponowana struktura danych na której moduł będzie się opierał to zestaw dwóch macierzy (4.1). Pierwsza zawierająca informacje o aktualnym stanie systemu składająca się z u informacji wejściowych nazywana macierzą stanu, druga składająca się z w skalarów opisująca zmiany w stanach nazywana macierzą przejścia.

$$\begin{aligned} \mathbf{S}_t &= [s_1, s_2, \ldots, s_u] \\ \mathbf{Z}_t &= [z_1, z_2, \ldots, z_w] \end{aligned} \tag{4.1}$$

Pierwsza z nich opisuje stan środowiska domowego w danym momencie t, druga z nich wskazuje na zmianę stanu urządzeń do następnego momentu w czasie $t + 1$. Jesteśmy w stanie określić pewną funkcję G w zależności od typów urządzeń i źródeł, która na podstawie aktualnego stanu oraz zmiany w danym momencie, generuje nam następną macierz $\mathbf{S}_{t+1}$ (4.2).

$$G\left(\mathbf{S}_t, \mathbf{Z}_t\right) = \mathbf{S}_{t+1} \tag{4.2}$$

$$\mathbf{Z}_k = P(\mathbf{S}_k) \tag{4.3}$$

Tabela 4.1. Tabela zawierająca listowanie warstw w pojedynczej sieci neuronowej

Warstwa	Aktywacja	Ilość neuronów
0	`linear`	Zależna od ilości urządzeń
1	`ReLU`	128
2	`ReLU`	64
3	`tanh`	1

Biorąc powyższe pod uwagę, proponowane rozwiązanie powinno na podstawie macierzy stanu $\mathbf{S}$ w momencie k przewidywać następny zestaw ruchów użytkownika, czyli macierz przejścia $\mathbf{Z}$ dla tego samego momentu w czasie (4.3).

4.2.2. Architektura głębokich sieci neuronowych

Sercem pracy całego modułu są modele matematyczne sieci neuronowych. Ze względu na wymóg minimalnej konfiguracji ich logiczna konfiguracja powinna być wybrana tak, aby były w stanie nauczyć się epizodów akcji użytkownika bez zbędnej wielkości i skomplikowania. Dodatkowym powodem ograniczenia skomplikowania takich sieci jest zdecydowane wydłużenie procesu uczenia sieci neuronowych dla tych, które zawierają dużo wag [25].

Model sieci neuronowej zawarty w takim module będzie pracował z różnymi typami urządzeń wejściowych, gdzie dane będą sformatowane w różny sposób. Dodatkowo dane wyjściowe będą musiały być interpretowane przez moduł inaczej, w zależności od celu. Tworzenie dużych i skomplikowanych sieci nie zawsze sprawia, że jest ona w stanie lepiej nauczyć się przekazywanych jej danych ze względu na klątwę wymiarowości [13].

Biorąc powyższe pod uwagę, proponuje się, aby moduł podczas tworzenia początkowych modeli nie tworzył jednej monolitycznej sieci odpowiedzialnej za wszystkie wejścia i wyjścia, a tworzył tyle sieci, ile jest urządzeń do sterowania. Umożliwi to sieciom lepsze dostosowanie sie do urządzeń i natury ich używania, kosztem nieznacznie dłuższego czasu uczenia.

Proponowana struktura takich sieci znajduje się w tabeli (4.1). Pierwsza warstwa w tym modelu będzie zależna od ilości urządzeń i źródeł dodatkowych danych, a ostatnia warstwa będzie zawierała jedno wyjście. Taka struktura będzie agregowała wszystkie wyjścia do jednej formy dostarczanej reszcie komponentów.

4.2.3. Eksport i import sieci

Aby móc zapisać modele sieci neuronowych wszystkie części systemu mające z nimi styczność muszą wspierać takie działanie. W szczególności dwoma krytycznymi elementami w przypadku tego modułu samouczącego jest biblioteka *Tensorflow* oraz tworzony przez nas system konwersji danych. Oba te elementy powinny współpracować ze sobą na

tyle dynamicznie, aby różne konfiguracje modeli oraz danych wejściowych były poprawnie obsługiwane.

Biblioteka *Tensorflow* dostarcza swoje metody na zapis i odczyt modeli do pamięci stałej, ale są one ograniczone z kilku powodów. Jednym z nich jest brak możliwości zapisywania struktury i wag do własnego bufora w pamięci, czy swojego deskryptora pliku. Sprawia to, że każda osobna zapisana sieć będzie tworzyła jeden osobny plik na dysku, dodatkowo bez możliwości dołączenia własnych danych. Operowanie na dużej ilości urządzeń obsługiwanych przez moduł będzie generowało jeszcze więcej plików, które mogą zostać uporządkowane do postaci jednego, czytanego raz w fazie rozruchu systemu.

W celu eksportu i importu sieci zostanie zaproponowany autorski sposób serializowania i deserializowania danych [22] w którym wszystkie definicje oraz równie ważne – wagi wyuczonych sieci neuronowych – zostaną zapisane w odpowiednim formacie do pliku archiwum `.zip`. Zapisanie wszystkich potrzebnych informacji do odtworzenia całej struktury w jednym pliku ułatwi przenoszenie użytkownikowi systemu między komputerami, ale także zniechęci go od ewentualnego ręcznego zmieniania wartości wag w systemie poprzez zaciemnienie widocznej struktury.

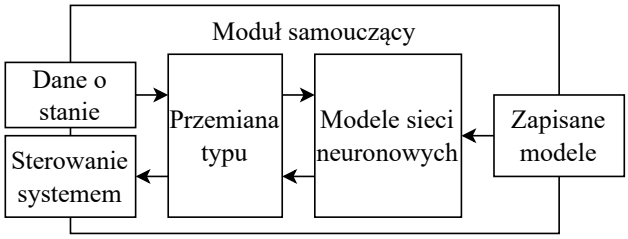
Proponowana architektura plików wewnątrz archiwum dla każdej z sieci osobno będzie składała się z:

1. Surowej deklaracji modelu, w której znajdzie się dokładny opis kształtu i wymiarów każdej z warstw potrzebnej do dokładnego jej odtworzenia.
2. Pliku z zapisanymi wagami każdej warstwy w sieci.
3. Pliku z informacjami pośrednimi, które nie są krytyczne do działania systemu.

Dodatkowo, w celu określenia czy nie doszło do zmiany dostępnych dla systemu urządzeń oraz w celu zapisania innych informacji ogólnych o wszystkich dostępnych w modelu sieciach, będzie potrzebny jeden dodatkowy plik znajdujący się katalogu głównym archiwum.

4.2.4. Podsumowanie

Na rysunku (4.3) znajduje się proponowana struktura całego modułu samouczącego. Dane o stanie i zmianach będą przechodziły strumieniem w głąb systemu. Dalej, po obliczonej na podstawie stanu systemu predykcji, system będzie sterował urządzeniami w środowisku domowym.



Rys. 4.3. Proponowana struktura działania modułu.

5. IMPLEMENTACJA

5.1. DANE WEJŚCIOWE

Krok przemiany danych wejściowych do wykorzystania w module został podzielony na dwa osobne etapy. Jeden z tych etapów, nazywany krokiem interpretacji, różni się w zależności od źródła skąd pochodzą dane. W zależności od tego, czy system korzysta z danych historycznych w procesie uczenia, czy korzysta z danych aktualnych w celu predykcji, inaczej przechodzą one proces interpretacji. Po etapie interpretacji jest krok transformacji i wygląda on tak samo dla każdego źródła niezależnie od typu.

