Politechnika Śląska
Wydział Mechaniczny Technologiczny
Katedra Podstaw Konstrukcji Maszyn

Rozprawa doktorska

Metodyka testowania sterowników systemów bateryjnych pojazdów wsparta modelem

mgr inż. Kamil Sternal
Promotor:
dr hab. inż. Marek Fidali, prof. PŚ

Gliwice, 2024

Spis treści

Spis rysunków

Spis tablic

Lista ważnych skrótów

Technologia pojazdów i akumulatorów

VAG Volkswagen Auto Group

Li-ion Lithium-ion

LFP Lithium Iron Phosphate

NCA Nickel Cobalt Aluminum

OBDII On-Board Diagnostics II

ESC Electronic Stability Control

ASIL Automotive Safety Integrity Level

Elektronika i systemy motoryzacyjne

ADC Analog Digital Converter

SPI Serial Peripheral Interface

CAN Controller Area Network

PWM Pulse Width Modulation

GPIO General Purpose Input/Output

EEPROM Electrically Erasable Programmable Read-Only Memory

EPROM Erasable Programmable Read-Only Memory

ROM Read-Only Memory

VHDL VHSIC Hardware Description Language

GDB GNU Debugger

DC Direct Current

AC Alternating Current

IGBT Insulated Gate Bipolar Transistor

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

Uczenie maszynowe i sztuczna inteligencja

AI Artificial Intelligence

CVAE Conditional Variational Autoencoder

VAE Variational Autoencoder

ReLU Rectified Linear Unit

VLB Variational Lower Bound

IWAE Importance Weighted Autoencoder

Metryki

MSE Mean Squared Error

RMSE Root Mean Square Error

ROC Receiver Operating Characteristic

NaN Not a Number

PCE Pesymistyczny Czas Egzekucji

OCE Optymistyczny Czas Egzekucji

WCET Worst Case Execution Time

BCET Best Case Execution Time

Inne akronimy

ISO International Organization for Standardization

IEC International Electrotechnical Commission

MISRA Motor Industry Software Reliability Association

SVN Subversion

IDE Integrated Development Environment

SWE Software Engineering Process

NHTSA National Highway Traffic Safety Administration

NASA National Aeronautics and Space Administration

QM Quality Management

HARA Hazard Analysis and Risk Assessment

RTX Ray Tracing Texel eXtreme

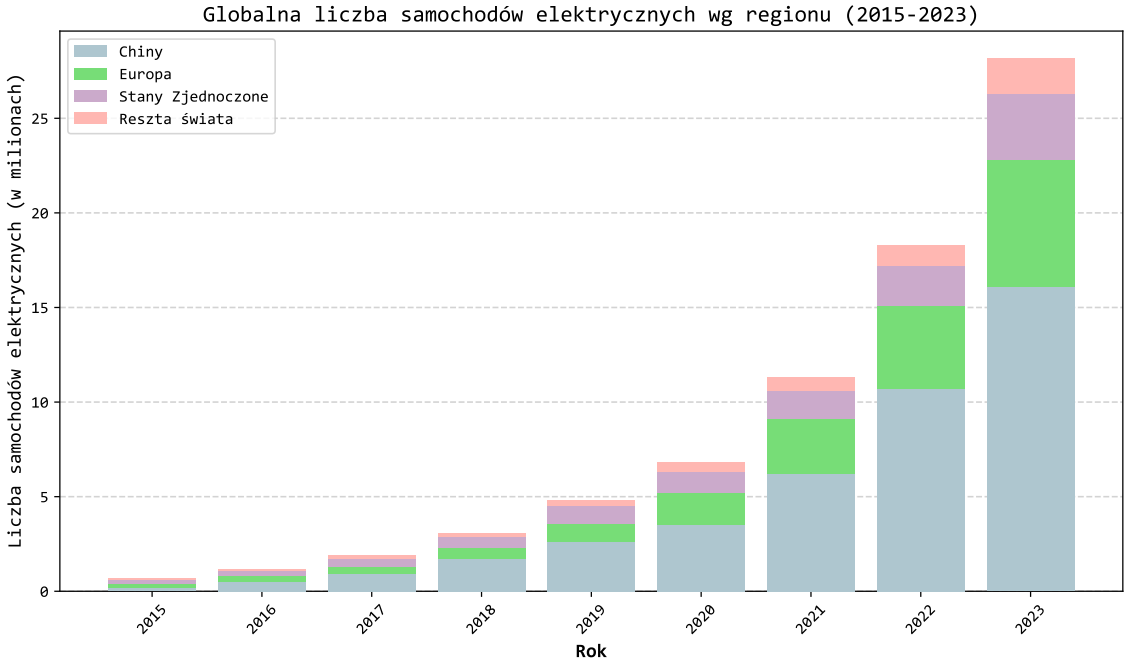
Rozdział 1

Wstęp

1.1 Geneza pracy

W roku 2023 liczba rejestracji nowych samochodów elektrycznych na świecie wyniosła około 40,5 miliona pojazdów. Dla porównania, w 2015 roku (rys. 1.1) liczba zarejestrowanych pojazdów elektrycznych wynosiła zaledwie 1,3 miliona, co oznacza, że liczba ta wzrosła ponad 31-krotnie w ciągu ośmiu lat. W Europie liczba zarejestrowanych samochodów elektrycznych wzrosła z 0,4 miliona w 2015 roku do 11,2 miliona w 2023 roku, co stanowi wzrost 28-krotny [43]. Przytoczone dane statystyczne wskazują na wyjątkowo dynamiczny trend wzrostowy w kierunku elektryfikacji gamy modelowej dostępnej na rynku motoryzacyjnym.

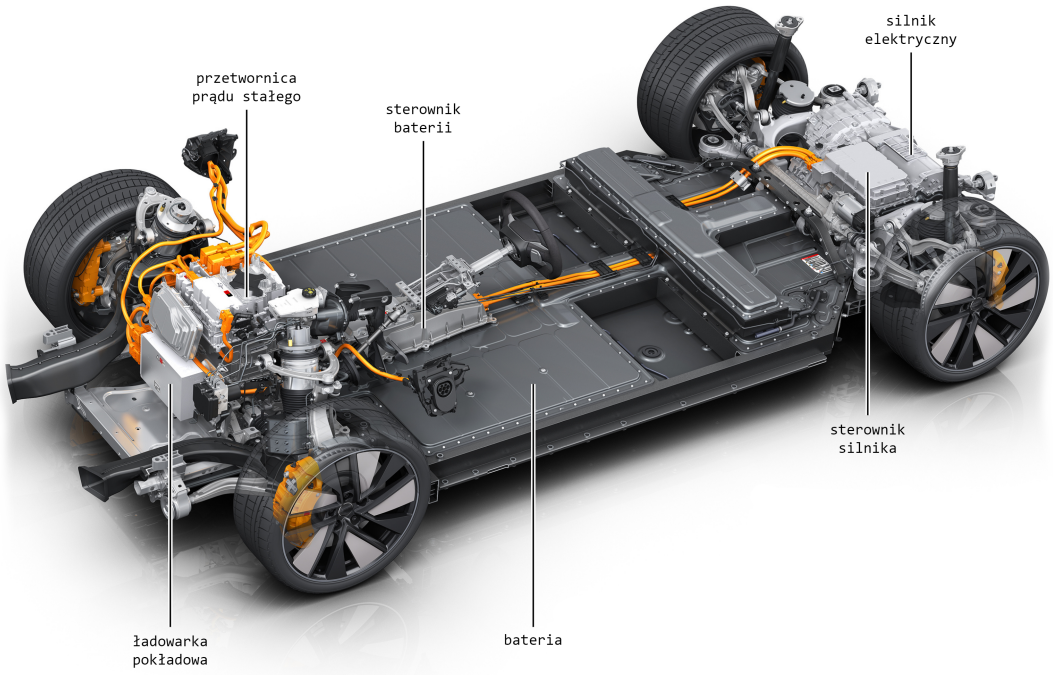
Wraz z tym trendem, producenci samochodów musieli zmierzyć się z rosnącymi wymaganiami rynku, co zmusiło ich do kompleksowej przebudowy architektury pojazdów. Przez ponad stulecie samochody opierały się głównie na wynalazku Karla Benza, który w 1885 roku zastosował silnik benzynowy własnej konstrukcji do napędu pojazdu, co zainicjowało rozwój przemysłu motoryzacyjnego [35]. Projekty samochodów utrzymywały się w zakresie rozwiązań wyłącznie mechanicznych aż do roku 1967. Przełom nastąpił, gdy firma Bosch opracowała dla Volkswagena pierwszy system elektronicznego wtrysku paliwa, znanego jako *D-Jetronic* [37], wprowadzając tym samym epokę elektroniki w motoryzacji. Ta innowacja umożliwiła włączenie elementów elektronicznych do konstrukcji mechanicznych, znacząco wpływając na rozwój technologii motoryzacyjnej. Od tego momentu udział komponentów elektronicznych systematycznie wzrastał, choć podstawowa zasada działania silnika spalinowego pozostała



Rysunek 1.1: Liczba rejestracji samochodów elektrycznych
w ciągu ostatnich ośmiu lat [43]

niezmieniona, opierając się na współdziałaniu ruchomych części [40]. Mimo że liczba ruchomych części rosła, to sterowaniem silnika spalinowego przeważnie zajmował się jeden sterownik. Jest to zupełnym przeciwieństwem rozwiązań stosowanych we współczesnych pojazdach elektrycznych gdzie liczba części mechanicznych znacząco spadła przykładowo [20]. Zauważyć więc można, że nastąpiła znaczna redukcja udziału części mechanicznej w układzie napędowym, co jest wynikiem postępu technologicznego i potrzeby efektywniejszego wykorzystania energii [21]. Jednakże ze względu na sposób sterowania silnikiem elektrycznym oraz sposób przechowywania energii potrzebnej do jego napędzania, konieczne stało się użycie wielu dodatkowych układów elektronicznych.

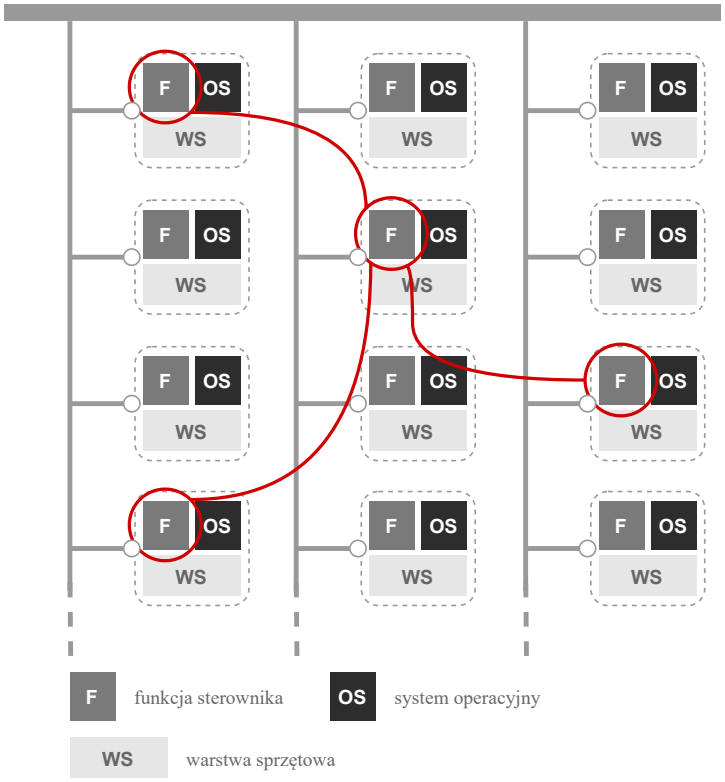
Na rysunku 1.2 przedstawia rysunek poglądowy pojazdu elektrycznego. Wyszczególnia on dwa główne układy niezbędne do wprawienia pojazdu w ruch: układ napędowy oraz układ bateryjny. Pozostałe układy zostały zgrupowane i oznaczone jako pomocnicze. Opisane elementy składają się na systemy, które realizują określone funkcje pojazdu, a w przeważającej części opierają się na układach elektronicznych. Jak można zauważyć, komponenty te stanowią większość, co oznacza, że liczba danych wymienianych pomiędzy układami znacząco wzrosła. Zwiększony przepływ danych wymaga od systemów sterowania realizacji większej liczby zadań, począwszy od ich odbioru, poprzez przetworzenie, aż po dystrybucję. Samochody elektryczne mają więcej komponentów elektronicznych niż konwencjonalne



Rysunek 1.2: Komponenty układu napędowego samochodu elektrycznego [20]

pojazdy spalinowe, co zwiększa złożoność i ilość danych przetwarzanych przez systemy sterowania pojazdu. Rys. 3.5 ilustruje tę zależność, ukazując, jak działanie jednej z funkcji sterownika pojazdu zależy od innych sterowników podłączonych do wspólnej magistrali danych [61].

Ze względu na to, że od samochodu wymaga się, by był bezpieczny zarówno dla użytkowników, jak i otoczenia, projektowane są w nim funkcje, które spełniają te wymagania. Dla przykładu pomiar i monitorowanie ciągłości izolacji przewodów wysokiego napięcia należy zaliczyć do funkcji bezpieczeństwa [48]. W związku z tym ważne jest, aby systemy pojazdów elektrycznych utrzymywały odpowiednią rezystancję izolacji, przekraczającą 80 kΩ (dla 800 V systemu bateryjnego) między liniami wysokiego napięcia a nadwoziem. Takie działanie zapewnia, że instalacja elektryczna będzie spełniała normy ISO 6469-3:2021 [3], co minimalizuje ryzyko porażenia elektrycznego. Systemy monitorowania izolacji są kluczowym elementem zapewniającym bezpieczeństwo elektryczne pojazdów, gdyż wymagają one stosowania specjalistycznych systemów pomiarowych. Ważne jest, aby były odporne na uszkodzenia, które mogą doprowadzić do poważnej awarii pojazdu i zagrozić bezpieczeństwu użytkownika. Jest to istotne, ponieważ aby prąd mógł przepływać z układu do ciała człowieka i z powrotem, muszą wystąpić dwa odrębne uszkodzenia izolacji instalacji. To podkreśla potrzebę ciągłego monitorowania sprawności i niezawodności sterowników podsystemów pojazdu odpowiedzialnych za realizację funkcji



Rysunek 1.3: Wizualizacja zależności pomiędzy realizowanymi funkcjami w pojeździe elektrycznym [25]

bezpieczeństwa. Te, jak i wiele innych funkcji spowodowały, że oprogramowanie je realizujące cechuje się dużym poziomem złożoności. Jedną z miar złożoności kodu źródłowego jest liczba linii zawierających instrukcje. W roku 2019 samochody dostępne na rynku sumarycznie posiadały około 100 milionów linii kodu. W roku 2025, z powodu rozwoju technologi samochodów elektrycznych, przewiduje się, że liczba ta wzrośnie do przedziału między 200 a 300 milionów linii kodu [25]. Wzrost ten będzie spowodowany zmianą rodzaju napędu na elektryczny oraz większymi możliwościami autonomicznej jazdy. Samochody elektryczne oraz z systemami wspomagającymi prowadzenie wymagają bardziej zaawansowanych i rozbudowanych systemów sterowania opartych na oprogramowaniu, stąd znaczący wzrost liczby linii kodu niezbędnego do ich funkcjonowania. W konsekwencji proces rozwoju oprogramowania oraz testowania jego staje się coraz bardziej wymagający.

1.2 Identyfikacja problemu badawczego

Rozwój technologiczny w motoryzacji, związany z wprowadzeniem elektromobilności, stawia coraz wyższe wymagania wobec systemów wbudowanych w pojazdach. System wbudowany rozumie się tutaj jako specjalizowany system komputerowy zaprojektowany do kontrolowania i zarządzania określonymi funkcjami pojazdu, takimi jak napęd elektryczny, systemy bezpieczeństwa czy systemy rozrywki. Systemy wbudowane pojazdów elektrycznych działają jako systemy czasu rzeczywistego. System czasu rzeczywistego to komputerowy system, który musi przetwarzać i odpowiadać na dane wejściowe w ściśle określonym czasie, niezależnie od obciążenia [53]. Jego poprawne działanie zależy od zakończenia operacji w wyznaczonym przedziale czasowym. Systemy czasu rzeczywistego są kluczowe w pojazdach elektrycznych, ponieważ zapewniają odpowiednią odpowiedź podukładów funkcjonalnych pojazdu na ciągle zmieniające się sygnały sterujące i warunki otoczenia. Muszą być zaprojektowane tak, aby spełniać wymagania czasowe zarówno podczas normalnej pracy, jak i w przypadku awarii. Dlatego analiza zależności czasowych zachodzących w sterownikach odgrywa istotną rolę w projektowaniu i testowaniu tych systemów. Precyzyjne określenie najdłuższej ścieżki wykonania programu sterownika jest niezbędne, aby testowany system sprostał krytycznym wymaganiom czasowym. Skuteczna analiza tych zależności pozwala także optymalizować wykorzystanie zasobów sprzętowych, przynosząc znaczące oszczędności czasowe. W analizie czasowej ważne są relacje między sygnałami wejściowymi. Zależności między nimi mogą wpływać na realizację czasowo najdłuższej ścieżki wykonania programu sterownika systemu wbudowanego, co bezpośrednio przekłada się na projektowanie i ocenę układów wbudowanych. Mimo postępów w inżynierii oprogramowania i systemów wbudowanych istnieje luka badawcza w metodach analizy czasów realizacji potencjalnych sekwencji kodu programu sterownika systemu wbudowanego w zależności od relacji między sygnałami wejściowymi. Dotychczasowe metody nie w pełni wykorzystują potencjał metod sztucznej inteligencji, które mogą zwiększyć skuteczność analizy zależności czasowych a tym samym podnieść efektywność projektowania systemów czasu rzeczywistego. Wykorzystanie metod sztucznej inteligencji w analizie zależności czasowych funkcji systemów wbudowanych może być pomocne w diagnozowaniu potencjalnych problemów na etapie projektowania i testowania prototypów sterowników baterii. Takie podejście może znacząco zwiększyć bezpieczeństwo, niezawodność i efektywność systemów stosowanych w motoryzacji, otwierając nowy rozdział w testowaniu i diagnozowaniu sterowników pojazdów elektrycznych.

1.3 Tezy pracy

Autor w swej pracy przyjął następujące tezy, będące podstawą do analizy i dyskusji w ramach prezentowanej pracy doktorskiej:

1. Istnie możliwość opracowania metodyki testowania sterowników systemów bateryjnych bazującej na metodach sztucznej inteligencji pozwalających na opracowanie modelu zależności czasowych uzależnionych od wzajemnych związków między sygnałami wejściowymi.
2. Istnieje metoda sztucznej inteligencji pozwalająca na detekcję potencjalnych anomalii w kodzie oprogramowania sterowników systemów bateryjnych pojazdów na podstawie wzajemnych zależności między sygnałami wejściowymi i potencjalnymi ścieżkami wykonania kodu.

1.4 Cel pracy

Celem niniejszej pracy jest opracowanie metodyki testowania sterowników układów bateryjnych pojazdów elektrycznych, która pozwala na detekcję anomalii w działaniu sterownika poprzez wykorzystanie modelu opracowanego z zastosowaniem metod sztucznej inteligencji. Odnosząc się do postawionych tez, praca ma na celu:

- Wykazanie, że istnieje możliwość opracowania metodyki testowania sterowników systemów bateryjnych bazującej na metodach sztucznej inteligencji, co pozwoli na stworzenie modelu zależności czasowych uzależnionych od wzajemnych związków między sygnałami wejściowymi.
- Zaprezentowanie metody sztucznej inteligencji umożliwiającej detekcję potencjalnych anomalii w kodzie oprogramowania sterowników systemów bateryjnych pojazdów, bazując na wzajemnych zależnościach między sygnałami wejściowymi i potencjalnymi ścieżkami wykonania kodu.

Realizacja tych celów pozwoli na usprawnienie procesu testowania i weryfikacji poprawności działania oprogramowania sterowników w kontekście spełnienia założeń projektowych. Poprzez zastosowanie algorytmów rozszerzonej inteligencji możliwe będzie analizowanie wpływu sygnałów wejściowych na czasy reakcji funkcji bezpieczeństwa w systemach bateryjnych. Opracowana metodyka umożliwi weryfikację, czy sterowniki bateryjne spełniają założenia projektowe oraz identyfikację potencjalnych anomalii, co przyczyni się do zwiększenia niezawodności i bezpieczeństwa pojazdów elektrycznych.

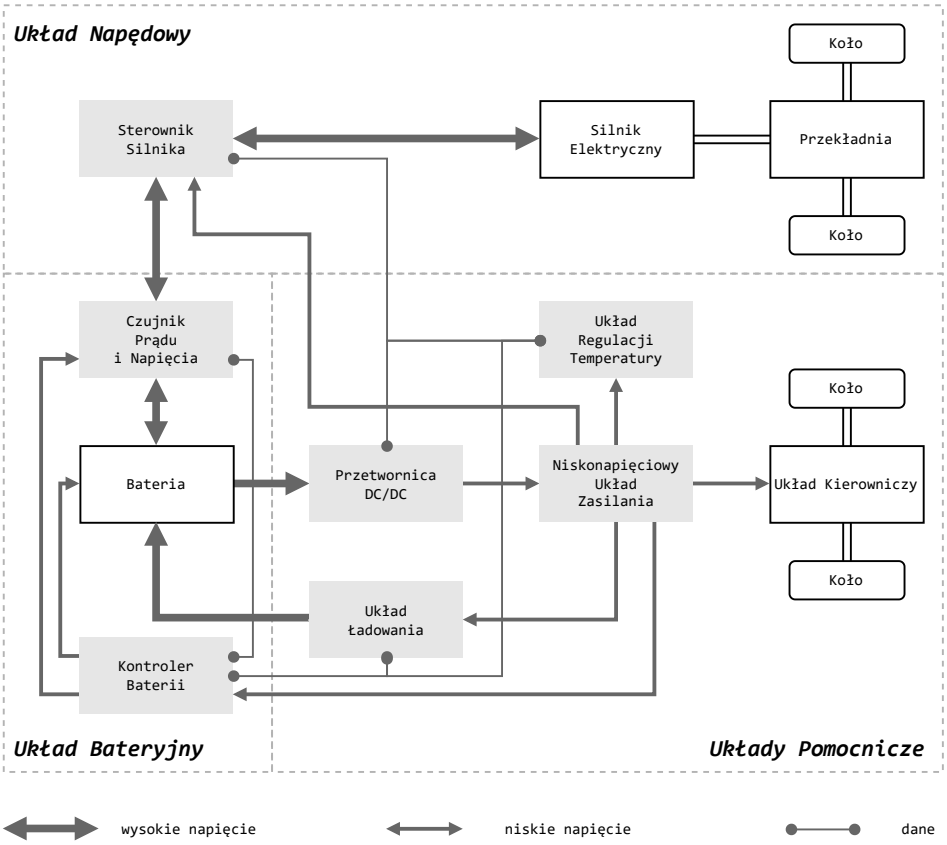
Rozdział 2

Przedmiot badań

2.1 Architektura pojazdów elektrycznych

Współczesne pojazdy elektryczne charakteryzują się zaawansowaną konstrukcją, która jest kluczowym elementem ich efektywności i funkcjonalności. Konstrukcja ta obejmuje nowoczesne układy napędowe, które są projektowane z myślą o maksymalnej wydajności energetycznej oraz minimalizacji strat mocy. Zaawansowane systemy zarządzania bateriami, znane jako BMS [33] (Battery Management System), odgrywają tutaj kluczową rolę, monitorując parametry pracy baterii, takie jak napięcie, temperatura i stan naładowania, aby zapewnić optymalne warunki pracy oraz bezpieczeństwo użytkowania.

Rysunek 2.1 przedstawia kompozycję, na którą składają się elementy elektroniczne i mechatroniczne, oznaczone kolorem szarym oraz elementy całkowicie mechaniczne oznaczone kolorem białym. Elementy elektroniczne obejmują jednostki sterujące, czujniki oraz moduły komunikacyjne, zapewniające integrację systemów oraz sprawną wymianę informacji między różnymi podzespołami pojazdu. Elementy mechatroniczne, takie jak układy napędowe i systemy hamowania regeneracyjnego, łączą w sobie funkcjonalności mechaniczne i elektroniczne, co pozwala na bardziej zaawansowane sterowanie i optymalizację działania.



Rysunek 2.1: Architektura pojazdów elektrycznych

2.1.1 Układ napędowy

Układ napędowy w samochodzie elektrycznym pełni kluczową funkcję, odpowiadając za napędzanie pojazdu oraz rekuperację energii. Napędzanie pojazdu polega na przekształcaniu energii elektrycznej zgromadzonej w akumulatorach w energię mechaniczną, która napędza koła pojazdu. Rekuperacja natomiast to proces odzyskiwania energii kinetycznej pojazdu podczas hamowania, która następnie jest przekształcana z powrotem w energię elektryczną i magazynowana w akumulatorach. Ten proces pozwala na zwiększenie efektywności energetycznej pojazdu i wydłużenie zasięgu na jednym ładowaniu [11].

Silnik Elektryczny

Silnik elektryczny jest kluczowym elementem układu napędowego w samochodzie elektrycznym. W pojazdach elektrycznych najczęściej stosowane są silniki prądu przemiennego, takie jak silniki indukcyjne lub synchroniczne z magnesami trwałymi (PMSM - Permanent Magnet Synchronous Motor).

W zależności od konstrukcji pojazdu, może on być wyposażony od jednego do czterech silników elektrycznych. W przypadku pojazdów elektrycznych z napędem na wszystkie koła, stosuje się zazwyczaj dwa lub więcej silników, co umożliwia skuteczniejsze przeniesienie momentu obrotowego na koła i precyzyjniejsze zarządzanie dystrybucją siły napędowej.

Głównym zadaniem silnika elektrycznego jest przekształcanie energii elektrycznej na energię mechaniczną. Typowe parametry silnika elektrycznego obejmują moc wyrażaną w kilowatach (kW) oraz moment obrotowy wyrażany w niutonometrach (Nm). Silniki elektryczne charakteryzują się wysoką sprawnością, często przekraczającą 90%, co znacząco przyczynia się do efektywności energetycznej pojazdu [19].

Sterownik Silnika Elektrycznego

Sterownik silnika elektrycznego jest odpowiedzialny za zasilanie i zarządzanie pracą silnika. Jego zadaniem jest przekształcanie prądu stałego (DC) z akumulatorów na prąd przemienny (AC) o odpowiedniej częstotliwości i amplitudzie, co pozwala na precyzyjne sterowanie prędkością i momentem obrotowym silnika. Typowe parametry sterownika obejmują maksymalne napięcie i prąd pracy, częstotliwość przełączania oraz sprawność energetyczną.

Budowa sterownika obejmuje moduł mocy, który składa się z tranzystorów IGBT (Insulated Gate Bipolar Transistor) lub MOSFET (Metal-Oxide-Semiconductor Field-Effect Transistor), oraz układ sterujący, który zarządza pracą tych tranzystorów. Relacja między sterownikiem a silnikiem elektrycznym jest kluczowa dla efektywnego działania układu napędowego. Sterownik musi precyzyjnie dostosowywać parametry prądu przemiennego, aby zapewnić optymalne warunki pracy silnika i maksymalną efektywność energetyczną.

Funkcje sterownika obejmują również zarządzanie rekuperacją energii podczas hamowania, ochronę przed przeciążeniami i przegrzaniem, oraz komunikację z innymi systemami pojazdu, takimi jak system zarządzania baterią. Dzięki zaawansowanym algorytmom sterowania, sterownik jest w stanie dynamicznie dostosowywać pracę silnika do warunków jazdy, zapewniając płynność i komfort prowadzenia pojazdu [18].

2.1.2 Układ bateryjny

Układ bateryjny odgrywa kluczową rolę w nowoczesnych pojazdach elektrycznych, działając jako magazyn energii elektrycznej. Energia ta może pochodzić z sieci energetycznej lub z procesu rekuperacji

i jest dostarczana do układu napędowego pojazdu. Dzięki temu, pojazdy elektryczne są w stanie efektywnie wykorzystać zgromadzoną energię, co przyczynia się do zmniejszenia emisji zanieczyszczeń oraz zwiększenia efektywności energetycznej. Podstawowe parametry charakteryzujące baterię to napięcie oraz pojemność. Najpopularniejsze obecnie architektury układów bateryjnych pracują przy napięciach 400V lub 800V [8]. Komercyjnie dostępne pojazdy elektryczne wyposażone są w pakiety bateryjne o łącznej pojemności w zakresie od 20 kWh do 100 kWh, co pozwala na różne zasięgi pojazdów w zależności od ich przeznaczenia i konstrukcji.

Ogniwa

Ogniwa są podstawowym elementem budowy układu bateryjnego. Połączone w pakiety, tworzą baterię. Na przykład, połączenie 198s2p oznacza, że 198 ogniw jest połączonych szeregowo, a dwa takie zestawy są połączone równolegle. Ze względu na kształt wyróżniamy następujące rodzaje ogniw: cylindryczne, pryzmatyczne oraz woreczkowe. Bez względu na ich kształt, współcześnie najpopularniejsze są ogniwa litowo-jonowe (Li-Ion), które występują w różnych wariantach chemicznych, takich jak litowo-żelazowo-fosforanowe (LFP) czy litowo-niklowo-kobaltowo-aluminiowe (NCA) [26].

Czujnik multimodalny

Czujnik prądu i napięcia w systemie baterii wysokonapięciowej umożliwia precyzyjne monitorowanie parametrów elektrycznych i zarządzanie energią. Urządzenie to, oparte na technologii efektu Halla, mierzy przepływ prądu oraz różnicę potencjałów elektrycznych na liniach wysokiego napięcia, takich jak linia pomiędzy ładowarką a baterią, baterią i silnikami. Dodatkowo stosuje się również czujniki oparte na bocznikach wykonanych w wysokiej tolerancji, o rezystancjach od 0.02 mΩ do 0.005 mΩ, które umożliwiają pomiar prądów do 1000A w systemach opartych na 800V architekturze. Czujnik prądu odpowiada za pomiar prądu wpływającego i wypływającego z baterii, monitorowanie przeciążeń prądowych oraz optymalizację zużycia energii. Czujnik napięcia monitoruje napięcie na liniach wysokiego napięcia, wykrywa degradację i awarie systemu, a także kontroluje moc wyjściową baterii. Dzięki dostępowi do linii wysokiego napięcia czujnik ten realizuje również pomiar ciągłości izolacji. Czujnik ten najczęściej komunikuje się z głównym sterownikiem baterii za pomocą magistrali CAN, co zapewnia niezawodną i szybką wymianę danych niezbędnych do efektywnego zarządzania systemem baterii [5].

Obwód bezpieczeństwa

Obwody bezpieczeństwa w pojeździe to pierścieniowe układy elektryczne, które przechodzą przez komponenty oraz jednostki sterujące i mostki kontaktowe w złączach. Układy te są zasilane prądem z 12-woltowego systemu elektrycznego pojazdu. Głównym celem obwodów bezpieczeństwa jest monitorowanie i zapewnienie prawidłowego funkcjonowania kluczowych komponentów pojazdu, co umożliwia szybką detekcję usterek, które mogą wpłynąć na bezpieczeństwo użytkowników pojazdu. W obwodach tych stosuje się dwie główne metody generowania sygnału diagnostycznego, które umożliwiają monitorowanie ich działania oraz detekcję usterek. Pierwsza metoda polega na zasilaniu obwodu stałym prądem diagnostycznym o natężeniu, na przykład 10 mA, co pozwala na ciągłe monitorowanie jego stanu. Druga metoda opiera się na generowaniu sygnału o określonym przebiegu, na przykład fali prostokątnej o ustalonej amplitudzie i częstotliwości. Tego rodzaju sygnały są kluczowe do monitorowania ciągłości obwodu oraz jego prawidłowego działania. W przypadku przerwania obwodu bezpieczeństwa, na przykład w wyniku odłączenia złącza, odpowiednia jednostka sterująca natychmiast powiadamia interfejs diagnostyczny magistrali danych, taki jak magistrala CAN, o zaistniałej usterce. Powiadomienie to umożliwia szybką identyfikację problemu oraz podjęcie stosownych działań naprawczych, co zapewnia dalsze bezpieczne użytkowanie pojazdu [2].

Sterownik pakietu

Sterownik pakietu, umieszczony w okolicy pakietu bateryjnego, monitoruje temperaturę oraz napięcia na poszczególnych ogniwach. Posiada również funkcję balansowania, dzięki której wszystkie ogniwa mają ten sam poziom naładowania. Balansowanie ogniw jest niezwykle istotne dla zachowania optymalnej wydajności i długowieczności pakietu bateryjnego. W praktyce, nawet niewielkie różnice w pojemności i stanie naładowania poszczególnych ogniw mogą prowadzić do ich nierównomiernego zużycia. Aby zapewnić właściwe działanie całego systemu, sterownik komunikuje się z głównym sterownikiem baterii za pomocą galwanicznie izolowanej magistrali, takiej jak SPI (Serial Peripheral Interface) lub CAN (Controller Area Network) [80].

Zespół styczników

Zespół styczników odpowiada za zwieranie i rozwieranie linii wysokiego napięcia pomiędzy układem napędowym, ładowarką pokładową a baterią. Styczniki są elementami mechanicznymi, sterowanymi

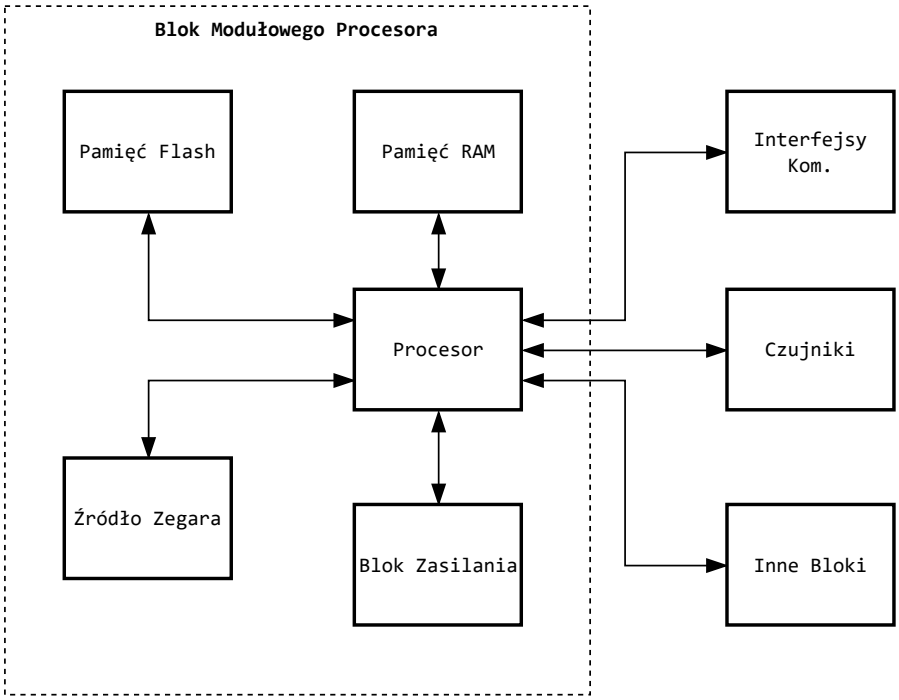
niskim napięciem. Dodatkowo stosuje się stycznik połączony w szeregu z rezystorem, którego zadaniem jest wstępne naładowanie układu przed włączeniem głównych przekaźników. Zapobiega to powstawaniu łuku elektrycznego, mogącego doprowadzić do zespawania styków głównych przekaźników [55].

Sterownik układu bateryjnego

Sterownik układu bateryjnego zarządza różnymi komponentami systemu bateryjnego, w tym systemami chłodzenia i ogrzewania baterii, aby utrzymać jej optymalną temperaturę pracy. Kontroluje pracę pompy chłodzącej, która cyrkuluje płyn chłodzący przez baterię, oraz automatycznie odłącza system w przypadku wykrycia krytycznego błędu, zapobiegając uszkodzeniom i zapewniając bezpieczeństwo. Inicjuje i nadzoruje proces balansowania ogniw po wyłączeniu zapłonu, aby utrzymać równomierne naładowanie ogniw. Wykonuje polecenia diagnostyczne przez złącze OBDII, zachowuje kody błędów oraz monitoruje obwody bezpieczeństwa. Sterownik monitoruje ogólny stan systemu bateryjnego, wpływając na decyzje dotyczące konserwacji i eksploatacji. Śledzi poziomy naładowania poszczególnych ogniw, co wpływa na strategie ładowania i decyzje operacyjne. Ocena pojemności ogniw i całego pakietu baterii oraz analiza starzenia się baterii są kluczowe dla przewidywania jej żywotności i wydajności. Przetwarza wartości dostarczone przez układ pomiaru izolacji przewodów wysokonapięciowych, wykrywając potencjalne uszkodzenia. Sterownik pełni również funkcje diagnostyczne, identyfikując i raportując problemy w systemie bateryjnym. Przechowuje i wyświetla wpisy dotyczące zdarzeń operacyjnych, wspomagając rozwiązywanie problemów. Komunikuje się z innymi sterownikami i systemami w pojeździe, zapewniając zintegrowane działanie. Wysyła komunikaty o błędach do interfejsu diagnostycznego pojazdu, wpływając na wyświetlanie ostrzeżeń i alertów dla kierowcy [5].

2.2 Systemy wbudowane

Urządzenia te można określić jako dedykowane systemy komputerowe, różniące się od pozostałych kategorii systemów komputerowych, takich jak komputery stacjonarne czy superkomputery. Typowo charakteryzują się one pewnymi ograniczeniami zarówno sprzętowymi, jak i programowymi w porównaniu do komputerów osobistych, w tym mniejszą mocą przetwarzania, zużyciem energii i ograniczonymi możliwościami sprzętowymi [59]. Ta definicja jednak ewoluuje w miarę rozwoju technologicznego i zmniejszania się kosztów implementacji rozmaitych komponentów sprzętowych oraz oprogramowania.



Rysunek 2.2: Diagram przedstawiający uproszczoną strukturę systemu wbudowanego

Rdzeniem każdego systemu wbudowanego jest mikrokontroler, czyli układ łączący w sobie procesor, pamięć RAM, pamięć flash i porty wejścia/wyjścia w jednym module (rys. 2.2). Przykładem takiego systemu jest sterownik samochodu elektrycznego, który odgrywa kluczową rolę w zarządzaniu pracą pojazdu, kontrolując silnik elektryczny, systemy ładowania baterii oraz inne funkcje pojazdu. Mikrokontrolery są powszechnie wykorzystywane w tych urządzeniach z uwagi na ich przystępną cenę, kompaktowe rozmiary oraz minimalne zapotrzebowanie energetyczne. Są one kluczowym komponentem dla działania tych urządzeń, umożliwiając interakcję z realnym światem poprzez porty wejścia/wyjścia oraz przetwarzanie danych i realizację określonych zadań. W tym kontekście ważne jest podkreślenie, że urządzenia są integrowane z układami mechanicznymi [79] [46].

Jednym z powszechnych aspektów, które są wykorzystywane w większości urządzeń wbudowanych, jest ich operowanie w czasie rzeczywistym. Zarządzanie czasem jest krytycznym zadaniem. Ważne jest nie tylko osiągnięcie odpowiedniego rezultatu, ale i osiągnięcie go w wymaganym momencie. System czasu rzeczywistego gwarantuje maksymalny czas reakcji od momentu otrzymania nowych danych wejściowych do ich przetwarzania [52]. Urządzenia wbudowane, z uwagi na swoją powszechność, szerokie zastosowanie oraz kluczową rolę w nowoczesnych technologiach, stanowią nieodłączny element pojazdów

elektrycznych, realizując specyficzne zadania w precyzyjnie zdefiniowanych ramach czasowych, często w interakcji z rzeczywistym światem. Projektowanie i wdrażanie tych systemów wymaga zrozumienia ograniczeń sprzętowych i programowych, oraz zdolności do pracy w środowisku multidyscyplinarnym, integrującym wiedzę z różnych obszarów inżynierii.

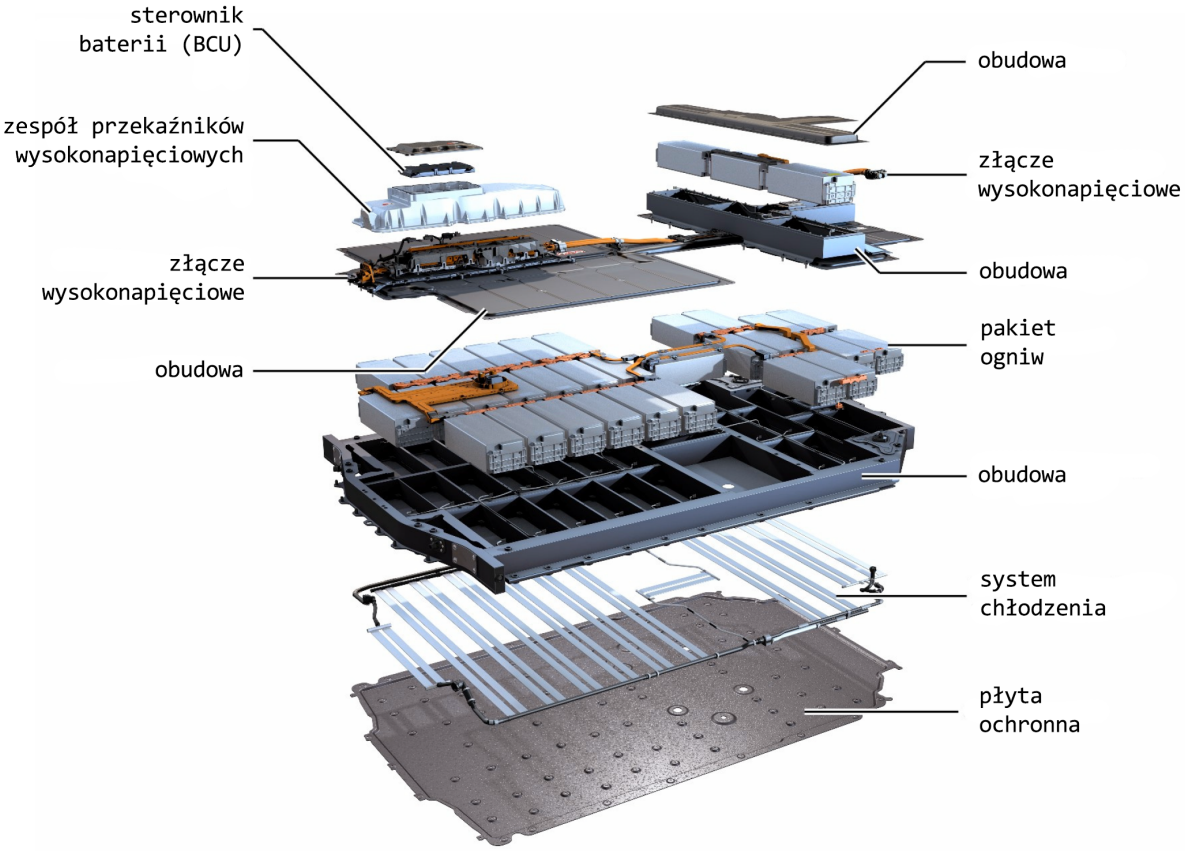
2.3 Funkcje bezpieczeństwa

Jednym z obszarów kompetencji firmy Dräxlmaier jest rozwój oraz produkcja komponentów dla pojazdów elektrycznych, z wyszczególnieniem układów bateryjnych. Firma ta dostarcza zaawansowane technologicznie rozwiązania, które znajdują zastosowanie w pojazdach elektrycznych produkowanych przez koncern VAG (Volkswagen Auto Group). Wśród nich znajduje się układ bateryjny składający się z zestawu ogniw o pojemności do 85 kWh, połączonych szeregowo, aby napięcie na zaciskach baterii wynosiło około 800 V w przypadku w pełni naładowanych ogniw. Takie rozwiązanie dostarcza moc na poziomie 500 kW do maksymalnie dwóch silników elektrycznych, zapewniając wysoką sprawność całego układu [5]. Wprowadzenie tak wysokiego napięcia wiąże się z ryzykiem porażenia prądem, które w większości przypadków może być śmiertelne. Dlatego zastosowano szereg technicznych środków zaradczych mających na celu zwiększenie bezpieczeństwa użytkowania.

