COLLEGIUM WITELONA
Uczelnia Państwowa

Wydział Nauk Technicznych i Ekonomicznych
Kierunek Informatka
Specjalność Programowanie Aplikacji Mobilnych i Internetowych

Sebastian Górski

Nethelt – wieloplatformowy system monitorowania urządzeń sieciowych z modułem wykrywania anomalii opartym na AI

Praca dyplomowa inżynierska
napisana pod kierunkiem
dr inż. Zbigniew Fryźlewicz

Legnica 2026 rok

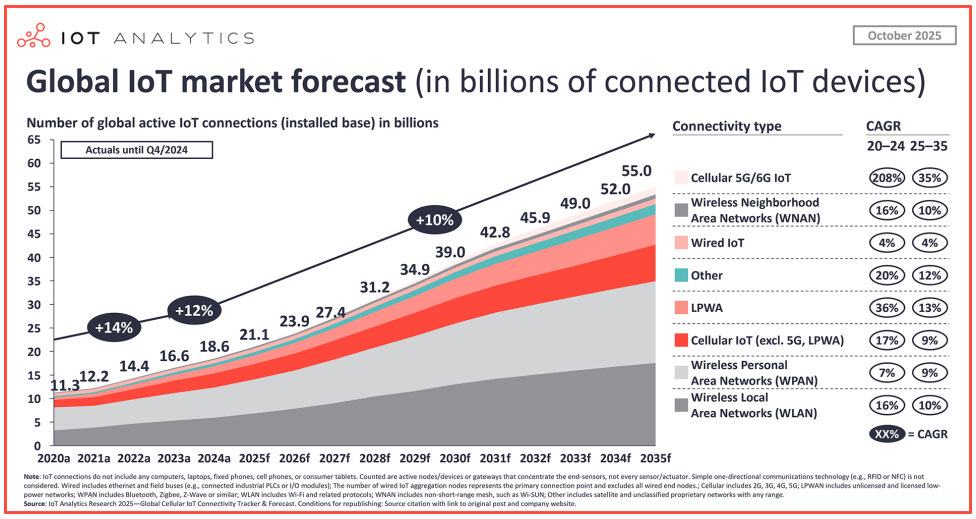
Spis treści

1. Wstęp

1.1 Wprowadzenie do problematyki

Aktualnie nie sposób wyobrazić sobie firmy, przedsiębiorstwa, a nawet gospodarstwa domowego, w którym nie korzysta się z internetu. Dostęp do zewnętrznych serwisów i usług stał się naszą codziennością. Korzystamy z internetu oraz urządzeń, łączących się do niego każdego dnia. W domach internet służy nam często do rozrywki i ułatwiania codziennych czynności, do wyszukiwania informacji i rozwiązywania problemów. Dzięki niemu jesteśmy w stałym kontakcie z innymi, mamy dostęp do płatności elektronicznych, a zakupy możemy zrobić nie wychodząc z domu. W pracy korzystamy z niego w celu wyszukiwania informacji i przesyłania ich dalej, wysyłania maili, pobierania danych z Państwowych spółek, czy chociażby robienia wideokonferencji z klientami z całego świata.

Poza dostępem do zewnętrznych usług coraz większą popularność zyskują urządzenia Smart oraz IoT (Internet of Things). Wielu z nas nawet nie ma pojęcia na temat tego jak wiele takich urządzeń znajduje się w naszym otoczeniu. Według IoT Analytics pod koniec 2025 r. liczba podłączonych urządzeń IoT na świecie miała osiągnąć 21,1 miliarda To wzrost o ok. 14% w stosunku do roku poprzedniego TODO: zweryfikować dane przed oddaniem pracy. Według prognozy liczba ta będzie tylko rosła i w 2030 roku może sięgnąć 39 miliardów[1]. Tendencję tę można zaobserwować na rysunku 1.1 przedstawiający liczbę aktywnych urządzeń typu IoT wraz z prognozą na przyszłe lata.



Rysunek 1.1: Liczba urządzeń typu IoT w latach 2020–2035
Źródło: https://iot-analytics.com/number-connected-iot-devices/

Pojawienie się tak dużej ilości urządzeń, z których specyfikacji wynika, że wręcz wymagają stałego połączenia z siecią, tworzy nowe problemy i pytania. Kluczowym pytaniem, które powinniśmy sobie zadać, jest to, czy przy utracie łączności z siecią takiego urządzenia pojawią się poważne komplikacje. Chociaż w przypadku np. lodówki bądź ekspresu do kawy działającym w sieci, brak połączenia nie wyrządzi sporych szkód, tak jednak w przypadku maszyn na hali produkcyjnej lub sygnalizacji świetlnej w centrum miasta, szkody mogą być spore i mogą zagrażać nie tylko sytuacji majątkowej, ale też bezpieczeństwu ludzi. W związku z powyższym należy monitorować stan takich urządzeń, a w przypadku ich awarii jak najszybciej dążyć do naprawy usterki.