5.1.1. Interpretacja historyczna

W przypadku danych ze źródła historycznego, system musi rozpoznawać epizody działań domowników w celu ich nauki. W tym celu, aby wygenerować zbiór danych uczących, algorytm przegląda posortowaną listę wydarzeń w czasie, celem określenia epizodów akcji. Począwszy od pierwszego wydarzenia, w dostępnej dla systemu historii, przegląda się ją w poszukiwaniu ciągu akcji o łącznym czasie nie większym, niż pewien z góry określony parametr nazwany `EPISODE_DELTA` od momentu wystąpienia pierwszej akcji w epizodzie. Ta wartość została wprowadzona w celu sprawdzenia, czy zmiana stanu w epizodzie się nie przedawniła. W przypadku, gdy jedno urządzenie zmienia swój stan kilkukrotnie w przeciągu jednego epizodu, brana jest tylko pod uwagę policzona zmiana i stan względem poprzedniego, przed liczonym epizodem. W tabeli (5.1) znajduje się znaleziony dla przykładowej historii epizod, a kolorem został zaznaczony znaleziony przez algorytm epizod akcji.

Warto zauważyć, że wybrana długość tego parametru będzie mocno wpływała na liczność i wielkość epizodów. Wybranie zbyt krótkiego czasu epizodu będzie rozdzielać powiązane ze sobą czynności, ale będzie rozróżniała zmiany stanu tego samego urządzenia w czasie, a wybranie zbyt długiego czasu będzie łączyło kilka niezależnych akcji ze sobą, możliwie je ze sobą niwelując. Sprawia to, że wybranie optymalnej długości epizodu jest bardzo ważne, aby system mógł dostosować się do użytkownika.

Do określenia działania, dla każdego innego typu urządzeń z osobna, skorzystano z abstrakcyjnej klasy `DeviceHistoryGeneric`, opisującej strukturę funkcji jakie powinna dana klasa implementować w celu poprawnego działania. Abstrakcyjna funkcja na podstawie stanu urządzenia w momencie t, stanu urządzenia w momencie k oraz czasu, do którego dany epizod powinien się skończyć, zwraca w postaci rzeczywistej liczby stan oraz przejście

Tabela 5.1. Tabela przedstawiająca działanie algorytmu szukania epizodów.

Urządzenie	Stan	Czas
...	...	...
światło kuchnia	on	16:52:48
światło kuchnia	off	17:32:12
klimatyzacja	17	17:33:00
światło salon	on	17:33:10
telewizor salon	on	17:33:15
światło balkon	on	19:32:42
...	...	...

stanu dla danego epizodu w czasie. Pseudokod takiej funkcji został zawarty w listingu (5.1). Warto zauważyć, że gdy nie ma poprzedniego stanu, tj. historia nie sięga na tyle wstecz, to kod musi obsługiwać wykrywanie obu wartości. W przypadku interpretacji prostych urządzeń nie jest to problemem, ponieważ jesteśmy w stanie wydedukować przejście stanu i aktualny stan na podstawie informacji pochodzącej z systemu. W przypadku gdy tej możliwości nie ma, pojedynczy błędny wynik powinien być zdecydowaną mniejszością po interpretacji reszty historii.

```
def get_past_state(aktualny, poprzedni, do_momentu):
    wartość_stanu = 1.0 jeśli aktualny = "on" wpw 0.0
    wartość_przejścia = 0.0
    # Czy mamy historię na temat poprzedniego?
    jeśli poprzedni nie istnieje:
        # zakładamy, że zmiana stanu odbyła się dawno w historii
        return wartość_stanu, wartość_przejścia

    # Czy zmiana się nie przedawniła?
    jeśli poprzedni.czas_zmiany < do_momentu:
        wartość_przejścia = 1.0 jeśli aktualny = "on" wpw -1.0

    return wartość_stanu, wartość_przejścia
```

Listing 5.1: Pseudokod funkcji interpretującej stany ze źródła historycznego dla typu urządzenia przełącznika stabilnego on/off.

5.1.2. Interpretacja bieżąca

Bieżąca interpretacja akcji użytkownika wygląda bardzo podobnie do analizy historycznej. Informacja o aktualnym stanie całego systemu przechowywana jest przez moduł w pamięci. Podczas pracy systemu, mamy pewność, że system *AppDaemon* zgłosi zmianę stanu tylko jednego urządzenia za każdym uruchomieniem asynchronicznej funkcji. Wiemy zatem, że musimy tylko obsłużyć zmianę stanu jednego urządzenia i zapisać jego stan do

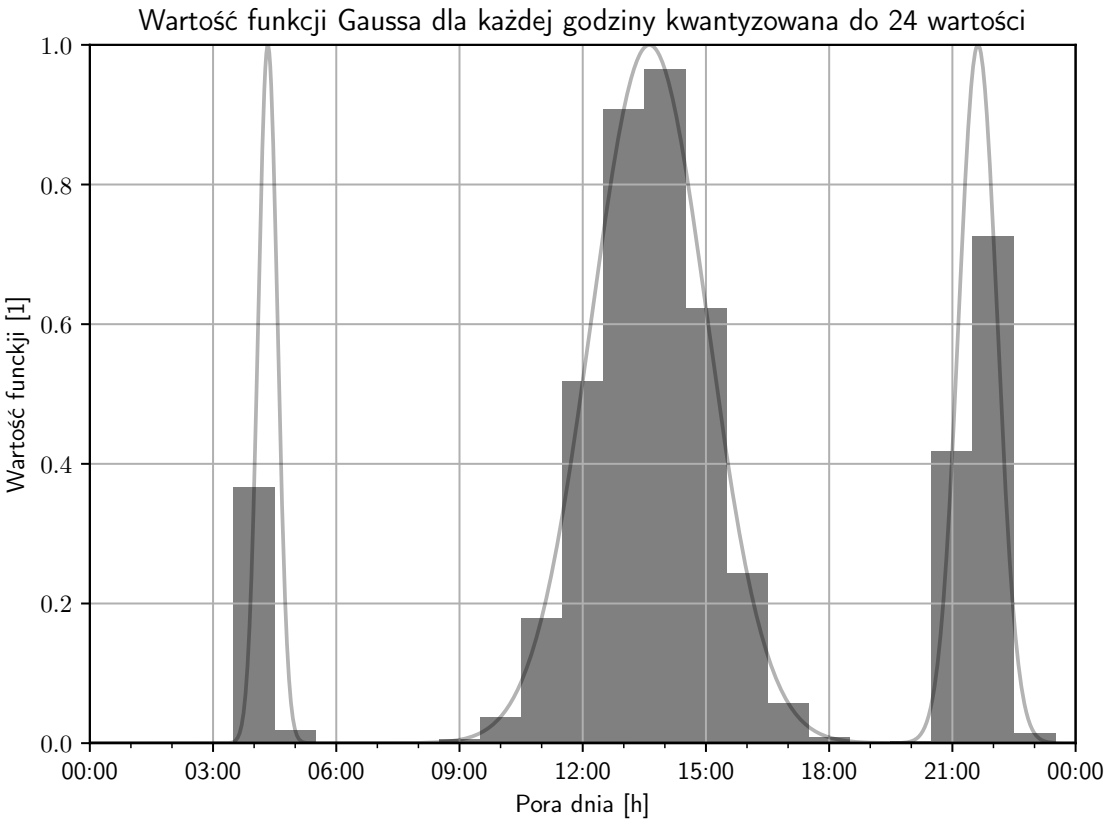
pamięci. Informacja o tym, jakie urządzenie zmieniło się do jakiego stanu będzie potrzebna w procesie predykcji, które będzie opisane w późniejszym etapie. Korzystając z tej samej klasy dla typu urządzenia co w przypadku interpretacji historycznej, w podobny sposób określamy stan i przejście stanu urządzenia. W tym wypadku ze względu na pewność, że dane urządzenie zmieniło swój stan dokładnie w momencie uruchomienia danej funkcji, obliczanie stanu i przejścia jest zdecydowanie prostsze, ponieważ nie trzeba rozważać przedawnienia się zmiany.