System jest kontrolowany przez trzy rodzaje komponentów elektronicznych, kluczowych dla zapewnienia efektywności i bezpieczeństwa eksploatacji pojazdu elektrycznego. Pierwszy rodzaj odpowiada za pomiar prądu oraz napięć. Aby zapewnić wysoką wiarygodność pomiaru prądu, stosuje się dwa niezależne tory pomiarowe, co spełnia rygorystyczne wymogi bezpieczeństwa. Drugi rodzaj to układy monitorujące napięcie oraz temperaturę ogniw. Budowa baterii oparta jest o ogniwa w technologii Li-ion [71], łączone w pakiety zamykane w hermetycznej obudowie odpornej na uszkodzenia mechaniczne. Obudowa wyposażona jest w system regulacji temperatury zdolny do chłodzenia lub ogrzewania pakietów w zależności od warunków środowiskowych. Trzeci rodzaj stanowi sterownik baterii, którego zadaniem jest fuzja informacji z wymienionych sensorów oraz komunikacja z pojazdem. Podsumowując, dla wyżej opisanego systemu bateryjnego wyszczególnia się następujące wymagania [5] [48]:

- monitorowanie parametrów ogniw takich jak temperatura i napięcie,
- diagnostyka oraz kontrola systemu odpowiedzialnego za utrzymanie optymalnej temperatury ogniw,

- balansowanie poziomu napięć pomiędzy poszczególnymi ogniwami,
- wielopunktowy pomiar wysokiego napięcia,
- wyzwalanie bezpiecznika pirotechnicznego,
- pomiar stanu izolacji na przewodach wysokiego napięcia,
- diagnostyka i sterowanie przekaźników wysokiego napięcia,
- informowanie pozostałych systemów pojazdu o niebezpiecznych zdarzeniach takich jak przegrzanie.



Rysunek 2.3: Przekrój baterii zamontowanej w samochodzie Audi RS e-tron GT opracowanej przez firmę DRÄXLMAIER [60]

Wszystkie wymienione wyżej wymagania są istotne z punktu widzenia bezpieczeństwa, a funkcje, które je realizują, nazywane w tej pracy będą funkcjami bezpieczeństwa. Oprócz funkcji bezpieczeństwa realizowane są również takie funkcje, których działanie nie jest wymagane ze względów bezpieczeństwa. Dążeniem jest, by te dwie grupy funkcji nie oddziaływały na siebie wzajemnie [78]. Jednakże z powodów takich jak błąd ludzki nie zawsze jest to możliwe. Celem autorów oprogramowania przed każdym

wypuszczeniem stabilnej wersji oprogramowania jest jego rzetelne przetestowanie. Obecnie stosowane metody w przedsiębiorstwie pozostawiają miejsce na optymalizację procesu testowania produktu, nim trafi on do klienta. Opracowana dzięki tej pracy metodyka testowania sterowników systemów bateryjnych ma na celu obniżenie nakładów pracy w procesie testowania przy jednoczesnym wzroście jakości testów oraz niezawodności systemów.

Niniejszy rozdział koncentruje się na funkcjach bezpieczeństwa w kontekście przemysłu motoryzacyjnego, z naciskiem na standard ISO 26262 [44], który dostarcza ramy dla zapewnienia funkcjonalnego bezpieczeństwa pojazdów drogowych. Zawiera przegląd kluczowych koncepcji, takich jak Bezpieczeństwo Funkcyjne oraz Automotive Safety Integrity Level (ASIL), które oceniają ryzyko związane z awariami systemów motoryzacyjnych. Przedstawione są również procesy analizy zagrożeń i oceny ryzyka oraz dekompozycja systemów w celu optymalizacji kosztów i złożoności projektów. Omówiono przykłady konkretnych funkcji bezpieczeństwa, ich implementacji oraz mechanizmów mających na celu minimalizację ryzyka i zapewnienie ochrony życia, zdrowia, mienia oraz środowiska [70].

2.3.1 Bezpieczeństwo funkcyjne

Przemysł samochodowy od zawsze charakteryzował się wysokimi wymaganiami dotyczącymi bezpieczeństwa, wynikającymi z potencjalnych zagrożeń związanych z awarią systemów pojazdu. W miarę rozwoju technologii oraz wzrostu liczby elektronicznych i elektrycznych systemów w pojazdach, konieczne stało się opracowanie specjalistycznych norm, które umożliwią systematyczne podejście do zarządzania bezpieczeństwem tych systemów. ISO 26262 została wprowadzona, aby sprostać tym wymaganiom, oferując ramy dla funkcjonalnego bezpieczeństwa pojazdów drogowych, szczególnie tych o masie do 3,5 tony. Standard ten opiera się na podstawowej normie IEC 61508 [1], adaptując jej założenia do specyfiki przemysłu motoryzacyjnego.

Bezpieczeństwo funkcyjne odnosi się do zdolności systemu do wykonywania swoich funkcji bez powodowania ryzyka dla zdrowia i życia ludzi, oraz bez generowania zagrożeń dla mienia i środowiska. W kontekście systemów elektronicznych i elektrycznych, bezpieczeństwo funkcyjne koncentruje się na zachowaniu systemu po wystąpieniu awarii. Zamiast skupiać się na pierwotnej funkcjonalności, priorytetem jest minimalizacja ryzyka związanego z awarią systemu, co osiąga się poprzez implementację odpowiednich mechanizmów bezpieczeństwa.

Automotive Safety Integrity Level jest systemem klasyfikacji ryzyka stosowanym w normie ISO 26262, który ocenia potencjalne zagrożenia związane z systemami motoryzacyjnymi. ASIL określa

wymagania dotyczące bezpieczeństwa, które muszą zostać spełnione, aby zminimalizować ryzyko związane z awariami systemów. Ocena ASIL obejmuje trzy kluczowe kryteria: dotkliwość (severity), narażenie (exposure) oraz możliwość kontrolowania (controllability). Na podstawie tych kryteriów określa się poziom ASIL, który może wynosić od A do D, gdzie ASIL D jest najwyższym poziomem bezpieczeństwa.

- **QM (Quality Management)**: Jest to poziom zarządzania jakością, który oznacza, że nie są wymagane dodatkowe środki bezpieczeństwa poza standardowymi procedurami zarządzania jakością. Systemy lub komponenty sklasyfikowane jako QM nie stwarzają istotnego ryzyka dla bezpieczeństwa i dlatego nie wymagają zastosowania szczególnych środków bezpieczeństwa funkcyjnego. Przykłady obejmują elementy, których awaria nie prowadzi do poważnych konsekwencji, takich jak oświetlenie wewnętrzne pojazdu.
- **ASIL A**: Najniższy poziom zagrożenia, wymagający najmniej rygorystycznych środków bezpieczeństwa. ASIL A stosuje się do funkcji, których awaria może prowadzić do niewielkich obrażeń. Środki bezpieczeństwa są stosowane, ale w ograniczonym zakresie.
- **ASIL B**: Średni poziom zagrożenia, wymagający umiarkowanych środków bezpieczeństwa. Funkcje sklasyfikowane jako ASIL B mogą w przypadku awarii powodować poważniejsze obrażenia, ale nadal kontrolowane przez kierowcę lub system.
- **ASIL C**: Wysoki poziom zagrożenia, wymagający znaczących środków bezpieczeństwa. ASIL C odnosi się do funkcji, których awaria może prowadzić do ciężkich obrażeń i wymaga zastosowania zaawansowanych mechanizmów zabezpieczeń.
- **ASIL D**: Najwyższy poziom zagrożenia, wymagający najbardziej rygorystycznych środków bezpieczeństwa. ASIL D stosuje się do funkcji, których awaria może prowadzić do śmierci lub poważnych obrażeń wielu osób. Funkcje te wymagają najostrzejszych standardów bezpieczeństwa i najbardziej zaawansowanych mechanizmów ochrony.

Wyznaczanie ASIL odbywa się poprzez proces analizy zagrożeń i oceny ryzyka (HARA - Hazard Analysis and Risk Assessment) [63], który obejmuje:

- **Ocena dotkliwości (Severity - S)**: Określenie potencjalnego stopnia uszkodzenia ciała lub zagrożenia życia w przypadku wystąpienia awarii. Dotkliwość klasyfikowana jest na cztery

poziomy: S0 (brak obrażeń), S1 (lekkie do umiarkowanych obrażeń), S2 (poważne obrażenia zagrażające życiu), S3 (obrażenia śmiertelne).

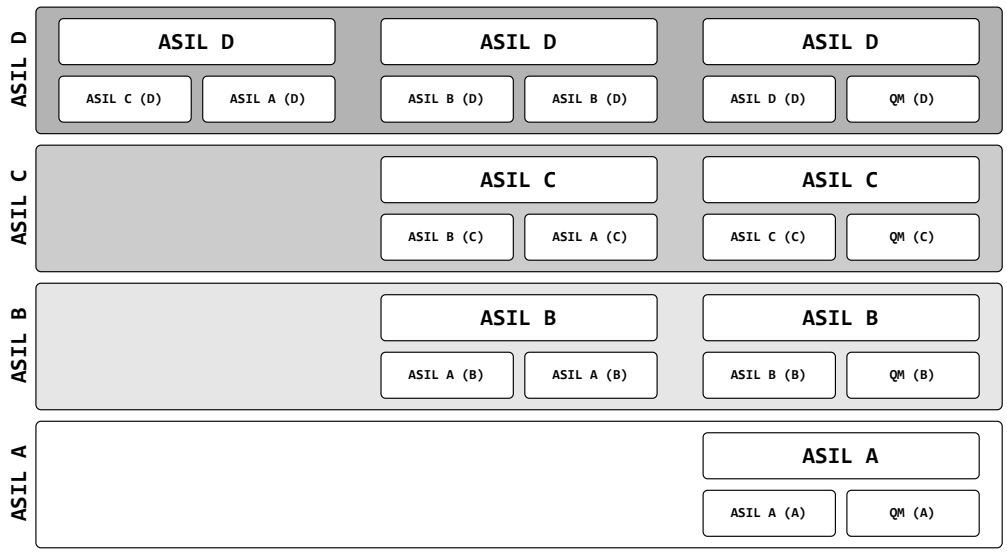
- **Ocena narażenia (Exposure - E)**: Określenie prawdopodobieństwa wystąpienia warunków operacyjnych prowadzących do zagrożenia. Narażenie klasyfikowane jest na pięć poziomów: E0 (niewiarygodnie małe), E1 (bardzo małe), E2 (małe), E3 (średnie), E4 (wysokie).
- **Ocena możliwości kontrolowania (Controllability - C)**: Określenie zdolności kierowcy lub innych uczestników ruchu do kontrolowania zagrożenia lub uniknięcia szkody. Możliwość kontrolowania klasyfikowana jest na cztery poziomy: C0 (ogólnie kontrolowalne), C1 (łatwe do kontrolowania), C2 (trudniejsze do kontrolowania), C3 (trudne do kontrolowania lub niekontrolowalne).

ASIL wyznacza się na podstawie kombinacji ocen dotkliwości, narażenia i możliwości kontrolowania, jak przedstawiono w Tabeli 2.1.

Dotkliwość (S)	**Narażenie (E)**	**Kontrola (C1)**	**Kontrola (C2)**	**Kontrola (C3)**
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A/QM
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Tabela 2.1: Tabela wyznaczania poziomu ASIL w ISO 26262 [44].

Dekompozycja w kontekście ISO 26262 odnosi się do procesu rozdzielania funkcji systemu na mniejsze, bardziej zarządzane komponenty, które mogą być oceniane i rozwijane niezależnie. Dekompozycja pozwala na obniżenie poziomu ASIL dla poszczególnych komponentów, co może prowadzić do znacznych oszczędności kosztów oraz uproszczenia procesu projektowania i walidacji [82]. Na przykład, zadanie o wysokim poziomie ASIL może być podzielone na kilka mniejszych zadań o niższych poziomach ASIL, co ułatwia spełnienie wymagań bezpieczeństwa przy jednoczesnym ograniczeniu złożoności systemu. Rysunek 2.4 ukazuje wszystkie dopuszczalne kombinacje poziomów ASIL.

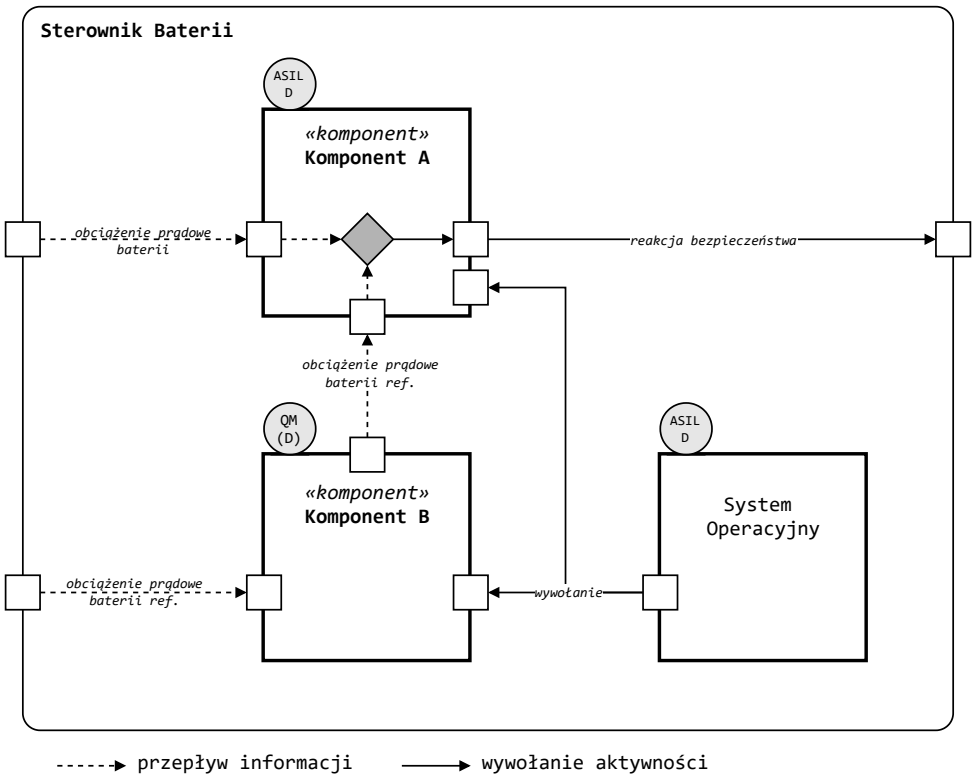


Rysunek 2.4: Dopuszczone kombinacje stosowane przy dekompozycji poziomów ASIL

Cel bezpieczeństwa

Cel bezpieczeństwa, zgodnie ze standardem ISO 26262, to wysokopoziomowe wymaganie mające na celu zapobieganie lub kontrolowanie zagrożeń w celu uniknięcia nieuzasadnionego ryzyka. Jego istota polega na zapewnieniu, że system działa bezpiecznie nawet w przypadku wystąpienia usterek, poprzez precyzyjne określenie sposobu unikania zdarzeń związanych z zagrożeniami. Realizacja celu bezpieczeństwa wymaga wdrożenia odpowiednich środków i mechanizmów, które minimalizują prawdopodobieństwo wystąpienia niebezpiecznych sytuacji oraz ograniczają skutki potencjalnych awarii.

Przykładem celu bezpieczeństwa może być system pomiaru prądu w akumulatorze samochodu elektrycznego. W tym kontekście cel bezpieczeństwa może brzmieć: „Bezpiecznik pirotechniczny zostanie aktywowany, gdy prąd przekroczy zdefiniowany próg. Celem tego jest ochrona przed zagrożeniami wynikającymi z nadmiernego przepływu prądu, który mógłby prowadzić do zwarcia, pożaru, stopienia izolacji przewodów oraz pojawienia się wysokiego napięcia na karoserii samochodu. Aktywacja bezpiecznika pirotechnicznego w momencie przekroczenia krytycznego progu prądu pozwala na szybkie odcięcie źródła zasilania, tym samym minimalizując ryzyko wystąpienia zagrożenia dla ludzkiego życia.



Rysunek 2.5: Architektura oprogramowania realizująca funkcję monitorowania obciążenia prądowego baterii

Aby osiągnąć określony cel bezpieczeństwa, należy zdefiniować i przypisać specyficzne Wymagania Funkcjonalnego Bezpieczeństwa (Functional Safety Requirements - FSR) w ramach projektu systemu. Wymagania te są następnie rozkładane na szczegółowe specyfikacje techniczne i implementowane w architekturze systemu. Przykładowo, w systemie pomiaru prądu, mogą być zastosowane następujące środki: wprowadzenie wielu czujników do pomiaru prądu, co pozwala na weryfikację poprawności danych i minimalizację ryzyka awarii pojedynczego czujnika; implementacja niezależnego systemu monitorowania, który aktywuje bezpiecznik pirotechniczny w momencie przekroczenia krytycznego progu prądu; zastosowanie zaawansowanych algorytmów detekcji błędów oraz mechanizmów korekty działania systemu w przypadku wykrycia nieprawidłowości.Przykład architektury oprogramowania realizujący wyżej opisany przykład znajduje się na rysunku 2.6. Dodatkowo rysunek ten przedstawia użycie podejścia dekompozycji przy realizacji funkcji o wysokim poziomie ASIL [29].

Funkcja bezpieczeństwa

Funkcja bezpieczeństwa w oprogramowaniu odnosi się do specyficznych działań lub operacji zaimplementowanych w systemie w celu eliminacji nieuzasadnionego ryzyka wynikającego z wadliwego działania. Funkcje te mają na celu łagodzenie zagrożeń, które mogą wynikać z wadliwego działania systemów elektrycznych i elektronicznych, w tym komponentów oprogramowania. W ramach normy ISO 26262 funkcja oprogramowania jest istotnym elementem, który wspiera realizację celu oprogramowania, zwłaszcza w kontekście celów bezpieczeństwa. Cel oprogramowania, szczególnie cel bezpieczeństwa, stanowi najwyższego poziomu wymaganie, którego zadaniem jest zapobieganie zdarzeniom niebezpiecznym. Cele te są następnie rozkładane na szczegółowe wymagania dotyczące bezpieczeństwa funkcjonalnego, które są realizowane przez konkretne funkcje oprogramowania.

Funkcje oprogramowania charakteryzują się określonymi wejściami i wyjściami. Te funkcje przetwarzają dane wejściowe, generując odpowiednie wyjścia, zgodnie z ustalonymi wymaganiami bezpieczeństwa, aby zapewnić prawidłowe i bezpieczne działanie oprogramowania w różnych warunkach. Ponadto funkcje te mogą być konstrukcjami związanymi z czasem, co oznacza, że dynamiczne aspekty architektury oprogramowania, takie jak czas wykonania i kolejność, są kluczowe dla zapewnienia terminowego i poprawnego wykonania funkcji bezpieczeństwa. Dotyczy to spełnienia wymagań czasu rzeczywistego oraz zapewnienia szybkich reakcji na wejścia, co jest niezbędne do utrzymania bezpieczeństwa systemu.

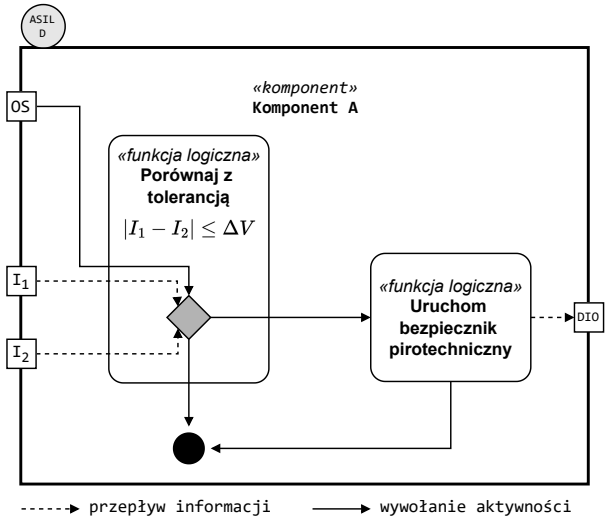
Poziom ASIL dla funkcji bezpieczeństwa musi być wyższy niż QM, ponieważ ASIL określa poziom redukcji ryzyka wymagany do zapewnienia bezpieczeństwa funkcjonalnego. Wyższe poziomy ASIL, od A do D (gdzie D jest najwyższy), wskazują na większą krytyczność i wymagają bardziej rygorystycznych środków bezpieczeństwa. Obejmują one szczegółową analizę, walidację i implementację mechanizmów bezpieczeństwa, które mają na celu złagodzenie ryzyka związanego z poważnymi zagrożeniami. Klasyfikacja QM jest stosowana do funkcji, które nie są krytyczne z punktu widzenia bezpieczeństwa, dlatego nie wymagają tak rygorystycznych środków ochrony [70].

Rys. 2.6 przedstawia przykład funkcji bezpieczeństwa. Na rysunku widoczny jest komponent A, który jest oznaczony jako posiadający najwyższy poziom ASIL (D), co wskazuje na jego krytyczność w kontekście bezpieczeństwa. Komponent ten realizuje dwie główne funkcje logiczne:

- **Porównanie z tolerancją**:
  - **Wejścia**: I1 i I2 - obciążenie prądowe baterii mierzone przez dwa niezależne układy pomiarowe.
  - **Logika działania**: Funkcja ta przetwarza dane wejściowe, porównując je z ustalonymi tolerancjami. Jeżeli wartości wejściowe I1 i I2 przekroczą określone limity, funkcja identyfikuje potencjalne zagrożenie.
  - **Wyjście**: Wynik porównania, który w przypadku przekroczenia tolerancji inicjuje kolejną funkcję.
- **Uruchomienie bezpiecznika pirotechnicznego**:
  - **Wejścia**: Sygnał z funkcji porównania z tolerancją.
  - **Logika działania**: Gdy funkcja porównania z tolerancją wykryje przekroczenie limitów, uruchamia bezpiecznik pirotechniczny. Ta akcja jest kluczowym mechanizmem bezpieczeństwa, który fizycznie odłącza system lub komponenty, aby zapobiec dalszym zagrożeniom.
  - **Wyjście**: Aktywacja bezpiecznika pirotechnicznego, co skutkuje fizycznym odłączeniem niebezpiecznego elementu systemu.

2.3.2 Funkcje bezpieczeństwa czujnika multimodalnego

Spośród wyżej opisanych komponentów, to czujnik multimodalny został wybrany jako obiekt do badań. Powody wyboru są trzy. Po pierwsze, jest on specyfikowany jako ASIL D, co stanowi najwyższy poziom bezpieczeństwa w przemyśle samochodowym. ASIL D (Automotive Safety Integrity Level D) jest najściślejszą klasą bezpieczeństwa zgodnie z normą ISO 26262, która określa wymagania dotyczące funkcjonalnego bezpieczeństwa systemów elektrycznych i elektronicznych w pojazdach. Wybór komponentu o tak wysokim poziomie bezpieczeństwa gwarantuje, że badania przeprowadzone na tym czujniku będą miały bezpośrednie zastosowanie do najbardziej krytycznych aplikacji, gdzie niezawodność i szybka reakcja na potencjalne zagrożenia są kluczowe.



Rysunek 2.6: Przykład funkcji bezpieczeństwa.

Po drugie, wszystkie funkcje czujnika określają czas reakcji na niebezpieczną sytuację, co jest kluczowe dla zapewnienia bezpieczeństwa pojazdu elektrycznego. Funkcje takie jak pomiar prądu, detekcja prądu zwarciowego czy monitorowanie napięcia i temperatury są fundamentalne dla prawidłowego działania systemu bateryjnego. Szybka i precyzyjna reakcja na zmieniające się warunki pozwala na uniknięcie awarii i potencjalnie niebezpiecznych sytuacji, takich jak przegrzanie czy zwarcia, które mogłyby prowadzić do poważnych uszkodzeń systemu lub nawet pożaru.

Po trzecie, autor pracy jest zaznajomiony z oprogramowaniem i specyfikacją opisywanego czujnika, co umożliwia dokładną analizę i interpretację wyników badań. Znajomość specyfikacji technicznej oraz dostęp do odpowiednich narzędzi programistycznych pozwala na precyzyjne przeprowadzenie eksperymentów oraz interpretację uzyskanych danych. Poniżej dokonano przeglądu funkcji czujnika oraz wyboru jednej z nich w celach badawczych.

Pomiar natężenia prądu

Jedną z głównych funkcji czujnika multimodalnego jest kompleksowe monitorowanie przepływu prądu pomiędzy baterią, układem napędowym a układem ładowania pojazdu elektrycznego. Czujnik ten zapewnia precyzyjne pomiary, ocenę, diagnostykę oraz transmisję danych dotyczących natężenia prądu, realizując pomiar w układzie redundancji sprzętowej przez dwa boczniki. Dzięki temu system

jest niezależny i spełnia wymagania poziomów bezpieczeństwa ASIL B (D) zgodnie z normą ISO 26262. Czujnik dwukrotnie i niezależnie mierzy natężenie prądu baterii oraz synchronizuje te pomiary z napięciem baterii w czasie krótszym niż 1 ms.

Proces pozyskiwania danych dzieli się na dwa główne tory: akwizycję wartości natężenia prądu baterii 1 oraz akwizycję wartości natężenia prądu baterii 2. Każdy tor spełnia szczegółowe wymagania dotyczące filtrowania wartości prądu, aktualizowania tych wartości w czasie krótszym niż 773 µs oraz dostarczania surowych i skorygowanych danych co 1 ms i 2424 ms. Przyjęty reżim czasowy ma na celu prawidłową detekcję przeciążeń obwodów baterii poprzez kontrolę bieżącej wartości z ustalonymi limitami dopuszczalnymi.

Ocena natężenia prądu obejmuje niezależne przetwarzanie danych z obu torów pomiarowych. Czujnik oblicza fizyczne wartości prądu, dokonuje kompensacji temperaturowej na podstawie aktualnych odczytów temperatury bocznika oraz generuje wartości średnich ruchomych w celu korekcji offsetu i detekcji przeciążeń prądowych. System priorytetowo traktuje średnie wartości prądu baterii, przypisując natężenie prądu na podstawie najwyższej wartości średniej spośród obu torów pomiarowych, co zapewnia precyzyjne wykrywanie zdarzeń przeciążeniowych.

Funkcja monitorowania natężenia prądu baterii wykorzystuje otrzymane wartości do wykrywania stanów przetężeniowych na podstawie zdefiniowanych progów. Jeśli natężenie prądu przekracza te progi lub zostaną wykryte warunki diagnostyczne, takie jak otwarty obwód, przekroczenie dopuszczalnych wartości natężenia prądu czy zwarcia, system identyfikuje zdarzenie przetężeniowe. Aby uniknąć fałszywych alarmów, implementowany jest mechanizm odfiltrowania, który wymaga, aby stan przetężeniowy trwał przez określony czas przed jego potwierdzeniem i uzyskaniem stabilnych wartości natężenia prądu do oceny.

Pomiar, przetwarzanie i analiza natężenia prądu umożliwiają również diagnozowanie niesprawności takich jak utrata bocznika (otwarty obwód), wartości natężenia prądu poza zakresem (zarówno niskie, jak i wysokie), zwarcia do masy oraz zwarcia do baterii. Proces oceny stanu obwodów bateryjnych jest realizowany co 10 ms dla obu torów pomiarowych. System generuje różne sygnały diagnostyczne w zależności od wartości i ich ważności, co pozwala na precyzyjne wykrywanie usterek oraz wdrażanie odpowiednich działań korygujących, zapewniając ciągłość działania systemu.

Na zakończenie, dane dotyczące natężenia prądu są przekazywane do innych funkcji systemowych, obejmując wartości natężenia prądu ładowania i rozładowania. Dzięki temu system może precyzyjnie monitorować i reagować na dynamiczne zmiany natężenia prądu w czasie rzeczywistym, wspierając zarówno operacyjne, jak i diagnostyczne funkcje systemu.

Pomiar wysokich napięć

Czujnik multimodalny realizuje pomiary wysokich napięć. Wartości napięć mierzone są na zaciskach w punktach takich jak bateria, bezpiecznik pirotechniczny oraz styczniki. Wartości zmierzone są uśredniane w krótkich odstępach czasu (5 ms), co pozwala na wygładzenie fluktuacji i zapewnienie stabilnych wyników pomiarowych. Bezpieczeństwo i niezawodność są zapewniane poprzez ciągłą diagnostykę, która obejmuje wykrywanie odchyleń od dopuszczalnych wartości, zwarć do masy lub akumulatora. Diagnostyka gwarantuje szybkie wykrywanie wszelkich usterek, a pomiary, których dotyczą te usterki, są oznaczane jako nieważne. Kontrole wiarygodności dodatkowo weryfikują poprawność pomiarów poprzez ich porównanie z przewidywanymi wartościami, co zwiększa niezawodność systemu.

Pomiar rezystancji izolacji

Pomiar rezystancji izolacji pozwala ocenić stan izolacji kabli wysokiego napięcia i jest wykonywany między przewodnikami wysokiego napięcia a karoserią pojazdu. Pomiar stanu izolacji kabli wysokiego napięcia w nowoczesnych pojazdach elektrycznych jest kluczową funkcją zapewniającą bezpieczeństwo i niezawodność układów baterii wysokiego napięcia. Układ pomiaru rezystancji izolacji pozwala na ciągłe monitorowanie stanu izolacji. Dane pomiarowe są zbierane co 10 milisekund, gdzie czujnik mierzy surowe wartości napięcia, które następnie są kalibrowane i normalizowane w celu odfiltrowania szumów pomiarowych. Przetworzone dane są używane do obliczania rezystancji izolacji i identyfikowania wszelkich odchyleń, które mogą wskazywać na pogorszenie lub uszkodzenie izolacji. System posiada rozbudowane funkcje diagnostyczne do wykrywania warunków takich jak zbyt niskie napięcie, zbyt wysokie napięcie, zwarcie do ziemi oraz zwarcie do baterii. Przeprowadza on również kontrole wiarygodności i diagnostykę zacięć przełączników, aby zapewnić niezawodność pomiarów. W przypadku wykrycia błędów system przechodzi w odpowiednie tryby błędów i unieważnia błędne odczyty. Dane dotyczące izolacji, w tym wartości rezystancji izolacji dla przewodów plusowych, minusowych oraz ogólny status izolacji, są przekazywane do odpowiednich komponentów pojazdu w celu dalszego przetwarzania i podjęcia działań.

Pomiar temperatury bocznika

Proces ten obejmuje pozyskiwanie wartości temperatury za pośrednictwem dwóch niezależnych kanałów w regularnych odstępach czasu, typowo co sekundę. Surowe dane temperatury z tych kanałów są

przetwarzane w celu obliczenia współczynników w odniesieniu do średnich wartości wysokiego napięcia, które następnie są używane do obliczenia średnich kroczących. Średnie te stabilizują dane, co pozwala systemowi na dokładne określenie temperatury bocznika przy użyciu wartości kalibracyjnych. W celu zapewnienia niezawodności pomiarów temperatury system przeprowadza diagnostykę w celu wykrycia takich warunków jak zbyt niska temperatura, zbyt wysoka temperatura, zwarcie do masy i zwarcie do baterii. Nieprawidłowe dane są identyfikowane i odpowiednio przetwarzane, aby zapobiec wpływowi błędnych odczytów na wydajność systemu. Dodatkowo, przeprowadzana jest kontrola wiarygodności poprzez porównanie dwóch temperatur bocznika w określonym oknie czasowym, zapewniając spójność i dokładność pomiarów w akceptowalnych granicach. Przetworzone i zweryfikowane dane temperatury bocznika są następnie przekazywane do innych komponentów systemu do dalszego przetwarzania i podejmowania decyzji.

Bezpiecznik pirotechniczny

Funkcja ta monitoruje różnorodne sygnały, w tym sygnały związane z kolizjami, przeciążeniami oraz komunikację w sieci CAN (Controller Area Network), które mogą wskazywać na konieczność uruchomienia bezpiecznika pirotechnicznego. Po wykryciu wypadku, sterownik poduszek powietrznych ocenia sytuację i informuje odpowiednie moduły, czy powinien zostać aktywowany bezpiecznik pirotechniczny. Dodatkowo funkcja monitoruje przepływ prądu w celu wykrycia przeciążenia. W momencie spełnienia warunków wyzwalających, system inicjuje sekwencję szybkiego otwarcia, generując impulsy aktywacyjne (co najmniej 1,75 A przez 0,5 ms) przy użyciu redundantnych wyjść dla zapewnienia niezawodności. System niezależnie ocenia potrzebę szybkiego otwarcia na różnych ścieżkach, aby efektywnie zarządzać odłączeniem wysokiego napięcia. Diagnostyka monitoruje gotowość i integralność bezpiecznika oraz jego mechanizmów wyzwalających, sprawdzając oporność zapalnika pirotechnicznego oraz diagnozując otwarte lub zwarte obwody w przełącznikach wysokiej i niskiej strony. Regularne testy diagnostyczne weryfikują stan systemu i zapewniają dokładną reakcję w razie potrzeby. Sekwencja działania bezpiecznika pirotechnicznego obejmuje: wykrycie wyzwalacza, generowanie impulsów aktywacyjnych, odłączenie ścieżki wysokiego napięcia oraz diagnostykę po aktywacji, która potwierdza działanie i aktualizuje status systemu.

2.3.3 Identyfikacja funkcji bezpieczeństwa czujnika multimodalnego

NA podstawie analizy budowy i działania czujnika multimodalnego można zidentyfikować podstawowe parametry funkcji bezpieczeństwa:

- **Liczba funkcji logicznych:** liczba niezależnych operacji logicznych realizowanych przez każdą funkcję.
- **Liczba wejść:** liczba sygnałów wejściowych wykorzystywanych przez funkcję.
- **Liczba wyjść:** liczba sygnałów wyjściowych generowanych przez funkcję.
- **Czas przetwarzania:** czas potrzebny na przetworzenie danych przez funkcję.
- **ASIL** poziom integralności bezpieczeństwa funkcji zgodnie z normą ISO 26262.

Dobór parametrów ma na celu rozróżnianie funkcji pod względem stopnia złożoności.

Funkcja	**Liczba funkcji logicznych**	**Liczba wejść**	**Liczba wyjść**	**Czas przetwarzania**	**ASIL**
Pomiar prądu	32	10	10	1 ms	B (D) + B (D)
Detekcja prądu zwarciowego	4	2	1	1 ms	D
Pomiar wysokich napięć	7	9	5	10 ms	B
Pomiar izolacji	9	6	7	10 ms	A
Pomiar temperatury bocznika	11	2	2	1 s	D
Bezpiecznik pirotechniczny	11	7	9	1 ms	D

Tabela 2.2: Tabela zawierająca liczbę funkcji logicznych, wejść, wyjść, czas przetwarzania oraz poziom ASIL dla poszczególnych funkcji.

2.3.4 Wybór funkcji bezpieczeństwa

Powyższe zestawienie funkcji bezpieczeństwa pozwoliło na wybór do dalszych badań funkcji bezpiecznika pirotechnicznego. Po pierwsze, analiza danych wskazuje, że reprezentuje ona średni poziom złożoności w systemie. Średnią liczbę funkcji logicznych w badanym systemie wynosi 12.33, podczas

gdy dla 'Bezpiecznika pirotechnicznego' jest to 11, co plasuje ją w pobliżu tej wartości. Ponadto średnia liczba wejść wynosi 6.00, a omawiana funkcja ma ich 7, co również wskazuje na bliskie pokrewieństwo ze średnią. Warto zauważyć, że liczba wyjść wynosi 9, co jest wartością znacząco powyżej średniej wynoszącej 5.67. Taka charakterystyka czyni 'Bezpiecznik pirotechniczny' funkcją o złożoności, która jest reprezentatywna dla systemu, lecz z wyższą niż przeciętną liczbą interakcji wyjściowych.

Kolejnym ważnym aspektem jest mechatroniczny charakter funkcji 'Bezpiecznik pirotechniczny'. Łączy ona elementy elektroniczne i mechaniczne, co jest kluczowe dla systemów mechatronicznych, które wymagają integracji sygnałów elektronicznych z mechaniką. Bezpiecznik pirotechniczny realizuje zaawansowane przetwarzanie sygnałów oraz precyzyjne sterowanie mechanicznymi elementami wykonawczymi. Dzięki temu funkcja ta stanowi idealny przypadek do testowania złożonych interakcji systemowych i zapewnia istotne dane na temat działania całego systemu w realistycznych warunkach operacyjnych.

2.3.5 Opis wybranej funkcji bezpieczeństwa

Głównymi zadaniami funkcji bezpieczeństwa bezpiecznika pirotechnicznego są aktywacja bezpiecznika oraz jego diagnostyka. Aktywacja bezpiecznika polega na wyzwoleniu mechanizmu detonującego w odpowiednich warunkach. Diagnostyka obejmuje badanie rezystancji bezpiecznika oraz aktywną diagnozę kondensatora buforowego. Kondensator ten ma za zadanie zapewnić wystarczającą poziom energii do detonacji bezpiecznika, niezależnie od warunków zewnętrznych. Diagnostyka kondensatora polega na sprawdzaniu czasu utraty ładunku, co pozwala określić jego teoretyczną pojemność. Prawidłowe funkcjonowanie tej diagnostyki jest kluczowe dla zapewnienia, że kondensator jest w stanie dostarczyć wymaganą energię do bezpiecznego i skutecznego uruchomienia bezpiecznika pirotechnicznego. Opisane funkcje oraz sygnały zostały przedstawione na rysunku 2.7.

Funkcja *Analiza Zewnętrznych Wyzwalaczy* monitoruje sygnały logiczne 'poduszka powietrzna' oraz 'wyzwalacz'. Gdy jeden z sygnałów przejdzie w stan aktywny, wyjścia 'wyzwalanie ścieżki 1' i 'wyzwalanie ścieżki 2' zostają ustawione na stan aktywny. Sygnały te z definicji nigdy nie przyjmą stanu aktywnego w tym samym czasie. W przypadku braku aktywności tych sygnałów funkcja analizuje wartość prądu baterii. Jeśli wartość prądu jest wadliwa lub przekracza ustalony próg, również aktywowane są wyjścia 'wyzwalanie ścieżki 1' i 'wyzwalanie ścieżki 2'. Wyjścia tej funkcji obejmują 'wyzwalanie ścieżki 1', 'wyzwalanie ścieżki 2' oraz 'status wyzwalacza'.

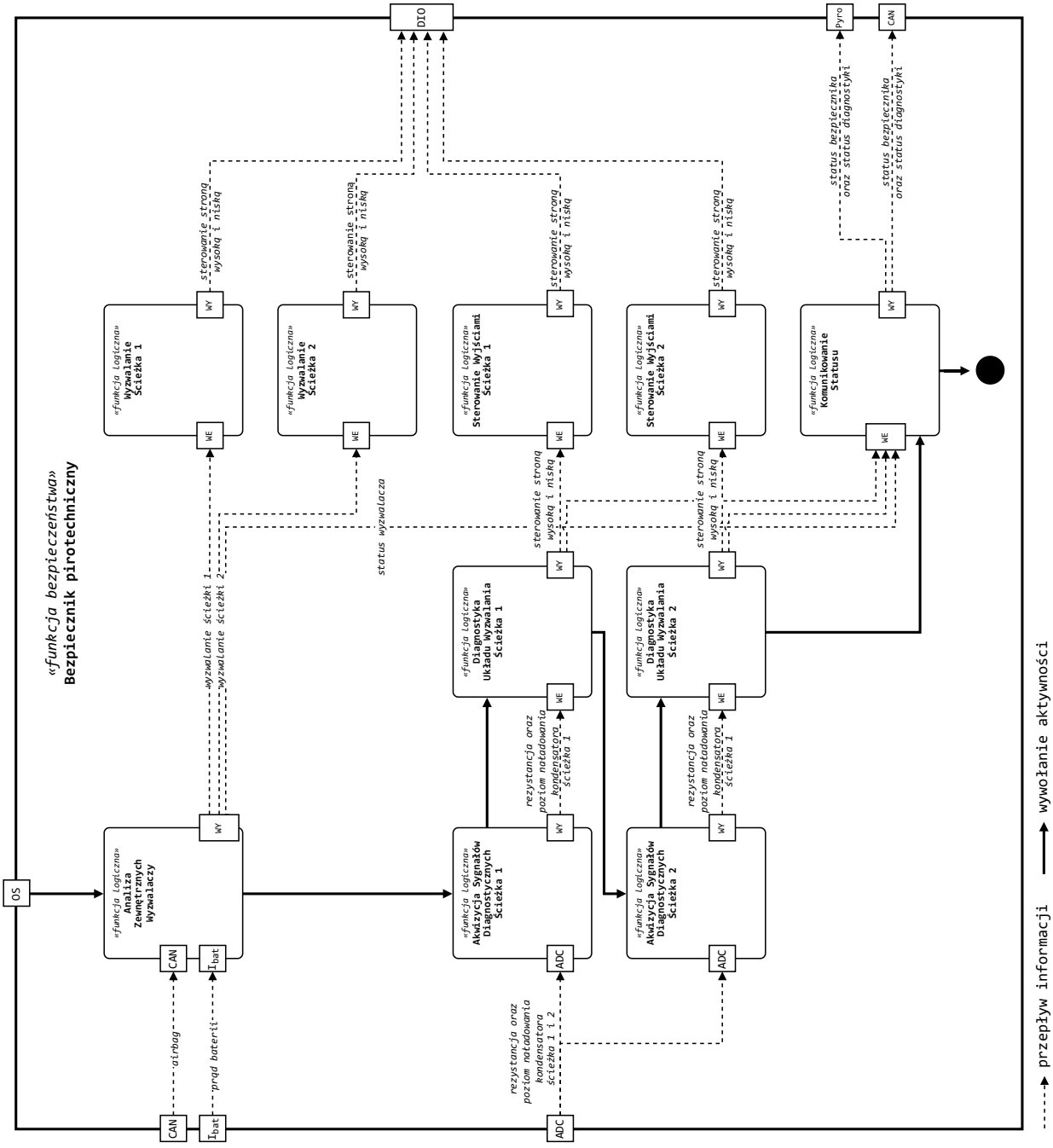
Funkcja *Wyzwalanie Ścieżka 1*, gdy sygnał wejściowy 'wyzwalanie ścieżki 1' jest aktywny, ustawia wyjścia 'sterowanie stroną wysoką' oraz 'sterowanie stroną niską' na stan wysoki (logiczne 1) dla ścieżki 1. Podobnie, *funkcja Wyzwalanie Ścieżka 2* działa na sygnale wejściowym 'wyzwalanie ścieżki 2'. Gdy sygnał ten jest aktywny, wyjścia 'sterowanie stroną wysoką' i 'sterowanie stroną niską' są ustawiane na stan wysoki (logiczne 1) dla ścieżki 2.