1.2 Cel pracy

Celem niniejszej pracy jest stworzenie systemu wielomodułowego do monitorowania i zarządzania urządzeniami sieciowymi w środowisku lokalnym (LAN). System ma umożliwiać rejestrację i logowanie użytkowników, konfigurację połączeń sieciowych oraz monitorowanie stanu urządzeń.

Dodatkowym celem jest opracowanie modułu detekcji anomalii z wykorzystaniem algorytmu *isolation forest*, który pozwoli na wczesne wykrywanie nieprawidłowości w funkcjonowaniu sieci. System powinien wspierać dwa typy użytkowników – administratora oraz użytkownika zwykłego – z odpowiednimi uprawnieniami do zarządzania systemem i dostępem do danych.

Projekt obejmuje także wdrożenie aplikacji webowej w środowisku chmurowym, zapewniając dostępność dla użytkowników końcowych niezależnie od lokalizacji. Aplikacja desktopowa będzie dostępna do pobrania bezpośrednio na stronie internetowej.

1.3 Zakres pracy

Zakres pracy obejmuje:

- implementację aplikacji webowej w technologii Java 25 z użyciem frameworka Spring Boot oraz interfejsu w TypeScript i Angular 21,
- stworzenie desktopowego klienta w Javie 25, działającego w trybie serwisu, umożliwiającego komunikację z serwerem,
- opracowanie modułu AI w Pythonie, wykorzystującego algorytm *isolation forest* do detekcji typowych anomalii w sieci,
- integrację modułów z centralną bazą danych PostgreSQL 18.2, współdzieloną między aplikacjami,
- wdrożenie aplikacji w środowisku chmurowym (Google Cloud Console) w celu zapewnienia dostępności dla użytkowników.

W zakresie pracy nie przewiduje się:

- stworzenia mobilnej aplikacji,
- obsługi wszystkich możliwych typów anomalii w sieci – moduł AI ograniczony jest do najczęściej występujących przypadków,
- rozbudowanego GUI w aplikacji desktopowej - wersja komputerowa będzie dostępna w trybie konsolowym. Wersja z interfejsem użytkownika zostanie zaimplementowana, jeśli pozostałe moduły będą gotowe przed ostatecznym terminem.

Zakres dokumentacji obejmuje przygotowanie instrukcji użytkownika, dokumentacji technicznej, diagramów UML oraz podstawowych statystyk działania systemu.

1.4 Założenia projektowe

Projekt ma charakter wielomodułowy i obejmuje trzy główne komponenty:

- aplikację webową,
- aplikację desktopową,
- moduł detekcji anomalii oparty na algorytmie AI typu *isolation forest*.

System zostanie zaprojektowany w taki sposób, aby umożliwić użytkownikowi rejestrację, logowanie, konfigurację połączenia ze swoją siecią LAN, dodawanie lokalnych urządzeń sieciowych, monitorowanie ich stanu oraz zmianę ustawień powiadomień. Aplikacja desktopowa będzie pełniła rolę klienta komunikującego się z serwerem, co jest niezbędne do prawidłowego funkcjonowania systemu w środowisku wielomodułowym. Dodatkowym wyzwaniem projektu będzie integracja wytrenowanego modelu AI, który będzie wspierał wczesne wykrywanie anomalii w sieci.

System przewiduje co najmniej dwa typy użytkowników: administratora oraz użytkownika zwykłego. Zalogowanie do systemu będzie obowiązkowe, a proces logowania zostanie udostępniony zarówno poprzez protokół OAuth2 (Google), jak i w formie lokalnej autoryzacji. Administrator będzie posiadał rozszerzone uprawnienia, w tym możliwość zarządzania użytkownikami oraz przeglądania podstawowych statystyk dotyczących korzystania z systemu.

Aplikacja webowa będzie realizowana w oparciu o nowoczesną wersję języka Java (wersja 25) oraz framework Spring Boot (w wersji 4.x.x), który ułatwia implementację i utrzymanie systemu. Interfejs użytkownika zostanie zbudowany przy użyciu TypeScript oraz frameworka Angular (wersja 21). Moduł AI zostanie opracowany w języku Python z wykorzystaniem odpowiednich bibliotek do uczenia maszynowego, co przyspieszy proces modelowania i analizy danych. Aplikacja desktopowa zostanie stworzona w tej samej wersji Javy, z której korzysta program internetowy, co pozwoli na jej uruchamianie na różnych systemach operacyjnych.

Dane pomiędzy modułami systemu będą współdzielone poprzez centralną bazę danych, której silnik wykorzysta PostgreSQL w wersji 18.2.