5.1.3. Transformacja

Po prawidłowej ekstrakcji zmian stanu, przed przekazaniem ich do sieci neuronowych, dane są dodatkowo przekształcane. W przypadku prostych urządzeń, typu przełącznik stabilny, ten proces nie wprowadza do danych żadnej zmiany. W przypadku bardziej skomplikowanych typów danych, takich jak zmienne temporalne wskazujące na np. dzień tygodnia czy porę dnia, przeprowadzane są pewne dodatkowe operacje rozkładające jedną konkretną informację na kilka różnych wartości, które lepiej będzie sieciom neuronowym powiązać z akcjami. W celu możliwości wprowadzenia rozszerzalności ponownie wykorzystano podejście obiektowe i skorzystano z klasy abstrakcyjnej. Zaproponowana klasa abstrakcyjna `Convertable` opisuje dwie funkcje, które dany konwerter danych musi zaimplementować. Jedna z nich opisuje przejście danych do wersji akceptowalnej przez sieci neuronowe, druga tłumaczy dane wygenerowane przez sieć neuronową do formy obsługiwanej przez resztę systemu. Jednym specjalnym przypadkiem takiego konwertowania danych, gdzie potrzebna jest zaimplementowana jedna z obu funkcji, jest przekazywanie zmiennych temporalnych do sieci.

W celu przekazania do sieci jak największej ilości rzeczywiście użytecznych informacji o porze dnia tak, aby była ona w stanie w swojej strukturze zapisać nawyki użytkownika, informacja o czasie jest podzielona. Podział jednej zmiennej czasowej odbywa się poprzez podzielenie jej na 24 różne informacje, gdzie każda z nich wskazuje na pewną wartość zależną od odległości indeksu danej zmiennej od godziny wydarzenia epizodu. Dokładniej, jest to wartość funkcji Gaussa dla wartości zależnej od indeksu, z parametrem μ ustawionym na moment w ciągu dnia podczas którego wydarzył się ten epizod.

Ważnym parametrem w przypadku funkcji Gaussa, poza parametrem μ, jest parametr opisujący szerokość dzwona, w zastosowaniach statystycznych nazywany odchyleniem standardowym σ. W przypadku zastosowania obliczania wartości funkcji na podstawie czasu szerokość mówi nam o tym, jak bardzo akcje występujące o konkretnej porze dnia mogą być proponowane o innych podobnych godzinach. Ustawienie tego parametru zbyt wąsko spowoduje, że system będzie rozpoznawał wykonywanie konkretnych akcji o bardzo szczegółowych godzinach w ciągu dnia. Ustawienie go natomiast za szeroko będzie proponowało wykonywanie akcji nieadekwatnych do konkretnej pory dnia. Na obrazie (5.1) znajduje się przykładowy zarys wartości funkcji dla kilku konkretnych pór dnia, razem



Rys. 5.1. Policzone wartości funkcji Gaussa wskazującej czas dla godziny 4:20:02, 13:37:21, 21:37:21.

z zaznaczonymi dokładnymi przebiegami krzywizny dzwonowej. Każda z zaznaczonych godzin posiada inny parametr σ w celu pokazania wpływu tego parametru na policzone wartości.

5.2. SIECI NEURONOWE

Ze względu na wymóg osobnego przetwarzania wielu źródeł różnego typu w systemie, gdzie część z nich może kolejno zostać jeszcze podzielona na dalsze drobniejsze informacje, całość systemu została umieszczona w obiekt będący menedżerem. Głównym zadaniem menedżera jest agregowanie do jednego obiektu wielu konwerterów i agentów. Agent w tym systemie to obiekt z pewnymi z góry określonymi funkcjami opakowujący modele matematyczne dostarczone przez bibliotekę *Tensorflow*, wykorzystywany do schowania za warstwą abstrakcji i detali implementacji [23], [16]. Takie wykorzystania menedżerów i obiektów agregujących przydaje się w przypadku tworzenia systemów dynamicznych, polegających na wczytywaniu i zapisywaniu obiektów do pamięci stałej. W przypadku tego modułu samouczącego daje to osobie rozszerzającej możliwości wykorzystania innych struktur niż wcześniej określono (4.1), lub nawet wykorzystanie kompletnie innych metod predykcji w celu osiągnięcia danego celu.

5.2.1. Dynamiczne tworzenie

Bardzo wygodną możliwością udostępnianą przez bibliotekę *Tensorflow* dla modeli sieci neuronowych są nazwane wyjścia oraz wejścia dla sieci. Pozwala to na dynamiczne tworzenie i eksploatację modeli bez żadnego bardzo skomplikowanego systemu przetwarzającego nazwę wejść na globalną pozycję w macierzy wejściowej. System w celu stworzenia modelu sieci neuronowej wykorzystuje obiekt `Model` udostępniany przez bibliotekę wykonawczą *Tensorflow* – Keras. Do stworzenia modelu sieci neuronowej korzysta się z informacji o nazwach wejść oraz nazwach wyjść. System podczas tworzenia nowej sieci tworzy listę warstw typu `Input` i każdej z nich przypisuje nazwę kolejnego wejścia do sieci, następnie każda z tych warstw, o kształcie `(wielkość próbki, 1)`, jest do siebie dodawana wzdłuż drugiego wymiaru za pomocą warstwy typu `Concatenate`. Powstała warstwa ma kształt `(wielkość próbki, ilość wejść)`. W tym momencie kolejne warstwy dodaje się tak samo jak w przypadku zwykłego modelu funkcyjnego biblioteki *Tensorflow* zgodnie z listą warstw podaną we wcześniejszym rozdziale (4.1). Ostatnia warstwa, wyjściowa, tak jak w przypadku warstw wejściowych, generowana jest z listy nazw wyjść. Dynamiczne tworzenie wyjść, razem z modelem modularnej transformacji opisanej wcześniej, daje bardzo potężne narzędzie do sterowania domem.

5.2.2. Serializowanie i deserializowanie

Zapis oraz import danych w systemach komputerowych z formy istniejącej w pamięci do formatu, którą można zapisać na dysk jest bardzo rozległym zagadnieniem samym w sobie. W przypadku języka Python jest bardzo dużo sposobów wykonania tego zadania. Istnieje wbudowana biblioteka *Pickle*, która zapisuje wprost bit po bicie zawartość części pamięci programu do pliku na dysku, jednak zapisuje też bardzo dużo redundantnych informacji nad którymi użytkownik nie ma kontroli. Istnieją rozwiązania, które w swojej formie są czytelne dla ludzi, ale osiągają mniejszą efektywność zapisu danych, takie jak *XML* [9] czy *JSON* [11].