Akwizycja Sygnałów Diagnostycznych Ścieżka 1 zajmuje się przetwarzaniem sygnałów wejściowych pochodzących z przetwornika ADC. Proces ten obejmuje kompensację surowej wartości offsetem z danych kalibracyjnych, sprawdzanie, czy wartość mieści się w zakresie (nie jest zwarta do masy lub zasilania), uśrednianie pięciu próbek oraz konwersję do wartości fizycznej, np. rezystancji w omach lub miliwoltach. Wyjściem tej funkcji jest 'status diagnostyki ścieżki 1'. *Akwizycja Sygnałów Diagnostycznych Ścieżka 2* wykonuje identyczne operacje dla sygnałów wejściowych ścieżki 2, a wynikiem jest 'status diagnostyki ścieżki 2'.

Diagnostyka Układu Wyzwalania Ścieżka 1 analizuje sygnały wejściowe, aby przeprowadzić diagnostykę bezpiecznika. Proces ten obejmuje sprawdzenie, czy rezystancja mieści się w tolerancji 100 Ω +/- 10 Ω oraz sprawdzenie kondensatora poprzez jego ładowanie i rozładowywanie. Jeżeli wszystkie wyniki diagnostyczne są prawidłowe, status diagnozy funkcji zostaje ustawiony na 'sprawny'; w przeciwnym przypadku zostaje ustawiony na 'niesprawny'. Po uruchomieniu systemu, dopóki wszystkie diagnostyki nie zostaną przeprowadzone przynajmniej raz, status wynosi 'w trakcie'. Diagnostyka Układu Wyzwalania Ścieżka 2 działa analogicznie, sprawdzając parametry rezystancji i kondensatora dla ścieżki 2, i ustala status diagnostyki na 'sprawny' lub 'niesprawny', z początkowym stanem 'w trakcie'.

Sterowanie Wyjściami zapewnia bezpieczne zarządzanie wyjściami, zapobiegając jednoczesnemu aktywowaniu stron wysokiej i niskiej, co mogłoby uruchomić bezpiecznik pirotechniczny. Funkcja ta przetwarza sygnały wejściowe 'sygnał sterowania ścieżka 1' oraz 'sygnał sterowania ścieżka 2', aby odpowiednio wysterować wyjścia.

Komunikowanie Statusu interpretuje sygnały diagnostyczne i określa ogólny stan systemu. Jeśli obie ścieżki są sprawne, stan jest 'sprawny'; w przeciwnym razie 'niesprawny'. Jeśli bezpiecznik został uruchomiony, stan bezpiecznika pozostaje 'aktywowany' aż do ponownego uruchomienia. W przeciwnym razie stan to 'gotowy' lub 'uszkodzony' w przypadku wykrycia błędów diagnostyki. Wyjściem są sygnały statusu systemu.



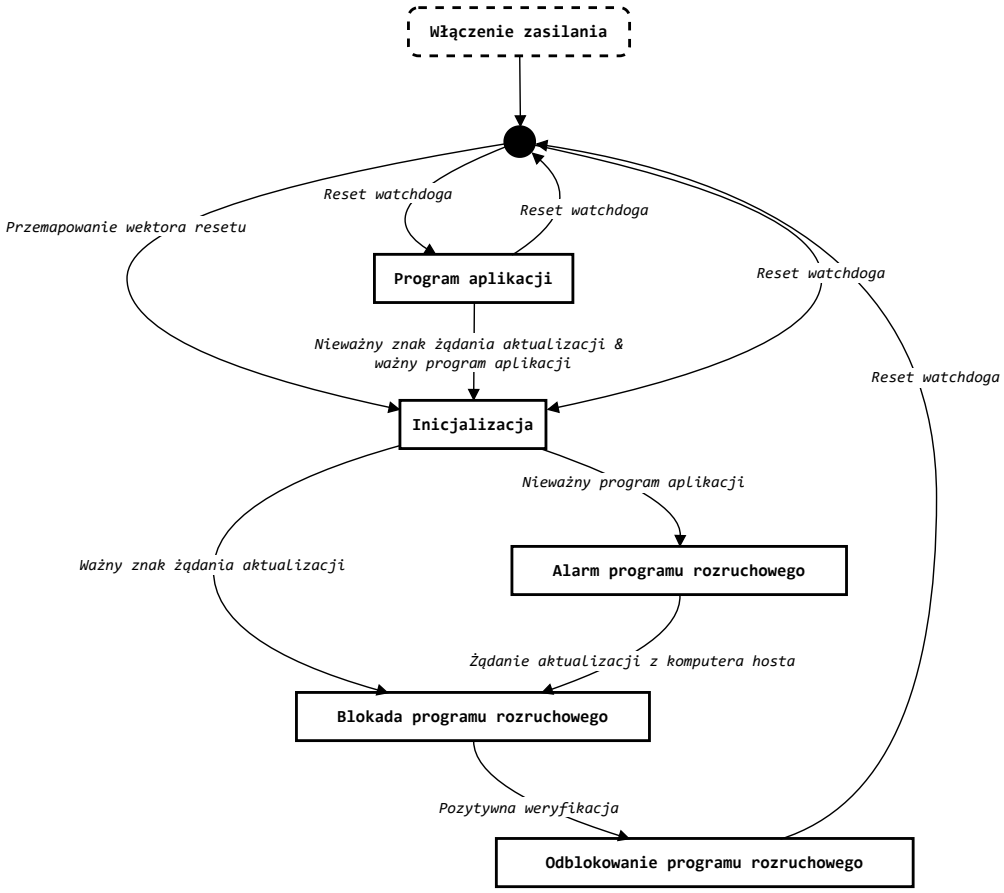
Rysunek 2.7: Schemat blokowy funkcji bezpieczeństwa - bezpiecznik pirotechniczny

2.4 Oprogramowanie w systemach wbudowanych

Nieodzownym elementem systemów wbudowanych jest oprogramowanie, które w dalszym ciągu będzie nazywane oprogramowaniem wbudowanym. Typowo, oprogramowanie wbudowane dzieli się na aplikację oraz bootloader [68], gdzie aplikacja odpowiada za realizację głównych funkcji systemu. W pracy doktorskiej główny nacisk położono na testowanie aplikacji działającej w systemach wbudowanych, z pominięciem bootloadera. Bootloader jest niewielkim programem odpowiedzialnym za inicjalizację sprzętu oraz załadowanie głównego oprogramowania systemu do pamięci operacyjnej. Jego podstawowe zadania obejmują sprawdzenie integralności oprogramowania, konfigurację niezbędnych rejestrów oraz przeprowadzenie wstępnej konfiguracji systemu, zanim kontrola zostanie przekazana do aplikacji. Zdecydowano o tym ze względu na to, że bootloader, mimo że jest kluczowym elementem inicjującym uruchomienie systemu, nie jest aktywny podczas działania aplikacji. Po zakończeniu procesu uruchamiania i załadowaniu głównego oprogramowania do pamięci operacyjnej, kontrola jest przekazywana do aplikacji, a bootloader przechodzi w stan bezczynności.

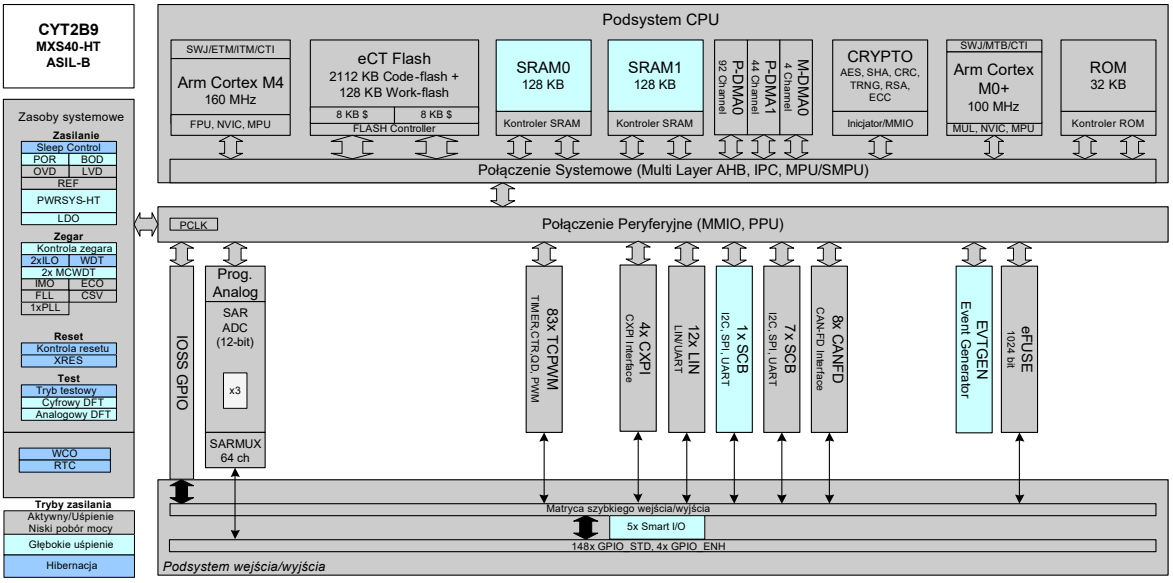
2.4.1 Aplikacja wbudowana

Aplikacja realizuje funkcjonalność produktu, zarządzając peryferiami mikrokontrolera, w tym przetwornikami analogowo-cyfrowymi (ADC - Analog Digital Converter), interfejsami komunikacyjnymi takimi jak SPI (Serial Peripheral Interface) i CAN (Controller Area Network), a także modulatorami szerokości impulsu (PWM - Pulse Width Modulation) jak i wieloma innymi. Peryferia mikrokontrolera, w kontekście pojazdów elektrycznych, są fundamentem dla zapewnienia precyzji, bezpieczeństwa oraz efektywności operacyjnej [73]. Należyte zarządzanie ADC, SPI, CAN oraz PWM jest krytyczne dla maksymalizacji wydajności pojazdu. Ponadto, elementy ogólnego przeznaczenia, takie jak porty wejścia/wyjścia (GPIO - General Purpose Input Output), timery i mechanizmy przerwań, stanowią podstawę dla szerokiego spektrum funkcji w systemach wbudowanych pojazdów. Na przykład, GPIO może być stosowane do monitorowania stanów czujników, takich jak przełączniki krańcowe, oraz do sterowania funkcjami takimi jak oświetlenie czy elementy pirotechniczne. Timery i mechanizmy przerwań umożliwiają precyzyjne zarządzanie czasem i sekwencjonowanie zadań, co jest kluczowe dla utrzymania synchronizacji między różnymi procesami oraz dla zapewnienia odpowiedzi systemu w czasie rzeczywistym [47].



Rysunek 2.8: Aplikacja oraz bootloader podczas startu systemu wbudowanego

Oprócz wymienionych istnieją inne peryferia mikrokontrolera, które, choć stosowane rzadziej, odgrywają istotną rolę w specyficznych aplikacjach. Należą do nich na przykład magistrala FlexRay, która jest wykorzystywana w bardziej wymagających systemach komunikacyjnych w samochodach. Rysunek 2.9 przedstawia różnorodne peryferia, podkreślając ich zróżnicowanie. Każde z tych peryferiów pełni kluczową rolę w odpowiednich systemach, gdzie wymagana jest szczególna funkcjonalność, jak na przykład w zaawansowanych systemach informacji i rozrywki kierowcy i pasażerów, telematyce pojazdowej, czy w algorytmach sterowania napędem elektrycznym.



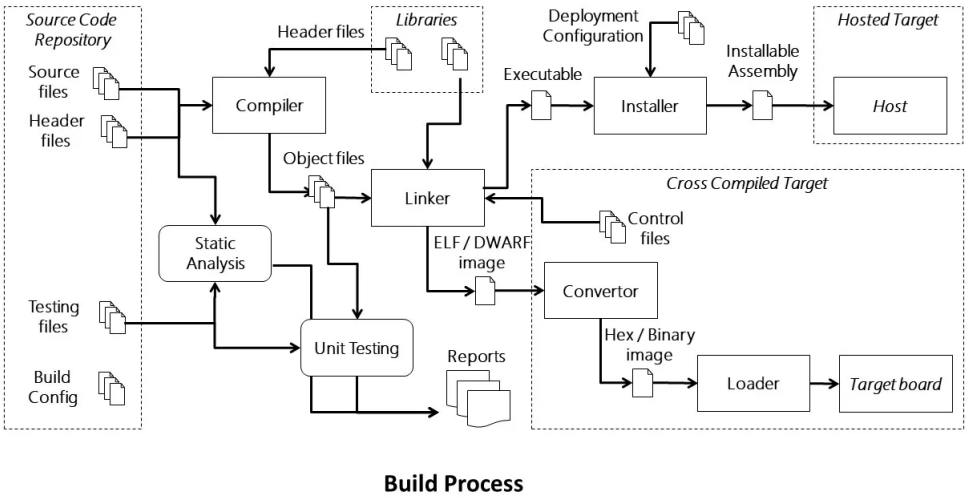
Rysunek 2.9: Diagram blokowy przedstawiający peryferia mikrokontrolera Traveo T2G przeznaczonego dedykowanego do aplikacji w przemyśle samochodowym [7]

2.4.2 Języki programowania

Oprogramowanie wbudowane w większości układów wbudowanych stosowanych w motoryzacji jest pisane w językach oprogramowania C i C++. Powodem tego jest to, że języki te oferują optymalną równowagę między kontrolą niskiego poziomu a wydajnością, umożliwiając bezpośredni dostęp do zasobów sprzętowych przy jednoczesnym zapewnieniu struktur i abstrakcji ułatwiających programowanie. Dzięki temu programiści mogą efektywnie tworzyć niezawodne i stabilne systemy, które są kluczowe w zastosowaniach motoryzacyjnych. Języki programowania niskiego poziomu, takie jak assembler, stosowane są sporadycznie jako sekcje w kodzie C. Wykorzystuje się je głównie do optymalizacji krytycznych fragmentów kodu lub do bezpośredniego sterowania specyficznymi funkcjami sprzętowymi, które wymagają precyzyjnej kontroli. Jednak ze względu na zwiększoną złożoność i trudność w zarządzaniu kodem assemblera, jego użycie jest ograniczone do sytuacji, gdzie korzyści z optymalizacji przewyższają koszty związane z utrzymaniem takiego kodu. Język C znajduje zastosowanie w systemach, w których sterowniki są relatywnie proste, na przykład w sterownikach pedału przyspieszenia oraz elektrycznego wspomagania kierownicy. Ponadto jest wykorzystywany w aplikacjach, gdzie priorytetem jest bezpieczeństwo, takich jak sterowniki baterii wysokonapięciowych pojazdów elektrycznych. Z kolei C++, z jego dodatkowymi abstrakcjami i mechanizmami obiektowymi, stosowany jest w bardziej złożonych systemach, takich jak sterowniki silnika, systemy rozpoznawania znaków, czy w rozrywce, gdzie obok niego znajdują zastosowanie również inne języki wysokiego poziomu, w tym język Java [15].

2.4.3 Proces projektowania i wytwarzania oprogramowania

Rozwój oprogramowania wbudowanego w przemyśle samochodowym jest procesem złożonym i wieloetapowym, który wymaga szczegółowej wiedzy i umiejętności technicznych [30]. Proces ten, rozpoczynając od pisania kodu źródłowego, poprzez kompilację, linkowanie, tworzenie pliku binarnego, aż do wgrywania oprogramowania przy użyciu debugera i finalnego debugowania, stanowi kluczowy element w tworzeniu zaawansowanych systemów pojazdów. Pisanie oprogramowania jest pierwszym i fundamentalnym krokiem w tworzeniu aplikacji dla systemów wbudowanych w pojazdach. W tym kontekście, istotne jest stosowanie systemów kontroli wersji, takich jak SVN czy GIT, które umożliwiają efektywne zarządzanie zmianami w kodzie oraz współpracę w zespołach programistycznych. Kontrola wersji jest nie tylko praktyką zwiększającą efektywność i bezpieczeństwo pracy, ale również wymogiem normy ISO26262, odnoszącej się do bezpieczeństwa funkcjonalnego w pojazdach. W kontekście standardów programistycznych, nie można pominąć znaczenia inicjatywy MISRA (Motor Industry Software Reliability Association), która publikuje zbiory zaleceń dotyczących bezpiecznego pisania oprogramowania w językach C i C++. Cel tych zaleceń jest prosty: minimalizacja ryzyka błędów i zapewnienie wysokiej jakości oraz niezawodności kodu, co jest kluczowe w aplikacjach krytycznych, jakimi są systemy wbudowane w pojazdach [75].



Rysunek 2.10: Proces rozwoju oprogramowania wbudowanego

Rozwijanie oprogramowania wbudowanego w przemyśle samochodowym wymaga przyjęcia holistycznego podejścia do procesu tworzenia, które wykracza poza tradycyjne metodyki. W tym kontekście,

znaczącą rolę odgrywa model Automotive SPICE (Software Process Improvement and Capability dEtermination), znany także jako ASPICE. Historia ASPICE sięga końca lat 90. XX wieku, kiedy to inicjatywa została zapoczątkowana przez grupę czołowych niemieckich przedsiębiorstw motoryzacyjnych, w tym gigantów jak BMW, Bosch, Continental, DaimlerChrysler oraz Volkswagen. Zasadniczym celem, jaki przyświecał twórcom ASPICE, było stworzenie ujednoliconych ram dla oceny oraz usprawniania procesów projektowania oprogramowania, mających na celu nie tylko optymalizację istniejących metod, ale również promowanie wysokich standardów jakości i bezpieczeństwa produktów. Pierwsza wersja modelu ASPICE, opublikowana w 2003 roku pod nazwą V-Model for Software Development (VDA 6.3), wykorzystywała metodologię V-Model. Jest to podejście, które akcentuje kluczową rolę testowania i weryfikacji na każdym etapie cyklu życia oprogramowania, podkreślając ich znaczenie dla finalnej jakości produktu. Rok 2005 przyniósł ze sobą drugą edycję ASPICE, która wprowadziła Model Oceny Procesu (PAM). PAM składa się z zestawu wytycznych i kryteriów służących do oceny efektywności oraz efektywności procesów tworzenia oprogramowania specyficznych dla branży motoryzacyjnej. Od tamtego czasu model ASPICE przeszedł kilka rewizji, każdorazowo zwiększając swoją precyzję i użyteczność. Dzisiaj ASPICE jest szeroko stosowany w całym sektorze motoryzacyjnym jako niezbędna platforma do oceny i udoskonalania procesów inżynierii oprogramowania, uzyskując uznania wielu organizacji motoryzacyjnych i stając się istotnym standardem w branży [38].

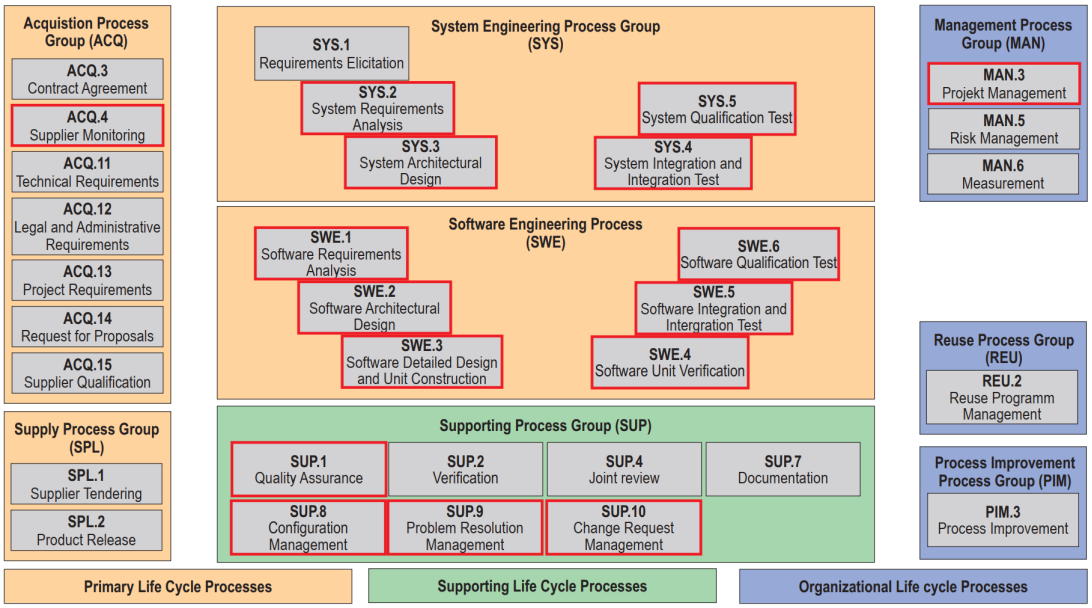
Przejście od koncepcji do gotowego produktu w dziedzinie inżynierii oprogramowania jest procesem skomplikowanym i wieloetapowym. Kluczowe jest tu nie tylko zrozumienie wymagań, ale również umiejętne zarządzanie nimi na każdym kroku tworzenia oprogramowania. Proces ten, składający się z sześciu podstawowych etapów, definiowany jest przez model ASPICE, który zapewnia kompleksowy przewodnik po procesach projektowania, implementacji i weryfikacji oprogramowania w branży motoryzacyjnej [32]:

- **SWE.1 Analiza wymagań oprogramowania** - Proces ten obejmuje zrozumienie i dokumentację wymagań systemowych oraz klienta, które przekładają się na wymagania oprogramowania. Wymagania te muszą być mierzalne i testowalne, co stanowi klucz do pomyślnej realizacji projektu. Dokładność tej fazy jest decydująca dla jakości i skuteczności całości przedsięwzięcia.
- **SWE.2 Projektowanie architektury oprogramowania** - Faza projektowania architektury oprogramowania polega na definiowaniu struktury systemu i głównych komponentów oprogramowania, a także zasad ich wzajemnych interakcji i integracji. Architektura powinna wspierać wszystkie zidentyfikowane wymagania oprogramowania, jednocześnie zapewniając jego łatwość

utrzymania i rozszerzalność.

- **SWE.3 Szczegółowe projektowanie i konstrukcja komponentów oprogramowania** - Etap ten dotyczy szczegółowego projektowania komponentów oprogramowania i ich realizacji. Konstrukcja komponentów oprogramowania wiąże się z kodowaniem poszczególnych modułów zgodnie z wcześniej zdefiniowanymi specyfikacjami i standardami.
- **SWE.4 Weryfikacja komponentów oprogramowania** - Proces weryfikacji komponentów oprogramowania ma na celu testowanie i przegląd kodu na poziomie modułów w celu sprawdzenia, czy spełniają one określone wymagania i czy są wolne od błędów. Techniki weryfikacyjne obejmują testy jednostkowe, które są istotne dla zapewnienia wysokiej jakości oprogramowania.
- **SWE.5 Integracja oprogramowania i testy integracyjne** - Proces integracji oprogramowania polega na łączeniu poszczególnych komponentów i weryfikacji ich współpracy. Testy integracyjne skupiają się na sprawdzeniu, czy zintegrowane komponenty prawidłowo współdziałają i realizują zaplanowane funkcjonalności.
- **SWE.6 Testy kwalifikacyjne oprogramowania** - Testy kwalifikacyjne oprogramowania mają na celu potwierdzenie, że całkowicie zintegrowane oprogramowanie spełnia wszelkie wymagania klienta oraz użytkownika końcowego, co jest kluczowe dla zapewnienia satysfakcji obu stron. Proces ten obejmuje kompleksowe testowanie funkcjonalne, wydajnościowe oraz zgodności z wymaganiami, które, jako niezbędne elementy, pozwalają na potwierdzenie gotowości produktu do wprowadzenia na rynek.

W kontekście szczegółowego rozwoju oprogramowania w branży samochodowej, kluczowe jest zrozumienie, że SWE.x reprezentuje jedynie fragment szerszej metodyki Automotive SPICE, odniesienia, do której można znaleźć na rysunku 2.11. Jest to ważne dla zapewnienia spójności procesów i wysokiej jakości końcowego produktu. Automotive SPICE składa się z różnych grup procesów, wśród których można wyróżnić Grupę Procesów Akwizycji (ACQ), System Engineering Process Group (SYS), Software Engineering Process Group (SWE), jak również Grupę Procesów Zarządzania (MAN), Grupę Procesów Wsparcia (SUP), Grupę Procesów Użytkowania (REU), oraz Grupę Procesów Poprawy Procesów (PIM). Każda z nich obejmuje kluczowe działania, takie jak na przykład nadzorowanie dostawców (ACQ.4), analiza i projektowanie architektury systemu (SYS.2 i SYS.3), zarządzanie ryzykiem (MAN.5), czy zapewnienie jakości (SUP.1). Koncentrując się na SWE.5, czyli etapie Integracji oprogramowania i testowania integracji, zaznaczyć należy, że prezentowana w pracy doktorskiej metodyka testowania



Rysunek 2.11: ASPICE

będzie zlokalizowana właśnie w tej części modelu ASPICE. Testy integracyjne są tu kluczowe dla zapewnienia, że wszystkie komponenty oprogramowania współpracują ze sobą poprawnie, zanim produkt zostanie wypuszczony na rynek.

2.4.4 Wyzwania rozwoju oprogramowania wbudowanego w przemyśle motoryzacyjnym

Rozwój oprogramowania wbudowanego w przemyśle samochodowym to jedno z największych wyzwań współczesnej inżynierii. Skomplikowanie i rozmiar oprogramowania w pojazdach osiągają poziomy nieznane dotąd w tej branży, ponieważ złożoność systemów rośnie wykładniczo. Tak dynamiczny rozwój wprowadza nowe wyzwania, w tym ryzyko błędów oprogramowania, zagrożenia bezpieczeństwa czy problemy z integracją milionów linii kodu. Wskazuje to na konieczność ciągłej pracy nad doskonaleniem metod testowania i analizy systemów oprogramowania w branży motoryzacyjnej. Przywołując cele niniejszej pracy, wskazuje się na konieczność identyfikacji i rozwiązania problemów związanych z zależnościami czasowymi w oprogramowaniu wbudowanym, co ma kluczowe znaczenie dla poprawy bezpieczeństwa i niezawodności pojazdów.

2.4.5 Metoda wsparta modelem

W przemyśle samochodowym, gdzie błąd w oprogramowaniu może mieć bezpośredni wpływ na bezpieczeństwo użytkowników, priorytetem jest bezpośrednia kontrola nad kodem źródłowym i jego dokładna weryfikacja. W firmie Dräxlmaier preferowane są metody zapewniające większą precyzję i bezpośredni nadzór nad tworzonym oprogramowaniem. Tym samym, skupienie się na tradycyjnych technikach programowania i ręcznym pisaniu kodu w języku C, wspieranym przez narzędzia do kontroli wersji i standardy pisania kodu, wynika nie tylko z przywiązania do sprawdzonych metod, ale również z konieczności spełnienia najwyższych standardów bezpieczeństwa funkcjonalnego [75] [47].

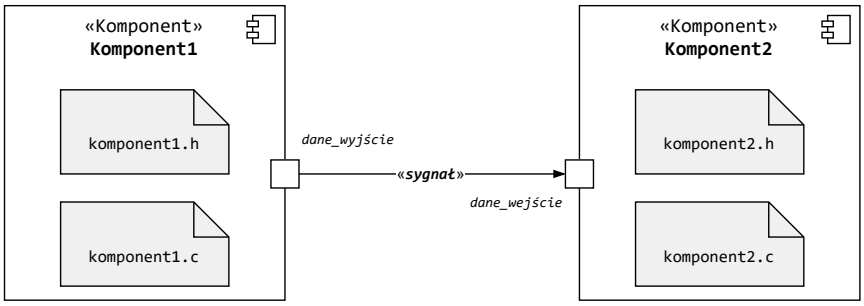
2.5 Wpływ struktury kodu oprogramowania na działanie układów wbudowanych

W oprogramowaniu wbudowanym istnieje struktura kodu wynikająca z zasad stosowanego języka programowania. Pomimo zdefiniowanej struktury kodu może dojść do nieprawidłowego działania programu. Dla potrzeb dalszych rozważań można przyjąć pewien fragment oprogramowania odpowiedzialny za realizacje określonego algorytmu będziemy nazywali komponentem.

2.5.1 Komponenty

Procedura, będąca podstawowym budulcem oprogramowania wbudowanego, może być częścią większego komponentu programowego lub też jego najprostszą implementacją. Ten komponent można rozumieć jako zbiór procedur, które są zarówno publiczne, jak i prywatne. Procedury wspólnie realizują określone zadanie, przyczyniając się do funkcjonalności całego systemu. Z abstrakcyjnego punktu widzenia, komponent stanowi logiczną całość, która może być postrzegana jako niezależny moduł. Natomiast w wymiarze fizycznym, komponent to jeden lub więcej plików źródłowych (.c) oraz plików nagłówkowych (.h), które razem tworzą program. Pliki źródłowe (.c) zawierają definicje procedur, implementując logikę działania poszczególnych komponentów, podczas gdy pliki nagłówkowe (.h) eksponują interfejs komponentu, definiując deklaracje procedur publicznych oraz typów danych, które mogą być wykorzystywane przez inne komponenty systemu. Taki podział sprzyja modularności oraz ukrywaniu implementacji, co jest zgodne z zasadą enkapsulacji - jedną z fundamentalnych koncepcji programowania. Odnosząc się do procedur, stanowią one kluczowe elementy komponentu. Każda

procedura posiada swoją specyfikację, określającą co robi (swoje zadanie) oraz jakie argumenty przyjmuje i jaki typ wartości zwraca. W kontekście oprogramowania napisanego w języku C, procedury dzielą się na publiczne i prywatne. Procedury publiczne stanowią interfejs komponentu, przez który komunikuje się on z resztą systemu. Są one dostępne dla innych komponentów i mogą być przez nie wywoływane w celu wykonania określonych zadań. Z kolei procedury prywatne są ukryte przed innymi częściami systemu i służą wyłącznie do wewnętrznych celów danego komponentu, co przyczynia się do lepszej organizacji kodu oraz jego bezpieczeństwa [66] [70].



Rysunek 2.12: Wizualizacja pokazująca kompozycję składającą się z dwóch komponentów, które wymieniają między sobą dane [66]

2.5.2 Wymiana danych

Ryzyko związane z nieprawidłowym działaniem oprogramowania wbudowanego może wynikać z kilku przyczyn: błędów komunikacji między komponentami, nieprawidłowej struktury procedur, użyciem nieprawidłowych poleceń języka do zapisu określonego algorytmu. Aby wyjaśnić istotę problemów z komunikacją komponentów, należy zwrócić uwagę na sposób, w jaki te komponenty komunikują się z otoczeniem. Kluczową rolę odgrywają tutaj interfejsy, będące zestawem procedur lub globalnych zmiennych, które umożliwiają wymianę danych między komponentami lub z zewnętrznymi systemami. Interfejsy te definiują, w jaki sposób komponenty mogą być wykorzystywane, co znacząco wpływa na modułowość oraz możliwości integracji systemu. Przykładowo, procedury publiczne stanowią część interfejsu komponentu, przez który jest on dostępny dla innych elementów systemu. Taka procedura może być zdefiniowana w pliku nagłówkowym (.h) i zaimplementowana w pliku źródłowym (.c), co zapewnia klarowne oddzielenie interfejsu od implementacji. Przykładem może być procedura dodaj(int a, int b), która realizuje dodawanie dwóch liczb i jest eksponowana do użytku przez inne komponenty. Z kolei globalne zmienne, choć rzadziej używane ze względu na ryzyko związane z bezpieczeństwem i trudnościami w utrzymaniu kodu, także mogą pełnić rolę w komunikacji między

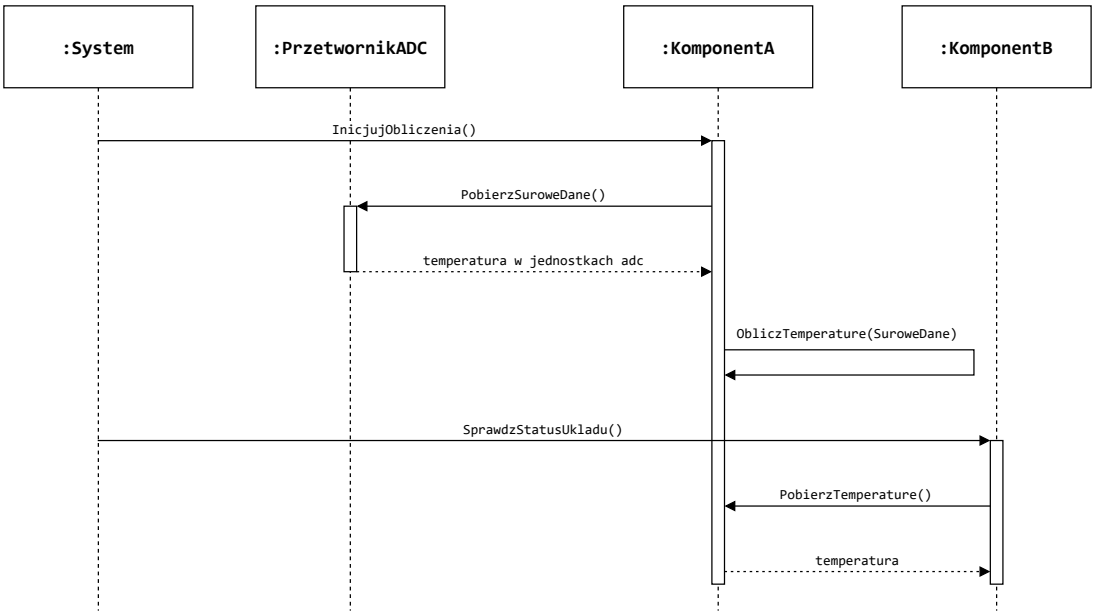
komponentami. Na przykład, zmienna `extern int licznik;` zadeklarowana w pliku nagłówkowym, może być współdzielona między różnymi komponentami, umożliwiając im dostęp do wspólnej wartości licznika. Jednakże nie jest to bezpieczna metoda na wymianę danych i stanowi ona jedynie przykład. Używanie globalnych zmiennych może prowadzić do problemów związanych z zarządzaniem stanem i współbieżnością, gdyż różne wątki lub procesy mogą jednocześnie modyfikować tę samą zmienną, co może prowadzić do nieprzewidywalnych błędów i trudności w debugowaniu [16].

2.5.3 Wyzwalanie aktywności

W nawiązaniu do przedstawionej wcześniej analizy interfejsów komponentów oprogramowania, istotne jest zrozumienie, w jaki sposób te elementy są aktywowane do wykonania określonych zadań. W szczególności, procedury wywoływane, będące specjalnym rodzajem interfejsu, odgrywają kluczową rolę w działaniu komponentów. Procedury te, charakteryzujące się brakiem bezpośrednich parametrów wejściowych oraz zwracanych wartości, stanowią punkt wejścia do komponentu, inicjując jego działanie. Tym samym, działają one na zasadzie sygnałów pobudzających komponent do realizacji zadania, nie przekazując mu jednak bezpośrednio danych do przetworzenia. Jako przykład można przytoczyć procedurę wywoływaną `ObliczTemperature`, która w momencie aktywacji pobiera surowe dane z przetwornika analogowo-cyfrowego (ADC). Następnie, na podstawie tych danych, procedura przeprowadza obliczenia w celu przekształcenia otrzymanej wartości na temperaturę wyrażoną w stopniach Celsiusza z dokładnością do jednego miejsca po przecinku. Taka specyfika działania procedur wywoływanych wskazuje na ich zasadniczą różnicę w porównaniu z klasycznymi procedurami, które wymagają podania argumentów i zwracają wyniki swojego działania. Rozważając sposób, w jaki komponent jest pobudzany do wykonania określonych procedur, należy podkreślić rolę zdarzeń systemowych lub sygnałów zewnętrznych. Komponent oczekuje na określone sygnały, które inicjują wywołanie procedur. Ten mechanizm pozwala na asynchroniczną pracę komponentów, co jest szczególnie ważne w złożonych systemach, gdzie wielozadaniowość i efektywne zarządzanie zasobami odgrywają kluczową rolę. W rezultacie procedury wywoływane umożliwiają efektywną implementację logiki biznesowej komponentu, jednocześnie minimalizując konieczność bezpośredniej interakcji z pozostałymi częściami systemu.

Odnosząc się do wcześniej omówionego sposobu aktywacji komponentów przez procedury wywoływane, nie należy zapomnieć o roli systemów operacyjnych w tym kontekście. System operacyjny, będący fundamentalnym składnikiem zarówno prostych, jak i skomplikowanych urządzeń elektronicznych, zarządza zasobami sprzętowymi i oprogramowaniem. Dzięki temu, użytkownicy oraz aplikacje

mogą efektywnie wykorzystywać dostępne zasoby systemowe. Systemy te, działając jako pośrednik między sprzętem a oprogramowaniem użytkownika, pełnią kluczową procedurę w zarządzaniu pamięcią, procesami, danymi oraz interakcjami między urządzeniami. Wykorzystują one różnorodne mechanizmy takie jak przerwania, sygnały i zdarzenia systemowe, które aktywują komponenty oprogramowania do działania. Na przykład, w odpowiedzi na przerwanie od zegara systemowego, system operacyjny może zainicjować procedurę, która odświeża odczyty z czujników, eliminując potrzebę ciągłego monitorowania tych urządzeń przez procesor.



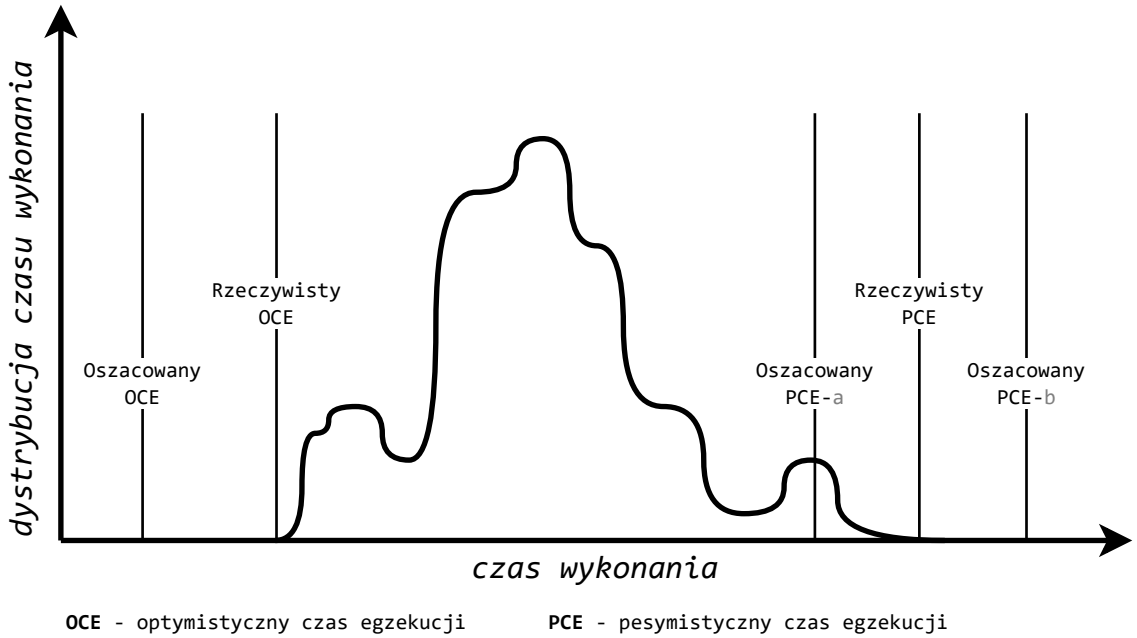
Rysunek 2.13: Przykład wywołań procedur na diagramie sekwencji

2.6 Zależności czasowe

W kontekście systemów czasu rzeczywistego istnieje fundamentalna potrzeba zapewnienia, że poszczególne funkcjonalności zaimplementowane w komponentach systemu reagują w ściśle określonym czasie. Odnosząc się do wcześniej opisanej definicji systemów czasu rzeczywistego (por. rozdział 2.2), należy podkreślić, że istotą tych systemów jest nie tylko poprawność logiczna wyników, ale także ich poprawność czasowa. Zatem implementacja poszczególnych funkcjonalności w komponentach musi być tak zaprojektowana, aby spełniała narzucone wymagania czasowe. Oznacza to, że czas potrzebny na wykonanie określonych operacji musi być przewidywalny i niezmienny, co umożliwia skuteczną kontrolę zachowania systemu w czasie rzeczywistym. Należy zauważyć, że w zależności od specyfiki

implementacji oraz kombinacji danych, poszczególne komponenty mogą wykonywać różną liczbę instrukcji. Ta zmienna liczebność instrukcji przekłada się bezpośrednio na zmienne czasy wykonania. Z tego powodu istotne jest, aby czas reakcji komponentów był poddany ścisłej kontroli i był zgodny z wymaganiami czasowymi całego systemu. W praktyce, różnice w czasach wykonania poszczególnych funkcjonalności mogą prowadzić do zakłóceń w działaniu systemu, a nawet do nieprzewidywalnych błędów. Dlatego też, zarządzanie zależnościami czasowymi między poszczególnymi komponentami stanowi kluczowy aspekt projektowania systemów czasu rzeczywistego. Ich właściwe zrozumienie i skuteczne uwzględnienie podczas projektowania są niezbędne dla zapewnienia nie tylko poprawności logicznej, ale także poprawności czasowej całego systemu [31].

W inżynierii oprogramowania związanej z systemami czasu rzeczywistego kluczowe jest precyzyjne oszacowanie czasu wykonania zadań. Poniższy tekst przedstawia wizualizację dwóch istotnych koncepcji: Optymistycznego Czasu Egzekucji (OCE) oraz Pesymistycznego Czasu Egzekucji (PCE). Dodatkowo omówione zostają warianty tych czasów, które pozwalają na głębsze zrozumienie ich wpływu na efektywność i niezawodność działania systemu.



Rysunek 2.14: Wizualizacja OCE oraz PCE

1. **PCE (Pesymistyczny Czas Egzekucji)** - Maksymalny czas wykonania zadania w najmniej korzystnych, ale możliwych do wystąpienia warunkach operacyjnych, nieuwzględniających awarii systemu, kluczowy w przypadkach, gdy wymagana jest gwarancja nieprzekraczania określonego limitu czasu.

2. **PCE-a** - Wariant Pesymistycznego Czasu Egzekucji, wskazujący na ryzyko niedoszacowania czasu wykonania, co może prowadzić do błędnej oceny zdolności systemu do zakończenia zadania w założonym czasie.

3. **PCE-b** - Wariant Pesymistycznego Czasu Egzekucji, reprezentujący przeszacowanie czasu wykonania, które może skutkować nieoptymalnym wykorzystaniem zasobów, przez niepotrzebnie zakładane buforowanie czasowe w planowaniu działania systemu.

4. **OCE (Optymistyczny Czas Egzekucji)** - Minimalny możliwy czas wykonania zadania, istotny w systemach, gdzie szybkie reakcje mogą prowadzić do problemów, takich jak niepożądane oscylacje czy zakłócenia.

5. **Rzeczywisty czas wykonania** - Czas, który faktycznie zajmuje wykonanie zadania, mierzony przy użyciu precyzyjnych narzędzi pomiarowych, odzwierciedlający rzeczywistą wydajność systemu.