W wypadku zapisu i odczytu modeli matematycznych potrzebna jest obsługa trzech różnych informacji, które zostały wymienione wcześniej (4.2.2). Opis logicznej struktury sieci składa się z informacji o każdej z warstw. Każda z warstw zawiera informacje o jej wielkości, typie, rodzaju aktywacji, nazwach oraz informacji z jakimi warstwami sąsiaduje. Dostarczana przez *Tensorflow* metoda dla obiektu modelu zwraca wszystkie te informacje w postaci słownika zawierającego tekst, listy oraz kolejne warstwy słowników. Klasa odpowiedzialna za model implementuje konstruktor [16], który korzystając z informacji zawartych w opisanym słowniku odtwarza dokładną strukturę modelu. Sytuacja wygląda bardzo podobnie dla wag w modelu. Każda warstwa udostępnia funkcję zwracającą listę wszystkich ważnych dla danego typu warstwy parametrów, a druga, w obiekcie modelu, przyjmuje listę tych parametrów w celu odtworzenia wag.

W celu implementacji zapisywania i odczytu danych z dysku skorzystano z wbudowanej w Python biblioteki obsługującej pliki `.zip` oraz z metody serializacji *JSON*. Do obsługi procesu zapisu i odczytu stworzono klasę `ModelSerializer`, implementującą metody potrzebne do skorzystania z danego obiektu jako menedżera kontekstu języka Python [14]. Taki sposób implementacji zapewni, że wszystkie wymagane operacje na pliku i buforach w pamięci zostaną wykonane bez ingerencji użytkownika, co pozwoli na uniknięcie błędów. Stworzony obiekt dostarcza dwie główne metody. Jedna z nich służąca do odczytu i odtworzenia całej struktury menedżerów do takiej wykorzystanej przez moduł oraz drugiej, wykorzystywanej do zapisu całego menedżera do pamięci stałej, które działają analogicznie w odwrotnym kierunku.

Pewną emergentną zaletą takiego połączenia serializowania za pomocą *JSON* wewnątrz kompresowalnego archiwum plików jest zmniejszenie miejsca, które zajmują wszystkie modele na dysku o nawet 57%. Opis wytworzonego w ten sposób przykładowego archiwum znajduje się w listingu (5.2).

```
Zip file size: 698977 bytes, number of entries: 19
?rw-------  2.0 unx     8626 b- 23-Nov-22 10:47 kuchnia/declaration.json
?rw-------  2.0 unx   264887 b- 23-Nov-22 10:47 kuchnia/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 kuchnia/meta/info.json
?rw-------  2.0 unx     8640 b- 23-Nov-22 10:47 ekspres/declaration.json
?rw-------  2.0 unx   265319 b- 23-Nov-22 10:47 ekspres/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 ekspres/meta/info.json
?rw-------  2.0 unx     8646 b- 23-Nov-22 10:47 salon/declaration.json
?rw-------  2.0 unx   265341 b- 23-Nov-22 10:47 salon/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 salon/meta/info.json
?rw-------  2.0 unx     8648 b- 23-Nov-22 10:47 telewizor/declaration.json
?rw-------  2.0 unx   265556 b- 23-Nov-22 10:47 telewizor/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 telewizor/meta/info.json
?rw-------  2.0 unx     8656 b- 23-Nov-22 10:47 balkon/declaration.json
?rw-------  2.0 unx   265197 b- 23-Nov-22 10:47 balkon/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 balkon/meta/info.json
?rw-------  2.0 unx     8658 b- 23-Nov-22 10:47 sypialnia/declaration.json
?rw-------  2.0 unx   265469 b- 23-Nov-22 10:47 sypialnia/weights.json
?rw-------  2.0 unx       59 b- 23-Nov-22 10:47 sypialnia/meta/info.json
?rw-------  2.0 unx      273 b- 23-Nov-22 10:47 info.json
19 files, 1644270 bytes uncompressed, 696557 bytes compressed:  57.6%
```

Listing 5.2: Listowanie plików wewnątrz archiwum zawierajacego przykładowe modele sieci pochodzące z programu zipinfo.

5.3. STEROWANIE SYSTEMEM

Gdy dane o stanie z dołączonymi innymi informacjami temporalnymi przejdą przez cały system i jest zwracana predykcja, system musi podjąć decyzję czy to, co aktualnie użytkownik tych urządzeń wykonuje, jest zgodne z tym co system przewiduje. Jak wcześniej zostało opisane, system stara się przewidzieć następną zmianę stanu na podstawie przekazanych do modelu wartości o aktualnym stanie. Aby system był jak najbardziej wygodny dla użytkownika, zdecydowano na dopełnianie akcji użytkownika, zamiast wykonywanie zadań autonomicznie. Sprawia to, że proces wyboru czasu kiedy system powinien dokończyć daną akcję staje się bardzo skomplikowany.

W celu wykonywania tylko tych akcji, które są w danym momencie zgodne z zamiarem użytkownika, porównywane są predykcje systemu z aktualną zmianą na którą moduł reaguje. Dane wejściowe policzone z interpretacji bieżącej (5.1.2) porównywane są z przewidywanymi przez system. Jeśli zamiar użytkownika jest wspólny z tym, co system chce wykonać, system podejmuje decyzję o dokończeniu przewidzianego epizodu akcji. Porównanie zamiaru jest trywialne i sprawdza podobieństwo obliczonego aktualnego przejścia stanu dla danego urządzenia z tym, które powstało z predykcji. Korzystając z informacji o aktualnym stanie oraz przewidzianej zmiany generowane są zapytania do systemu, mające na celu sprowadzenie systemu do nowego stanu.