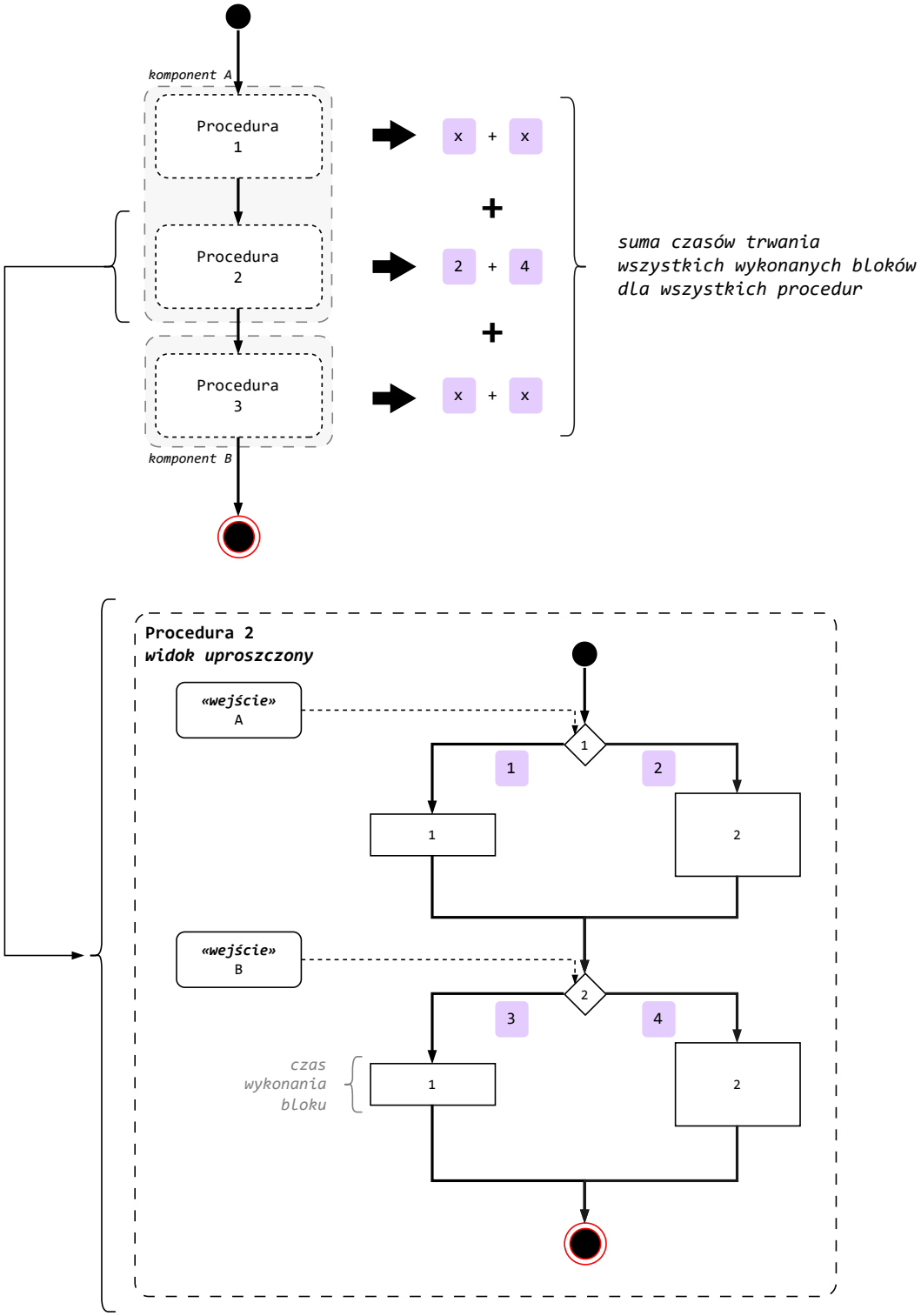
6. **Oszacowany czas wykonania** - Przewidywany czas wykonania zadania na podstawie modeli symulacyjnych lub analitycznych, który może być podatny na błędy estymacji i wymaga walidacji poprzez rzeczywiste pomiary.

2.6.1 Algorytm koncepcyjny - kontekst

W celu zobrazowania znaczenia zależności czasowych w oprogramowaniu układów wbudowanych można posłużyć się przykładem trzech funkcji realizujących określone zadanie. Na rysunku 2.15 zaprezentowano diagram opisujących współdziałanie procedur, na którym można wyróżnić:

- **Blok instrukcji:** Obejmuje zbiór instrukcji, które są wykonywane sekwencyjnie jako jedna jednolita ścieżka. Każdy blok instrukcji odpowiada specyficznym działaniom realizowanym w ramach określonej procedury lub procesu i posiada tylko jedną, stałą ścieżkę wykonania.
- **Węzeł decyzyjny:** Jest to element sterujący, który decyduje o wyborze odpowiedniej ścieżki wykonania na podstawie stanu wejściowych sygnałów lub danych. Węzły decyzyjne są kluczowe w strukturach kontrolnych, gdzie zależnie od spełnienia warunków wybierane są różne kierunki przepływu wykonania.
- **Ścieżka wykonania:** Jest to sekwencja instrukcji przetwarzanych w ustalonym porządku. Ścieżka wykonania ukazuje, jak poszczególne instrukcje są realizowane w zależności od spełnienia określonych warunków.

Na rysunku 2.15 dla procedury nr 2, wyróżniono specyficzne warunki dla wejść A i B. Spełnienie warunku na wejściu A prowadzi do wykonania bloku instrukcji numer 1, natomiast jego niespełnienie kieruje proces do bloku instrukcji numer 2. Analogicznie, dla wejścia B realizacja warunku skutkuje wykonaniem bloku instrukcji 3, a brak spełnienia – bloku instrukcji 4. Każdy z tych bloków instrukcji ma przypisany indywidualny czas wykonania, który wpływa na sumaryczny czas działania procedury. Suma czasów wykonania wszystkich bloków instrukcji dla procedury 2 jest zmienna i zależy od spełnienia warunków wejściowych. Taka reprezentacja podkreśla, jak różnorodność czasów wykonania poszczególnych bloków instrukcji wpływa na całkowity czas realizacji procedury. Znajomość tych czasów jest niezbędna przy projektowaniu systemów czasu rzeczywistego, ponieważ zapewnia stabilność i niezawodność działania systemu poprzez umożliwienie przewidywalności jego zachowania.



Rysunek 2.15: Wizualizacja zależności czasowych

2.6.2 Algorytm koncepcyjny - implementacja w języku C

Na rysunku 2.15 zaprezentowano szczegółowy algorytm, jaki realizuje procedura nr 2, gdzie wyróżniono warunki dla wejść A i B, które decydują o aktywacji poszczególnych ścieżek wykonania instrukcji. Algorytm ten można zapisać w postaci kodu języka C (Listing 2.1).

```c
#include <stdbool.h>

// Deklaracja procedur zewnetrznych
bool KomponentB_PobierzA();
bool KomponentB_PobierzB();

// Implementacja procedury publicznej KomponentA_Oblicz
void KomponentA_Oblicz() {
    // Warunek pierwszy
    if (KomponentB_PobierzA() == true) {
        Blok_1_1(); // czas wykonania ~2 ms
    } else {
        Blok_1_2(); // czas wykonania ~4 ms
    }

    // Warunek drugi
    if (KomponentB_PobierzB() == true) {
        Blok_2_1(); // czas wykonania ~2 ms
    } else {
        Blok_2_2(); // czas wykonania ~4 ms
    }
}
```

Listing 2.1: Implementacja przykładu z rysunku 2.12

Algorytm, jak wskazuje listing 2.1, składa się z dwóch głównych warunków zależnych od wyników procedury KomponentB_PobierzA() oraz KomponentB_PobierzB(). W przypadku pierwszego warunku, gdy procedura KomponentB_PobierzA() zwraca wartość true, wykonywany jest blok instrukcji o nazwie Blok_1_1, którego czas wykonania hipotetycznie może wynosić 2 ms. W przeciwnym wypadku,

system kieruje proces do bloku Blok_1_2, którego wykonanie trwa około 4 milisekundy. Podobnie, drugi warunek w algorytmie polega na wykonaniu procedury KomponentB_PobierzB(). Jeśli wynik tej procedury również jest równy true, instrukcje są przekierowywane do Blok_2_1, co zajmuje kolejne 2 milisekundy. W sytuacji, gdy wynik jest negatywny, algorytm kieruje do bloku Blok_2_2, który wymaga 4 milisekund na wykonanie. Jak wynika z powyższego opisu, zależności czasowe są kluczowym aspektem algorytmu, gdyż czas odpowiedzi systemu może się różnić w zależności od stanu wejściowego procedur zewnętrznych. Przyjęte czasy wykonanie są hipoteczne i założono je w celu prezentacji przykładu.

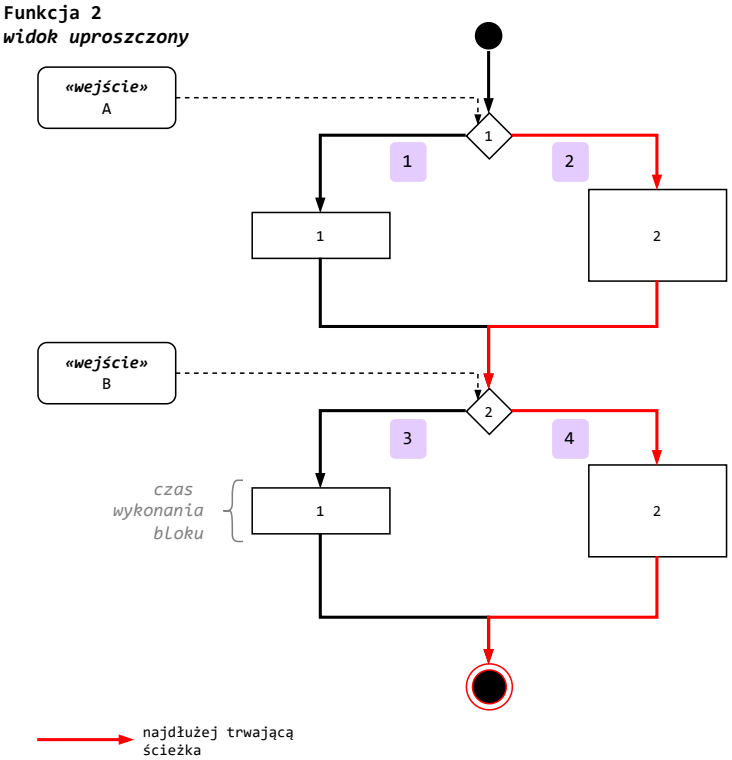
2.6.3 Ścieżka wykonania

Ścieżka wykonania programu, będąca kluczowym elementem analizy wydajności oprogramowania, może być postrzegana jako sekwencja procedur, a w szerszym ujęciu komponentów, które program wykonuje podczas działania. Jest to szczególnie istotne w kontekście systemów wbudowanych, gdzie każdy cykl procesora ma znaczenie. Różne aspekty ścieżki wykonania mogą być analizowane w zależności od poziomu szczegółowości: od pojedynczej procedury po całe komponenty systemu. Dla przykładu, można przyjrzeć się pojedynczej procedurze, która zawiera sekwencję operacji arytmetycznych, a jej wykonanie jest przewidywalne i niezależne od zewnętrznych czynników. W takim przypadku, analiza czasu wykonania może skupić się na optymalizacji tych operacji, aby zapewnić możliwie najkrótszy czas wykonania oraz maksymalną efektywność działania. Z drugiej strony, analizując cały komponent systemu, punktem wyjścia może być główna procedura, a analiza rozszerza się na wszystkie zagnieżdżone procedury i ich interakcje, co znacznie komplikuje proces optymalizacji. Konkretna ścieżka wykonania jest bezpośrednio powiązana z czasem, jaki jest potrzebny do jej realizacji. Czas ten może być różny w zależności od wielu czynników, w tym od skomplikowania logiki programu oraz od wydajności i obciążenia systemu. Przykładowo, czas wykonania ścieżki, która zawiera wiele zagnieżdżonych wywołań procedur, będzie dłuższy w porównaniu do ścieżki, która wykonuje proste operacje arytmetyczne [12]. Dla potrzeb dalszych rozważań można zdefiniować dwa pojęcia: możliwa i niemożliwa ścieżka wykonania sekwencji komponentów programu:

- **Możliwa ścieżka wykonania** odnosi się do sekwencji instrukcji w programie, która może być rzeczywiście zrealizowana podczas jego działania, przy założeniu, że spełnione są wszystkie warunki logiczne i zależności. Taka ścieżka musi być zgodna z logiką programu oraz spełniać wszystkie wymagania wynikające z jego struktury i danych wejściowych.

- **Niemożliwa ścieżka wykonania** to sekwencja instrukcji, której osiągnięcie jest niemożliwe ze względu na korelację sygnałów wejściowych, nawet jeśli wewnętrzna logika algorytmu na to pozwala. Oznacza to, że chociaż program teoretycznie mógłby wykonać daną ścieżkę, to w rzeczywistości, ze względu na ograniczenia wynikające z wzajemnych zależności danych wejściowych, nigdy nie zostanie ona przebyta.

Korzystając z przykładu algorytmu zamieszczonego na rys.2.16 i zapisanego w postaci kodu można założyć, że niemożliwa ścieżka wykonania będzie, wtedy gdy wartości na wejściach A i B będą przyjmowały wartość logiczna 1 (prawda), zatem algorytm może wykonać 3 ścieżki możliwe.



Rysunek 2.16: Wizualizacja zależności czasowych

2.6.4 Blok instrukcji

W języku C istnieją różne kategorie instrukcji, które są wykorzystywane wewnątrz procedury bez wpływu na zmianę ścieżki wykonania. Można je zastosować w bloku instrukcji. Należą do nich instrukcje takie jak operacje arytmetyczne, bitowe, operacje na danych, dostęp do danych oraz deklaracje i inicjalizacje zmiennych. Na przykład, w kategorii operacje arytmetyczne znajdują się proste operacje

arytmetyczne, takie jak dodawanie czy mnożenie, które są fundamentem logiki obliczeniowej. Ponadto, istnieją operatory przypisania łączone z operacjami arytmetycznymi, takie jak += czy *=, które skracają składnię kodu, zwiększając jego czytelność i efektywność. Definiowanie bloku instrukcji, obejmuje zatem zbiór instrukcji, które są wykonywane sekwencyjnie i które nie zmieniają ścieżki wykonania programu ani czasu wykonania [14].

Czas wykonania wspomnianego zestawu instrukcji może wykazywać zmienność. Zmienna szybkość wykonania bloku instrukcji może być często wynikiem działania mechanizmów takich jak stronicowanie pamięci dla instrukcji i danych, co jest kluczowe szczególnie w systemach o ograniczonych zasobach, takich jak mikrokontrolery. Stronicowanie może wpływać na czas wykonania bloku instrukcji poprzez zmiany w dostępności danych i instrukcji. Jeśli dane lub instrukcje nie znajdują się w pamięci podręcznej (cache) i wymagane jest załadowanie ich ze wolniejszej pamięci RAM, może to prowadzić do opóźnień.

Kolejnym aspektem, który zasługuje na uwagę, są przerwania. Przerwania to mechanizmy, które pozwalają reagować na określone zdarzenia w systemie poprzez zatrzymanie bieżącego zadania i wykonanie procedury przerywającej. Chociaż przerwania są niezbędne do zarządzania określonymi zdarzeniami, mogą wprowadzać opóźnienia w działaniu systemu. Z tego powodu, w projektowaniu systemów, gdzie kluczowa jest przewidywalność czasowa, często dąży się do minimalizacji użycia przerwań lub do ich optymalizacji tak, aby wprowadzały jak najmniejsze opóźnienia. Warto zauważyć, że zarówno stronicowanie, jak i przerwania są zewnętrznymi elementami, które mają znaczący wpływ na czas wykonania bloku instrukcji. Optymalne zarządzanie tymi elementami jest niezbędne dla zapewnienia stabilności i efektywności działania systemów czasu rzeczywistego, gdzie każda milisekunda może być kluczowa.

2.6.5 Węzeł decyzyjny

Na wybór możliwej ścieżki realizacji kodu programu ma wpływ węzeł decyzyjny. Węzły decyzyjne w kodzie mogą być zaimplementowane za pomocą jednej z kilku instrukcji takich jak `for`, `while`, `do-while`, `switch`, `break`, `continue`, oraz `goto` [49].

Analizując przedstawione struktury, należy zwrócić uwagę, iż standardy programowania, takie jak MISRA [64], zalecają unikanie niektórych z tych konstrukcji w celu zapewnienia wyższego poziomu bezpieczeństwa i czytelności kodu. Przykładowo, użycie instrukcji `goto` jest często odradzane, ponieważ może prowadzić do kodu trudniejszego do śledzenia i utrzymania. Z kolei struktury takie jak `for`, `while`, i `do-while` są powszechnie akceptowane i szeroko stosowane, ze względu na ich zdolność do

efektywnego zarządzania powtarzalnymi zadaniami w jasny i strukturalny sposób. Jednakże nawet w ramach tych akceptowanych pętli, MISRA ma konkretne zalecenia dotyczące stosowania instrukcji `break` oraz `continue`. MISRA zaleca unikanie instrukcji `break` w pętlach, ponieważ jej użycie może prowadzić do przedwczesnego zakończenia iteracji, co potencjalnie może wprowadzać trudności w analizie przepływu programu i zarządzaniu stanem programu. Wykorzystanie `break` powinno być zatem ograniczone i stosowane tylko wtedy, gdy jest to absolutnie konieczne i jasno uzasadnione, aby zachować przejrzystość przepływu sterowania. Co więcej, instrukcja `continue` również jest odradzana przez standard MISRA, ponieważ może ona zakłócać naturalny przepływ iteracji pętli, co utrudnia śledzenie i utrzymanie logicznego przepływu programu. Zamiast korzystać z `continue`, zaleca się reorganizację logiki warunkowej wewnątrz pętli, aby utrzymać czytelność i uproszczenie kodu. Tym samym, użycie `continue`, podobnie jak `break`, powinno być rozważane z ostrożnością i stosowane tylko wtedy, gdy nie wprowadza dodatkowej złożoności lub ryzyka błędów.

Przechodząc dalej do analizy węzłów decyzyjnych w kontekście struktur sterujących, kluczowe jest zrozumienie, jakie elementy języka C mogą wpływać na te kluczowe punkty decyzyjne w programie. Węzeł decyzyjny, który w języku C najczęściej reprezentowany jest przez strukturę `if`, może być również modyfikowany przez użycie operatorów logicznych i relacyjnych, które determinują warunki przejścia przez różne ścieżki wykonania programu.

Elementy, które mają bezpośredni wpływ na węzeł decyzyjny, obejmują:

- **Operatory porównania** – takie jak `==`, `!=`, `<`, `>`, `<=`, `>=`, które są używane do porównywania wartości i decydowania o kierunku przepływu na podstawie tych porównań.
- **Operatory logiczne** – takie jak `&&` (i), `||` (lub), oraz `!` (negacja), które pozwalają na łączenie wielu warunków w jednym wyrażeniu decyzyjnym, co umożliwia bardziej złożone decyzje.
- **Wyrażenia arytmetyczne** – które mogą być używane do obliczania wartości w warunkach, na przykład `if ((x + y) > z)`.
- **Procedury** – których wyniki mogą być wykorzystywane do podejmowania decyzji, na przykład `if (isValid(x))`, gdzie `isValid()` jest procedurą sprawdzającą pewien warunek i zwracającą wartość boolowską.
- **Zmienne** – które przechowują dane wpływające na przebieg programu. Ich aktualne wartości mogą determinować wynik warunku w instrukcji decyzyjnej.

- **Operatory warunkowe** – takie jak operator trójargumentowy `?:`, który może służyć do zwięzłego formułowania wyrażeń decyzyjnych, wpływających na przepływ programu bez konieczności używania pełnych struktur `if-else`.

Wspomniane wcześniej elementy, takie jak operatory porównania i logiczne, funkcje, oraz zmienne, mają zastosowanie nie tylko w ramach instrukcji warunkowych, ale również w innych strukturach sterujących, takich jak pętle. Ich uniwersalność umożliwia szerokie wykorzystanie w różnorodnych kontekstach programistycznych, co znacząco wpływa na elastyczność i wydajność kodu. Listing 2.2 pokazuje użycie wyżej wspomnianych elementów. W kontekście języka C, węzeł decyzyjny stanowi centralny element, wokół którego koncentruje się logika sterowania przepływem programu. Każdy z omówionych elementów języka ma kluczowe znaczenie dla efektywnego i precyzyjnego kierowania działaniem programu, umożliwiając tworzenie zarówno prostych, jak i zaawansowanych aplikacji, które są w stanie odpowiedzieć na zmienne warunki operacyjne. To zrozumienie podkreśla, jak węzły decyzyjne wpływają na strukturalną i funkcjonalną złożoność programowania w języku C.

```c
#include <stdio.h>
#define PROG_KRYTYCZNY 75

int czyKrytyczna(int temperatura) {
    // Operator porownania sterujacy przeplywem
    return temperatura > PROG_KRYTYCZNY;
}
void sprawdzIAlarmuj(int temperatura) {
    // Instrukcja warunkowa if sterujaca przeplywem
    if (czyKrytyczna(temperatura)) {
        // W aplikacjach wbudowanych, szczegolnie w przemysle samochodowym,
        // funkcje takie jak printf
        // nie sa stosowane, poniewaz systemy te czesto nie posiadaja
        // interfejsu do wyswietlania tekstu,
        // a takze nie wymagaja tego typu interakcji.
        // Funkcja printf zostala uzyta tutaj wylacznie
        // w celach demonstracyjnych, aby uczynic przyklad bardziej zrozumialym
        printf("Temperatura %d przekracza prog %d stopni C.\n", temperatura,
            PROG_KRYTYCZNY);
    } else {
        printf("Temperatura %d jest w bezpiecznym zakresie.\n", temperatura);
    }
}
int main() {
    int temperatury[] = {68, 72, 77, 80, 65, 70};
    int liczbaCzujnikow = sizeof(temperatury) / sizeof(temperatury[0]);

    // Petla for sterujaca przeplywem
    for (int i = 0; i < liczbaCzujnikow; i++) {
        sprawdzIAlarmuj(temperatury[i]);
    }
    return 0;
}
```

Listing 2.2: System monitorowania temperatury z funkcja alarmowa

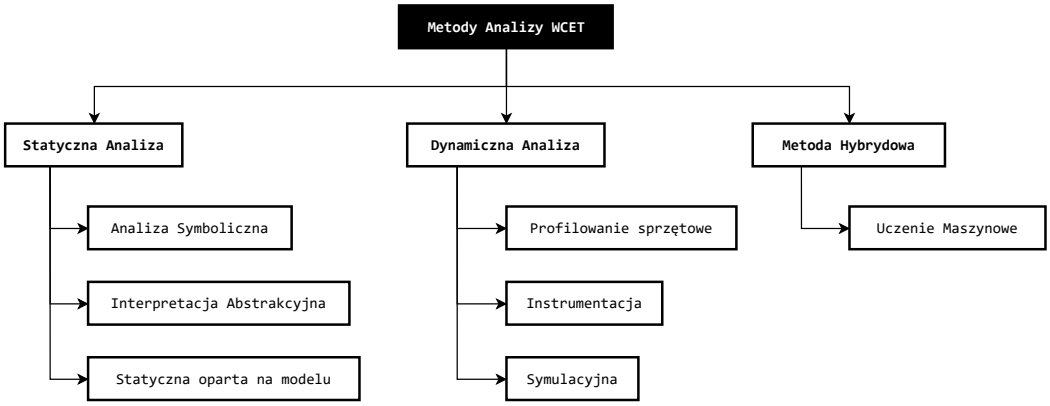
2.6.6 Charakterystyki czasowe algorytmu

Przy omawianiu wpływu wejść A i B (Procedura 2) na czas wykonania algorytmu, nie można pominąć koncepcji Pesymistycznego Czasu Egzekucji (PCE), znanego również jako Worst Case Execution Time (WCET) [31]. PCE definiuje się jako maksymalny czas wykonania zadania w najmniej korzystnych, lecz możliwych do wystąpienia warunkach operacyjnych, nie uwzględniając awarii systemu. W przypadku omawianego algorytmu (rys. 2.16), PCE może być osiągnięty w sytuacji, gdy obie procedury wejściowe zwrócą wartości, które spowodują wywołanie bloków o dłuższym czasie trwania, czyli Blok_1_2 i Blok_2_2. Równie istotną miarą jest Optymistyczny Czas Egzekucji (OCE), znany również jako Best Case Execution Time (BCET). OCE określa minimalny możliwy czas wykonania zadania. Jest to szczególnie ważne w systemach, gdzie nie tylko opóźnienia, ale i zbyt szybkie reakcje mogą prowadzić do niepożądanych skutków, jak na przykład w systemach sterowania przemysłowego czy interaktywnych aplikacjach czasu rzeczywistego. W omawianym algorytmie OCE nastąpiłby, gdy obie procedury wejściowe KomponentB_PobierzA() i KomponentB_PobierzB() zwróciłyby wartości prawdziwe, co spowodowałoby wywołanie Blok_1_1 i Blok_2_1, każdy z czasem trwania około 2 ms.

2.7 Metody detekcji możliwych ścieżek wykonania i pomiaru zależności czasowych

Precyzja w szacowaniu czasów wykonania ma bezpośredni wpływ na efektywność i stabilność całego systemu. W dalszej części rozdziału zostaną omówione różne podejścia badawcze, w tym metody statyczne, dynamiczne oraz hybrydowe, umożliwiające dogłębne zrozumienie i optymalizację czasu wykonania krytycznych funkcji oprogramowania. Początkowo omówione zostaną metody statyczne, które nie wymagają uruchamiania programu do oceny jego czasu wykonania, zawierające techniki, takie jak analiza symboliczna czy interpretacja abstrakcyjna, które pozwalają na modelowanie potencjalnych ścieżek wykonania bez rzeczywistego uruchomienia programu. Dzięki temu możliwe jest wygenerowanie danych pozwalających na teoretyczne przewidzenie najgorszego scenariusza wykonania. Metody dynamiczne, w odróżnieniu od statycznych, opierają się na rzeczywistym wykonaniu programu i analizie wyników w czasie rzeczywistym, umożliwiając identyfikację najdłuższych ścieżek wykonania oraz obserwację zachowania systemu podczas różnych warunków operacyjnych. Metoda hybrydowa, łącząca analizę statyczną i dynamiczną, wykorzystuje zalety obu podejść, minimalizując ich ograniczenia. Poprzez integrację danych uzyskanych zarówno z analiz statycznych, jak i wyników dynamicznych

pomiarów, możliwe jest uzyskanie bardziej kompleksowego i wiarygodnego obrazu czasów wykonania w aplikacjach o wysokim stopniu skomplikowania. W kontynuacji rozdziału szczegółowo omówione zostaną narzędzia i techniki wykorzystywane w każdej z metod, wraz z analizą ich zalet i ograniczeń, a także odpowiedniości do specyficznych zastosowań.



Rysunek 2.17: Metody estymacji najgorszego czasu wykonania (WCET)

2.7.1 Metoda statyczna

Statyczna analiza kodu stanowi metodę analizy oprogramowania, realizowaną bez potrzeby jego uruchamiania. Definiowana jako proces offline, umożliwia ocenę potencjalnych parametrów czasu wykonania oraz innych aspektów kluczowych dla bezpieczeństwa i niezawodności systemów. Działanie to opiera się na wczytywaniu pliku wykonywalnego, jego dekompilacji na instrukcje asemblera oraz generowaniu drzew przepływu sterowania i drzew wywołań funkcji, które reprezentują wzajemne powiązania i zależności między różnymi funkcjami w kodzie. Kluczowe znaczenie w statycznej analizie kodu ma określenie maksymalnej liczby iteracji pętli, które bezpośrednio wpływają na obliczenia czasu wykonania w najgorszym przypadku. Analiza ta wymaga dokładnych modeli matematycznych procesora oraz specyfikacji konfiguracji pamięci, co pozwala na uwzględnienie wpływu pamięci podręcznej i potokowości procesora na wynikowy czas wykonania. Statyczna analiza kodu znajduje zastosowanie przede wszystkim w systemach, gdzie niezbędne jest gwarantowanie maksymalnych czasów wykonania, co jest wymagane przez różnorodne standardy bezpieczeństwa. Proces ten jest także używany do automatycznej weryfikacji WCET w ramach procesu kompilacji oraz może być stosowany do optymalizacji czasu wykonania oprogramowania. Pomimo swoich zalet, statyczna analiza kodu wiąże się z istotnymi wyzwaniami. Trudności te obejmują nierozwiązane wywołania funkcji pośrednich, rekurencje

oraz błędnie identyfikowane górne granice pętli, które wymagają ręcznego dodania adnotacji przez użytkownika. Dodatkowo, analiza może prowadzić do znacznego przeszacowania czasów wykonania, co jest szczególnie problematyczne w złożonych systemach wielordzeniowych, gdzie analiza nie bierze pod uwagę przerwań czy konfliktów dostępu do pamięci. Proces ten, mimo wyzwań, pozostaje niezbędnym narzędziem w wielu krytycznych dla bezpieczeństwa aplikacjach, oferując rzetelne oszacowania czasów wykonania bez konieczności uruchamiania kodu. W kontekście rosnących wymagań dotyczących bezpieczeństwa i niezawodności systemów, statyczna analiza kodu prezentuje się jako kluczowa metodologia, umożliwiająca głębokie zrozumienie i optymalizację czasu wykonania krytycznych funkcji oprogramowania [28] [77].

Metoda wsparta modelem

Metoda statyczna oparta na modelu do analizy WCET skupia się na wykorzystaniu wysokopoziomowego modelu aplikacji w celu systematycznego generowania kodu niższego poziomu, takiego jak kod C, a następnie kod maszynowy. Zachowanie semantyki programu na każdym z tych etapów jest kluczowe, gdyż umożliwia zachowanie spójności pomiędzy pierwotnym modelem a jego implementacją sprzętową. Metoda ta, w przeciwieństwie do dynamicznych, nie wymaga wykonania programu, aby określić najgorsze możliwe czasy wykonania, co jest istotne z punktu widzenia aplikacji krytycznych, gdzie niezawodność i przewidywalność są niezbędne. Porównując statyczne metody oparte na modelu z innymi rodzajami analiz, takimi jak metody dynamiczne czy symboliczne, można zauważyć zasadnicze różnice. Metody dynamiczne, opierając się na faktycznym wykonaniu programu, mogą nie uwzględnić wszystkich teoretycznych ścieżek wykonania, co potencjalnie skutkuje niedoszacowaniem ryzyka. Z kolei metody symboliczne, choć potrafią analizować różne stany programu bez jego wykonania, często oddalają się od realnych warunków pracy aplikacji, skupiając się na bardziej abstrakcyjnych modelach. Metoda statyczna oparta na modelu zapewnia zaś bezpośrednie przełożenie z modelu wysokopoziomowego na kod wykonawczy, co zwiększa dokładność analizy. Podstawowe elementy wykorzystywane w tej metodzie obejmują model wysokopoziomowy, kod C oraz kod binarny. Model wysokopoziomowy stanowi abstrakcyjne odwzorowanie aplikacji, które jest następnie transformowane do postaci kodu C, zachowując kluczowe elementy logiki programu. Końcowe przełożenie na kod binarny pozwala na szczegółową analizę interakcji ze sprzętem, co jest decydujące dla dokładności obliczeń WCET. Proces analizy rozpoczyna się od opracowania modelu wysokopoziomowego, który jest systematycznie przekształcany w niższe reprezentacje kodu, aż do poziomu binarnego. Na każdym z tych poziomów

przeprowadzana jest statyczna analiza struktury i przepływu, co umożliwia wyodrębnienie kluczowych informacji dotyczących czasu wykonania. Następnie, bazując na tych danych, obliczany jest WCET, biorąc pod uwagę wszystkie możliwe scenariusze wykonania programu, co jest szczególnie istotne w systemach o wysokich wymaganiach bezpieczeństwa. Zalety stosowania metody statycznej opartej na modelu obejmują gwarancje bezpieczeństwa, wynikające z kompleksowego pokrycia potencjalnych ścieżek wykonania oraz systematyczność rozwoju od projektu do implementacji, co zwiększa efektywność i jakość projektowania. Dodatkowo, metoda ta często integruje się z nowoczesnymi narzędziami programistycznymi, co wspiera automatyzację i zarządzanie projektami. Wady tej metody wynikają przede wszystkim z jej złożoności oraz możliwości przeszacowania czasów wykonania, co może prowadzić do nieoptymalnego wykorzystania zasobów systemowych. Ponadto, metoda ta jest silnie zależna od dokładności modelu początkowego, co oznacza, że jakiekolwiek niedokładności na wczesnym etapie projektowania mogą mieć długofalowe konsekwencje dla całej analizy [12].

Metoda analizy symbolicznej

Analiza symboliczna, jako specyficzna forma statycznej analizy kodu, umożliwia zbadanie programu bez wykorzystywania rzeczywistych danych wejściowych, operując na symbolach. Symboliczne reprezentacje wartości wejściowych pozwalają na jednoczesne rozważenie wielu możliwych ścieżek wykonania programu. Istotnym składnikiem tej metody jest generowanie formuł matematycznych, które opisują stan programu na każdym kroku jego wykonania. Te formuły mogą być następnie używane do weryfikacji, czy program spełnia określone wymagania bezpieczeństwa czy poprawności działania. Porównując analizę symboliczną z innymi metodami statycznymi, widoczne jest jej wyższe zaawansowanie. Tradycyjne metody statyczne, które skupiają się na bezpośredniej analizie kodu źródłowego, nie posiadają zdolności do elastycznego modelowania nieokreślonych stanów danych, co jest możliwe dzięki użyciu symboli w analizie symbolicznej. Z kolei metody dynamiczne, wymagające wykonania kodu, nie są w stanie dostarczyć informacji o wszystkich potencjalnych ścieżkach wykonania, co stanowi kolejny atut analizy symbolicznej. Podstawowymi elementami wykorzystywanymi w analizie symbolicznej są symbole oraz narzędzie do rozwiązywania zapytań logicznych opartych na teoriach spójności modułowej. Symbole służą do abstrakcyjnej reprezentacji wartości zmiennych, natomiast to narzędzie umożliwia sprawdzenie, czy określone formuły logiczne są spełnione. Dodatkowo, drzewo stanów jest strukturą danych pozwalającą na śledzenie różnych możliwych stanów programu, które wynikają z różnorodnych ścieżek wykonania.Proces analizy symbolicznej rozpoczyna się od inicjalizacji

symboli, które reprezentują potencjalne dane wejściowe. Następnie, analizowane są możliwe ścieżki wykonania programu, z których każda jest reprezentowana przez odpowiednie symbole wynikające z operacji. Formułowane są zapytania dotyczące własności, które program powinien spełniać, a następnie za pomocą narzędzia do rozwiązywania problemów logicznych rozstrzyga się, czy te formuły są prawdziwe dla dowolnych wartości symboli. W rezultacie, analiza wyników pozwala na interpretację bezpieczeństwa, poprawności działania czy innych właściwości programu. Zaletami analizy symbolicznej są jej wszechstronność, dokładność oraz elastyczność. Metoda ta pozwala na dokładne zidentyfikowanie błędów, które mogłyby umknąć podczas tradycyjnych testów, oraz na analizę złożonych warunków bez potrzeby dostępu do konkretnych danych wejściowych. Jednakże, metoda ta posiada również wady, takie jak wysokie wymagania obliczeniowe, które są szczególnie problematyczne w przypadku dużych lub skomplikowanych programów, a także złożoność implementacji i ograniczenia narzędzi do rozwiązywania problemów logicznych, które mogą wpływać na skuteczność analizy w pewnych scenariuszach. Stosując analizę symboliczną, można zatem uzyskać dogłębne zrozumienie programu, co jest nieocenione w kontekście zapewnienia jego bezpieczeństwa i niezawodności. Oferuje ona szczegółowy wgląd w potencjalne problemy, które mogą nie być widoczne przy użyciu innych metod analizy [13].

Metoda interpretacji abstrakcyjnej

Metoda statyczna oparta na modelu z zastosowaniem interpretacji abstrakcyjnej do analizy WCET wykorzystuje tę technikę jako sposób na solidne przybliżenie semantyki programów. Metoda ta pozwala na estymację najgorszego przewidywanego czasu wykonania przez analizę statyczną kodu źródłowego, bez potrzeby jego uruchamiania. Semantyka zbierająca, definiowana jako najmniejszy możliwy zbiór stanów osiągalnych podczas wykonania programu, jest przybliżana przez semantykę abstrakcyjną — nadzbiór semantyki zbierającej, który może zawierać stany nie występujące rzeczywiście w wykonaniu, ale zapewniające kompleksowe i solidne informacje. Analiza różni się od innych metod, takich jak analiza symboliczna, która skupia się na dokładności poprzez użycie wyrażeń symbolicznych do definiowania możliwych wartości zmiennych, nie zapewniając jednak takiego samego poziomu bezpieczeństwa. W przeciwieństwie do tego, statyczne metody oparte na modelu wykorzystują matematyczny model kodu oraz sprzętu do konserwatywnego szacowania, co pozwala na bezpieczne przewidywanie granic czasowych wykonania różnych sekwencji instrukcji. Proces analizy rozpoczyna się od analizy przepływu w celu zrozumienia struktury programu, a następnie przechodzi do analizy niskopoziomowej, modelując cechy sprzętu takie jak pamięci podręczne i potoki. Kluczowe jednostki używane w modelu, takie

jak stany abstrakcyjne i grafy przepływu sterowania (CFG), umożliwiają szczegółowe mapowanie potencjalnych ścieżek przez program. Interpretacja abstrakcyjna jest stosowana, by konserwatywnie oszacować granice pętli i ścieżek wykonania, co umożliwia integrację wyników z analizą niskopoziomową do ostatecznego oszacowania WCET. Korzyści z metody obejmują przede wszystkim jej bezpieczeństwo, gwarantujące, że szacowany czas wykonania nigdy nie jest niższy niż rzeczywisty, co jest kluczowe w systemach czasu rzeczywistego. Metoda jest niezależna od wykonania, co sprawia, że jest odpowiednia do wczesnych etapów rozwoju oprogramowania i oferuje teoretyczną możliwość rozważenia wszystkich ścieżek wykonania. Wśród wad metody można wymienić przeszacowanie, które może prowadzić do niewydajnego wykorzystania zasobów, oraz jej złożoność, wymagającą znacznej wiedzy fachowej, zwłaszcza przy modelowaniu złożonych interakcji sprzętowych. Pomimo tych wyzwań, metoda ta stanowi ważne narzędzie w projektowaniu oprogramowania, gdzie bezpieczeństwo i przewidywalność wykonania są priorytetami [24].

2.7.2 Metoda statystyczna

W niniejszym rozdziale szczegółowo omówione zostaną metody analizy charakterystyk czasowych, skupiając się na grupie technik opartych na pomiarach. W odróżnieniu od statycznej analizy kodu, metody te bazują na rzeczywistych danych pochodzących z działania systemu lub jego dokładnej symulacji. Pozwala to na empiryczne oszacowanie czasu potrzebnego na wykonanie określonych operacji lub funkcji, co z kolei przekłada się na większą dokładność wyników i umożliwia bardziej szczegółową analizę rzeczywistego zachowania systemów. Pierwszym podejściem w ramach metod pomiarowych są metody instrumentacyjne. Wykorzystują one wstawianie specjalnych instrukcji w kodzie, co umożliwia dokładne śledzenie parametrów wykonania. Takie techniki, mimo że mogą wpływać na naturalne działanie systemu, dostarczają wartościowych danych do analizy czasowej. Drugie podejście, metody profilowania sprzętowego, opierają się na generowaniu szczegółowych śladów wykonania programu. Informacje te dotyczą sekwencji wywołań funkcji oraz czasów ich wykonania i pozwalają na głęboką analizę charakterystyk czasowych bez istotnego ingerowania w funkcjonowanie systemu. Trzecią grupę stanowią metody symulacyjne, które za pomocą zaawansowanych narzędzi symulacyjnych modelują działanie aplikacji w różnorodnych warunkach sprzętowych i systemowych. Wyniki takich symulacji są cenne, choć należy je traktować jako przybliżenie rzeczywistych warunków wykonania. Pierwszym krokiem w realizacji tych metod jest przygotowanie scenariuszy testowych, które stanowią kluczowy element badania czasowych charakterystyk systemu. Scenariusze te muszą być skonstruowane tak,

by obciążać system w sposób maksymalny, co pozwala na identyfikację przypadków długotrwałego działania. Do stworzenia efektywnych scenariuszy niezbędna jest dogłębna znajomość systemu, która umożliwia wskazanie i testowanie jego najbardziej wrażliwych i krytycznych obszarów. Następnym krokiem jest pomiar czasu trwania wykonania scenariuszy. W tym kontekście wykorzystuje się trzy główne metody: instrumentację, profilowanie sprzętowe oraz symulację. Instrumentacja polega na wstawianiu do kodu specjalnych instrukcji pozwalających na śledzenie czasów wykonania poszczególnych operacji. Metoda ta, choć dokładna, może wpływać na naturalne działanie systemu poprzez dodatkowe obciążenie. Profilowanie sprzętowe używa dedykowanych narzędzi sprzętowych do monitorowania wykonania programu, co minimalizuje ingerencję w jego działanie. Symulacja pozwala na modelowanie działania systemu w kontrolowanych warunkach, co jest szczególnie przydatne w przypadkach trudnych do zreplikowania w rzeczywistym środowisku. Ostatnim etapem jest analiza statystyczna wyników pomiarów. W tym kroku wykonuje się różnorodne operacje statystyczne, takie jak obliczanie średnich, median, odchylenia standardowego czy estymacja wartości skrajnych, co ma na celu oszacowanie najgorszych czasów wykonania scenariuszy. Takie podejście pozwala na bardziej precyzyjne zrozumienie, jak długo system może zajmować się wykonaniem najbardziej wymagających operacji. Dalsza część rozdziału będzie koncentrować się na szczegółowej analizie każdej z wymienionych metod. Omówione zostaną ich zalety oraz ograniczenia [4].

Metoda instrumentacji

W metodzie analizy opartej na instrumentacji kodu źródłowego, działania skupiają się na docelowym sprzęcie i kodzie źródłowym, umożliwiając badanie czasu wykonania określonych funkcji w ramach systemu. Instrumentacja kodu źródłowego polega na wstawieniu dodatkowych instrukcji do kodu programu, co pozwala na dokładne monitorowanie i rejestrację różnych parametrów wykonania. Przykładowo, w procesie pomiarowym dodawane są instrukcje mierzące czas rozpoczęcia i zakończenia wykonania każdego bloku kodu. Działanie tej metody jest wieloetapowe. Początkowo, podczas fazy analizy, kod źródłowy jest przetwarzany w celu ekstrakcji informacji o ścieżkach wykonania, które następnie są segmentowane. Segmentacja ta pozwala na ograniczenie liczby różnych ścieżek poddawanych dalszym pomiarom, co jest kluczowe dla zachowania przejrzystości analizy i zarządzania złożonością danych testowych. W fazie pomiarów, generowane automatycznie dane testowe kierują wykonanie programu po zadanych ścieżkach, a wynikające z tego czasy wykonania są rejestrowane dzięki wcześniej dodanym instrumentacjom. Jedną z głównych zalet metody instrumentacji jest możliwość dokładnego

monitorowania wykonania programu na żywo, co pozwala na realistyczną ocenę czasów wykonania. Dodatkowo, metoda ta jest stosunkowo elastyczna i pozwala na dostosowanie poziomu szczegółowości pomiarów do potrzeb użytkownika, co jest szczególnie ważne w złożonych systemach czasu rzeczywistego. Wadami tej techniki są natomiast potencjalne zakłócenia w naturalnym zachowaniu systemu spowodowane dodatkowym kodem instrumentacji, co może prowadzić do nienaturalnie dłuższych czasów wykonania. Ponadto, ze względu na konieczność precyzyjnego projektowania testów i generowania odpowiednich danych wejściowych, metoda ta może być czasochłonna i wymagać zaawansowanych narzędzi do generowania oraz zarządzania testami [81].