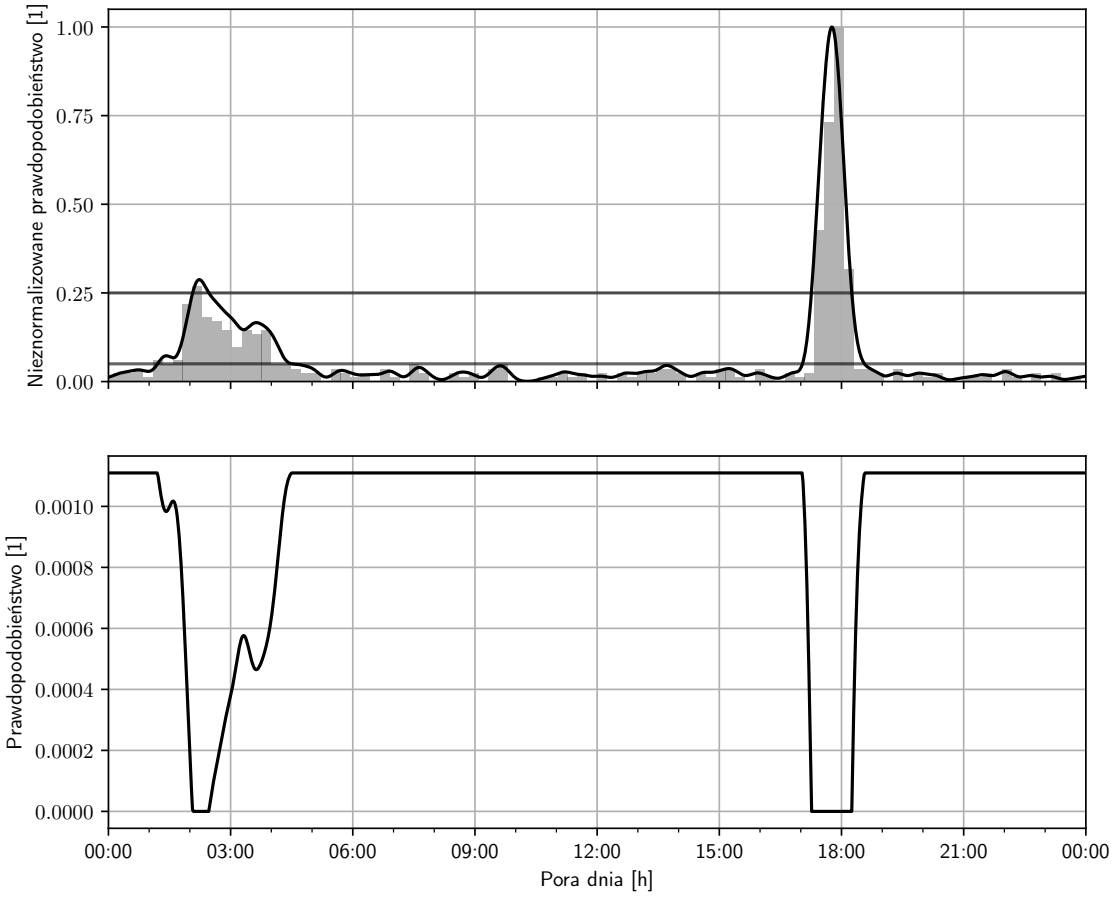
Do stworzenia funkcjonalności cofania niechcianych akcji wykorzystano istniejące w systemie rozwiązania. W momencie gdy system wykonuje za użytkownika dowolny epizod, zapisywany jest stan tuż przed wysłaniem do systemu polecenia o zmianie. Gdy użytkownik zdecyduje, że wykonana przez system zmiana nie odpowiada jego oczekiwaniom i poinformuje o tym moduł, to, korzystając z poprzedniego oraz aktualnego stanu, system wygeneruje zapytania sprowadzające system do stanu bez przewidzianej zmiany. Na podstawie wyliczonej docelowej zmiany oraz aktualnego stanu system wygeneruje zapytanie mające na celu usunięcie niewłaściwej predykcji.

5.4. FORMATOWANIE DANYCH UCZĄCYCH

Uczenie sieci neuronowych z danych pochodzących z historii HomeAssistant często samo w sobie jest niewystarczające. Sieci mimo tego, że uczone są na dużej ilości danych temporalnych, poza samymi stanami systemu, nie są w stanie nauczyć się konkretnych epizodów. W tym celu zaproponowano generowanie dodatkowych danych uczących na podstawie wygenerowanych informacji o przejściach, aby poprawić celność systemu.

Algorytm generowania danych tworzy dodatkowe informacje wejściowe zawierające brak zmiany stanu dla konkretnego stanu systemu o innej porze dnia, niż wtedy kiedy dany epizod jest wykonywany. Aby to umożliwić funkcja tworzy dwa słowniki, jeden dla którego wartościami są listy zawierające czas pewnego wydarzenia, druga dla której wartości to zbiory zmiany stanów. W obu tych słownikach kluczami są stany, dla którego rozpatrywany

jest czas czy zmiana. Dalej, dla każdego czasu kiedy dane zdarzenie się wydarzyło, czyli dla elementów w pierwszym z obu słowników, generowany jest estymator jądrowy danego wydarzenia. W tym celu wykorzystano sumę rozkładów normalnych. Następnie wygenerowany estymator jest odwracany i progowany. Wszystkie wartości powyżej lub poniżej pewnego przedziału są spłaszczane do wartości brzegowych, a następnie przekształcane i skalowane tak, aby suma pola pod wykresem wynosiła 1, a najmniejszą wartością było 0. Proces ten został zobrazowany na obrazie (5.2). Korzystając z tak wygenerowanego estymatora znormalizowanego i funkcji losującej wartość z przedziału pod warunkiem macierzy prawdopodobieństw, dostarczonej przez *numpy*, generowany jest zestaw zerowych przejść z czasem pochodzącym z tego rozkładu, a następnie dołączany do listy danych do uczenia. Ten proces ma na celu stworzenie listy dodatkowych danych bez żadnej zmiany dla konkretnego stanu o innej porze w ciągu dnia, tak aby sieci neuronowe powiązały konkretny stan w ciągu dnia z konkretną porą dla danej odpowiedzi. Tak stworzona lista dla konkretnych stanów jest bardzo rzadka – posiada same zera w przejściach stanów – co może negatywnie wpłynąć na wynik uczenia sieci neuronowych. W tym celu, aby zrównoważyć stosunek ilości pustych akcji do tych rzeczywistych, do listy danych uczących dodawane są kolejne dane. Dane te losowane są z drugiego słownika dla konkretnie obsługiwanego stanu systemu i dodawane do danych uczących, czyli dopisywane są kopie danego rzeczywistego wydarzenia.



Rys. 5.2. Wizualizacja wygenerowanego estymatora jądrowego bez progowania oraz przebieg gotowej funkcji gęstości prawdopodobieństwa.

5.5. NAPOTKANE PROBLEMY

Ta sekcja ma na celu opisanie zaistniałych problemów podczas implementowania modułu. Są to trudności związane z przewidzeniem, jak zachowywać się będzie gotowy system bez żadnej istniejącej implementacji ani prototypu, ale także związane ze zwykłą naturą programistyczną.

5.5.1. Reprezentacja czasu

Reprezentacja czasu w modelach uczenia maszynowego jest bardzo ważna. Wysoka i mała rzadkość danych sprawia, że modele potrzebują bardzo dużo obserwacji aby rzeczywiście były w stanie generować predykcje zgodne z prawdą [13]. Podczas implementacji modułu początkowo wszystkie dane temporalne reprezentowane były jako kosinusoida znormalizowanej pory dnia. Wykorzystanie funkcji trygonometrycznej było uwarunkowane jej okresowością i spokojnym przebiegiem przez całą rozpatrywaną domenę. Podczas testów prototypu, bez dodatkowej funkcji tworzącej nowe dane uczące (5.4), modele mimo testów z różnymi parametrami nie były w stanie dopasować się do obserwacji. Zdecydowano

wówczas o rozbiciu danych temporalnych do zdecydowanie większej i rzadszej formy, zainspirowanej metodami radialnych funkcji bazowych, opisanej w (5.1.3). Testy prototypu przynosiły zdecydowanie lepsze wyniki, ale wciąż nie były one wystarczająco zadowalające. Kolejnym etapem który podjęto było stworzenie wyżej opisanej funkcji generującej sztuczne dane. Dopiero od tego momentu sieć uczyła się obserwacji w dostatecznym stopniu. Po wstępnych badaniach okazało się, że poprzednio stworzone rozwiązanie korzystające z funkcji trygonometrycznej i, w dalszej iteracji, złożenie sinusoidy i kosinusoidy w połączeniu z funkcją, również dają zadowalające wyniki. Zdecydowano zatem o pozostawieniu części kodu, odpowiedzialnego za pozostały sposób transformacji danych, jako możliwej alternatywy dla użytkownika.