Metoda profilowania sprzętowego

Metoda profilowania w analizie charakterystyk czasowych systemów wbudowanych polega na generowaniu szczegółowych śladów wykonania programu za pomocą specjalnych jednostek sprzętowych, które rejestrują instrukcje wykonywane przez procesor. Dzięki temu możliwe jest dokładne śledzenie, które linie kodu są wykonywane i którymi ścieżkami podąża wykonanie kodu, bez konieczności modyfikacji kodu źródłowego. Ta technika umożliwia głęboką analizę czasu wykonania aplikacji bez istotnego ingerowania w jej naturalne działanie, co jest kluczowe przy badaniu systemów czasu rzeczywistego. Zaawansowany debugger, który wspiera tego rodzaju profilowanie sprzętowe, oferuje możliwości zbierania i analizowania danych śledzenia w czasie rzeczywistym. Umożliwia konfigurację śledzenia za pomocą dedykowanych modułów sprzętowych, które rejestrują działanie CPU bez wpływu na kod źródłowy. Zaletą tej metody profilowania jest jej minimalny wpływ na działanie systemu – monitorowanie jest prowadzone przez zewnętrzne jednostki sprzętowe, które nie wpływają na ogólną wydajność aplikacji. Metoda ta oferuje również wysoką dokładność i szczegółowość danych, umożliwiając precyzyjne określenie przyczyn problemów wydajnościowych oraz optymalizację systemu. Wadą metody profilowania sprzętowego jest konieczność dostępu do dedykowanego sprzętu pomiarowego oraz potencjalne trudności związane z jego konfiguracją, co może być czasochłonne i wymagać specjalistycznej wiedzy technicznej. Narzędzie do debugowania dostarcza jednak opcji, które umożliwiają łatwiejszą analizę zebranych danych, co pomaga w przeglądaniu ścieżek wykonania kodu, analizowaniu zmian w zmiennych, czy nawet symulowaniu różnych scenariuszy wykonania programu na podstawie zebranych danych, bez potrzeby ingerencji w kod źródłowy [56].

Metoda symulacyjne

Metoda symulacyjna, stosowana w analizie charakterystyk czasowych systemów wbudowanych, koncentruje się na wykorzystaniu zaawansowanych narzędzi symulacyjnych do modelowania działania aplikacji w zmiennych warunkach sprzętowych i systemowych. Podobnie jak w przypadku metod opartych na instrumentacji, proces rozpoczyna się od przygotowania odpowiedniego modelu aplikacji, który odwzorowuje strukturę i zachowanie oryginalnego systemu. Podstawą metody symulacyjnej jest tworzenie szczegółowych modeli systemów, które mogą być testowane w różnych scenariuszach. Symulacja umożliwia przeprowadzanie testów na wirtualnych replikach rzeczywistego sprzętu, co pozwala na analizę zachowania systemu bez ryzyka uszkodzenia fizycznego sprzętu czy konieczności angażowania zasobów produkcyjnych. Modele te są zazwyczaj tworzone przy użyciu języków opisu sprzętu takich jak VHDL czy Verilog, bądź przy użyciu wysokopoziomowych narzędzi symulacyjnych oferowanych przez oprogramowanie inżynieryjne. Działania w ramach symulacji składają się z kilku etapów. Początkowo, tworzone są modele sprzętu oraz systemu operacyjnego, które mają odzwierciedlać rzeczywiste zachowanie systemu pod różnymi obciążeniami i w różnych konfiguracjach. Następnie, na tych modelach uruchamiane są aplikacje lub ich fragmenty, co pozwala na obserwację reakcji systemu na specyficzne polecenia i zadania. W trakcie symulacji, możliwe jest śledzenie różnych parametrów, takich jak czas wykonania, zużycie zasobów czy odpowiedzi na zdarzenia zewnętrzne. Jedną z głównych zalet metody symulacyjnej jest jej bezpieczeństwo i wszechstronność – symulacje mogą być łatwo modyfikowane i dostosowywane do specyficznych potrzeb projektowych bez dodatkowych kosztów związanych z fizycznymi prototypami. Ponadto, metoda ta umożliwia testowanie systemów w warunkach, które mogłyby być niebezpieczne lub niedostępne dla rzeczywistych urządzeń. Wadą metody symulacyjnej jest jej zależność od dokładności modeli – wszelkie uproszczenia czy błędy w modelowaniu mogą prowadzić do wyników, które nie odzwierciedlają w pełni rzeczywistego zachowania systemów. Symulacje są również zazwyczaj czasochłonne i wymagają dużych mocy obliczeniowych, zwłaszcza w przypadku skomplikowanych systemów z wieloma interakcjami [57].

2.7.3 Metoda hybrydowa

Metoda hybrydowa analizy najgorszego przypadku czasu wykonania (WCET) łączy statyczną analizę programu z pomiarami dynamicznymi, co umożliwia uzyskanie bardziej wiarygodnych wyników dla złożonych systemów informatycznych. Hybrydowe bloki, będące mniejszymi jednostkami kodu, są analizowane i mierzone pod kątem czasu wykonania, a ich granice można dostosowywać w celu

optymalizacji precyzji prognoz WCET. Dostosowanie granic bloków hybrydowych polega na dynamicznym określaniu, jak duże fragmenty kodu powinny być analizowane jako pojedyncze bloki. Granice te wpływają na szczegółowość analizy — mniejsze bloki umożliwiają bardziej szczegółową ocenę, lecz zwiększają koszt obliczeniowy. Z tego względu kluczowe jest, aby algorytmy uczenia maszynowego optymalnie dobierały te granice, uwzględniając równowagę między precyzją analizy a zasobami niezbędnymi do jej przeprowadzenia. Zastosowanie uczenia maszynowego umożliwia dalsze usprawnienie tego procesu poprzez automatyzację ekstrakcji cech z dostępnych danych, co przekłada się na trafniejsze identyfikowanie krytycznych ścieżek wykonania oraz modelowanie predykcyjne. Algorytmy te pozwalają również na wykrywanie nieoczywistych wzorców w danych dotyczących czasu wykonania, co jest kluczowe w analizie scenariuszy najgorszego przypadku. Dzięki temu możliwe jest bardziej precyzyjne dostosowanie granic bloków hybrydowych, wspierające estymację WCET. Techniki te umożliwiają także modelowanie marginesów bezpieczeństwa, uwzględniających zmienność czasów wykonania wynikającą z różnorodnych danych wejściowych i warunków operacyjnych. W rezultacie hybrydowa analiza WCET, wspierana przez uczenie maszynowe, otwiera nowe możliwości zwiększenia efektywności i niezawodności szacunków, minimalizując ryzyko niedoszacowania krytycznych parametrów systemów informatycznych [41].

Przegląd potencjalnych algorytmów

1. **Drzewa decyzyjne i lasy losowe**:
    - Są to algorytmy uczące się nadzorowane (supervised learning), które doskonale radzą sobie z klasyfikacją oraz regresją. Ich zaletą jest interpretowalność wyników, co może być pomocne w analizie decyzji podejmowanych przez system bezpieczeństwa [62].
    - Jednak w przypadku dużych zbiorów danych mogą być mniej efektywne obliczeniowo.
2. **Sieci neuronowe**:
    - Mogą być stosowane zarówno do klasyfikacji, jak i regresji. Są w stanie modelować skomplikowane zależności w danych i efektywnie rekonstruować brakujące informacje [23].
    - Sieci neuronowe typu autoenkodery [22] są szczególnie przydatne do redukcji wymiarowości danych oraz imputacji brakujących wartości.
    - Wymagają jednak dużej mocy obliczeniowej i mogą być trudne w interpretacji.
3. **Algorytmy klasteryzacji (np. k-średnich)**:

- Są stosowane w uczeniu nienadzorowanym do grupowania podobnych danych. Mogą być używane do wykrywania anomalii poprzez identyfikację niepasujących do żadnego klastru danych [84].
- Mogą jednak mieć problemy z danymi o złożonej strukturze.

4. **Algorytmy redukcji wymiarowości (np. PCA, t-SNE)**:

- Pomagają w uproszczeniu struktury danych, co może być pomocne przy dużej liczbie zmiennych wejściowych.
- Nie są one samodzielnymi algorytmami uczenia maszynowego, ale wspierają inne algorytmy poprzez przetwarzanie wstępne danych [65].

2.8 Autoenkoder wariacyjny z warunkowaniem

Po dogłębnej analizie kluczowych cech i zdolności różnych algorytmów uczenia maszynowego oraz specyfiki danych wejściowych, wybrany został autoenkoder wariacyjny z warunkowaniem (CVAE - Conditional Variational Autoencoder, pol. warunkowy autoenkoder wariacyjny) [45]. CVAE stanowi rozszerzenie klasycznego autoenkodera wariacyjnego (VAE) poprzez wprowadzenie dodatkowych zmiennych warunkujących [51]. Architektura CVAE składa się z enkodera, dekodera oraz warstwy próbkowania [74]. Enkoder przekształca dane wejściowe w reprezentację w przestrzeni utajonej, dekoder odtwarza dane wejściowe na podstawie próbek pobranych z tej przestrzeni, a warstwa próbkowania implementuje tzw. reparametryzację, umożliwiającą różniczkowanie i optymalizację modelu. Funkcja straty CVAE obejmuje zarówno błąd rekonstrukcji, jak i dywergencję Kullbacka-Leiblera, co zapewnia efektywne uczenie i generalizację modelu [36].

2.8.1 Zdolność do przetwarzania sygnałów dyskretnych różnego pochodzenia

Autoenkoder wariacyjny z warunkowaniem wykazuje doskonałą zdolność do przetwarzania zarówno sygnałów dyskretnych różnego pochodzenia. CVAE może być efektywnie stosowany w analizie danych, które pochodzą z różnych źródeł, takich jak sensoryczne dane ciągłe oraz zdarzenia logiczne. Przykłady takich danych obejmują:

- **Sygnały dyskretne pochodzące z przetworników analogowo-cyfrowych**: prąd baterii (mierzony w amperach, zakres 0-1000A), napięcie na ogniwach baterii (mierzone w woltach, zakres 200-400V), temperatura systemu (mierzona w stopniach Celsjusza, zakres -40°C do +85°C).
- **Sygnały dyskretne — wielostanowe**: sygnał uruchomienia poduszki powietrznej (stan logiczny 0 lub 1), stany kontaktorów (otwarte/zamknięte, kodowane jako 0/1).

CVAE przetwarza te różnorodne sygnały poprzez zastosowanie odpowiednich warstw wejściowych i funkcji aktywacji [50]. Dla sygnałów ciągłych stosuje się najczęściej funkcje aktywacji takie jak ReLU lub tanh, podczas gdy dla sygnałów dyskretnych wykorzystuje się funkcje sigmoid lub softmax [69].

2.8.2 Odporność na szum

Wariacyjny charakter autoenkodera umożliwia mu efektywne radzenie sobie z zaszumionymi sygnałami, co jest szczególnie istotne w środowisku samochodowym [42], gdzie występuje znaczne zakłócenie elektromagnetyczne oraz fluktuacje sygnałów ciągłych. CVAE, dzięki swoim właściwościom probabilistycznym, może modelować szum w danych i oddzielać go od istotnych informacji. Model CVAE implementuje to poprzez wprowadzenie losowości do procesu kodowania i dekodowania, co w efekcie prowadzi do lepszej generalizacji i odporności na szum. Technicznie, odbywa się to poprzez próbkowanie z rozkładu w przestrzeni utajonej, co można interpretować jako normalizację modelu. Dodatkowo, CVAE może być trenowany z wykorzystaniem technik augmentacji danych, takich jak dodawanie szumu gaussowskiego do danych wejściowych, co dodatkowo zwiększa jego odporność na zakłócenia [72].

2.8.3 Wysoka czułość i dokładność

Autoenkoder wariacyjny z warunkowaniem wykazuje wysoką czułość i dokładność w rozpoznawaniu zmian w sygnałach. Jego architektura umożliwia uchwycenie skomplikowanych zależności między danymi wejściowymi. Algorytm ten precyzyjnie reaguje na zmiany zarówno w sygnałach ciągłych, jak i dyskretnych. CVAE osiąga to poprzez optymalizację funkcji straty, która składa się z dwóch głównych komponentów [9]: błędu rekonstrukcji (najczęściej średni błąd kwadratowy lub binarna entropia krzyżowa) oraz dywergencji Kullbacka-Leiblera. Błąd rekonstrukcji zapewnia dokładne odtwarzanie danych wejściowych, podczas gdy dywergencja KL normalizuje przestrzeń utajoną, co przyczynia się

do lepszej generalizacji i czułości modelu. W praktyce, CVAE może wykrywać subtelne zmiany w sygnałach, rzędu miliwoltów dla napięcia lub dziesiątych części stopnia dla temperatury [10].

2.8.4 Zdolność do rekonstrukcji brakujących informacji

CVAE jest wysoce efektywny w rekonstrukcji brakujących danych, co stanowi jedno z kluczowych wymagań systemu [67]. Dzięki swojej strukturze, autoenkoder ten może generować brakujące wartości na podstawie dostępnych informacji, co zapewnia ciągłość i dokładność działania systemu. Jego probabilistyczna natura pozwala na efektywne modelowanie rozkładu danych i uzupełnianie braków. Technicznie, CVAE realizuje to poprzez kodowanie dostępnych danych do przestrzeni utajonej [58], a następnie dekodowanie z tej przestrzeni, uwzględniając warunki nałożone przez zmienne warunkujące. W przypadku brakujących danych, model może generować prawdopodobne wartości poprzez próbkowanie z wyuczonego rozkładu w przestrzeni utajonej. Dokładność rekonstrukcji można ocenić za pomocą miar takich jak średni błąd kwadratowy (MSE) lub współczynnik determinacji (R^2) dla danych ciągłych, oraz dokładność klasyfikacji dla danych dyskretnych.

2.8.5 Wykrywanie anomalii

Dzięki zdolności do modelowania danych wejściowych, CVAE może efektywnie wykrywać anomalie [83]. Algorytm ten identyfikuje nietypowe wzorce w danych, co umożliwia szybkie wykrycie potencjalnych problemów i zapobieganie awariom systemu. Anomalie mogą być identyfikowane poprzez porównanie rzeczywistych danych z danymi rekonstruowanymi przez autoenkoder. W praktyce, proces wykrywania anomalii przy użyciu CVAE obejmuje następujące etapy:

1. Kodowanie danych wejściowych do przestrzeni utajonej.
2. Dekodowanie z przestrzeni utajonej do przestrzeni wejściowej.
3. Obliczenie błędu rekonstrukcji (np. za pomocą MSE).
4. Porównanie błędu rekonstrukcji z ustalonym progiem.

Jeśli błąd rekonstrukcji przekracza ustalony próg, dana obserwacja jest klasyfikowana jako anomalia. Próg ten może być ustalany dynamicznie, na przykład jako wielokrotność odchylenia standardowego błędów rekonstrukcji dla danych treningowych. Dodatkowo, CVAE może być rozszerzony o mechanizmy

uwagi (attention mechanisms) [86], co pozwala na identyfikację, które konkretnie cechy lub elementy sygnału przyczyniają się do wykrycia anomalii.

2.8.6 Architektura CVAE

Przedstawiony na rysunku 2.18 diagram ilustruje architekturę Warunkowego Autoenkodera Wariacyjnego (CVAE - Conditional Variational Autoencoder) [76]. CVAE to zaawansowany model uczenia maszynowego, który łączy cechy autoenkodera z podejściem wariacyjnym, umożliwiając generowanie danych warunkowych.

Architektura CVAE składa się z kilku kluczowych elementów, z których każdy posiada szczegółową strukturę i specyfikację. Poniżej przedstawiono rozszerzony opis poszczególnych komponentów wraz z dodatkowymi informacjami technicznymi.

1. **Sieć Propozycji (Enkoder)**:
   - **Wejście**:
     - Przyjmuje znormalizowane dane oraz maski określające obserwowane cechy.
     - Warstwa transformująca kategorie na wektory binarne (maski).
   - **Architektura**:
     - Pierwsza warstwa liniowa (nn.Linear) przetwarza wejście.
     - Następnie, sieć składa się z 5 głębokich warstw liniowych z funkcjami aktywacji Leaky-ReLU oraz połączeniami rezydualnymi, co zwiększa zdolność modelu do uchwycenia złożonych zależności w danych.
     - Ostatnia warstwa liniowa generuje podwójne wartości dla każdego wymiaru latentnego, odpowiadające średniej i logarytmowi wariancji (32×2 wymiarów latentnych).
     - Szerokość (liczba neuronów w warstwach ukrytych) wynosi 128, a głębokość (łączna liczba warstw ukrytych poza pierwszą i ostatnią) to 5.
   - **Funkcje Aktywacji**:
     - Używane funkcje aktywacji to LeakyReLU, które pomagają w uniknięciu problemu znikającego gradientu i pozwalają na przepływ gradientów nawet dla ujemnych wartości wejściowych.
   - **Reprezentacja Rozkładu**:

- Sieć Propozycji jest reprezentowana przez rozkład $q_\phi(z|x, b)$, gdzie z oznacza zmienne latentne, x to dane wejściowe, a b to maska określająca obserwowane cechy.

2. **Sieć Prior (Enkoder)**:

   - **Wejście**:
     - Opiera się wyłącznie na maskach oraz obserwowanych cechach (x_{1-b}).
     - Warstwa transformująca kategorie na wektory binarne (maski).
   - **Architektura**:
     - Sieć zawiera 5 głębokich warstw liniowych z funkcjami aktywacji LeakyReLU oraz połączeniami rezydualnymi, z których każda wykorzystuje warstwę pamięci do przechowywania danych wejściowych.
     - Ostatnia warstwa liniowa generuje parametry rozkładu latentnego (32×2 wymiarów latentnych).
     - Szerokość i głębokość sieci są analogiczne do sieci Propozycji, co zapewnia spójność architektoniczną.
   - **Funkcje Aktywacji**:
     - Podobnie jak w sieci Propozycji, używane są funkcje aktywacji LeakyReLU.
   - **Reprezentacja Rozkładu**:
     - Sieć Prior jest reprezentowana przez rozkład $p_\psi(z|x_{1-b}, b)$, gdzie x_{1-b} oznacza obserwowane cechy obiektu.
   - **Zastosowanie**:
     - Umożliwia modelowi wnioskowanie o rozkładzie zmiennych latentnych przy braku niektórych obserwacji, co zwiększa elastyczność i odporność modelu na brakujące dane.

3. **Sieć Generatywna (Dekoder)**:

   - **Wejście**:
     - Przyjmuje próbkowane zmienne latentne z, a także obserwowane cechy x_{1-b} i maski b.
   - **Architektura**:
     - Zaczyna się od jednej warstwy liniowej.

  - Następnie, sieć zawiera 6 warstw liniowych z funkcjami aktywacji LeakyReLU oraz połączeniami rezydualnymi, z których każda wykorzystuje MemoryLayer do integracji z siecią Prior.
  - Ostatnia warstwa liniowa generuje rekonstrukcję danych x_b.
  - Dodatkowo, warstwa ustawiająca sigmy rozkładu gaussowskiego na jedynkę na końcu sieci zapewnia stabilność generowanych rozkładów.

- **Funkcje Aktywacji**:
  - Używane funkcje aktywacji to LeakyReLU, co wspiera efektywne trenowanie głębokiej sieci.

- **Reprezentacja Rozkładu**:
  - Sieć Generatywna jest reprezentowana przez rozkład $p_\theta(x_b|z, x_{1-b}, b)$, gdzie x_b oznacza nieobserwowane cechy obiektu.

- **Zastosowanie**:
  - Generuje rekonstrukcję oryginalnych danych, uwzględniając jednocześnie obserwowane cechy i maski, co pozwala na dokładne odtworzenie brakujących informacji.

4. **Funkcja Straty**:

- **Składniki**:
  - Oblicza rozbieżność KL (Kullbacka-Leiblera) między rozkładami z sieci Propozycji $q_\phi(z|x, b)$ i Prior $p_\psi(z|x_{1-b}, b)$.
  - Ocenia jakość rekonstrukcji generowanej przez sieć Generatywną $p_\theta(x_b|z, x_{1-b}, b)$.

- **Cel**:
  - Minimalizacja funkcji straty pozwala na optymalizację parametrów wszystkich sieci w architekturze CVAE, zapewniając zarówno dokładne odwzorowanie danych, jak i zgodność rozkładów latentnych.

Dodatkowe Informacje Techniczne

- **Warstwy Pamięci**:

  - W sieci Prior oraz Generatywnej zastosowano warstwy pamięci (MemoryLayer), które umożliwiają przechowywanie i integrację danych wejściowych, co zwiększa zdolność modelu do uchwycenia długoterminowych zależności w danych.

- **Maski**:
  - Maski binarne przekształcają kategorie danych na format odpowiedni do przetwarzania przez sieci, umożliwiając modelowi identyfikację obserwowanych i nieobserwowanych cech.

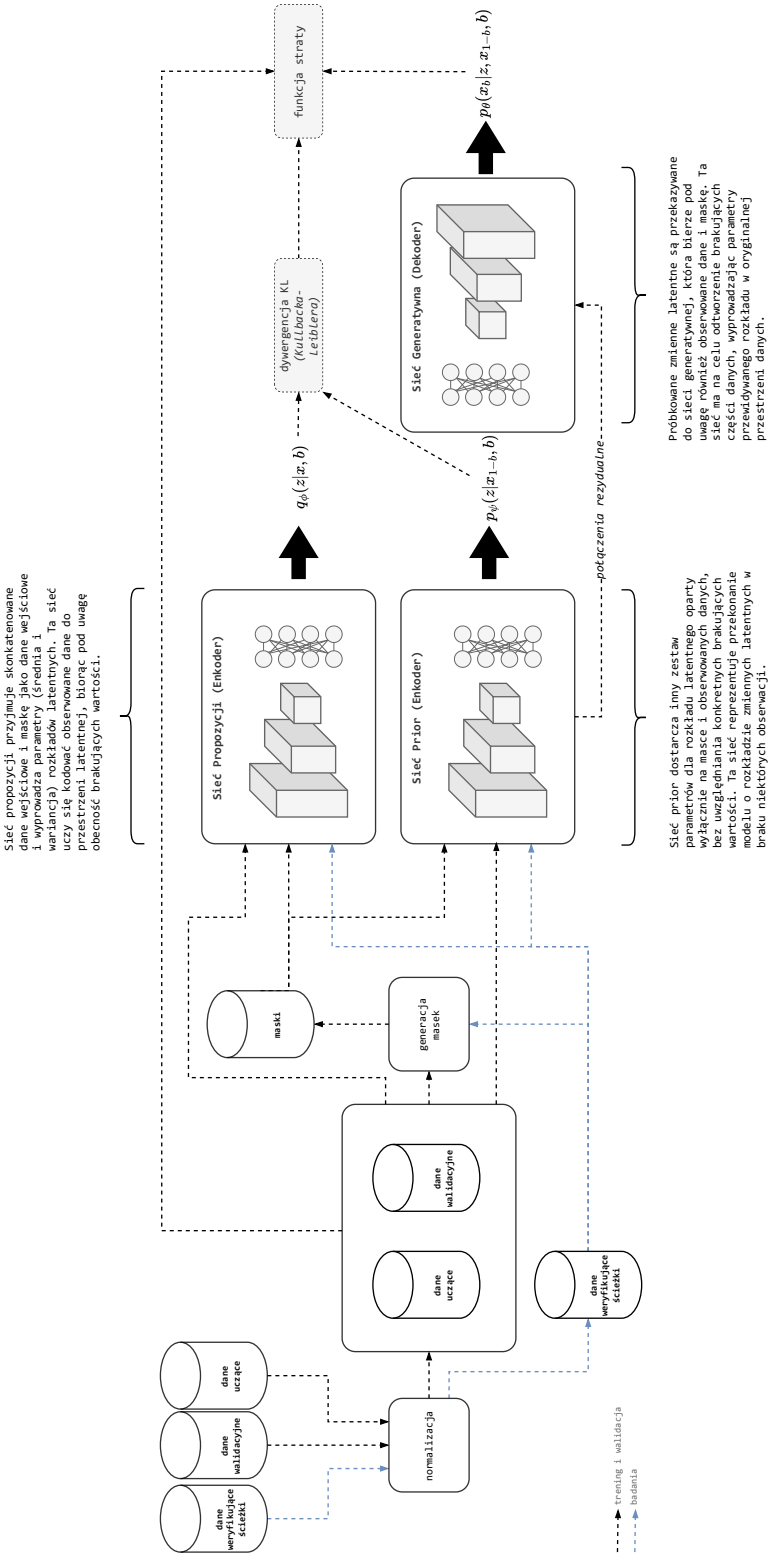
- **Rozmiary Latentne**:
  - Wymiar latentny wynosi 32×2, gdzie każda para odpowiada średniej i logarytmowi wariancji dla danej zmiennej latentnej.

Proces działania CVAE jest złożony i obejmuje szereg kluczowych etapów, które współpracują ze sobą, aby umożliwić efektywne generowanie warunkowych danych. Poniżej przedstawiono szczegółowy opis kolejnych faz przetwarzania danych w CVAE:

1. Dane wejściowe są normalizowane i generowane są odpowiednie maski.
2. Sieć Propozycji $q_\phi(z|x, b)$ koduje dane wejściowe do przestrzeni latentnej, uwzględniając maski.
3. Równolegle, sieć Prior $p_\psi(z|x_{1-b}, b)$ dostarcza alternatywny zestaw parametrów rozkładu latentnego.
4. Zmienne latentne są próbkowane z wykorzystaniem techniki reparametryzacji.
5. Sieć Generatywna $p_\theta(x_b|z, x_{1-b}, b)$ dekoduje próbkowane zmienne latentne, generując rekonstrukcję danych.
6. Funkcja straty ocenia jakość rekonstrukcji oraz zgodność rozkładów latentnych $q_\phi(z|x, b)$ i $p_\psi(z|x_{1-b}, b)$.

CVAE uczy się kodować obserwowane dane do przestrzeni latentnej, biorąc pod uwagę obecność brakujących wartości. Model ten charakteryzuje się zdolnością do generowania nowych danych, uwzględniając zarówno obserwowane cechy, jak i brakujące wartości. Dzięki temu CVAE znajduje zastosowanie w zadaniach generowania warunkowego oraz imputacji brakujących danych.

Warto podkreślić, że architektura ta wykorzystuje połączenia rezydualne między różnymi komponentami [34], co może poprawiać przepływ gradientów podczas treningu i stabilizować proces uczenia. Dodatkowo, zastosowanie dwóch enkoderów (sieci Propozycji $q_\phi(z|x, b)$ i Prior $p_\psi(z|x_{1-b}, b)$) umożliwia modelowi lepsze radzenie sobie z niepełnymi danymi.



Rysunek 2.18: Architekturę Warunkowego Autoenkodera Wariacyjnego [76]

Rozdział 3

Koncepcja metodyki

Celem opracowanej metodologii jest stworzenie ram, które umożliwią wiarygodną i efektywną ocenę funkcji bezpieczeństwa pod kątem spełniania wymagań czasowych. Metodologia ma także na celu usprawnienie procesu testowania poprzez wykorzystanie technik sztucznej inteligencji, które umożliwiają precyzyjniejszą analizę zależności czasowych oraz identyfikację potencjalnych zagrożeń wynikających z opóźnień w działaniu funkcji bezpieczeństwa.

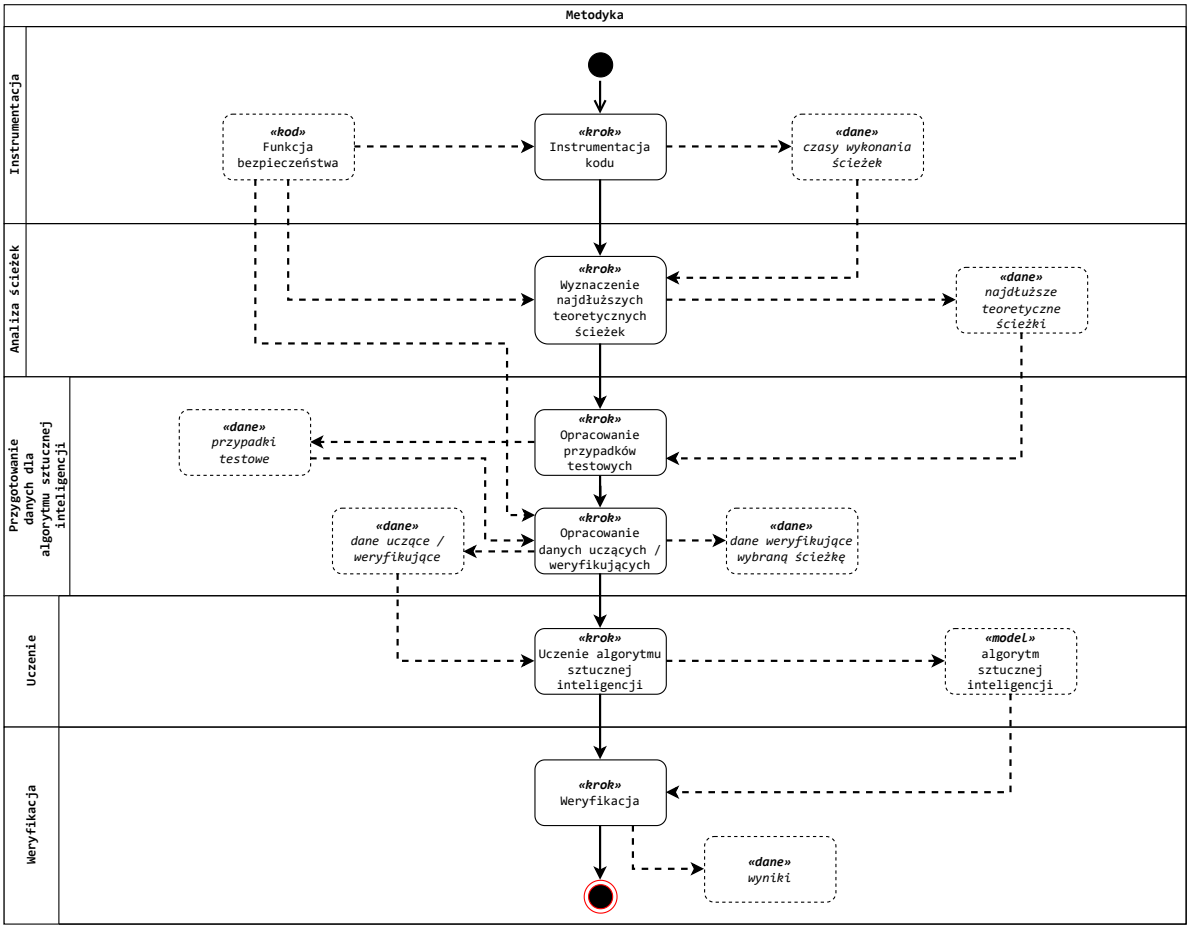
Aby ocenić stan sterowników układów bateryjnych samochodów elektrycznych, opracowano procedurę testowania tych układów w odniesieniu do wymagań czasowych dotyczących funkcji bezpieczeństwa. Głównym celem tej procedury jest sprawdzenie, czy funkcje bezpieczeństwa w sterownikach systemów bateryjnych spełniają rygorystyczne wymagania czasowe, co jest kluczowe dla bezpieczeństwa użytkowników pojazdów oraz innych uczestników ruchu drogowego.

Metodologia zakłada kompleksowe podejście do oceny czasowej funkcji bezpieczeństwa, uwzględniające zarówno aspekty teoretyczne, jak i praktyczne. Istotne jest uzyskanie pełnego pokrycia wszystkich możliwych ścieżek wykonania funkcji, co umożliwia identyfikację najdłuższych czasów wykonania oraz ich analizę.

Proces ten składa się z sześciu etapów:

1. **Instrumentacja kodu**: Modyfikacja kodu funkcji bezpieczeństwa w celu pozyskania informacji o wykonywanych ścieżkach.
2. **Analiza ścieżek**: Identyfikacja i ocena wszystkich możliwych ścieżek wykonania funkcji bezpieczeństwa.
3. **Wybór algorytmu sztucznej inteligencji**: Dobór odpowiedniego algorytmu, który będzie skutecznie testował funkcje pod kątem możliwych ścieżek wykonania dla testowanej funkcji bezpieczeństw.
4. **Przygotowanie danych dla algorytmu**: Zbieranie i przygotowanie niezbędnych danych wejściowych do procesu testowania.
5. **Uczenie algorytmu**: Trenowanie algorytmu w celu optymalizacji jego zdolności do wykrywania opóźnień i zagrożeń.
6. **Weryfikacja poprawności modelu**: Ocena wyników testów w celu potwierdzenia skuteczności modelu oraz jego zgodności z założonymi wymaganiami.

Każdy z tych etapów przyczynia się do osiągnięcia głównych celów metodologii, zapewniając kompleksową i dokładną ocenę funkcji bezpieczeństwa. Dzięki zastosowaniu zaawansowanych metod sztucznej inteligencji proces testowania staje się bardziej efektywny, a analiza zależności czasowych pozwala na wczesne wykrywanie potencjalnych problemów, co znacząco podnosi poziom bezpieczeństwa systemów bateryjnych w pojazdach elektrycznych.



Rysunek 3.1: Koncepcja metodyki

3.0.1 Opis metodyki

Proponowana metodyka testowania sterowników systemów bateryjnych pojazdów wsparta modelem zakłada realizację następujących etapów:

Etap 1: Instrumentacja

Pierwszym etapem metodyki jest instrumentacja kodu wybranej funkcji bezpieczeństwa. Celem tego kroku jest uzyskanie dokładnych pomiarów czasów wykonania dla każdej ze ścieżek kodu. Proces ten odbywa się na docelowym sprzęcie, co zapewnia zaufanie do uzyskanych wyników. W trakcie instrumentacji dobierane są odpowiednie parametry wejściowe dla węzłów decyzyjnych, tak aby uzyskać pełne pokrycie wszystkich możliwych ścieżek wykonania. Wynikiem tego etapu jest zestaw ścieżek

oznaczonych unikalnym identyfikatorem, dla których zapisany jest najdłuższy zarejestrowany czas wykonania.

Etap 2: Analiza ścieżek

Na podstawie danych uzyskanych w poprzednim etapie przeprowadzana jest analiza ścieżek wykonania. W tym kroku wyznaczana jest najdłuższa teoretyczna ścieżka wykonania wybranej funkcji bezpieczeństwa. Do tego celu wykorzystuje się odpowiednią metodę analizy, która może obejmować techniki takie jak analiza statyczna lub dynamiczna, czy też bardziej zaawansowane metody jak analiza formalna. Wynikiem tego etapu jest zidentyfikowanie najdłuższej teoretycznej ścieżki wykonania, która stanowi kluczowy punkt odniesienia dla dalszych etapów metodyki.

Etap 3: Wybór algorytmu sztucznej inteligencji

W trzecim etapie metodyki następuje wybór odpowiedniego algorytmu sztucznej inteligencji, który zostanie wykorzystany do analizy i predykcji ścieżek wykonania funkcji bezpieczeństwa. Decyzja o wyborze algorytmu jest kluczowa dla skuteczności całego procesu testowania i opiera się na specyfice problemu oraz charakterystyce danych zebranych w poprzednich etapach.

Etap 4: Przygotowanie danych dla algorytmu sztucznej inteligencji

Kolejnym etapem jest przygotowanie danych, które zostaną wykorzystane do trenowania i weryfikacji algorytmu sztucznej inteligencji. Proces ten obejmuje dwa kroki: opracowanie przypadków testowych oraz przygotowanie danych uczących i weryfikujących. Przypadki testowe są projektowane w taki sposób, aby sprawdzić, czy wyznaczona teoretycznie najdłuższa ścieżka wykonania jest rzeczywiście najdłuższą ścieżką w rzeczywistym działaniu systemu. Dane uczące i walidujące są następnie przygotowywane w formacie akceptowanym przez wybrany algorytm, uwzględniając strukturę funkcji bezpieczeństwa oraz przypadki testowe.

Etap 5: Uczenie algorytmu

Na tym etapie algorytm sztucznej inteligencji jest trenowany przy użyciu przygotowanych wcześniej danych. Celem tego kroku jest uzyskanie modelu, który będzie w stanie przewidzieć najdłuższą ścieżkę wykonania funkcji bezpieczeństwa na podstawie danych wejściowych. Proces uczenia obejmuje iteracyjne

dostosowywanie modelu, aby osiągnąć jak największą dokładność w predykcjach czasów wykonania oraz w identyfikacji krytycznych ścieżek wykonania.

Etap 6: Weryfikacja

Ostatnim etapem metodyki jest weryfikacja uzyskanego modelu. W tym kroku przy użyciu wytrenowanego modelu przeprowadzana jest weryfikacja, czy wyznaczona teoretyczna najdłuższa ścieżka wykonania jest zgodna z rzeczywistą najdłuższą ścieżką wykonaną przez system. Weryfikacja ta ma kluczowe znaczenie dla oceny, czy funkcje bezpieczeństwa spełniają wymagania czasowe. Wyniki weryfikacji są podstawą dalszej optymalizacji systemu.

3.1 Transformacja funkcji bezpieczeństwa

Po wyborze odpowiedniego algorytmu uczenia maszynowego, którym jest wariacyjny autoenkoder z warunkowaniem, kluczowe jest szczegółowe przedstawienie sposobu, w jaki elementy funkcji bezpieczeństwa, takie jak węzły decyzyjne, są transformowane na wektor danych. Proces ten ma fundamentalne znaczenie dla zrozumienia, jak dane wejściowe są przekształcane w zestaw danych wykorzystywany przez algorytm uczenia maszynowego, co jest kluczowe dla uzyskania dokładnych i efektywnych wyników. Transformacja funkcji bezpieczeństwa rozpoczyna się od przypisania każdemu sygnałowi wejściowemu unikalnej liczby całkowitej, zaczynając od 1. Sygnały wejściowe mogą obejmować różnorodne dane, takie jak informacje sensoryczne, dane użytkownika czy parametry systemowe, które są niezbędne do prawidłowego działania algorytmu.

Kolejnym etapem transformacji jest przypisanie każdej ścieżce, rozpoczynającej się od węzła decyzyjnego i kończącej blokiem instrukcji, kolejnej liczby całkowitej, przy czym numeracja zaczyna się od liczby o jeden większej niż liczba sygnałów wejściowych. Ścieżki te mogą reprezentować różne scenariusze lub decyzje, jakie algorytm podejmuje w odpowiedzi na dane wejściowe. W ten sposób tworzy się ciąg liczb składający się z indeksów sygnałów oraz ścieżek. Indeksy te mają kluczowe znaczenie, gdyż wskazują dokładną lokalizację sygnału wejściowego lub ścieżki w wektorze danych, który powstaje na każdym etapie działania algorytmu.

Proces ten można opisać matematycznie, co pomaga w jego precyzyjnym zrozumieniu i implementacji. Niech $s_1, s_2, s_3, \ldots$ oznaczają kolejne indeksy sygnałów wejściowych, a $p_1, p_2, p_3, \ldots$ oznaczają kolejne indeksy ścieżek. Wektor danych $\mathbf{d}$ dla danego kroku k tworzą wartości sygnałów wejściowych oraz liczba wykonań ścieżek, zapisana jako liczba całkowita pod indeksem danej ścieżki:

$$\mathbf{d} = \begin{bmatrix} s_1 & s_2 & s_3 & \ldots & p_1 & p_2 & p_3 & \ldots \end{bmatrix} \tag{3.1}$$

Dla każdego kolejnego kroku wykonania funkcji bezpieczeństwa $k_1, k_2, k_3, \ldots$, powstaje nowy wektor danych, a zestaw tych wektorów tworzy dwuwymiarową tablicę danych wykorzystywaną przez algorytm uczenia maszynowego:

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_{k_1} \\ \mathbf{d}_{k_2} \\ \mathbf{d}_{k_3} \\ \vdots \end{bmatrix} \tag{3.2}$$

Wektor danych, w tym przypadku, to uporządkowany zestaw wartości liczbowych, który reprezentuje złożoną strukturę danych wejściowych i decyzji podejmowanych przez wariacyjny autoenkoder z warunkowaniem [17]. Każda wartość w wektorze odpowiada albo konkretnej wartości sygnału wejściowego, albo ścieżce decyzyjnej, którą algorytm wybiera w odpowiedzi na te sygnały. W uproszczeniu wektor danych jest numerycznym odwzorowaniem aktualnego stanu algorytmu w danym momencie, który jest analizowany przez algorytm uczenia maszynowego. W kontekście tego procesu wektor danych umożliwia wariacyjnemu autoenkoderowi efektywne przetwarzanie informacji oraz podejmowanie decyzji na podstawie wzorców wykrytych w zbiorze danych.

Wektor danych, utworzony z tych indeksów, jest kluczowym elementem w procesie przetwarzania informacji przez wariacyjny autoenkoder z warunkowaniem. Każdy wektor reprezentuje zbiór danych dla jednego kroku funkcji bezpieczeństwa, obejmując zarówno wartości sygnałów wejściowych, jak i informacje o ścieżkach decyzyjnych. Zestaw tych wektorów stanowi dane wejściowe dla algorytmu uczenia maszynowego, tworząc dwuwymiarową tablicę danych, gdzie każdy wiersz odpowiada pojedynczemu wektorowi z jednego kroku wykonania funkcji. Rysunek **??** przedstawia graficzne odwzorowanie procesu transformacji sygnałów wejściowych i ścieżek decyzyjnych na wektor danych.

3.1.1 Normalizacja wektorów danych

W celu zapewnienia efektywnego działania algorytmów uczenia maszynowego, niezbędne jest przeprowadzenie normalizacji wektorów danych. Normalizacja pozwala na skalowanie wartości wektorów do porównywalnego zakresu, co poprawia konwergencję i stabilność działania autoenkodera. Najczęściej stosowaną metodą normalizacji jest przeskalowanie wartości do przedziału [0, 1] [39].

Niech $\mathbf{d}_{i,j}$ oznacza wartość i-tego wektora danych w j-tym kroku. Przeskalowanie wartości do przedziału [0, 1] można przeprowadzić według wzoru:

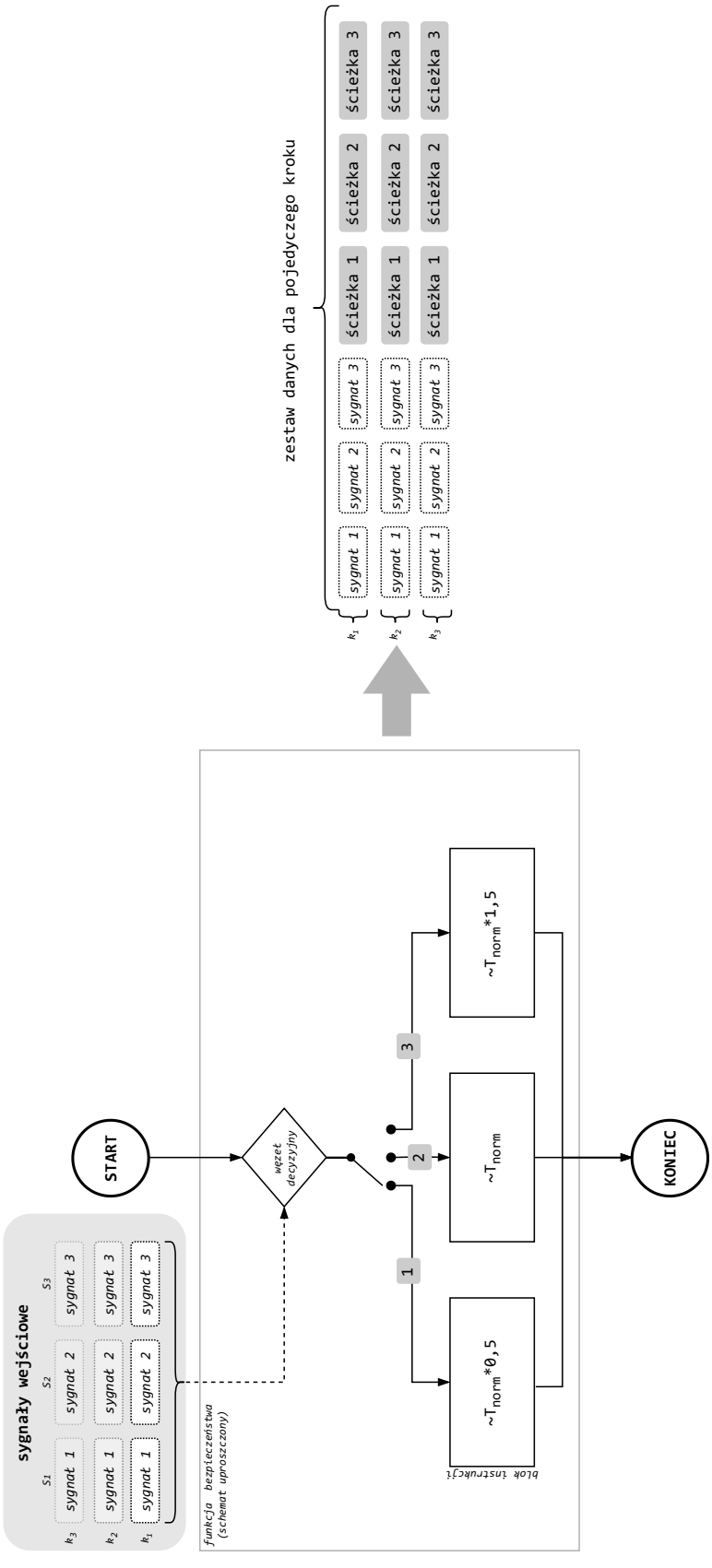
$$\mathbf{d}'_{i,j} = \frac{\mathbf{d}_{i,j} - \min(\mathbf{d}_j)}{\max(\mathbf{d}_j) - \min(\mathbf{d}_j)} \tag{3.3}$$

gdzie $\min(\mathbf{d}_j)$ i $\max(\mathbf{d}_j)$ oznaczają odpowiednio minimalną i maksymalną wartość w kolumnie j.

Kolumny w wektorze danych $\mathbf{d}$ reprezentują wartości sygnałów wejściowych lub ścieżek decyzyjnych. Każda kolumna odpowiada jednej zmiennej:

1. **Sygnały wejściowe**: Kolumny $s_1, s_2, s_3, \ldots$ w wektorze danych $\mathbf{d}$ reprezentują wartości poszczególnych sygnałów wejściowych.
2. **Ścieżki decyzyjne**: Kolumny $p_1, p_2, p_3, \ldots$ w wektorze danych $\mathbf{d}$ reprezentują liczby wykonań poszczególnych ścieżek decyzyjnych.

Wartości te są zbierane dla każdego kroku wykonania funkcji bezpieczeństwa i tworzą dwuwymiarową tablicę danych $\mathbf{D}$, gdzie każdy wiersz odpowiada pojedynczemu wektorowi danych z jednego kroku wykonania funkcji.



Rysunek 3.2: Diagram objaśniający sposób generowania przykładów uczących dla autoenkodera wariacyjnego.

3.2 Charakterystyka procesu wyboru algorytmu uczenia maszynowego

Celem tego podrozdziału jest określenie kryteriów wyboru algorytmu uczenia maszynowego. Decyzja o wyborze powinna opierać się na precyzyjnie zdefiniowanych kryteriach technicznych, które algorytm musi spełnić, aby efektywnie przetwarzać dane wejściowe oraz generować wartościowe wyniki. Te kryteria są kluczowe dla zapewnienia efektywności i praktyczności zastosowań w systemach bezpieczeństwa.

Pierwszym krokiem w określeniu oczekiwań wobec modelu jest analiza jego możliwości w generowaniu odpowiedzi na pytania dotyczące wykonalności ścieżek w rzeczywistych warunkach. Oczekiwania te można podzielić na dwie główne kategorie:

Potwierdzenie aktywności ścieżki

Potwierdzenie aktywności ścieżki opiera się na założeniu, że dane wejściowe zawierają informację o aktywności danej ścieżki. W celu matematycznego opisu tego procesu, niech:

- $\mathbf{d} = [s_1, s_2, \ldots, s_n, p_1, p_2, \ldots, p_m]$ oznacza wektor danych, gdzie $s_1, s_2, \ldots, s_n$ to sygnały wejściowe, a $p_1, p_2, \ldots, p_m$ to ścieżki.

Dane wejściowe $\mathbf{d}$ zawierają informacje o aktywności ścieżki p_i, jeżeli wartość p_i w wektorze $\mathbf{d}$ jest większa od zera. Potwierdzenie aktywności ścieżki można zdefiniować jako:

$$p_i > 0 \implies \text{ścieżka } p_i \text{ jest aktywna}$$

Algorytm powinien być zdolny do identyfikacji tej aktywności na podstawie danych wejściowych $\mathbf{d}$ i zwrócić odpowiedni wektor wyjściowy $\mathbf{d}$, gdzie:

$$\mathbf{d}_{\text{wyjściowy}} = f(\mathbf{d}) \quad \text{takie, że} \quad p_i > 0 \implies \text{ścieżka } p_i \text{ jest aktywna}$$

Wnioskowanie o możliwości wykonania ścieżki

Wnioskowanie o możliwości wykonania ścieżki polega na analizie danych wejściowych, które nie zawierają bezpośredniej informacji o aktywności ścieżki. W takim przypadku algorytm musi doko-

nać wnioskowania na podstawie korelacji między sygnałami wejściowymi. Matematycznie można to przedstawić następująco:

- $\mathbf{d} = [s_1, s_2, \ldots, s_n, p_1, p_2, \ldots, p_m]$ oznacza wektor danych, gdzie $s_1, s_2, \ldots, s_n$ to sygnały wejściowe, a $p_1, p_2, \ldots, p_m$ to ścieżki.

Dane wejściowe $\mathbf{d}$ nie zawierają bezpośredniej informacji o aktywności ścieżki p_i, co oznacza, że wartość p_i w wektorze $\mathbf{d}$ wynosi 0. Algorytm musi wnioskować o możliwości ścieżki na podstawie korelacji między danymi wejściowymi:

$$p_i = 0 \implies \text{wnioskowanie na podstawie } \mathbf{d}$$

Algorytm wykonuje wnioskowanie poprzez funkcję $g(\mathbf{d})$, która określa prawdopodobieństwo aktywności ścieżki p_i:

$$g(\mathbf{d}) = \hat{p}_i$$

gdzie $\hat{p}_i$ to przewidywana wartość dla ścieżki p_i na podstawie korelacji pomiędzy sygnałami wejściowymi $\mathbf{d}$. Ostatecznie, algorytm dokonuje predykcji, czy ścieżka p_i jest możliwa, porównując $\hat{p}_i$ z odpowiednim progiem decyzyjnym τ:

$$\hat{p}_i \geqslant \tau \implies \text{ścieżka } p_i \text{ jest możliwa}$$

3.2.1 Przebieg procesu uczenia

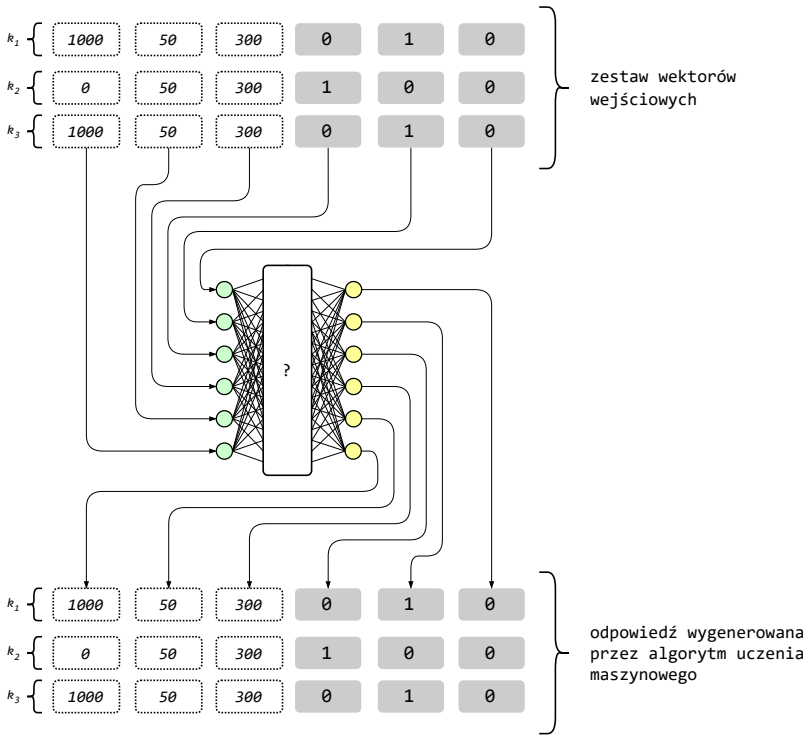
Proces uczenia algorytmu wykorzystał autoenkoder wariacyjny z warunkowaniem, zakładając, że wektor danych wyjściowych miał identyczną postać jak wektor danych wejściowych. Proces ten został podzielony na kilka etapów:

1. **Przygotowanie danych**: Dane wejściowe odzwierciedlały rzeczywiste warunki działania funkcji bezpieczeństwa. Zestaw danych został podzielony na dwie części: dane uczące i dane walidujące.

2. **Podział na zestawy**: Pierwsza część danych została użyta do trenowania algorytmu, a druga do jego weryfikacji. Do tego zestawu przygotowano wektory na podstawie znajomości węzła

decyzyjnego, pokazujące kombinacje sygnałów wejściowych, przy których aktywowana jest ścieżka.

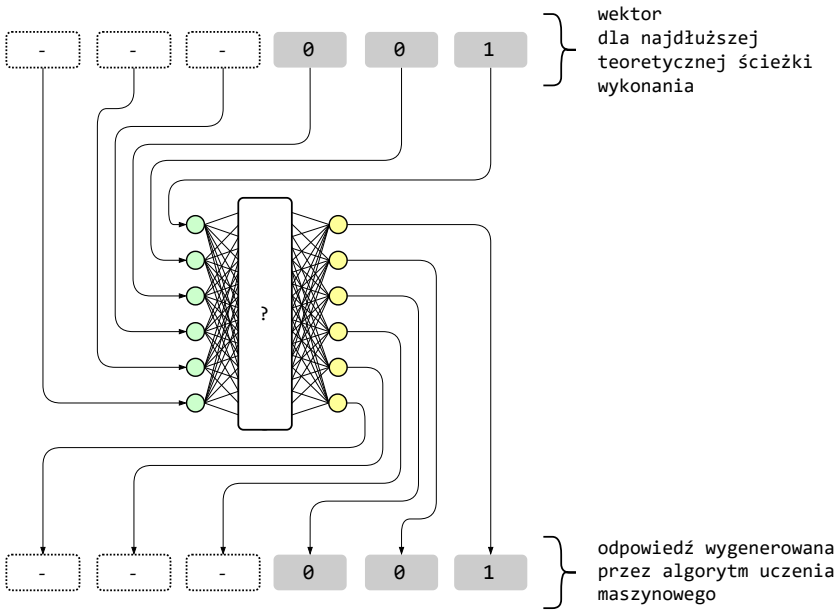
3. **Proces uczenia**: Algorytm uczył się na podstawie zestawu danych uczących, dostosowując swoje parametry tak, aby poprawnie rekonstruować brakujące informacje i wykrywać zależności między sygnałami wejściowymi.



Rysunek 3.3: Wizualizacja procesu uczenia

3.2.2 Przebieg procesu weryfikacji

Weryfikacja algorytmu polegała na przygotowaniu wektorów, gdzie część sygnałowa była ukryta, a część zawierająca informacje o ścieżkach odzwierciedlała wybraną ścieżkę do weryfikacji. Algorytm dostarczył wyjściowy wektor identyczny z wejściowym, jeśli wybrana ścieżka była możliwa w rzeczywistości. W przeciwnym razie różnica w wektorze wyjściowym sugerowała, że ścieżka była tylko teoretyczna.



Rysunek 3.4: Wizualizacja procesu weryfikacji

3.2.3 Analiza sygnałów wejściowych

W celu skutecznego doboru algorytmu uczenia maszynowego konieczne było dokładne zrozumienie charakterystyki sygnałów wejściowych. Funkcja bezpieczeństwa operowała wyłącznie na sygnałach dyskretnych pochodzących z różnych źródeł. Algorytmy musiały być zdolne do przetwarzania sygnałów o różnorodnym charakterze, co wymagało zastosowania odpowiednich metod analizy i przetwarzania danych.

Podział sygnałów dyskretnych

Sygnały pochodzące z przetworników analogowo-cyfrowych – są to sygnały reprezentujące fizyczne wartości [27]. Przykłady:

- Natężenie prądu baterii zmieniające się w zależności od obciążenia systemu, reprezentowany przez sekwencję dyskretnych wartości.
- Napięcie na ogniwach baterii, które zmienia się w czasie i jest przetwarzane na formę dyskretną za pomocą przetwornika.

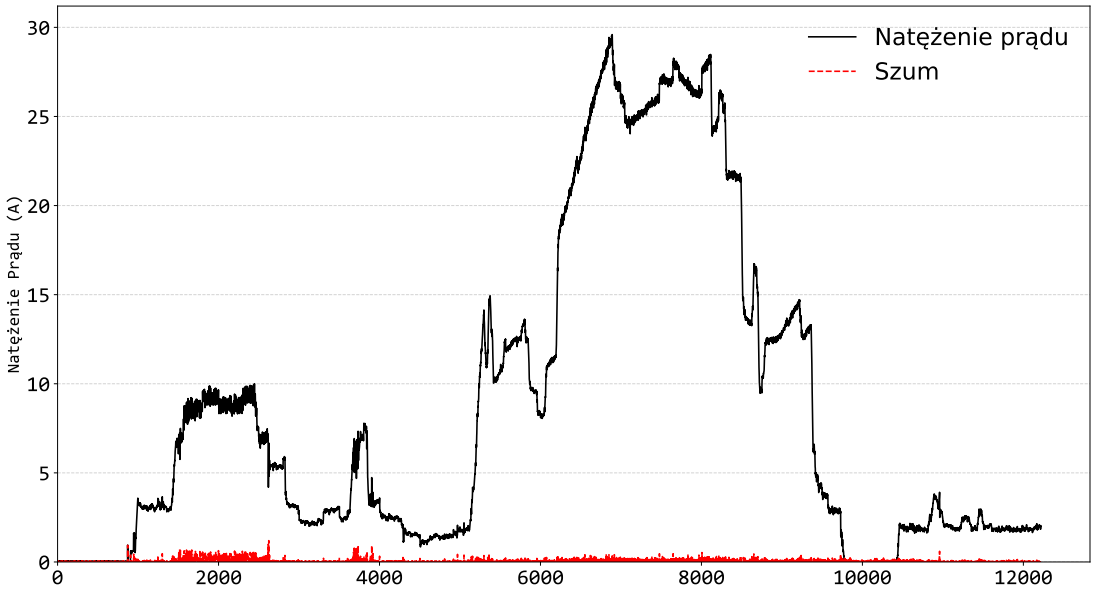
Sygnały wielostanowe, logiczne – sygnały te przyjmują jeden z określonych stanów z puli możliwych wartości, a każdy stan odpowiada określonej sytuacji operacyjnej układu lub systemu. Przykłady:

- Sygnał uruchomienia poduszki powietrznej, który może być aktywny lub nieaktywny.
- Sygnały wynikowe procesów diagnostycznych, które informują o rodzaju rozpoznanej usterki.

Charakterystyka sygnałów pochodzących z przetworników analogowo-cyfrowych

Sygnały pochodzące z przetworników, takie jak prąd baterii, są przekształcane na formę dyskretną, co umożliwia ich dalszą analizę w algorytmach uczenia maszynowego. Mimo że sygnały te są dyskretne, mogą zawierać szum, wynikający z fluktuacji w systemie pojazdu elektrycznego. Na przykład prąd baterii może podlegać nagłym zmianom spowodowanym przyspieszaniem, hamowaniem rekuperacyjnym czy uruchamianiem systemów pokładowych, takich jak klimatyzacja czy wspomaganie kierownicy [85].

Na rysunku 3.5 przedstawiono zmienność prądu pobieranego z baterii podczas spokojnej jazdy. Zakłócenia i szumy stanowią wyzwanie dla algorytmów uczenia maszynowego, które muszą wyodrębniać istotne informacje z zanieczyszczonych danych. Skuteczny algorytm musi rozróżniać wzorce istotne od zakłóceń, aby zapewnić niezawodność i stabilność wyników.



Rysunek 3.5: Pobór prądu baterii podczas spokojnej jazdy

Charakterystyka sygnałów wielostanowych

Sygnały wielostanowe, takie jak te sterujące pracą podzespołów samochodowych, przyjmują jeden z kilku możliwych stanów. Ich analiza wymaga identyfikacji bieżącego stanu oraz przewidywania możliwych przejść między stanami. Na przykład sygnały sterujące poduszką powietrzną są aktywowane tylko w określonych warunkach, takich jak wykrycie kolizji pojazdu [6].

3.3 Symulowane błędy - scenariusze testowe

W niniejszym rozdziale przedstawiona zostanie struktura scenariuszy testowych, opracowana w celu weryfikacji skuteczności proponowanej metodyki testowania sterowników pojazdów elektrycznych. Zasadniczym celem tego podejścia jest identyfikacja najdłuższych ścieżek wykonania, które z racji istniejących zależności między sygnałami wejściowymi są niemożliwe do zrealizowania. Opisane poniżej scenariusze testowe mają na celu zapewnienie kompleksowej oceny omawianej metodyki, a ich wyniki mogą służyć jako podstawy do jej optymalizacji i wdrożenia w przemyśle.

3.3.1 Elementy scenariuszy testowych

W celu zapewnienia kompleksowej i skutecznej weryfikacji zaproponowanej metodyki testowania, scenariusze testowe muszą być odpowiednio zaprojektowane i zawierać kluczowe elementy. Poniżej przedstawiono te, które powinny zostać uwzględnione w strukturze każdego scenariusza testowego.

- **Algorytm funkcji bezpieczeństwa**: Zawiera węzły decyzyjne, więcej niż jedną ścieżkę wykonania oraz określone sygnały wejściowe. Może być syntetyczny jak i rzeczywisty, czyli oparty o realną funkcję bezpieczeństwa.
- **Zestaw danych wejściowych**: Tworzony na podstawie sygnałów wejściowych przypisanych do wybranego algorytmu.
- **Dane walidujące**: Opracowywane w zależności od celu scenariusza i wybranego algorytmu, służące do sprawdzenia wykonania wybranej ścieżki.
- **Dane uczące (opcjonalne)**: Stosowane do nauczania algorytmu, jak poszczególne węzły reagują na dane wejściowe, co wpływa na ścieżkę wykonania.

3.3.2 Podział scenariuszy testowych

Scenariusze testowe podzielono ze względu na pochodzenie danych i charakter sygnałów wejściowych. Sposób podziału zaprezentowano na rysunku 3.6.

Podział ze względu na pochodzenie danych

Scenariusze testowe mogą być również klasyfikowane według pochodzenia danych, które są używane w testach. Dzieli się je na dane syntetyczne oraz dane rzeczywiste.

- **Dane syntetyczne**

  Dane syntetyczne to dane, które są generowane specjalnie na potrzeby weryfikacji metodyki testowej. Pozwalają one na pełną kontrolę nad badanym środowiskiem, co umożliwia dokładne modelowanie scenariuszy testowych. Wykorzystanie danych syntetycznych pozwala na odseparowanie logiki testowanego algorytmu od samej procedury testowej, co ułatwia ocenę wyników i minimalizuje wpływ nieprzewidzianych zmiennych.

- **Dane rzeczywiste**

  Dane rzeczywiste są pozyskiwane z funkcjonującego systemu w pojeździe elektrycznym, w tym przypadku z czujnika multimodalnego. Użycie rzeczywistych danych pozwala na weryfikację metodyki w warunkach zbliżonych do tych, w których będzie ona rzeczywiście wykorzystywana. Dzięki temu możliwe jest potwierdzenie użyteczności metodyki w rzeczywistych zastosowaniach przemysłowych.

Podział ze względu na charakter sygnałów wejściowych

Scenariusze testowe mogą być podzielone w zależności od rodzaju sygnałów wejściowych, na których bazuje weryfikowany algorytm. W tej kategorii wyróżnia się dwa typy sygnałów reprezentujące wartości logiczne oraz fizyczne.

- **Wartości logiczne**: Scenariusze te weryfikują działanie metodyki w sytuacjach, gdy sygnały wejściowe mają postać sygnału wielostanowego, który reprezentuje określone wartości logiczne, takie jak sygnał wyzwalający bezpiecznik pirotechniczny. Tego rodzaju scenariusze są kluczowe dla oceny funkcji bezpieczeństwa, gdzie precyzja w rozpoznawaniu sygnałów logicznych jest niezwykle istotna.

- **Wartości fizyczne**: Ta kategoria obejmuje scenariusze, w których sygnały wejściowe reprezentują wartości fizyczne, pochodzące z przetworników analogowo-cyfrowych, takie jak natężenie prądu czy napięcie. Scenariusze te mają na celu sprawdzenie, jak metodyka radzi sobie z przetwarzaniem i interpretacją sygnałów fizycznych, co jest kluczowe dla funkcji kontrolnych w systemach sterowania pojazdów.

Podział ten został wprowadzony, aby uwzględnić specyficzne właściwości sygnałów wejściowych, które mogą znacząco wpływać na działanie testowanego algorytmu. W praktyce, scenariusze testowe muszą być odpowiednio dostosowane do charakterystyki sygnałów, aby zapewnić wiarygodne wyniki testów.

Podział ze względu na ścieżkę wykonania

Kolejnym kryterium podziału scenariuszy testowych jest ścieżka wykonania algorytmu. W tym kontekście można wyróżnić dwie kategorie scenariuszy:

- **Możliwa ścieżka**: Obejmuje scenariusze, w których dla danego algorytmu oraz zestawu danych wejściowych istnieje możliwość wykonania określonej ścieżki. Scenariusze te pozwalają na ocenę, czy metodyka prawidłowo identyfikuje ścieżki, które są zgodne z zaprogramowanymi regułami i warunkami.
- **Niemożliwa ścieżka**: Dotyczy scenariuszy, w których dla danego algorytmu oraz zestawu danych wejściowych ścieżka wykonania nie jest możliwa do realizacji. Analiza takich scenariuszy jest niezbędna do oceny, czy metodyka skutecznie wykrywa ścieżki, które ze względu na zależności pomiędzy sygnałami wejściowymi są niemożliwe do wykonania.

Podział ten jest kluczowy dla pełnej oceny użyteczności metodyki, gdyż pozwala na sprawdzenie, czy jest ona w stanie rozpoznać zarówno ścieżki możliwe, jak i niemożliwe do wykonania.

Podział ze względu na obecność danych uczących dla węzłów decyzyjnych

W strukturze scenariuszy testowych uwzględnia się również podział związany z obecnością danych uczących. Wyróżnia się dwa główne przypadki:

- **Dane uczące obecne**: Scenariusze, w których algorytm jest wspierany przez dane uczące, co pozwala na jego adaptację i optymalizację w reakcji na różne sygnały wejściowe.
- **Dane uczące nieobecne**: Scenariusze, w których algorytm działa bez wsparcia danych uczących, co pozwala na ocenę jego podstawowej zdolności do przetwarzania sygnałów wejściowych oraz rozpoznawania ścieżek wykonania.

Analiza różnic w wynikach pomiędzy scenariuszami z obecnością oraz bez obecności danych uczących jest istotna dla zrozumienia wpływu tych danych na skuteczność metodyki.

Podział ze względu na liczbę próbek w zestawach danych

Ostatnim omawianym kryterium podziału scenariuszy testowych jest liczba próbek w zestawach danych. W tej kategorii wyróżnia się następujące poziomy:

- **Niska liczba próbek**: Zestaw danych stanowi 10% oryginalnego zestawu.
- **Średnia liczba próbek**: Zestaw danych stanowi 50% oryginalnego zestawu.
- **Wysoka liczba próbek**: Zestaw danych stanowi 100% oryginalnego zestawu.

Podział ten został wprowadzony w celu zbadania, w jaki sposób różna liczba próbek danych wpływa na wyniki metodyki testowej. Analiza ta pozwala na ocenę, czy skuteczność metodyki jest zależna od skali zestawu danych oraz czy może być ona optymalizowana w zależności od dostępnych zasobów.

Rysunek 3.6: Wizualizacja scenariuszy testowych

3.4 Ocena poprawności działania autoenkodera

Do oceny poprawności działania autoenkodera zastosowano kilka miar. Pierwszą z nich był błąd średniokwadratowy (RMSE), który pozwala na ocenę różnicy pomiędzy przewidywanymi a rzeczywistymi wartościami danych. Kolejnymi miarami są Variational Lower Bound (VLB) oraz Importance Weighted Autoencoder (IWAE), które dostarczają dodatkowych informacji na temat jakości rekonstrukcji i efektywności modelu.

3.4.1 Błąd średniokwadratowy (RMSE - Root Mean Square Error)

Root Mean Square Error (RMSE) jest miarą różnicy pomiędzy wartościami przewidywanymi przez model a wartościami rzeczywistymi. W kontekście wariacyjnych autoenkoderów z warunkowaniem, RMSE jest często wykorzystywany jako funkcja strat służąca do oceny jakości rekonstrukcji danych wejściowych. Im niższa wartość RMSE, tym lepsza jakość rekonstrukcji danych. Przykładowo, RMSE równy 0 oznacza idealną rekonstrukcję bez żadnych błędów, natomiast wyższe wartości wskazują na większe odchylenia między danymi rzeczywistymi a przewidywanymi [54].

Formuła matematyczna:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.4}$$

gdzie:

- N – liczba próbek,
- y_i – rzeczywista wartość,
- $\hat{y}_i$ – wartość przewidziana przez model.

3.4.2 Dolna granica funkcji wiarygodności (VLB - Variational Lower Bound

Variational Lower Bound (VLB), znana również jako Evidence Lower Bound (ELBO) [17], jest dolną granicą logarytmu funkcji wiarygodności (log-likelihood) danych w modelach wariacyjnych. W kontekście wariacyjnych autoenkoderów z warunkowaniem, VLB służy jako funkcja celu do optymalizacji podczas treningu modelu. VLB składa się z dwóch składników: oczekiwanej log-wiarygodności rekonstrukji

danych oraz negatywnej dywergencji KL między dystrybucją wariacyjną a priorytetową. Optymalizacja VLB prowadzi do maksymalizacji prawdopodobieństwa danych przy jednoczesnym minimalizowaniu różnicy między dystrybucjami $q(\mathbf{z}|\mathbf{x})$ i $p(\mathbf{z})$. Dzięki temu model uczy się zarówno dokładnej rekonstrukcji danych, jak i spójnej reprezentacji zmiennych ukrytych.

W praktyce, wartości VLB są zazwyczaj negatywne, ponieważ reprezentują logarytmiczne prawdopodobieństwo, które jest mniejsze lub równe zero. Zwiększanie VLB (tj. jego mniej ujemne wartości) wskazuje na poprawę modelu. Typowe wartości VLB zależą od skali danych i specyfiki problemu, ale celem jest systematyczne zwiększanie VLB w trakcie procesu treningowego.

VLB jest wyrażona jako:

$$\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, q) = \log p(\mathbf{v}; \boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\mathbf{h} \mid \mathbf{v}) \| p(\mathbf{h} \mid \mathbf{v}; \boldsymbol{\theta})) \tag{3.5}$$

gdzie:

- $\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, q)$: Dolne ograniczenie dowodu (ELBO). Jest funkcją danych zaobserwowanych $\mathbf{v}$, parametrów modelu $\boldsymbol{\theta}$ i rozkładu q.
- $\log p(\mathbf{v}; \boldsymbol{\theta})$: Logarytm prawdopodobieństwa danych zaobserwowanych przy danych parametrach modelu $\boldsymbol{\theta}$. Jest to wartość, którą chcemy ostatecznie zmaksymalizować, ale może być trudna do obliczenia bezpośrednio.
- $D_{\mathrm{KL}}(q(\mathbf{h} \mid \mathbf{v}) \| p(\mathbf{h} \mid \mathbf{v}; \boldsymbol{\theta}))$: Rozbieżność Kullbacka-Leiblera (KL) między przybliżonym rozkładem $q(\mathbf{h} \mid \mathbf{v})$ a rzeczywistym rozkładem a posteriori $p(\mathbf{h} \mid \mathbf{v}; \boldsymbol{\theta})$. Ten składnik mierzy, jak bliski jest rozkład q do prawdziwego rozkładu a posteriori.
- $q(\mathbf{h} \mid \mathbf{v})$: Dowolny rozkład prawdopodobieństwa zmiennych ukrytych $\mathbf{h}$ przy danych zaobserwowanych $\mathbf{v}$. Jest on używany jako przybliżenie rzeczywistego rozkładu a posteriori.
- $p(\mathbf{h} \mid \mathbf{v}; \boldsymbol{\theta})$: Rzeczywisty rozkład a posteriori zmiennych ukrytych $\mathbf{h}$ przy danych zaobserwowanych $\mathbf{v}$ i parametrach modelu $\boldsymbol{\theta}$.

3.4.3 Technika ważonego próbkowania (IWAE - Importance Weighted Autoencoder)

Definicja: Importance Weighted Autoencoder (IWAE) [17] jest rozszerzeniem tradycyjnego wariacyjnego autoenkodera (VAE), które wykorzystuje technikę ważonego próbkowania do lepszego oszacowania dolnej granicy wariacyjnej (VLB) funkcji wiarygodności logarytmicznej danych. IWAE poprawia oszacowanie funkcji wiarygodności logarytmicznej poprzez zwiększenie liczby próbek K, co pozwala na lepsze aproksymowanie prawdziwej dystrybucji a posteriori $p(\mathbf{z}|\mathbf{x})$. Dzięki zastosowaniu wielu ważników, IWAE jest w stanie lepiej eksplorować przestrzeń zmiennych ukrytych, co prowadzi do dokładniejszych modeli generatywnych.

Wartość IWAE jest dolną granicą logarytmu funkcji wiarygodności danych, podobnie jak VLB. W praktyce, wartości IWAE są zazwyczaj ujemne, ponieważ logarytmiczna funkcja wiarygodności często przyjmuje wartości mniejsze lub równe zero. Analiza wartości IWAE obejmuje następujące scenariusze:

- **IWAE dodatnie:** Gdy IWAE przyjmuje wartości dodatnie, oznacza to, że model osiąga bardzo wysoką wiarygodność logarytmiczną danych. Taki przypadek może wskazywać na doskonałe dopasowanie modelu do danych treningowych, jednakże w praktyce jest to rzadkie i może sugerować przeuczenie modelu (overfitting).
- **IWAE ujemne:** Typowe wartości IWAE są ujemne, co odzwierciedla rzeczywiste prawdopodobieństwo danych w modelu generatywnym. Im wartość IWAE jest mniej ujemna, tym lepsze jest dopasowanie modelu do danych. Redukcja wartości ujemnych wskazuje na poprawę jakości rekonstrukcji i lepsze odwzorowanie struktury danych przez model.
- **IWAE dążące do zera:** Kiedy IWAE dąży do zera, oznacza to, że model generatywny osiąga maksymalne możliwe dopasowanie do danych, minimalizując różnicę między modelowaną dystrybucją a rzeczywistą dystrybucją danych. Wartość IWAE zbliżająca się do zera sugeruje, że model prawie idealnie odwzorowuje prawdziwą dystrybucję danych, co jest celem optymalizacji modeli generatywnych.

Dolna granica IWAE jest dana przez:

$$\mathcal{L}_k(x, q) = \mathbb{E}_{z^{(1)},\ldots,z^{(k)}\sim q(z|x)} \left[\log \frac{1}{k} \sum_{i=1}^{k} \frac{p_{\text{model}}(x, z^{(i)})}{q(z^{(i)}|x)} \right]. \tag{3.6}$$

gdzie:

1. $\mathcal{L}_k(x, q)$: Funkcja celu ważonego autoenkodera dla k próbek.
2. x: Dane wejściowe.
3. q: Rozkład wariacyjny używany do aproksymacji prawdziwego rozkładu a posteriori.
4. $\mathbb{E}_{z^{(1)},\ldots,z^{(k)}\sim q(z|x)}$: Wartość oczekiwana względem k próbek $z^{(1)}, \ldots, z^{(k)}$ pobranych z rozkładu $q(z|x)$.
5. $p_{\text{model}}(x, z^{(i)})$: Wspólne prawdopodobieństwo danych x i ukrytej zmiennej $z^{(i)}$ w modelu.
6. $q(z^{(i)}|x)$: Rozkład wariacyjny (lub aproksymowany rozkład a posteriori) oceniony dla próbki $z^{(i)}$ przy danym x.
7. $z^{(i)}$: i-ta próbka ukrytej zmiennej z.

3.5 Struktura danych użytych w metodyce

Głównym elementem struktury danych wykorzystywanym w przedstawionej metodyce jest wektor. W kontekście analizy algorytmów definiuje się go jako uporządkowany ciąg liczb sprowadzonych do zakresu od 0 do 1. Takie skalowanie wartości umożliwia porównywanie różnych sygnałów i sekcji algorytmu w jednolity sposób. Dla celów wizualizacji wektora stosuje się skalę kolorów, gdzie 0 reprezentuje kolor niebieski, 1 – czerwony, a wartości pośrednie – odcienie żółtego. Wektor ten powstaje w wyniku połączenia jego poszczególnych składowych, do których zaliczają się:

Sygnały wejściowe algorytmu

- Każdy sygnał, który wpływa na przebieg wykonania badanego algorytmu.
- Stosuje się sygnały dyskretne, odzwierciedlające wartości fizyczne lub logiczne.
- Na przykładowym rysunku 3.7 oznaczono je jako 'sygnał A' oraz 'sygnał B'.

Stan aktywacji poszczególnych sekcji ścieżki badanego algorytmu

- Każdemu przejściu pomiędzy węzłami decyzyjnymi algorytmu przypisuje się indeks w wektorze.

- Liczba wykonań danej sekcji zapisywana jest pod odpowiednią pozycją w wektorze.
- W przedstawionej ilustracji 3.7 zastosowano numeryczne oznaczenia w szarych polach w celu identyfikacji poszczególnych segmentów analizowanej ścieżki.

3.5.1 Wektory wejściowe i wyjściowe

Rozróżnia się dwa podstawowe rodzaje wektorów:

Wektor wejściowy

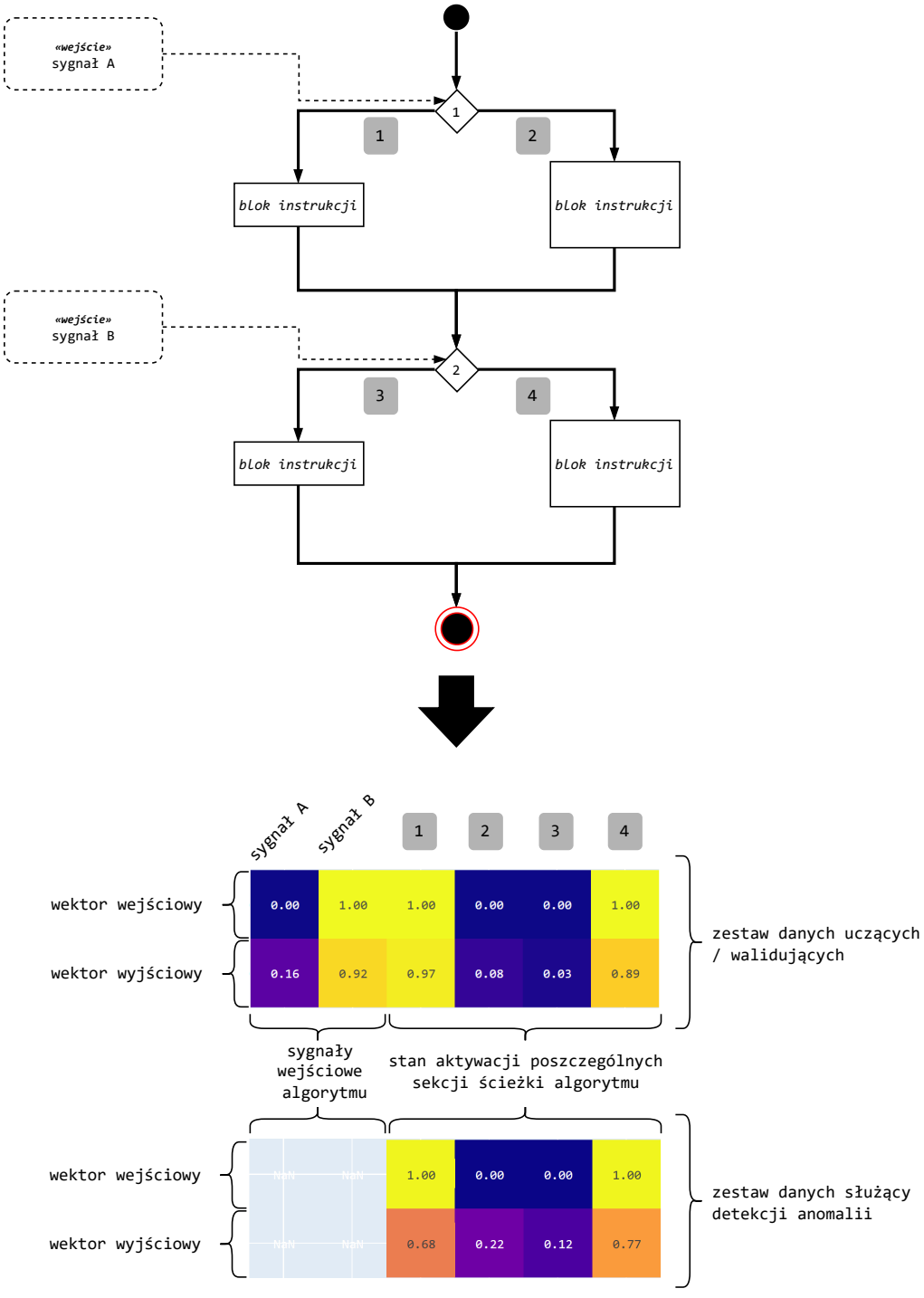
- Stanowi dane wejściowe dla algorytmu sztucznej inteligencji.
- Jego celem jest wymuszenie odpowiedzi, która umożliwia wykrycie potencjalnych anomalii, przy czym anomalią jest ścieżka o najdłuższym czasie wykonania, której osiągnięcie jest niemożliwe dla badanego algorytmu z danym zestawem sygnałów wejściowych.

Wektor wyjściowy

- Jest odpowiedzią algorytmu sztucznej inteligencji na wektor wejściowy i posiada on taką samą strukturę jak wektor wejściowy.
- Różnice wartości między symetrycznymi elementami wektora wejściowego i wyjściowego (na tych samych indeksach) interpretowane są jako informacja o prawdopodobieństwie wystąpienia anomalii.
- Wartości, dla których RMSE jest bliska zeru, wskazują na brak anomalii.

3.5.2 Wektory w etapach detekcji anomalii

- **Etap uczenia/walidacji:** W wektorze na tym etapie wszystkim polom przypisuje się wartości, z wyjątkiem sytuacji, gdy proces uczenia obejmuje także węzły decyzyjne. W takich przypadkach, pola niezwiązane z danym węzłem (sygnały wejściowe i fragmenty ścieżki) oznacza się wartością NaN (*Not a Number*), którą algorytm uczenia maszynowego ignoruje.
- **Etap detekcji anomalii:** W tym etapie, zarówno w wektorze wejściowym, jak i wyjściowym, zakrywa się wszystkie pola zawierające wartości sygnałów wejściowych badanego algorytmu.
- Normalizacja wartości do zakresu 0–1 umożliwia porównywanie i analizę różnych sygnałów i sekcji algorytmu w jednolity sposób.
- Stosowanie wartości NaN w procesie uczenia pozwala na elastyczne traktowanie różnych części wektora w zależności od kontekstu analizy.
- Wykorzystanie RMSE jako miary różnicy między wektorem wejściowym a wyjściowym jest kluczowe dla identyfikacji potencjalnych anomalii, co zwiększa skuteczność algorytmu w wykrywaniu nieprawidłowości.



Rysunek 3.7: Wizualizacja reprezentacji danych użytych przy weryfikacji metodyki

Detekcja anomali na podstawie oceny wartości RMSE

Niech $X = \{x_1, x_2, \ldots, x_n\}$ będzie wektorem wejściowym, a $Y = \{y_1, y_2, \ldots, y_n\}$ będzie wektorem wyjściowym, gdzie x_i i y_i to wartości na danym polu i-tym odpowiednio dla wektora wejściowego i wyjściowego, dla części zawierającej stany aktywacji poszczególnych sekcji ścieżki badanego algorytmu.

RMSE dla danego wektora:

$$\mathrm{RMSE}(X, Y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.7}$$

RMSE dla pojedynczego pola i-tego:

$$\mathrm{RMSE}_i = \sqrt{(x_i - y_i)^2} \tag{3.8}$$

Prawdopodobieństwo anomalii dla pola i-tego:

$$P(A_i) = f(\mathrm{RMSE}_i) \tag{3.9}$$

gdzie $f(\mathrm{RMSE}_i)$ jest funkcją zależną od RMSE_i. Dla liniowej zależności możemy zapisać:

$$P(A_i) = k \cdot \mathrm{RMSE}_i \tag{3.10}$$

gdzie k jest współczynnikiem skalującym.