5.5.2. Obraz Docker – AppDaemon

W celu minimalizacji reimplementacji często używanych i powtarzanych funkcji w językach programowania korzysta się ze standardowych bibliotek, które zapewniają szkielet podstawowych funkcjonalności programowania [20]. Pomaga to uniknięcia części błędów i zmniejszenia wielkości powstałego oprogramowania. Każda rodzina systemów operacyjnych opiera swoją strukturę na innej bibliotece, które zapewniają w praktyce taką samą funkcjonalność inną implementacją i konwencją, zatem programy nie są kompatybilne między rodzinami systemów operacyjnych, a czasami nawet różnymi dystrybucjami tego samego systemu operacyjnego. Przykładem takiego oprogramowania, które nie jest kompatybilne między dystrybucjami tej samej rodziny systemów, jest *Tensorflow*, który jest dostępny w gotowej do zainstalowania wersji, między innymi dla systemów wspierających *glibc*. W celu stworzenia modułu samouczącego należało połączyć ze sobą obraz *AppDaemon* wraz z *Tensorflow*, które w oryginalnych konfiguracjach korzystają z zupełnie innych bibliotek standardowych języka C [4] [2]. W celu rozwiązania tego problemu stworzono własny obraz modułu *AppDaemon*, który powstał na bazie systemu wspierającego *glibc*. Podczas tworzenia instalowana jest biblioteka *Tensorflow* oraz opisywany w tej pracy moduł samouczący.

6. BADANIA

W celu przeprowadzenia badania jakości tego, jak dobrze system uczy się nawyków użytkownika dla różnych konfiguracji parametrów systemu, zaproponowano pewien sztucznie wygenerowany scenariusz użycia kilku urządzeń domowych. Przykładowy scenariusz zawiera następującą listę urządzeń: światło w kuchni, ekspres do kawy, światło w salonie, telewizor w salonie, światła na balkonie, lampka w sypialni.

Użytkownik codziennie około godziny 5:30 gasi światło na balkonie przed domem, zapala światło w kuchni, a następnie włącza ekspres i na chwilę w salonie włącza telewizor. Gdy wróci do domu popołudniem, wchodzi do kuchni, zapala światło, włącza ekspres. Po czasie wyłącza w kuchni ekspres, światło, włącza w salonie telewizor i światło. Wieczorem, o godzinie 22, wyłącza w salonie telewizor oraz światło i włącza światło w kuchni. Po 15 minutach wyłącza światło i idzie do sypialni gdzie włącza lampkę. Późniejszą porą w nocy użytkownik wyłącza lampkę w sypialni i włącza światło na balkonie.

W celu zbadania różnych konfiguracji przeprowadzono serię wielu testów. Do danych uczących, tych, które pochodziłyby z systemu *HomeAssistant*, zostały dodane pewne zakłócenia w postaci włączeń i wyłączeń różnych urządzeń o różnych porach dnia, tak, aby nie zakłócały one surowego planu dnia użytkownika. Dane ewaluacyjne to te same dane co w przypadku uczenia systemu, pozbawione dodanych zakłóceń. Do badań użyto różnych kombinacji opcji konfiguracyjnych z długością epizodu równą 600, czyli 10 minut. W tabelach (6.1), (6.2), (6.3), zawarto badania z optymalizatorem Adam z włączonym pędem Nesterova, a w tabelach (6.4), (6.5), (6.6) bez. W tabelach (6.1) i (6.4) wykorzystano zaproponowany sposób przemiany danych temporalnych na reprezentację Gaussa. W tabelach (6.2) i (6.5) wykorzystano przemianę danych w postaci złożenia sinusoidy i kosinusoidy, a w tabelach (6.3) i (6.6) zastosowano samą kosinusoidę. Obie te metody zostały opisane w (5.5.1). Oba zestawy tabel podają wartość błędu walidacyjnego dla każdego sterowanego urządzenia wraz z sumą dla całego systemu. Każda kolumna oznacza inną badaną konfigurację.

I – Parametr prędkości uczenia 0.005, zakres progowania od 0.50 do 0.95,
II – Parametr prędkości uczenia 0.005, zakres progowania od 0.75 do 0.90,
III – Parametr prędkości uczenia 0.001, zakres progowania od 0.50 do 0.95,
IV – Parametr prędkości uczenia 0.001, zakres progowania od 0.75 do 0.90,

Tabela 6.1. Tabela przedstawiająca wyniki dla czasu w reprezentacji Gaussa z optymalizatorem Nadam.

	I	II	III	IV
Światła kuchnia	1.3184×10^{-1}	1.2615×10^{-1}	1.3509×10^{-1}	1.3616×10^{-1}
Ekspres	1.2637×10^{-1}	1.2612×10^{-1}	1.3058×10^{-1}	1.2807×10^{-1}
Światła salon	1.2604×10^{-1}	1.2636×10^{-1}	1.2509×10^{-1}	1.2330×10^{-1}
Telewizor	5.0368×10^{-1}	3.7748×10^{-1}	2.5453×10^{-1}	2.5345×10^{-1}
Światła balkon	3.2991×10^{-4}	1.1964×10^{-1}	6.4132×10^{-3}	4.7199×10^{-3}
Lampka sypialnia	1.6458×10^{-4}	1.2587×10^{-1}	8.6326×10^{-2}	8.6425×10^{-2}
Suma	8.8843×10^{-1}	1.0016×10^{0}	7.3802×10^{-1}	7.3212×10^{-1}

Tabela 6.2. Tabela przedstawiająca wyniki dla czasu używającego złożenia sinusoidy i kosinusoidy ułamka czasu z optymalizatorem Nadam.

	I	II	III	IV
Światła kuchnia	3.3997×10^{-1}	3.4003×10^{-1}	2.2071×10^{-1}	2.1171×10^{-1}
Ekspres	1.2645×10^{-1}	1.2621×10^{-1}	1.4054×10^{-1}	1.4197×10^{-1}
Światła salon	1.2584×10^{-1}	1.2632×10^{-1}	1.2765×10^{-1}	1.2775×10^{-1}
Telewizor	3.7751×10^{-1}	5.0319×10^{-1}	3.3696×10^{-1}	3.3481×10^{-1}
Światła balkon	6.1214×10^{-3}	6.3107×10^{-3}	2.2572×10^{-2}	2.4086×10^{-2}
Lampka sypialnia	2.4572×10^{-1}	7.7277×10^{-4}	2.0910×10^{-2}	2.2086×10^{-2}
Suma	1.2216×10^{0}	1.1028×10^{0}	8.6934×10^{-1}	8.6241×10^{-1}

Tabela 6.3. Tabela przedstawiająca wyniki dla kosinusoidy czasu z optymalizatorem Nadam.

	I	II	III	IV
Światła kuchnia	3.4200×10^{-1}	1.8050×10^{-1}	2.1940×10^{-1}	2.2070×10^{-1}
Ekspres	2.5275×10^{-1}	1.2672×10^{-1}	1.3860×10^{-1}	1.3782×10^{-1}
Światła salon	1.2608×10^{-1}	1.2595×10^{-1}	1.2898×10^{-1}	1.2910×10^{-1}
Telewizor	3.7796×10^{-1}	3.7781×10^{-1}	3.7902×10^{-1}	3.7880×10^{-1}
Światła balkon	1.2032×10^{-1}	1.1964×10^{-1}	5.0807×10^{-2}	5.2643×10^{-2}
Lampka sypialnia	2.4561×10^{-1}	2.4562×10^{-1}	2.6154×10^{-2}	2.9102×10^{-2}
Suma	1.4647×10^{0}	1.1762×10^{0}	9.4295×10^{-1}	9.4817×10^{-1}

Tabela 6.4. Tabela przedstawiająca wyniki dla czasu w reprezentacji Gaussa z optymalizatorem Adam.