Prawdopodobieństwo wystąpienia anomalii dla całego wektora:

$$P(A) = \frac{1}{n}\sum_{i=1}^{n} P(A_i) \tag{3.11}$$

co po podstawieniu daje:

$$P(A) = \frac{k}{n}\sum_{i=1}^{n} \mathrm{RMSE}_i \tag{3.12}$$

lub po uproszczeniu:

$$P(A) = \frac{k}{n}\sum_{i=1}^{n} \sqrt{(x_i - y_i)^2} \tag{3.13}$$

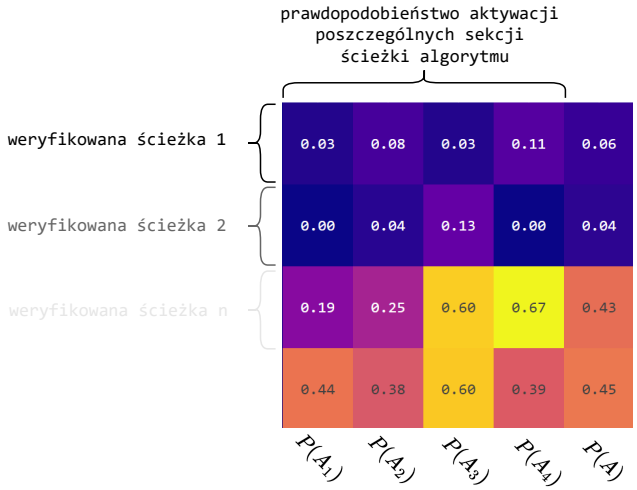
co w rezultacie jest równoważne:

$$P(A) = \frac{k}{n}\text{RMSE}(X, Y) \tag{3.14}$$

Gdzie:

- $P(A_i)$ oznacza prawdopodobieństwo wystąpienia anomalii w polu i-tym.
- $P(A)$ oznacza ogólne prawdopodobieństwo anomalii w całym wektorze.
- k to współczynnik skalujący, który można dostosować na podstawie specyfiki danego algorytmu i jego tolerancji na błędy.

Ten zapis pokazuje, że im większe są różnice między wartościami wektora wejściowego i wyjściowego, tym większe jest prawdopodobieństwo wystąpienia anomalii w analizowanym algorytmie. $P(A_i)$ oraz $P(A)$ zostały zwizualizowane na rysunku 3.8.



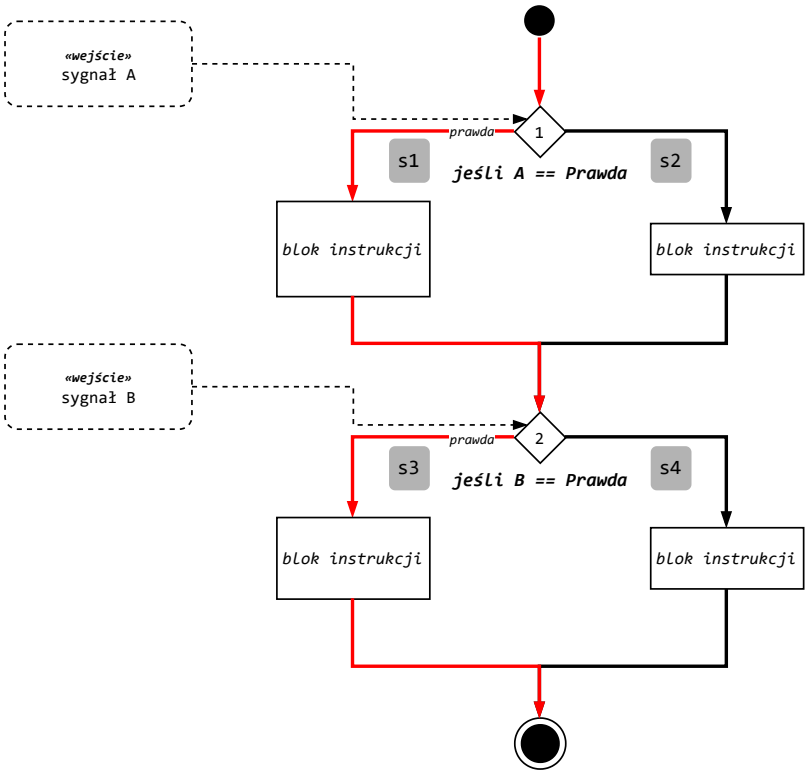
Rysunek 3.8: Wizualizacja prawdopodobieństwa anomalii w poszczególnych ścieżkach algorytmu

Rozdział 4

Wyniki badań

4.1 Zestaw Danych Syntetycznych — Logicznych

Do badania wykorzystano algorytm przedstawiony na rysunku 4.1, zaprojektowany w taki sposób, aby zminimalizować liczbę parametrów wejściowych oraz węzłów decyzyjnych. Celem tego było ocenienie efektywności autoenkodera w możliwie najprostszym scenariuszu, który jednocześnie znajduje zastosowanie w różnych obszarach funkcji bezpieczeństwa. Analizowany algorytm posiada dwa sygnały wejściowe: A i B, oba typu logicznego, mogące przyjmować wartości prawda lub fałsz. Założono, że oba parametry nigdy nie mogą jednocześnie przyjąć wartości prawda. Pełny zestaw danych uczących zawiera 100 rekordów, które zostały podzielone na 90% danych trenujących oraz 10% danych walidacyjnych. Taki podział wynika z niewielkiej liczby dostępnych danych, co jest konsekwencją faktu, że każda zmienna wejściowa może przyjmować maksymalnie dwa stany.



Rysunek 4.1: Wizualizacja algorytmu I

Algorytm dopuszcza cztery ścieżki działania, zależnie od wartości parametrów A i B. Ścieżki te są następujące:

- **Ścieżka s1**: Wykonywana jest, gdy A przyjmuje wartość prawda.
- **Ścieżka s2**: Wykonywana, gdy A przyjmuje wartość fałsz.
- **Ścieżka s3**: Wykonywana, gdy B przyjmuje wartość prawda.
- **Ścieżka s4**: Wykonywana, gdy B przyjmuje wartość fałsz.

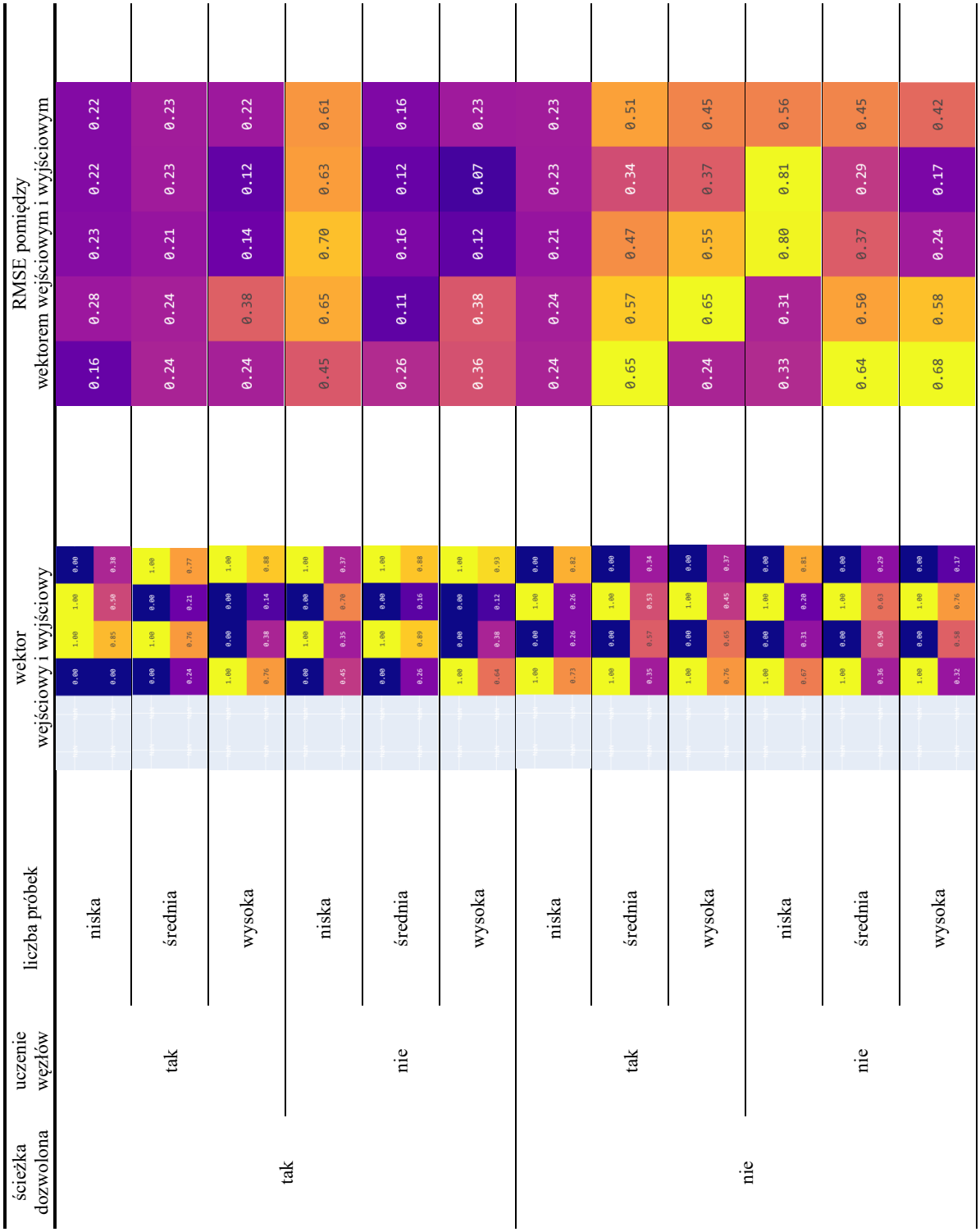
Ze względu na relację między parametrami, istnieje kombinacja, która jest niedozwolona: **Przypadek A = `prawda`, B = `prawda`**: W tej sytuacji algorytm miałby wykonać ścieżki **s1** i **s3**. Jednakże, zgodnie z warunkiem, że A i B nigdy nie mogą jednocześnie przyjmować wartości `prawda`, ta kombinacja jest niedozwolona. Jednocześnie ta ścieżka jest najdłuższa i niedozwolona w badanym algorytmie. Na rysunku 4.1 została ona oznaczona kolorem czerwonym.

Analiza przedstawionych wyników w tabeli 4.2 wskazuje na skuteczność wariacyjnego autoenkodera w detekcji anomalii poprzez monitorowanie wartości RMSE, choć z pewnymi zastrzeżeniami. Dla możliwej ścieżki (danych normalnych) RMSE zazwyczaj utrzymuje się na stabilnym, niskim poziomie, wynoszącym od 0,16 do 0,23, zależnie od liczby rekordów oraz trybu uczenia węzłów. W trybie bez uczenia węzłów RMSE dla możliwej ścieżki wynosi 0,22 dla niskiej, 0,23 dla średniej oraz 0,22 dla wysokiej liczby próbek. Jednak w trybie uczenia węzłów, wartość błędu średniokwadratowego dla możliwej ścieżki znacznie wzrasta do 0,61 przy niskiej liczbie próbek, co sugeruje trudności modelu w efektywnej rekonstrukcji danych przy ograniczonej ilości informacji. Przy średniej i wysokiej liczbie próbek RMSE dla możliwej ścieżki spada do 0,16 i 0,23, odpowiednio, wskazując na poprawę jakości rekonstrukcji wraz ze wzrostem liczby rekordów. W przypadku niemożliwej ścieżki (anomalii) wartości RMSE są znacznie wyższe w porównaniu do możliwej ścieżki. Bez uczenia węzłów RMSE wynosi 0,23 dla niskiej, 0,51 dla średniej oraz 0,45 dla wysokiej liczby próbek. Z kolei w trybie uczenia węzłów RMSE wynosi 0,56 przy niskiej liczbie próbek, 0,45 przy średniej oraz 0,42 przy wysokiej liczbie próbek. Te wyniki potwierdzają, że autoenkoder jest efektywny w wykrywaniu nieprawidłowości, zwłaszcza przy średniej i wysokiej liczbie próbek, gdzie RMSE dla anomalii jest znacznie wyższe niż dla danych normalnych. Dodatkowo, wprowadzenie trybu uczenia węzłów wpływa na wartość RMSE w zależności od liczby rekordów. Dla możliwej ścieżki, tryb uczenia węzłów prowadzi do znaczącego wzrostu RMSE przy niskiej liczbie próbek, co sugeruje, że model ma trudności z rekonstrukcją danych w tych warunkach. Jednak przy średniej i wysokiej liczbie próbek, RMSE dla danych normalnych spada, co wskazuje na poprawę jakości rekonstrukcji wraz ze wzrostem liczby rekordów. W przypadku niemożliwej ścieżki, tryb uczenia węzłów nie powoduje istotnych zmian w RMSE przy średniej i wysokiej liczbie próbek, co dalej potwierdza zdolność modelu do wykrywania anomalii.

ścieżka dozwolona	uczenie węzłów	liczba próbek	liczba epok	IWAE	VLB	całkowity czas uczenia [s]	średni czas epoki [s]
tak	tak	niska	91	0,05	0,05	66,74	0,33
		średnia	61	-0,48	-0,54	64,69	0,32
		wysoka	169	-0,73	-0,77	63,93	0,32
	nie	niska	13	0,07	0,07	65,85	0,33
		średnia	28	-0,54	-0,59	66,27	0,33
		wysoka	141	-0,62	-0,70	64,36	0,32
nie	tak	niska	104	-0,07	-0,08	51,85	0,50
		średnia	65	-0,69	-0,69	59,95	0,93
		wysoka	125	-0,74	-0,74	69,32	0,35
	nie	niska	14	-0,09	-0,40	85,65	6,04
		średnia	32	-0,61	-0,47	99,03	3,12
		wysoka	120	-0,57	-0,54	114,51	0,57

Tabela 4.1: Statystyki procesu uczenia modelu - algorytm I

Analiza statystyk z tabeli 4.1 dla procesu uczenia wariacyjnego autoenkodera z warunkowaniem wykazuje, że wybór wariantu ścieżki oraz liczby rekordów ma kluczowy wpływ na efektywność i jakość modelu. Wariant możliwej ścieżki osiąga wartości metryk IWAE (0,05) oraz VLB (0,05) przy niskiej liczbie rekordów, co wskazuje na bliskość do zera, zgodnie z celem metryk. Jednakże, zwiększenie liczby rekordów do średniej (IWAE = $-0{,}48$, VLB = $-0{,}54$) oraz wysokiej (IWAE = $-0{,}73$, VLB = $-0{,}77$) skutkuje większą odległością metryk od zera, sugerując trudności optymalizacyjne przy większej liczbie danych. W przypadku wariantu niemożliwej ścieżki, wszystkie konfiguracje prezentują wartości metryk IWAE i VLB oddalone od zera. Najlepsze wyniki w tej kategorii uzyskano przy niskiej liczbie rekordów (IWAE = $-0{,}07$, VLB = $-0{,}08$), które są bliższe zeru w porównaniu do innych konfiguracji. Wzrost liczby rekordów do średniej (IWAE = $-0{,}69$, VLB = $-0{,}69$) oraz wysokiej (IWAE = $-0{,}74$, VLB = $-0{,}74$) nie przynosi poprawy, a wartości metryk oddalają się od zera.

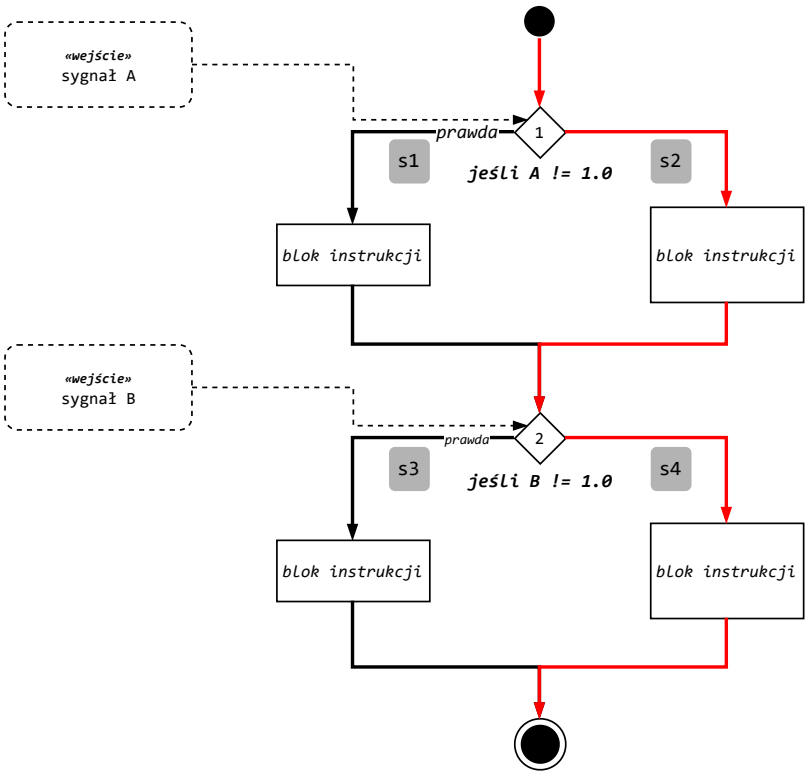


ścieżka dozwolona	uczenie węzłów	liczba próbek	wektor wejściowy i wyjściowy	RMSE pomiędzy wektorem wejściowym i wyjściowym			
tak	tak	niska	[0.00, 0.00] [1.00, 1.00] [0.00, 0.38] [0.85, 0.50]	0.16	0.28	0.23	0.22
		średnia	[0.00, 0.24] [1.00, 0.76] [0.00, 0.21] [1.00, 0.77]	0.24	0.24	0.21	0.23
		wysoka	[1.00, 0.76] [0.00, 0.38] [0.00, 0.14] [1.00, 0.88]	0.24	0.38	0.14	0.12
	nie	niska	[0.00, 0.45] [1.00, 0.35] [0.00, 0.70] [1.00, 0.37]	0.45	0.65	0.70	0.63
		średnia	[0.00, 0.26] [1.00, 0.89] [0.00, 0.16] [1.00, 0.88]	0.26	0.11	0.16	0.12
		wysoka	[1.00, 0.64] [0.00, 0.38] [0.00, 0.12] [1.00, 0.93]	0.36	0.38	0.12	0.07
nie	tak	niska	[1.00, 0.73] [0.00, 0.26] [1.00, 0.26] [0.00, 0.82]	0.24	0.24	0.21	0.23
		średnia	[1.00, 0.35] [0.00, 0.57] [1.00, 0.53] [0.00, 0.34]	0.65	0.57	0.47	0.34
		wysoka	[1.00, 0.76] [0.00, 0.65] [1.00, 0.45] [0.00, 0.37]	0.24	0.65	0.55	0.37
	nie	niska	[1.00, 0.67] [0.00, 0.31] [1.00, 0.20] [0.00, 0.81]	0.33	0.31	0.80	0.81
		średnia	[1.00, 0.36] [0.00, 0.50] [1.00, 0.63] [0.00, 0.29]	0.64	0.50	0.37	0.29
		wysoka	[1.00, 0.32] [0.00, 0.58] [1.00, 0.76] [0.00, 0.17]	0.68	0.58	0.24	0.17

Tabela 4.2: Tabela wyników - algorytmu I

4.2 Zestaw Danych Syntetycznych — Fizycznych

Do badania został użyty algorytm realizujący hipotetyczną funkcję bezpieczeństwa, który uwzględnia dane fizyczne. Algorytm, pokazany na rysunku 4.2, został opracowany tak, aby liczba parametrów wejściowych oraz węzłów decyzyjnych była minimalna. Podobnie jak w przypadku badanego algorytmu z rysunku 4.1, ma to na celu sprawdzenie, jak autoenkoder sprawdzi się w możliwie najprostszym przypadku. W podanym algorytmie są dwa sygnały wejściowe: A i B, oba typu fizycznego, które mogą przyjmować wartości w zakresie od -1 do 1. Parametr A pochodzi z funkcji sinus, natomiast parametr B z funkcji cosinus. Relacja między tymi parametrami zakłada, że gdy parametr A wynosi 1, B zawsze wynosi -1, i odwrotnie - innymi słowy sygnały te są w przeciw fazie. Pełen zestaw danych uczących zawiera 2000 rekordów. Dane zostały podzielone na 80% danych trenujących oraz 20% danych walidujących.



Rysunek 4.2: Wizualizacja algorytmu II

Algorytm analizuje cztery możliwe ścieżki działania, zależnie od wartości parametrów A i B. Ścieżki

te są następujące:

- **Ścieżka s1**: Wykonywana, gdy A przyjmuje wartość różną od 1.
- **Ścieżka s2**: Wykonywana, gdy A przyjmuje wartość 1.
- **Ścieżka s3**: Wykonywana, gdy B przyjmuje wartość różną od 1.
- **Ścieżka s4**: Wykonywana, gdy B przyjmuje wartość 1.

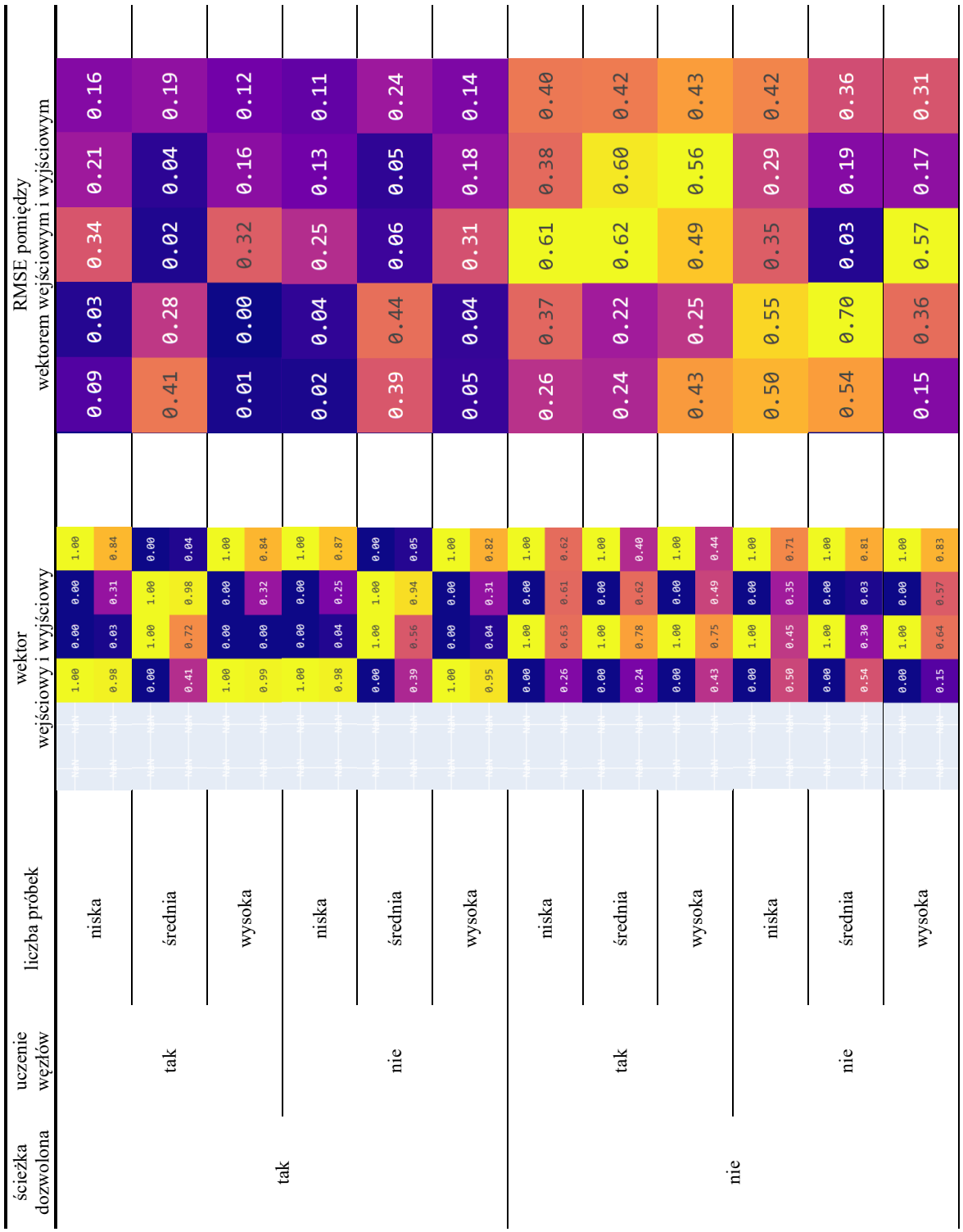
Ze względu na relację między parametrami, istnieje kombinacja, która jest niedozwolona: **Przypadek A = 1, B = 1**: W tej sytuacji algorytm miałby wykonać ścieżki s2 i s4. Jednakże, zgodnie z warunkiem, że A i B są w przeciw fazie, ta kombinacja jest niedozwolona. Jest to więc najdłuższa ścieżka dla badanego algorytmu. Na rysunku 4.2 została ona oznaczona kolorem czerwonym.

Analiza przedstawionych wyników w tabeli 4.4 jednoznacznie wykazała, że wariacyjny autoenkoder skutecznie odróżnia normalne scenariusze od anomalii poprzez znaczące różnice w wartościach RMSE. Dla wariantu ścieżki „Możliwa ścieżka" RMSE oscyluje w zakresie od 0,11 do 0,24, natomiast dla „Niemożliwej ścieżki" wartości te są znacznie wyższe, wynosząc od 0,31 do 0,43. Ponadto, zastosowanie trybu „Uczenia węzłów" wpływa na RMSE w przypadku możliwej ścieżki w zależności od liczby rekordów. Przykładowo, dla niskiej liczby rekordów RMSE spada z 0,16 do 0,11, co świadczy o poprawie dokładności rekonstrukcji danych normalnych. Jednakże, dla średniej liczby rekordów RMSE wzrasta z 0,19 do 0,24, a dla wysokiej liczby rekordów wzrasta z 0,12 do 0,14. Wzrost RMSE przy średniej i wysokiej liczbie rekordów wskazuje na konieczność dalszej optymalizacji modelu w tych zakresach. Należy zauważyć, że mimo wzrostu RMSE przy wyższej liczbie rekordów, wartości te pozostają nadal niższe niż dla scenariuszy „Niemożliwej ścieżki". Dodatkowo, niezależnie od trybu uczenia oraz liczby rekordów, wartości RMSE dla „Niemożliwej ścieżki" pozostają znacząco wyższe niż dla ścieżki możliwej, co potwierdza stabilność i niezawodność modelu w detekcji anomalii. Najwyższa wartość RMSE dla anomalii wynosi 0,43 przy podejściu bez uczenia węzłów i wysokiej liczbie rekordów, co nadal jest wyraźnie wyższe niż wartości dla scenariuszy nieweryfikujących anomalii.

ścieżka dozwolona	uczenie węzłów	liczba próbek	liczba epok	IWAE	VLB	całkowity czas uczenia [s]	średni czas epoki [s]
tak	tak	niska	173	-1,01	-1,00	208,92	1,04
		średnia	195	-1,09	-1,11	762,46	3,81
		wysoka	142	-1,08	-1,12	1489,09	7,45
	nie	niska	101	-0,96	-1,08	205,26	1,03
		średnia	171	-1,05	-1,12	761,68	3,81
		wysoka	195	-1,09	-1,11	1485,72	7,43
nie	tak	niska	190	-1,11	-1,10	229,00	1,21
		średnia	209	-1,17	-1,18	817,22	3,91
		wysoka	150	-1,14	-1,18	1571,83	10,49
	nie	niska	109	-1,04	-1,15	220,98	2,03
		średnia	185	-1,13	-1,20	825,25	4,45
		wysoka	217	-1,20	-1,23	1654,92	7,62

Tabela 4.3: Statystyki procesu uczenia modelu - algorytm II

Analiza statystyk z tabeli 4.3 przedstawionej powyżej dla procesu uczenia wariacyjnego autoenkodera z warunkowaniem wykazuje, że wybór wariantu ścieżki oraz liczby rekordów ma kluczowy wpływ na efektywność i jakość modelu. Wariant możliwej ścieżki generalnie osiąga lepsze wartości metryk IWAE oraz VLB w porównaniu do wariantu niemożliwych ścieżek. Dla niskiej liczby rekordów, wariant możliwej ścieżki bez uczenia węzłów osiąga (IWAE $= -1,01$, VLB $= -1,00$), podczas gdy wariant niemożliwych ścieżek osiąga (IWAE $= -1,11$, VLB $= -1,10$). Przy średniej liczbie rekordów, wariant możliwej ścieżki bez uczenia węzłów uzyskuje (IWAE $= -1,09$, VLB $= -1,11$), w porównaniu do (IWAE $= -1,17$, VLB $= -1,18$) w wariancie niemożliwych ścieżek. Wysoka liczba rekordów w wariancie możliwej ścieżki bez uczenia węzłów daje (IWAE $= -1,08$, VLB $= -1,12$), podczas gdy wariant niemożliwych ścieżek osiąga (IWAE $= -1,14$, VLB $= -1,18$).



ścieżka dozwolona	uczenie węzłów	liczba próbek	wektor wejściowy i wyjściowy	RMSE pomiędzy wektorem wejściowym i wyjściowym
tak	tak	niska	[1.00, 0.98] [0.00, 0.03] [0.00, 0.31] [1.00, 0.84]	0.09 0.03 0.34 0.21 0.16
		średnia	[0.00, 0.41] [1.00, 0.72] [1.00, 0.98] [0.00, 0.04]	0.41 0.28 0.02 0.04 0.19
		wysoka	[1.00, 0.99] [0.00, 0.00] [0.00, 0.32] [1.00, 0.84]	0.01 0.00 0.32 0.16 0.12
	nie	niska	[1.00, 0.98] [0.00, 0.04] [0.00, 0.25] [1.00, 0.87]	0.02 0.04 0.25 0.13 0.11
		średnia	[0.00, 0.39] [1.00, 0.56] [1.00, 0.94] [0.00, 0.05]	0.39 0.44 0.06 0.05 0.24
		wysoka	[1.00, 0.95] [0.00, 0.04] [0.00, 0.31] [1.00, 0.82]	0.05 0.04 0.31 0.18 0.14
nie	tak	niska	[0.00, 0.26] [1.00, 0.63] [1.00, 0.61] [0.00, 0.62]	0.26 0.37 0.61 0.38 0.40
		średnia	[0.00, 0.24] [1.00, 0.78] [1.00, 0.62] [0.00, 0.40]	0.24 0.22 0.62 0.60 0.42
		wysoka	[0.00, 0.43] [1.00, 0.75] [0.00, 0.49] [1.00, 0.44]	0.43 0.25 0.49 0.56 0.43
	nie	niska	[0.00, 0.50] [1.00, 0.45] [0.00, 0.35] [1.00, 0.71]	0.50 0.55 0.35 0.29 0.42
		średnia	[0.00, 0.54] [1.00, 0.30] [0.00, 0.03] [1.00, 0.81]	0.54 0.70 0.03 0.19 0.36
		wysoka	[0.00, 0.15] [1.00, 0.64] [0.00, 0.57] [1.00, 0.83]	0.15 0.36 0.57 0.17 0.31

Tabela 4.4: Wyniki z walidacji modelu - algorytm II

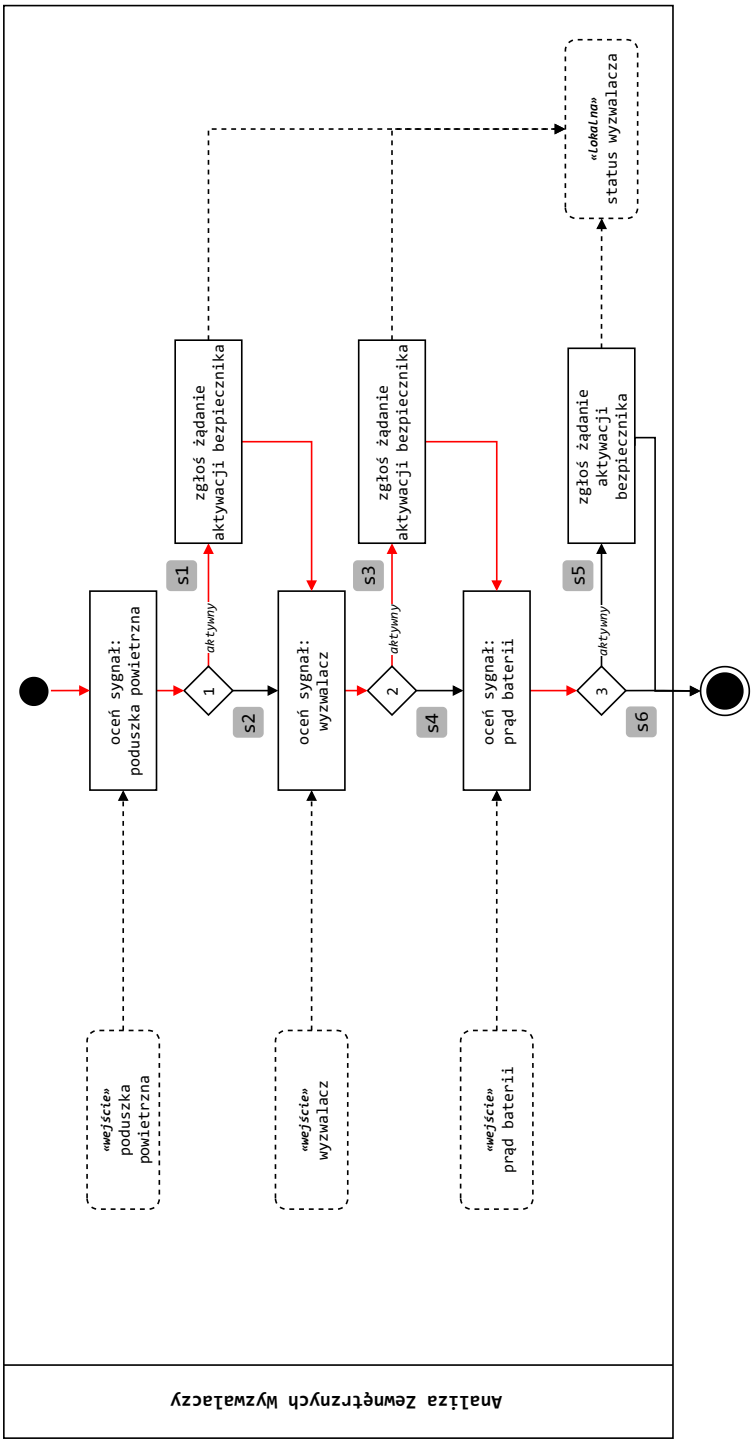
4.3 Zestaw Danych Rzeczywistych — Logicznych

Do badania został użyty algorytm, pokazany na rysunku 4.3, który został wyodrębniony z funkcji bezpieczeństwa o nazwie bezpiecznik pirotechniczny, przedstawionej na rysunku 2.7. W podanym algorytmie są trzy sygnały wejściowe: poduszka powietrzna, wyzwalacz oraz prąd akumulatora. Parametry poduszka powietrzna i wyzwalacz są typu logicznego i mogą przyjmować wartości 0 (fałsz) lub 1 (prawda). Dodatkowo istnieje parametr prąd akumulatora, który jest typu fizycznego i może przyjmować wartości od 0 do 1200. Nie ma on jednak wpływu na wystąpienie anomalii. Relacja między parametrami logicznymi zakłada, że poduszka powietrzna i wyzwalacz nie mogą być jednocześnie prawdziwe. Pełen zestaw danych uczących zawiera 2000 rekordów. Dane zostały podzielone na 80% danych trenujących oraz 20% danych walidujących. Dane pochodzą z pojazdu testowego, w którym zamontowany był czujnik multimodalny opisany w rozdziale 2.3.2.

Algorytm rozważa sześć ścieżek działania, zależnie od wartości parametrów `poduszka powietrzna`, `wyzwalacz` i `prąd akumulatora`. Ścieżki te są następujące:

- **Ścieżka s1**: Wykonywana, gdy poduszka powietrzna przyjmuje wartość 1.
- **Ścieżka s2**: Wykonywana, gdy poduszka powietrzna przyjmuje wartość 0.
- **Ścieżka s3**: Wykonywana, gdy wyzwalacz przyjmuje wartość 1.
- **Ścieżka s4**: Wykonywana, gdy wyzwalacz przyjmuje wartość 0.
- **Ścieżka s5**: Wykonywana, gdy wartość prądu akumulatora jest większa niż 1000.
- **Ścieżka s6**: Wykonywana, gdy wartość prądu akumulatora jest mniejsza lub równa 1000.

Ze względu na relację między parametrami, istnieje kombinacja, która jest niedozwolona:
Przypadek poduszka powietrzna = 1, wyzwalacz = 1: W tej sytuacji algorytm miałby wykonać ścieżki s1 i s3. Jednakże, zgodnie z warunkiem, że poduszka powietrzna i wyzwalacz nie mogą być jednocześnie prawdziwe, ta kombinacja jest niedozwolona. Jest to więc najdłuższa ścieżka dla badanego algorytmu. Na rysunku 4.3 została ona oznaczona kolorem czerwonym.



Rysunek 4.3: Wizualizacja algorytmu - algorytmu III

Analiza wyników wykazała, że wariacyjny autoenkoder z warunkowaniem efektywnie rozróżnia normalne scenariusze („Możliwa ścieżka") od anomalii („Niemożliwa ścieżka") na podstawie wartości RMSE. Dla możliwych ścieżek RMSE oscyluje w przedziale 0,07 – 0,19, podczas gdy dla niemożliwych ścieżek wartości te znacznie wzrastają do zakresu 0,37 – 0,45, co stanowi ponad dwukrotny wzrost. Ta znacząca różnica umożliwia precyzyjne wykrywanie anomalii. Ponadto, liczba rekordów wpływa na jakość rekonstrukcji w przypadku możliwych ścieżek, gdzie średnia liczba rekordów redukuje RMSE do 0,07 – 0,08, co wskazuje na optymalną wydajność modelu przy umiarkowanym zestawie danych treningowych. Wysoka liczba rekordów nieznacznie zwiększa RMSE do 0,13 – 0,15, co może sugerować, że większy zestaw danych wprowadza dodatkowe zmienności, które model musi uwzględnić, lub że model zaczyna mieć trudności z generalizacją przy bardzo dużej liczbie próbek. Natomiast niska liczba rekordów utrzymuje wyższe wartości RMSE na poziomie 0,18 – 0,19, co potwierdza potrzebę odpowiedniej ilości danych treningowych dla zachowania niskich błędów rekonstrukcji. Dodatkowo, tryb uczenia węzłów wykazuje minimalny wpływ na wartości RMSE zarówno dla możliwych, jak i niemożliwych ścieżek, co sugeruje stabilność modelu niezależnie od konfiguracji uczenia. W przypadku anomalii, tryb uczenia węzłów powoduje niewielkie zmniejszenie RMSE do 0,37, jednakże wartość ta nadal pozostaje znacznie wyższa niż dla możliwych ścieżek, co nie zakłóca efektywności detekcji anomalii. Stabilne utrzymanie wysokich wartości RMSE dla anomalii gwarantuje, że model nadal skutecznie identyfikuje odchylenia od normy.

ścieżka dozwolona	uczenie węzłów	liczba próbek	liczba epok	IWAE	VLB	całkowity czas uczenia [s]	średni czas epoki [s]
tak	tak	niska	173	-1,59	-1,62	275,17	1,38
		średnia	154	-1,64	-1,63	1033,07	5,17
		wysoka	181	-1,66	-1,67	2034,50	10,17
	nie	niska	116	-1,57	-1,61	277,54	1,39
		średnia	199	-1,64	-1,68	1032,72	5,16
		wysoka	129	-1,67	-1,69	2042,11	10,21
nie	tak	niska	198	-1,74	-1,76	314,19	1,59
		średnia	166	-1,72	-1,71	1111,99	6,71
		wysoka	191	-1,71	-1,72	2146,45	11,24
	nie	niska	123	-1,63	-1,67	294,44	2,39
		średnia	220	-1,74	-1,79	1142,67	5,19
		wysoka	136	-1,73	-1,75	2156,57	15,83

Tabela 4.5: Statystyki procesu uczenia modelu - algorytm III

Analiza statystyk z tabeli przedstawionej powyżej dla procesu uczenia wariacyjnego autoenkodera z warunkowaniem wykazuje, że wybór wariantu ścieżki oraz liczby rekordów ma kluczowy wpływ na efektywność i jakość modelu. Wariant możliwej ścieżki osiąga wartości metryk IWAE ($-1{,}59$) oraz VLB ($-1{,}62$) przy niskiej liczbie rekordów, co wskazuje na stosunkowo dobre dopasowanie modelu do danych.

Dla wariantu niemożliwej ścieżki, wartości metryk są bardziej negatywne, na przykład VLB wynosi $-1{,}76$ przy niskiej liczbie rekordów, co sugeruje znaczne trudności optymalizacyjne w tym scenariuszu. Ponadto, włączenie uczenia węzłów wpływa na wyniki metryk VLB, ale nie zawsze w sposób korzystny. Przykładowo, dla średniej liczby rekordów w wariancie możliwej ścieżki, VLB pogarsza się z $-1{,}63$ do $-1{,}68$ przy zastosowaniu uczenia węzłów, co wskazuje na oddalanie się metryki od wartości docelowej. Jednocześnie, zwiększenie liczby rekordów wpływa na wydłużenie czasu treningu oraz średniego czasu na epokę. Dla wariantu możliwej ścieżki bez uczenia węzłów, czas uczenia wzrasta z 275 sekund przy niskiej liczbie rekordów do 2034 sekund przy wysokiej liczbie rekordów. Analogicznie, średni czas na epokę dla tego wariantu rośnie z 1,38 sekund do 10,17 sekund. Mimo pewnych korzyści w dopasowaniu modelu, wzrost liczby rekordów generuje istotne obciążenie obliczeniowe. W świetle powyższych obserwacji, optymalizacja konfiguracji modelu wymaga starannego zbalansowania między jakością wyników a efektywnością czasową. Preferowanie wariantu możliwej ścieżki, zwłaszcza przy niższej liczbie rekordów, umożliwia osiągnięcie lepszych rezultatów w krótszym czasie treningu. Decyzja o włączeniu uczenia węzłów powinna być podejmowana w kontekście dostępnych zasobów obliczeniowych oraz specyfiki danego scenariusza, gdzie korzyści z poprawy jakości modelu mogą, ale nie zawsze muszą, przeważać nad zwiększonym kosztem czasowym.

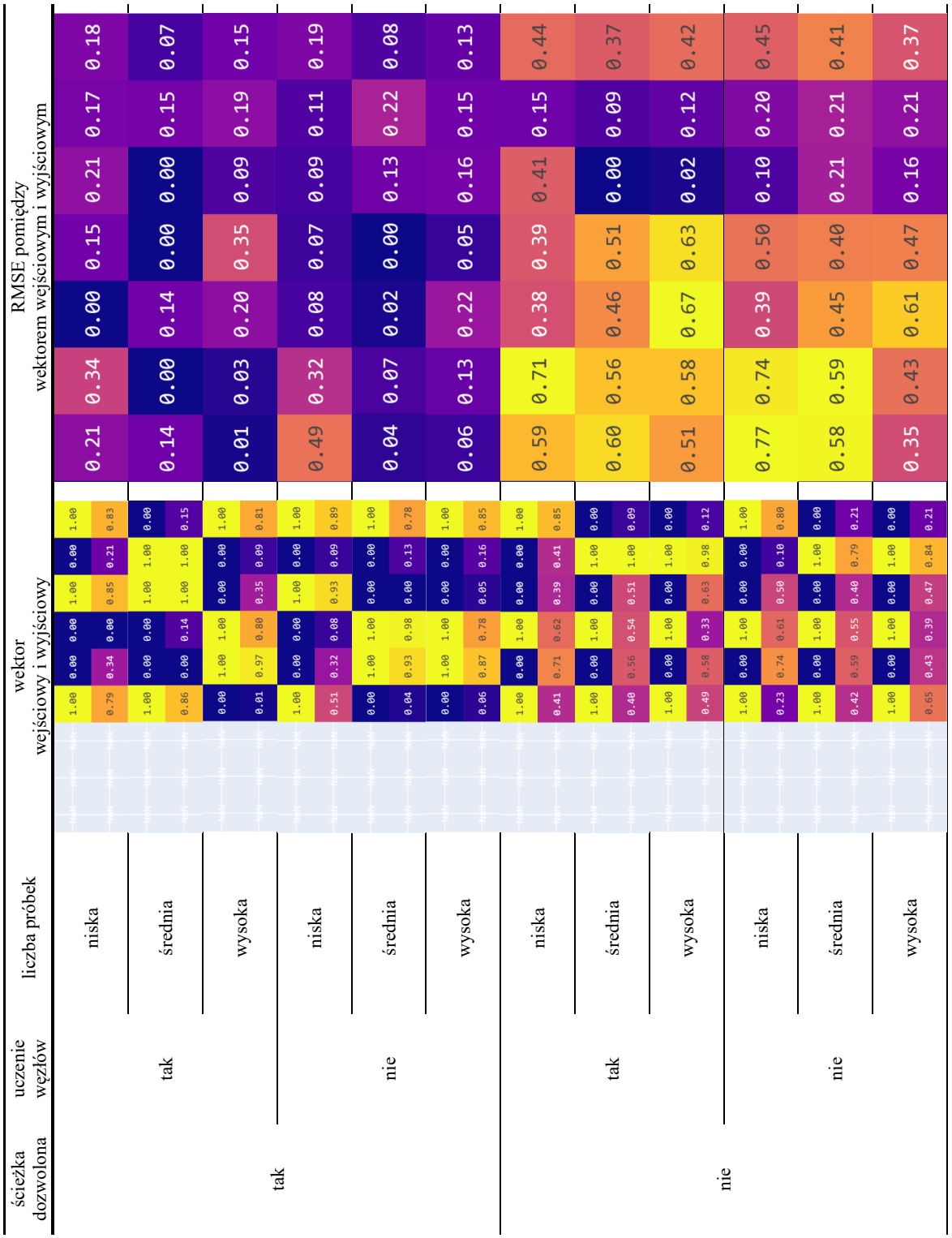


Tabela 4.6: Tabela wyników - algorytmu III

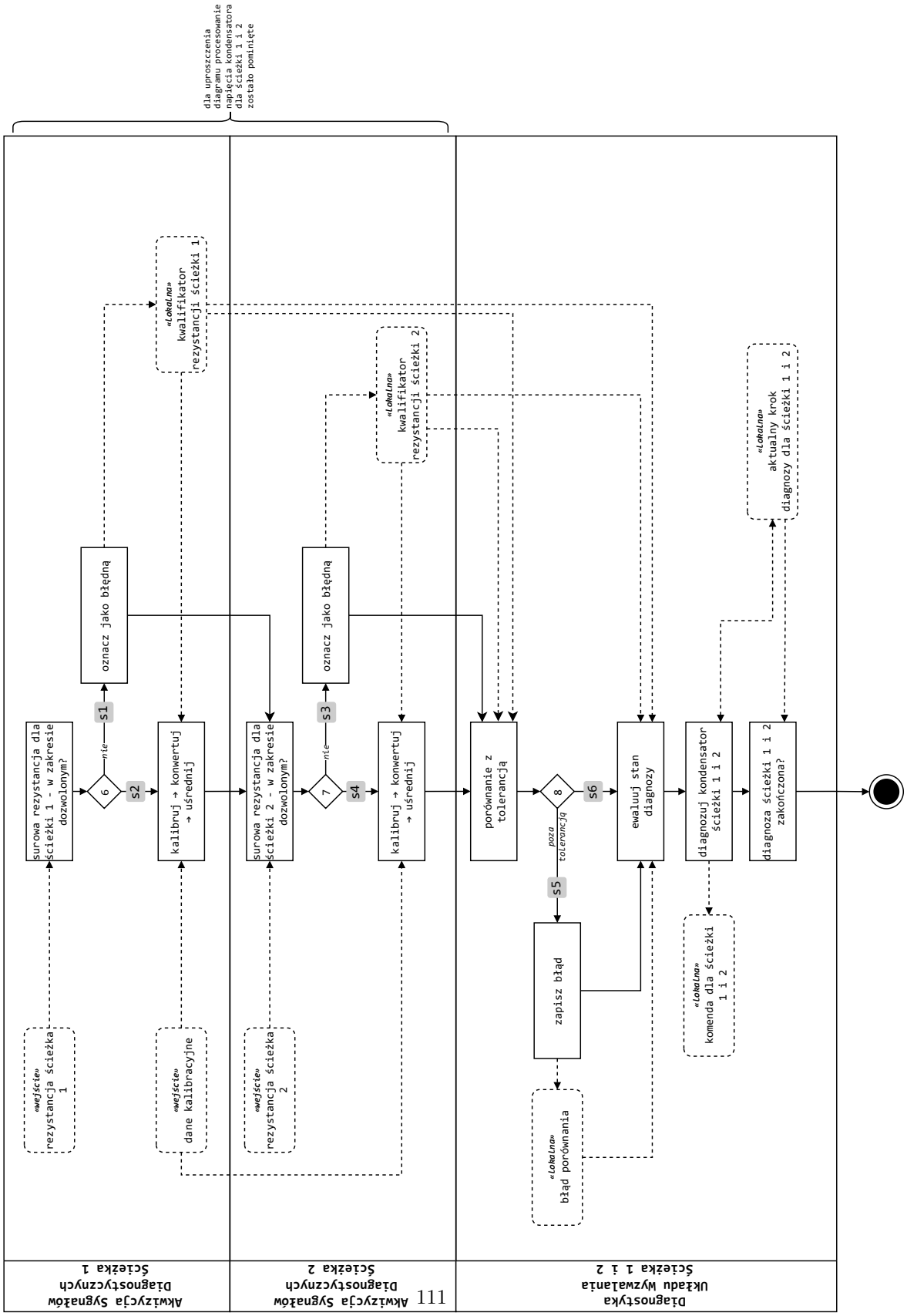
4.4 Zestaw Danych Rzeczywistych — Fizycznych

Do badania został użyty algorytm, pokazany na rysunku 4.3, który został wyodrębniony z funkcji bezpieczeństwa o nazwie bezpiecznik pirotechniczny, przedstawionej na rysunku 2.7. W podanym algorytmie są dwa sygnały wejściowe: rezystancja ścieżka 1 oraz rezystancja ścieżka 2, oba reprezentują wartości fizyczne, które mogą przyjmować wartości w zakresie od 0 do 2500. Relacja między tymi parametrami oznacza, że oba parametry muszą albo znajdować się w zakresie, albo poza nim, co oznacza, że nie ma sytuacji, w której jeden parametr jest w zakresie, a drugi nie. Pełen zestaw danych uczących zawiera 6000 rekordów. Dane zostały podzielone na 80% danych trenujących oraz 20% danych walidujących. Dane pochodzą z pojazdu testowego, w którym zamontowany był czujnik multimodalny opisany w rozdziale 2.3.2.

Algorytm rozważa sześć ścieżek działania, zależnie od wartości parametrów `rezystancja ścieżka 1` i `rezystancja ścieżka 2`. Ścieżki te są następujące:

- **Ścieżka s1**: Wykonywana, gdy rezystancja ścieżka 1 przyjmuje wartość spoza dopuszczalnego zakresu.
- **Ścieżka s2**: Wykonywana, gdy rezystancja ścieżka 1 przyjmuje wartość w dopuszczalnym zakresie.
- **Ścieżka s3**: Wykonywana, gdy rezystancja ścieżka 2 przyjmuje wartość spoza dopuszczalnego zakresu.
- **Ścieżka s4**: Wykonywana, gdy rezystancja ścieżka 2 przyjmuje wartość w dopuszczalnym zakresie.
- **Ścieżka s5**: Wykonywana, gdy różnica między rezystancja ścieżka 1 a rezystancja ścieżka 2 przekracza tolerancję.
- **Ścieżka s6**: Wykonywana, gdy różnica między rezystancja ścieżka 1 a rezystancja ścieżka 2 mieści się w tolerancji.

Ze względu na relację między parametrami, istnieje kombinacja, która jest niedozwolona: **Przypadek, gdy jeden parametr jest w zakresie, a drugi poza zakresem**: W tej sytuacji algorytm miałby wykonać ścieżki s2 i s3 lub s1 i s4. Jednakże, zgodnie z warunkiem, że oba parametry muszą jednocześnie znajdować się w zakresie lub poza zakresem, te kombinacje są niedozwolone. Są to najdłuższe i niedozwolone ścieżki w badanym algorytmie. Na rysunku 4.4 zostały one oznaczone kolorem czerwonym.



Rysunek 4.4: Wizualizacja algorytmu IV

Analiza wyników wykazała, że wariacyjny autoenkoder skutecznie wykrywa anomalie, co potwierdzają znacznie wyższe wartości RMSE w scenariuszu „Niemożliwa ścieżka" (od 0,34 do 0,67) w porównaniu do „Możliwej ścieżki" (od 0,03 do 0,52). Dodatkowo, zastosowanie trybu uczenia węzłów powoduje ogólny wzrost wartości RMSE zarówno w scenariuszach normalnych, jak i anomalnych. Przykładowo, RMSE dla „Niemożliwej ścieżki" wzrasta z 0,56 do 0,67 przy wysokiej liczbie rekordów, natomiast dla „Możliwej ścieżki" RMSE wzrasta z 0,03 do 0,33 przy niskiej liczbie próbek. Ponadto, liczba rekordów ma istotny wpływ na efektywność detekcji anomalii. Zarówno niska (RMSE = 0,03 vs. RMSE = 0,34) jak i wysoka liczba rekordów (RMSE = 0,13 vs. RMSE = 0,55) sprzyjają lepszemu rozróżnieniu między scenariuszami normalnymi a anomaliami. Jednakże, średnia liczba rekordów generuje najwyższe wartości RMSE zarówno dla „Możliwej" (0,27) jak i „Niemożliwej ścieżki" (0,56), co może wskazywać na większą niepewność modelu przy tej liczbie danych. Wzrost RMSE w scenariuszach normalnych przy zastosowaniu trybu uczenia węzłów sugeruje, że model może być bardziej podatny na fałszywe alarmy, co wymaga dalszej analizy w celu zbalansowania czułości modelu. Dodatkowo, choć RMSE jest istotnym wskaźnikiem efektywności detekcji anomalii, analiza mogłaby zostać wzbogacona o inne metryki, takie jak precyzja, czułość czy specyficzność, aby uzyskać pełniejszy obraz wydajności wariacyjnego autoenkodera. Ponadto, specyfika danych oraz kontekst ich zastosowania powinny być uwzględnione w interpretacji wyników, aby zapewnić lepszą generalizację modelu do różnych scenariuszy.

ścieżka dozwolona	uczenie węzłów	liczba próbek	liczba epok	IWAE	VLB	całkowity czas uczenia [s]	średni czas epoki [s]
tak	tak	niska	185	-1,37	-1,40	580,26	2,90
		średnia	165	-1,42	-1,43	7035,50	14,07
		wysoka	678	-1,43	-1,43	26849,12	26,85
	nie	niska	57	-1,40	-1,43	587,98	2,94
		średnia	180	-1,45	-1,46	2811,04	14,06
		wysoka	941	-1,43	-1,44	40886,42	40,89
nie	tak	niska	201	-1,12	-1,07	1411,87	7,02
		średnia	187	-1,23	-1,23	1632,55	8,72
		wysoka	145	-1,42	-1,43	1887,71	9,44
	nie	niska	64	-1,13	-1,10	1415,15	22,03
		średnia	199	-1,30	-1,28	1636,33	8,24
		wysoka	187	-1,45	-1,48	1892,08	9,46

Tabela 4.7: Statystyki procesu uczenia modelu - algorytm IV

Analiza statystyk z tabeli 4.1 dla procesu uczenia wariacyjnego autoenkodera z warunkowaniem wykazuje, że wybór wariantu ścieżki oraz liczby rekordów ma kluczowy wpływ na efektywność i jakość modelu. Wariant możliwej ścieżki osiąga wartości metryk IWAE ($-1{,}43$) oraz VLB ($-1{,}43$) przy

wysokiej liczbie rekordów, co wskazuje na umiarkowane dopasowanie modelu do danych. Przy średniej liczbie rekordów (IWAE = −1,42, VLB = −1,43) oraz niskiej liczbie rekordów (IWAE = −1,37, VLB = −1,40), wyniki metryk są bardziej korzystne, wskazując na lepsze dopasowanie modelu przy mniejszej liczbie próbek. W przypadku wariantu niemożliwej ścieżki, wszystkie konfiguracje prezentują mniej korzystne wartości IWAE i VLB w porównaniu z wariantem możliwej ścieżki, co wskazuje na trudności modelu w dopasowaniu do danych. Najlepsze wyniki w tej kategorii uzyskano przy niskiej liczbie rekordów (IWAE = −1,12, VLB = −1,07), co sugeruje lepsze dopasowanie modelu w tym przypadku. Wzrost liczby rekordów do średniej (IWAE = −1,23, VLB = −1,23) oraz wysokiej (IWAE = −1,42, VLB = −1,43) w przypadku niemożliwej ścieżki powoduje pogorszenie wyników, co sugeruje, że większa liczba próbek utrudnia modelowi wykrycie anomalii.

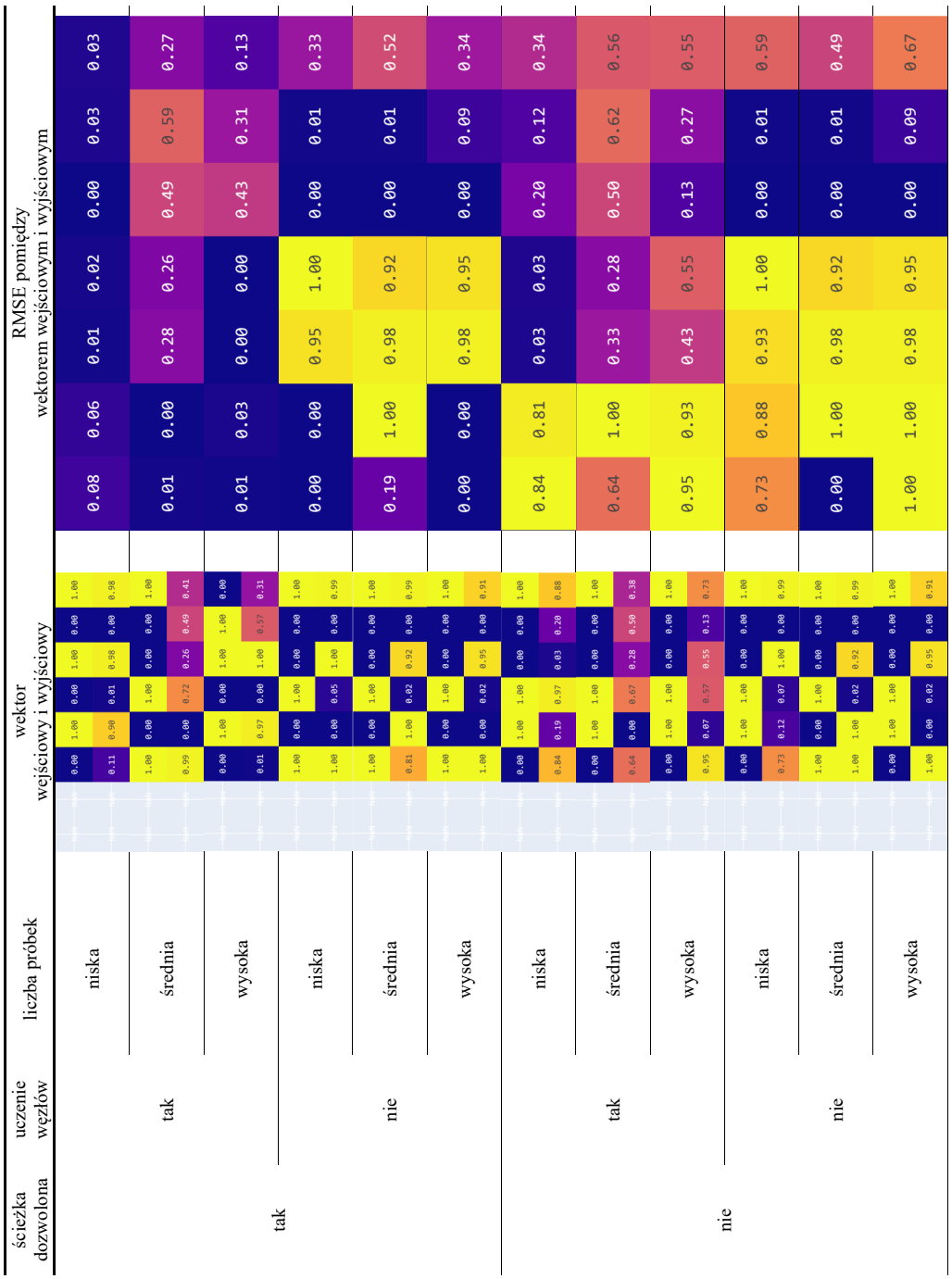


Tabela 4.8: Tabela wyników - algorytmu IV

Rozdział 5

Podsumowanie i Wnioski

W ramach projektu doktoratu wdrożeniowego opracowano nowatorską metodykę testowania sterowników systemów bateryjnych, która umożliwia wykrywanie anomalii w oprogramowaniu realizującym za funkcje bezpieczeństwa systemu bateryjnego pojazdu elektrycznego. Wyniki badań potwierdziły skuteczność tej metodyki, szczególnie w kontekście detekcji ścieżek uznawanych za możliwe na podstawie analizy statycznej, które w rzeczywistości nie mogą zostać zrealizowane z podanymi zestawami danych wejściowych. Opracowana metoda eliminuje zagrożenia wynikające z nieprawidłowości w działaniu oprogramowania sterownika, co ma kluczowe znaczenie dla zapewnienia bezpieczeństwa systemów bateryjnych.

Metodyka ta pozwala na usprawnienie procesu testowania sterowników na etapie badań walidacyjnych w u partnera przemysłowego. Zastosowanie autoenkodera wariacyjnego z warunkowaniem pozwala na automatyzację procesu wykrywania anomalii, co odciąża specjalistów i zwiększa precyzję oraz wydajność testów.

Innowacyjnym elementem opracowanego podejścia jest zastosowanie algorytmu autoenkodera wariacyjnego z warunkowaniem, który umożliwia automatyczne modelowanie zależności między sygnałami wejściowymi a czasami reakcji funkcji bezpieczeństwa. Algorytm ten, w przeciwieństwie do tradycyjnych metod opartych na analizie statycznej, pozwala na bardziej precyzyjną detekcję anomalii oraz zmniejsza konieczność ingerencji eksperta w proces weryfikacji. Automatyzacja ta zapewnia wyższą precyzję testowania systemów bateryjnych, co jest znaczącym wkładem w rozwój metod testowania tych systemów.

W ramach badań zweryfikowano działanie autoenkodera dla 4 przypadków kodu działających wg różnych algorytmów i przy różnych typach danych wejściowych. Dla każdego algorytmu wyodrębniono pary wartości błędu średniokwadratowego dla wektorów wynikowych, gdzie każda para składała się z dwóch wartości:

1. Średnia wartość RMSE dla wektorów oceniających ścieżkę jako możliwą
2. Średnia wartość RMSE dla wektorów oceniających ścieżkę jako niemożliwą

Warto zaznaczyć, że RMSE dla pojedynczego wektora oceniano na podstawie wcześniej ustalonych założeń, zamiast oznaczać je jako $P(A)$.

Potwierdzenie hipotez badawczych świadczy o skuteczności zaproponowanej metodyki, która nie tylko umożliwia wykrycie anomalii w algorytmach bezpieczeństwa, ale również zapewnia spełnienie założeń projektowych przez sterowniki systemów bateryjnych.

- **Teza 1:** Istnie możliwość opracowania metodyki testowania sterowników systemów bateryjnych bazującej na metodach sztucznej inteligencji pozwalających na opracowanie modelu zależności czasowych uzależnionych od wzajemnych związków między sygnałami wejściowymi.
- **Teza 2:** Istnieje metoda sztucznej inteligencji pozwalająca na detekcję potencjalnych anomalii w kodzie oprogramowania sterowników systemów bateryjnych pojazdów na podstawie wzajemnych zależności między sygnałami wejściowymi i potencjalnymi ścieżkami wykonania kodu.

Potwierdzenie tych hipotez świadczy o wysokiej skuteczności opracowanego podejścia i stanowi solidną podstawę do dalszych badań oraz rozwijania metodyki na inne typy funkcji bezpieczeństwa.

5.1 Podsumowanie wyników badań

Dane zebrano na podstawie analiz wyników czterech badanych algorytmów, z których każdy dostarczał sześć par wartości RMSE. Dla każdej z par wyznaczono średnie wartości RMSE, które następnie zostały pogrupowane według algorytmów. W celu określenia, czy ścieżka jest możliwa, czy niemożliwa, dla każdej grupy obliczono medianę. Mediana została uznana za wskaźnik decyzyjny, a jej wartość porównywano do progu wynoszącego 0,2.

Na podstawie obliczeń ustalono, że mediana RMSE dla poszczególnych algorytmów wahała się od 0,21 do 0,3. Próg decyzyjny wynoszący 0,2 został przyjęty jako wartość graniczna, powyżej której wynik

uznaje się za pewny i świadczący o większej możliwości zrealizowania ścieżki. Dla wartości zbliżonych do progu, np. 0,21 dla Algorytmu I, zachodzi konieczność przeprowadzenia dodatkowej analizy.

Wyniki badania pokazują, że algorytmy o medianach przekraczających próg 0,2, jak Algorytm I (mediana 0,21) czy Algorytm IV (mediana 0,3), wskazują na wysoką pewność w ocenie ścieżek jako możliwych. Algorytmy te dostarczają wartości RMSE, które jednoznacznie przekraczają próg decyzyjny, co świadczy o ich dużej skuteczności w ocenie ścieżek.

Z kolei dla algorytmów o medianach bliskich progowi, jak Algorytm II (mediana 0,235) oraz Algorytm III (mediana 0,265), pojawia się konieczność przeprowadzenia dodatkowej analizy wyników, szczególnie dla par, które zbliżają się do progu 0,2. Dla wyników takich jak 0,21 dla Algorytmu I lub 0,22 dla Algorytmu III, wskazana jest dodatkowa weryfikacja, aby wyeliminować ewentualne niepewności.

Algorytm	**Mediana RMSE**	**Przekroczenie progu (0,2)**	**Komentarz**
Algorytm I	0,21	Tak	Wyniki bliskie progowi, konieczność dodatkowej analizy dla par o wartościach zbliżonych do 0,2
Algorytm II	0,235	Tak	Mediana powyżej progu, jednak wyniki zbliżone do 0,2 wymagają dodatkowej analizy
Algorytm III	0,265	Tak	Mediana powyżej progu, algorytm pewny, lecz niektóre pary wymagają weryfikacji
Algorytm IV	0,3	Tak	Algorytm zdecydowanie przekracza próg, wyniki są pewne

Tabela 5.1: Zbiorcze wyniki mediany RMSE dla czterech algorytmów

Na podstawie przeprowadzonych obliczeń, stwierdzono, że wszystkie algorytmy przekraczają próg decyzyjny wynoszący 0,2, co świadczy o wysokiej pewności w ocenie możliwości badanych ścieżek. Jednakże, dla wyników zbliżonych do progu, konieczne jest przeprowadzenie dodatkowej weryfikacji, co zapewni większą pewność oceny w przypadkach granicznych.

5.2 Wkład pracy w rozwój dziedziny

Opracowana metodyka stanowi istotny wkład w rozwój metod testowania sterowników systemów bateryjnych. Zastosowanie algorytmów rozszerzonej inteligencji, takich jak autoenkoder wariacyjny, umożliwia automatyzację procesów testowych, co zwiększa precyzję i efektywność wykrywania anomalii. Dzięki tej automatyzacji możliwe jest zastąpienie subiektywnych ocen eksperckich bardziej obiektywną

analizą techniczną, co ma istotne znaczenie w kontekście rozwoju pojazdów elektrycznych.

5.3 Ograniczenia badań

Mimo że metodyka została przetestowana na różnych algorytmach i zestawach danych, konieczne są dalsze badania w celu zapewnienia jej pełnej uniwersalności. Przeprowadzone testy wykazały wysokie wymagania obliczeniowe – scenariusz wykorzystujący rzeczywiste dane wejściowe z czujnika multimodalnego pojazdu elektrycznego z użyciem pełnego zestawu danych wymagał 941 epok trenowania, a każda epoka trwała średnio 40 sekund na karcie graficznej RTX 4090. Te obciążenia obliczeniowe mogą stanowić barierę dla praktycznego wdrożenia metodyki w bardziej złożonych systemach.

Dalsze badania powinny skupić się na optymalizacji obliczeniowej oraz zastosowaniu algorytmów o zmniejszonej złożoności, aby uczynić metodykę bardziej dostępną i skalowalną.

5.4 Kierunki dalszych badań

Na podstawie uzyskanych wyników zaleca się następujące kierunki przyszłych badań:

- Opracowanie zasad dotyczących przygotowania zestawów danych uczących i walidujących w celu zmniejszenia nakładu pracy na proces uczenia i zwiększenia precyzji wyników.
- Zastosowanie technologii optymalizacji obliczeniowej, takich jak sieci neuronowe o zmniejszonej złożoności, w celu zwiększenia efektywności metodyki.
- Dalsza automatyzacja procesu generowania danych testowych oraz analiza wpływu liczby przykładów uczących na model wypracowny przez autoenkoder, co pozwoli na lepsze zrozumienie zależności pomiędzy liczbą danych a precyzją wykrywania anomalii.
- Badania nad wpływem obecności szumu w danych wejściowych na wyniki autoenkodera, w celu określenia odporności modelu na zakłócenia oraz możliwości poprawy jakości rekonstrukcji sygnału.
- Analiza wpływu częstotliwości próbkowania sygnałów pomiarowych na korelację pomiędzy różnymi zmiennymi, co pozwoli na dokładniejsze modelowanie i zwiększenie precyzji wykrywania anomalii.

Metodyka testowania sterowników systemów bateryjnych pojazdów wsparta modelem

Rozprawa doktorska - streszczenie

Autor: mgr inż. Kamil Sternal

Promotor: dr hab. inż. Marek Fidali, prof. PŚ

Opiekun z przemysłu: dr inż. Wojciech Sebzda

Politechnika Śląska, Wydział Mechaniczny Technologiczny

Dräxlmaier Group

Opracowana w ramach niniejszej pracy doktorskiej metodyka testowania sterowników systemów bateryjnych pojazdów elektrycznych wykorzystuje zaawansowane modele sztucznej inteligencji, w szczególności autoenkodery wariacyjne z warunkowaniem. Celem badania było stworzenie innowacyjnego podejścia, które umożliwi efektywne i w pełni zautomatyzowane wykrywanie anomalii w funkcjach bezpieczeństwa sterowników bateryjnych, z naciskiem na precyzyjną analizę czasową. W obliczu dynamicznego rozwoju elektromobilności oraz wzrastającej złożoności systemów wbudowanych, kluczowe jest zapewnienie, że funkcje bezpieczeństwa tych systemów działają zgodnie z rygorystycznymi wymaganiami czasowymi.

W pracy zastosowano autoenkoder wariacyjny z warunkowaniem do analizy zarówno danych syntetycznych, jak i rzeczywistych. Ta technika umożliwiła dokładne modelowanie czasów reakcji funkcji bezpieczeństwa oraz identyfikację odchyleń od normy wskazujących na potencjalne anomalie. Proces obejmował analizę danych w różnych scenariuszach testowych, co pozwoliło na ocenę skuteczności proponowanej metodyki w szerokim zakresie warunków operacyjnych. Zastosowanie sztucznej inteligencji w tym kontekście jest innowacyjne, gdyż pozwala na automatyzację procesu testowania oraz zwiększenie jego dokładności.

Wyniki badań wykazały, że opracowana metodyka skutecznie wykrywa anomalie w sterownikach systemów bateryjnych, przewyższając pod względem efektywności tradycyjne metody heurystyczne. Przeprowadzone testy na danych rzeczywistych potwierdziły, że metoda ta znacząco zmniejsza ryzyko przekroczenia dopuszczalnych czasów reakcji funkcji bezpieczeństwa, co jest kluczowe w projektowaniu bezpiecznych i niezawodnych systemów sterowania. Ponadto, metodyka ta pozwala na precyzyjne prognozowanie czasów reakcji, co dodatkowo zwiększa jej wartość praktyczną.

Model-Based Testing Methodology for Vehicle Battery System Controllers

Doctoral Dissertation - Abstract

Author: M.Sc. Eng. Kamil Sternal

Supervisor: Dr. hab. Eng. Marek Fidali, Prof. at SUT

Industrial Supervisor: Ph.D. Eng. Wojciech Sebzda

Silesian University of Technology, Faculty of Mechanical Engineering

Dräxlmaier Group

The testing methodology for electric vehicle battery system controllers developed in this doctoral dissertation utilizes advanced artificial intelligence models, particularly conditional variational autoencoders. The aim of the research was to create an innovative approach that enables efficient and fully automated anomaly detection in the safety functions of battery controllers, with an emphasis on precise timing analysis. In the face of the dynamic development of electromobility and the increasing complexity of embedded systems, it is crucial to ensure that the safety functions of these systems operate in accordance with strict timing requirements.

In the study, a conditional variational autoencoder was employed to analyze both synthetic and real data. This technique enabled accurate modeling of the response times of safety functions and the identification of deviations from the norm indicating potential anomalies. The process encompassed data analysis in various test scenarios, which allowed for assessing the effectiveness of the proposed methodology across a wide range of operational conditions. The application of artificial intelligence in this context is innovative, as it allows for the automation of the testing process and enhances its accuracy.

The research results demonstrated that the developed methodology effectively detects anomalies in battery system controllers, surpassing traditional heuristic methods in terms of efficiency. Tests conducted on real data confirmed that this method significantly reduces the risk of exceeding permissible response times of safety functions, which is crucial in designing safe and reliable control systems. Moreover, this methodology allows for precise forecasting of response times, further increasing its practical value.

Bibliografia

[1] IEC 61508-1:2010 - Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems - Part 1: General Requirements, April 2010.

[2] Ti designs: Tida-01445 automotive high-voltage interlock reference design. `https://www.ti.com/tool/TIDA-01445`, 2018. Accessed: 2024-05-16.

[3] ISO 6469-3:2021 electrically propelled road vehicles — safety specifications — part 3: Protection of persons against electric shock. International Organization for Standardization, 2021.

[4] Jaume Abella, Carles Hernández, Eduardo Quiñones, Francisco J Cazorla, Philippa Ryan Conmy, Mikel Azkarate-Askasua, Jon Perez, Enrico Mezzetti, and Tullio Vardanega. Wcet analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10. IEEE, 2015.

[5] Audi AG. Audi e-tron gt (type f8). Self-study Programme 684, Audi AG, Ingolstadt, Germany, 2021.

[6] Audi AG. *Audi Self-Study Program 686: TFSI e-Models*, 2021. Training manual for internal use in workshops and service centers.

[7] Infineon Technologies AG. *TRAVEO™ T2G CYT2B9 Series DataSheet (v11.00)*, 2024. Accessed: 2024-06-16.

[8] Iman Aghabali, J. Bauman, P. Kollmeyer, Yawei Wang, B. Bilgin, and A. Emadi. 800-v electric vehicle powertrains: Review and analysis of benefits, challenges, and future trends. *IEEE Transactions on Transportation Electrification*, 7:927–948, 2020.

[9] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Fixing a broken elbo. In *35th International Conference on Machine Learning (ICML)*, pages 159–168, 2018.

[10] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. In *Proceedings of the 33rd International Conference on Machine Learning Workshops*, pages 1–18, 2015.

[11] Krishan Arora, Suman Lata Tripathi, and Himanshu Sharma. *Electric Vehicle Design: Design, Simulation, and Applications*. Wiley-Scrivener, May 2024.

[12] Mihail Asavoae, Claire Maiza, and Pascal Raymond. Program semantics in model-based wcet analysis: A state of the art perspective. In Claire Maiza, editor, *13th International Workshop on Worst-Case Execution Time Analysis*, volume 30 of *Open Access Series in Informatics (OASIcs)*, pages 32–41, Dagstuhl, Germany, 2013. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[13] Clément Ballabriga, Julien Forget, and Giuseppe Lipari. Symbolic wcet computation. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(2):1–26, 2017.

[14] Arvind Kumar Bansal. *Introduction to programming languages*. CRC Press, 2014.

[15] Michael Barr. *Programming embedded systems in C and C++*. Ó'Reilly Media, Inc.", 1999.

[16] Michael Barr and Anthony Massa. *Programming Embedded Systems in C and C++*. O'Reilly Media, Inc., 2006.

[17] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[18] Johannes Buerger and James Anderson. Robust control for electric vehicle powertrains. *Control Theory and Technology*, 17:382 – 392, 2019.

[19] William Cai, Xiaogang Wu, Minghao Zhou, Yafei Liang, and Yujin Wang. Review and development of electric motor systems and electric powertrains for new energy vehicles. *Automotive Innovation*, 4:3–22, 2021.

[20] C.C. Chan. The state of the art of electric and hybrid vehicles. *Proceedings of the IEEE*, 90(2):247–275, 2002.

[21] C.C. Chan. The state of the art of electric, hybrid, and fuel cell vehicles. *Proceedings of the IEEE*, 95(4):704–718, 2007.

[22] Shuangshuang Chen and Wei Guo. Auto-encoders in deep learning—a review with new perspectives. *Mathematics*, 2023.

[23] Rene Y. Choi, Aaron S. Coyner, Jayashree Kalpathy-Cramer, M. Chiang, and J. Campbell. Introduction to machine learning, neural networks, and deep learning. *Translational Vision Science Technology*, 9, 2020.

[24] Marianne De Michiel, Armelle Bonenfant, Hugues Cassé, and Pascal Sainrat. Static loop bound analysis of c programs based on flow analysis and abstract interpretation. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 161–166. IEEE, 2008.

[25] Dr. Herbert Diess. Leading the transformation. page 13, March 2014.

[26] Yuanli Ding, Z. Cano, A. Yu, Jun Lu, and Zhongwei Chen. Automotive li-ion batteries: Current status and future perspectives. *Electrochemical Energy Reviews*, 2:1–28, 2019.

[27] Paulo SR Diniz. *Signal processing and machine learning theory*. Elsevier, 2023.

[28] Christian Ferdinand. Worst case execution time prediction by static program analysis. page 125, 2004.

[29] Alessandro Frigerio. Functional-safety analysis of asil decomposition for redundant automotive systems. 2022.

[30] Andreas Gerstlauer, Christian Haubelt, Andy Pimentel, and Jürgen Teich. *Design of Embedded Systems: Formal Models, Validation, and Synthesis*. Springer Science & Business Media, 2002.

[31] Peter Gliwa. *Embedded Software Timing*. Springer, 2021.

[32] Knüvener Mackert GmbH. *Automotive SPICE Guide: ASPICE 3.1 with Guidelines*. Knüvener Mackert GmbH, 4th edition, 2020.

[33] Robert Bosch GmbH. *Automotive Handbook*. Friedrichshafen and Karlsruhe, 11th edition, 2022.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[35] John Heitmann. *The Automobile and American Life*. McFarland, 2011.

[36] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a

constrained variational framework. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[37] Victor Albert Walter Hillier. *Hillier's fundamentals of automotive electronics*. Nelson Thornes, 1996.

[38] Klaus Hörmann, Markus Müller, Lars Dittmann, and Jörg Zimmer. *Automotive SPICE in Practice: Surviving Implementation and Assessment*. Rocky Nook, 2008.

[39] Lei Huang. *Normalization Techniques in Deep Learning*. Springer, 2022.

[40] Iqbal Husain. *Electric and Hybrid Vehicles: Design Fundamentals*. CRC Press, 2011.

[41] Thomas Huybrechts, Siegfried Mercelis, and Peter Hellinckx. A New Hybrid Approach on WCET Analysis for Real-Time Systems Using Machine Learning. In Florian Brandner, editor, *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*, volume 63 of *Open Access Series in Informatics (OASIcs)*, pages 5:1–5:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[42] Daniel Jiwoong Im, Sungjin Ahn, Roland Memisevic, and Yoshua Bengio. Denoising criterion for variational auto-encoding framework. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2059–2065, 2017.

[43] International Energy Agency. Global electric car stock, 2013-2023, 2023. Accessed: 2024-04-05.

[44] International Organization for Standardization. Road vehicles — functional safety (iso 26262). Technical Report ISO 26262, ISO, Geneva, Switzerland, 2018.

[45] Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. Variational autoencoder with arbitrary conditioning. *arXiv preprint arXiv:1806.02382*, 2018.

[46] Klaus Janschek. *Mechatronic systems design: methods, models, concepts*. Springer Science & Business Media, 2011.

[47] Ronald K. Jurgen, editor. *Automotive Microcontrollers, Volume 2*. SAE International, 2008.

[48] M Kathiresh, GR Kanagachidambaresan, and Sheldon S Williamson. *E-Mobility*. Springer, 2022.

[49] Brian W Kernighan and Dennis M Ritchie. The c programming language. 2002.

[50] Diederik P. Kingma, Danilo Jimenez Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 3581–3589, 2014.

[51] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations (ICLR)*, 2014.

[52] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Science & Business Media, 2011.

[53] Hermann Kopetz and Wilfried Steiner. Real-time communication. In *Real-time systems: Design principles for distributed embedded applications*, pages 177–200. Springer, 2022.

[54] Zdzislaw Kowalczuk, editor. *Intelligent and Safe Computer Systems in Control and Diagnostics*, volume 545 of *Lecture Notes in Networks and Systems*. Springer, Cham, 2023.

[55] Huimin Liang, Delong Liu, Xue Zhou, X. Liao, Dan Chen, Z. Cai, and Shuqing Chen. Arc behaviors of three kinds of bridge-type contacts when opening a resistive load in range of from 300vdc to 750vdc. *2018 IEEE Holm Conference on Electrical Contacts*, pages 81–86, 2018.

[56] Marcus Lindner, J. A. Rivera, Henrik Tjader, P. Lindgren, and Johan Eriksson. Hardware-in-the-loop based wcet analysis with klee. *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1:345–352, 2018.

[57] Yue Lu. *Pragmatic approaches for timing analysis of real-time embedded systems*. PhD thesis, Mälardalen University, 2012.

[58] Jingwei Ma, Jianfei Chen, and Han Xiao. Partial VAE for hybrid data imputation. *arXiv preprint arXiv:1806.02920*, 2018.

[59] Peter Marwedel. *Embedded System Design*. Springer Science & Business Media, 2011.

[60] Audi MediaCenter. Audi rs e-tron gt - liquid cooled lithium-ion battery a210363. https://www.audi-mediacenter.com/en/photos. (accessed: 15.07.2022).

[61] Peter Mertens and Michael Schlesinger. *Integrierte Informationsverarbeitung 1: Operative Systeme in der Industrie*. Gabler, Wiesbaden, 15 edition, 2008.

[62] Ibomoiye Domor Mienye, Yanxia Sun, and Zenghui Wang. Prediction performance of improved decision tree-based algorithms: a review. *Procedia Manufacturing*, 35:698–703, 2019. The 2nd

International Conference on Sustainable Materials Processing and Manufacturing, SMPM 2019, 8-10 March 2019, Sun City, South Africa.

[63] Joseph D Miller. *Automotive system safety: Critical considerations for engineering and effective management*. John Wiley & Sons, 2019.

[64] MISRA Consortium Limited. *MISRA C:2023 Guidelines for the use of the C language in critical systems*, third edition, second revision edition, April 2023. Incorporates Amendments 2–4 and Technical Corrigendum 2, supporting C11 and C18 language features.

[65] S. Nanga, A. T. Bawah, Ben Acquaye, Mac-Issaka Billa, Francisco Baeta, N. Odai, Samuel Kwaku Obeng, and Ampem Darko Nsiah. Review of dimension reduction methods. *Journal of Data Analysis and Information Processing*, 2021.

[66] Nicolas Navet and Françoise Simonot-Lion. *Automotive embedded systems handbook*. CRC press, 2017.

[67] Alberto Nazábal, Pablo Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using variational autoencoders. *Pattern Recognition*, 107:107501, 2020.

[68] Tammy Noergaard. *Embedded systems architecture: a comprehensive guide for engineers and programmers*. Newnes, 2012.

[69] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[70] Hans-Leo Ross and Ross. *Functional Safety for Road Vehicles*. Springer, 2016.

[71] Sesha Gopal Selvakumar. Electric and hybrid vehicles – a comprehensive overview. In *2021 IEEE 2nd International Conference On Electrical Power and Energy Systems (ICEPES)*. IEEE, December 2021.

[72] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[73] Andrew N. Sloss, Dominic Symes, and Chris Wright. *ARM System Developer's Guide: Designing and Optimizing System Software*. Elsevier, 2005.

[74] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 3483–3491, 2015.

[75] Miroslaw Staron. *Automotive software architectures*. Springer, 2021.

[76] Ryan R. Strauss and Junier B. Oliva. Any variational autoencoder can do arbitrary conditioning. *ArXiv*, abs/2201.12414, 2022.

[77] Wei-Tsun Sun, Eric Jenn, and Hugues Cassé. Validating Static WCET Analysis: A Method and Its Application. 72:6:1–6:10, 2019.

[78] Uranchimeg Tudevdagva and Dipl Inf Rene Schmidt. Autosar partitioning. page 16.

[79] Jonathan W Valvano. *Introduction to Embedded Systems*. CreateSpace Independent Publishing Platform, 2023.

[80] Volkswagen Group of America, LLC. *SSP 811213 - The High-Voltage System in the ID.4*. Volkswagen Group of America, United States, January 2021. Technical Service Bulletin.

[81] Ingomar Wenzel, Raimund Kirner, Bernhard Rieder, and Peter Puschner. Measurement-based timing analysis. In *Leveraging Applications of Formal Methods, Verification and Validation: Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings 3*, pages 430–444. Springer, 2008.

[82] Guoqi Xie, Yawen Zhang, Renfa Li, Kenli Li, and Keqin Li. *Functional Safety for Embedded Systems*. CRC Press, 2023.

[83] Haowen Xu, Wei Chen, Lili Zhao, Peng Li, Guanjie Chen, Jiqun Bu, and Deng Chen. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 27th International Conference on World Wide Web*, pages 187–196. International World Wide Web Conferences Steering Committee, 2018.

[84] Haoxiang Xu. Research on clustering algorithms in data mining. pages 652–655, 2022.

[85] Jisu Yu, Kilho Lee, Junho Cho, and Beomjin Choi. The study on emi characteristics under various operational conditions in dc-dc converter for electric vehicle. In *2024 IEEE 15th International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, pages 1–4, 2024.

[86] Xi Zhang, Liangpei Zhang, and Bo Du. Deep learning for remote sensing data: A technical tutorial on the state of the art. *IEEE Geoscience and Remote Sensing Magazine*, 7(2):22–40, 2019.