	I	II	III	IV
Światła kuchnia	3.3984×10^{-1}	1.2775×10^{-1}	1.3552×10^{-1}	1.3403×10^{-1}
Ekspres	1.2613×10^{-1}	1.2696×10^{-1}	1.2883×10^{-1}	1.2869×10^{-1}
Światła salon	8.8793×10^{-2}	1.2635×10^{-1}	1.1933×10^{-1}	1.1832×10^{-1}
Telewizor	3.7789×10^{-1}	3.7740×10^{-1}	2.7103×10^{-1}	2.5464×10^{-1}
Światła balkon	8.6420×10^{-6}	3.0359×10^{-4}	3.1579×10^{-3}	3.0881×10^{-3}
Lampka sypialnia	5.3721×10^{-4}	1.9876×10^{-3}	8.5490×10^{-2}	8.3690×10^{-2}
Suma	9.3319×10^{-1}	7.6075×10^{-1}	7.4336×10^{-1}	7.2245×10^{-1}

Tabela 6.5. Tabela przedstawiająca wyniki dla czasu używającego złożenia sinusoidy i kosinusoidy ułamka czasu z optymalizatorem Adam.

	I	II	III	IV
Światła kuchnia	3.7779×10^{-1}	1.3758×10^{-1}	1.9831×10^{-1}	2.0386×10^{-1}
Ekspres	1.6384×10^{-1}	1.2675×10^{-1}	1.3649×10^{-1}	1.4022×10^{-1}
Światła salon	1.2603×10^{-1}	1.2617×10^{-1}	1.2749×10^{-1}	1.2719×10^{-1}
Telewizor	3.7989×10^{-1}	3.7785×10^{-1}	3.5703×10^{-1}	3.5197×10^{-1}
Światła balkon	8.4263×10^{-4}	1.1967×10^{-1}	1.7433×10^{-2}	1.3986×10^{-2}
Lampka sypialnia	1.0623×10^{-3}	1.2384×10^{-1}	1.6005×10^{-2}	2.1704×10^{-2}
Suma	1.0495×10^{0}	1.0119×10^{0}	8.5275×10^{-1}	8.5893×10^{-1}

Tabela 6.6. Tabela przedstawiająca wyniki dla kosinusoidy czasu z optymalizatorem Adam.

	I	II	III	IV
Światła kuchnia	2.7359×10^{-1}	2.2487×10^{-1}	2.2312×10^{-1}	2.1595×10^{-1}
Ekspres	1.2599×10^{-1}	1.2722×10^{-1}	1.4030×10^{-1}	1.3696×10^{-1}
Światła salon	2.5258×10^{-1}	1.2679×10^{-1}	1.2886×10^{-1}	1.2817×10^{-1}
Telewizor	5.0317×10^{-1}	5.0326×10^{-1}	3.8027×10^{-1}	3.8196×10^{-1}
Światła balkon	1.2039×10^{-1}	1.1987×10^{-1}	5.9904×10^{-2}	5.4583×10^{-2}
Lampka sypialnia	2.4636×10^{-1}	1.2597×10^{-1}	2.9190×10^{-2}	2.1298×10^{-2}
Suma	1.5221×10^{0}	1.2280×10^{0}	9.6165×10^{-1}	9.3893×10^{-1}

Porównując do siebie otrzymane wyniki ewaluacji można zauważyć pewne wnioski. Przewaga optymalizatora zależy od użytych sposobów przekształcenia danych temporalnych. W dużej części przypadków suma wartości błędu dla reprezentacji Gaussa i reprezentacji złożenia funkcji trygonometrycznych jest mniejsza dla optymalizatora Adam, w porównaniu do Nadam z wyjątkiem samej kosinusoidy. Najmniejszy błąd sumaryczny osiąga reprezentacja dzwonowa, osiągając go w każdym możliwym przypadku. Użycie węższego zakresu progowania w pojedynczych wynikach daje gorsze wyniki, niż dla szerokiego, mniej skupionego na podstawie obszaru. Wszystkie przypadki mniejszej prędkości uczenia osiągnęły lepsze wyniki, niż tam, gdzie sieci otrzymały wyższy parametr. Wybór wariantu IV z użyciem funkcji Gaussa i optymalizatorem Nadam przynosi wystarczające wyniki dla ogółu wszystkich urządzeń. Nie jest to jednak przesłanka do tego, że wynik ten będzie też najniższy dla każdego urządzenia z osobna. Porównując do siebie wariant pierwszy z czwartym w tabeli (6.4), można zauważyć, że część urządzeń lepiej przystosowała się do nawyków użytkownika, a mimo to wciąż osiągnęły gorszy wynik sumaryczny.

Gwoli podsumowania warto wspomnieć, że badania nie oddają w pełni jakości użycia modułu w rzeczywistości, ponieważ w celu przeprowadzenia tych badań użyto pewnych uogólnień które upraszczają realia. Niższa wartość wyniku ewaluacji nie sprawi, że dane urządzenie będzie lepiej przystosowywać się do zamiarów użytkownika, a wskaże tylko dobór parametrów lepszy dla pewnej części odbiorców modułu.

PODSUMOWANIE

ZREALIZOWANA PRACA

W ramach tej pracy przeanalizowano i omówiono zagadnienie związane z automatyką domową. Przejrzano i skategoryzowano istniejące rozwiązania problemu pracy zawarte w literaturze. Każde z podejść zostało poddane analizie w celu rozpoznania mocnych i słabych stron rozwiązania. W dalszej części zagłębiono się w problem zastosowań urządzeń IoT do celów automatyki. Na podstawie literatury i istniejących rozwiązań stworzono zakres wymagań, jakie powinien spełniać działający system w celu wspierania użytkownika w codziennych czynnościach domowych. Stworzone wymogi – funkcjonalne i niefunkcjonalne – pomogły w określeniu technologii oraz całego środowiska aplikacji w którym stworzony moduł pracuje. Wykorzystanie konkretnych aplikacji, modułów i bibliotek ukształtowało architekturę logiczną samego modułu. Korzystając z planów i architektury sporządzono w pełni funkcjonujące rozwiązanie. W celu sprawdzenia jakości systemu przeprowadzono badania i opisano wyniki.

CEL PRACY

Cel pracy został osiągnięty. Sporządzono moduł samouczący, który wykonuje epizody nauczonych akcji za użytkownika. Dodatkowo w celu integracji z rzeczywistym środowiskiem, rozwiązanie współpracuje z oprogramowaniem HomeAssistant w celu sterowania urzadzeniami IoT w domu. Zastosowanie paradygmatu zorientowanego na obiekty do implementacji modułu ułatwiło integrację systemu w dynamiczną całość i pozwoli na dalszy rozwój aplikacji. System podczas pierwszego uruchomienia stworzy modele sieci neuronowych i nauczy je na podstawie danych historycznych. W trybie bieżącej pracy system będzie przewidywał następne akcje użytkownika i w wypadku wspólnego zamiaru, wykonywał pozostałe czynności.

SUGESTIE DALSZYCH PRAC

Każde stworzone oprogramowanie nigdy nie jest skończone, jest ono jedynie publikowane. Każda praca zostawia pole do dalszych usprawnień i modyfikacji. Opisany w tej pracy moduł samouczący nie jest wyjątkiem w tej zasadzie. Jedną dużą zmianą, która mogłoby poprawić jakość działania całego systemu, byłoby dynamiczne tworzenie różnych struktur sieci neuronowych w zależności od docelowego urządzenia. Usprawnienie miałoby

na celu określenie dokładnej definicji ostatnich warstw w modelu, co poprawiłoby jakość połączenia między konwerterami a sercem modułu. Dodatkowo, w celu poprawy jakości działania całego systemu, należałoby rozważyć inny algorytm generowania dodatkowych danych oraz ziarnisty sposób określenia dla każdego urządzenia parametrów uczenia, warstw sieci i generowania epizodów. Polem do rozważań jest problem identyfikacji i ewentualnego interweniowania w przypadku, gdy dwa podobne przejścia stanów wykonują się pod warunkiem tego samego stanu systemu.

Dalsze prace będą niewątpliwym efektem tworzenia rozszerzalnego i wolnego oprogramowania. Stworzony system wspierajacy tworzenie nowych, dodatkowych urządzeń jest jedynie zachętą dla użytkownika aby ten moduł rozszerzać.

WNIOSKI

Połączenie ze sobą HomeAssistant i AppDaemon to bardzo silne połączenie, w tym przypadku wzmacniane jeszcze bardziej przez uczenie maszynowe. Daje ono możliwość połączenia wszystkiego, co użytkownik jest w stanie wyrazić oprogramowaniem, ze światem rzeczywistym, a dokładniej, jego środowiskiem domowym. Daje narzędzia do wykreowania czegoś większego, co świadomie będzie opiekowało się swoimi domownikami. Takie zastosowanie sterowania domem i automatyką tworzy możliwość reagowania środowiska domowego na wydarzenia dla niego zewnętrznych, do których nie zostało specjalnie przygotowane.

Ta praca to tylko kropla w morzu możliwości jakie taka kombinacja oprogramowania jest w stanie osiągnąć i kroplą w oceanie możliwości wykorzystania systemów IoT.

BIBLIOGRAFIA

[1] *Internet Protocol*, RFC 791. 1981.

[2] acockburn, *Kod źródłowy dockerfile dla obrazu appdaemon*, `https://github.com/AppDaemon/appdaemon/blob/ac5f70ed90f4e43b575c978c456a7705803d2c17/Dockerfile`.

[3] Alam, M.R., Reaz, M.B.I., Mohd Ali, M.A., *Speed: An inhabitant activity prediction algorithm for smart homes*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans. 2012, tom 42, 4, str. 985–990.

[4] Alpine Linux Development Team, *About alpine linux*, `https://web.archive.org/web/20231125205542/https://www.alpinelinux.org/about/`. Dostęp 25-11-2023.

[5] Cockburn, A., *Dokumentacja appdaemon*, `https://web.archive.org/web/20231111025200/https://appdaemon.readthedocs.io/en/latest/`. Dostęp 11-11-23.

[6] Cook, D.J., Youngblood, M., Heierman, E.O., Gopalratnam, K., Rao, S., Litvin, A., Khawaja, F., *Mavhome: An agent-based smart home*, w: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).* (IEEE, 2003), str. 521–524.

[7] Cordero, J., Yi, J., Clausen, T., Baccelli, E., *Enabling multihop communication in spontaneous wireless networks*, ACM SIGCOMM eBook on" Recent Advances in Networking. 2013, tom 1, str. 413–457.

[8] Domoticz Team, *Dokumentacja pisania skryptów domoticz*, `https://web.archive.org/web/20221217175847/https://www.domoticz.com/wiki/Scripts`. Dostęp 17-12-22.

[9] Goldfarb, C.F., Prescod, P., *Charles F. goldfarb's XML handbook* (Prentice Hall PTR, Harlow, England, 2002).

[10] Heierman, E., Cook, D., *Improving home automation by discovering regularly occurring device usage patterns*, w: *Third IEEE International Conference on Data Mining* (2003), str. 537–540.

[11] I Code Academy, *Json for beginners* (Whiteflowerpublsihing, 2020).

[12] Jaihar, J., Lingayat, N., Vijaybhai, P.S., Venkatesh, G., Upla, K.P., *Smart home automation using machine learning algorithms*, w: *2020 international conference for emerging technology (INCET)* (IEEE, 2020), str. 1–4.

[13] Keogh, E., Mueen, A., *Curse of Dimensionality* (Springer US, Boston, MA, 2010), str. 257–258.

[14] Lutz, M., *Learning Python* (O'REILLY', 2013).

[15] Mannila, H., Toivonen, H., Inkeri Verkamo, A., Data Mining and Knowledge Discovery. 1997, tom 1, 3, str. 259–289.

[16] Martin, R.C., *Czysty Kod* (Helion, 2014).

[17] Marufuzzaman, M., Tumbraegel, T., Rahman, L.F., Sidek, L.M., *A machine learning approach to predict the activity of smart home inhabitant*, J. Ambient Intell. Smart Environ. 2021, tom 13, 4, str. 271–283.

[18] openHAB Community openHAB Foundation, *Dokumentacja pisania skryptów*, `https://web.archive.org/web/20231009171101/https://www.openhab.org/addons/automation/jsscripting/`. Dostęp 09-10-23.

[19] Pang, B., Nijkamp, E., Wu, Y.N., *Deep learning with tensorflow: A review*, Journal of Educational and Behavioral Statistics. 2020, tom 45, 2, str. 227–248.

[20] Plauger, P.J., *The Standard C Library* (Prentice Hall, Philadelphia, PA, 1991).

[21] Preu, S., Cap, C., *Overview of spontaneous networking - evolving concepts and technologies*. 2002.

[22] Reis, J., *Fundamentals of data engineering* (O'Reilly Media, Sebastopol, CA, 2022).

[23] Roberts, E.S., *Programming Abstractions in C++* (Pearson, Upper Saddle River, NJ, 2013).

[24] Sequeiros, H., Oliveira, T., Thomas, M.A., *The impact of iot smart home services on psychological well-being*, Information Systems Frontiers. 2021, tom 24, 3, str. 1009–1026.

[25] Shah, B., Bhavsar, H., *Time complexity in deep learning models*, Procedia Computer Science. 2022, tom 215, str. 202–210. 4th International Conference on Innovative Data Communication Technology and Application.

[26] Silverio-Fernández, M., Renukappa, S., Suresh, S., *What is a smart device? - a conceptualisation within the paradigm of the internet of things*, Visualization in Engineering. 2018, tom 6, 1.

[27] Szabłowski, S., *Projektowanie dydaktycznych systemów automatyki domowej*, Dydaktyka informatyki. 2019, , 14, str. 137–146.

[28] Umamageswari, S.S.M., Kannan, M.K.J., *Tensorflow-based smart home using semi-supervised deep learning with context-awareness*, Journal of Mathematical and Computational Science. 2022.

[29] Wu, E., Zhang, P., Lu, T., Gu, H., Gu, N., *Behavior prediction using an improved hidden markov model to support people with disabilities in smart homes*, w: *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2016), str. 560–565.

[30] Zaki, M.J., Machine Learning. 2001, tom 42, 1/2, str. 31–60.

SPIS RYSUNKÓW

SPIS LISTINGÓW

SPIS TABEL