Projekt przewiduje wdrożenie aplikacji w środowisku chmurowym w celu zapewnienia dostępności dla użytkowników końcowych z dowolnej lokalizacji. Wstępnie planowane jest wykorzystanie platformy Google Cloud Console do tego celu.

2. Warstwa API

2.1 Struktura bazy danych

Na rysunku 2.1 przedstawiono schemat relacyjnej bazy danych aplikacji NetHelt. Diagram obrazuje strukturę tabel oraz relacje pomiędzy encjami systemu.

Centralnym elementem modelu jest tabela `users`, przechowująca dane użytkowników systemu wraz z przypisaną rolą (`roles`). Każdy użytkownik może posiadać wiele urządzeń sieciowych zapisanych w tabeli `devices`.

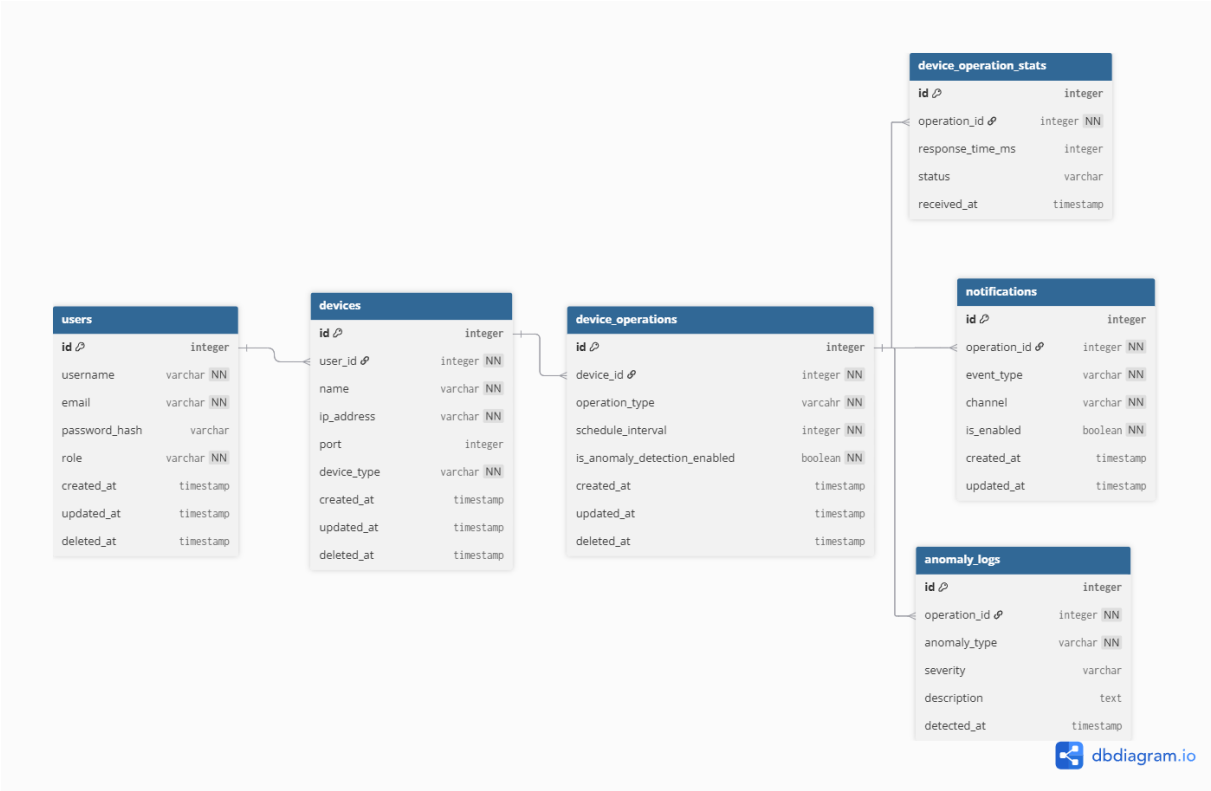
Z każdym urządzeniem powiązane są operacje sieciowe (`device_operations`), określające typ wykonywanej operacji oraz czas między kolejnymi wywołaniami operacji. Dodatkowo w tej tabeli można przełączać wykrywanie anomalii.

Wyniki wykonywanych operacji zapisywane są w `device_operation_stats`, gdzie gromadzone są informacje dotyczące odpowiedzi z urządzenia. Na podstawie zgromadzonych statystyk mogą być tworzone okna czasowe, wysyłane do analizy do modułu AI. W przypadku wykrycia anomalii, wpis taki generowany jest w tabeli `anomaly_logs`.

Konfiguracja powiadomień dotyczących zdarzeń monitorujących przechowywana jest w tabeli `notifications`. Powiadomienia są powiązane z operacjami monitorującymi i definiują kanał komunikacji, typ zdarzenia wywołującego alert oraz status danych powiadomień.

Niektóre tabele posiadają kolumnę `deleted_at`, która pozwala na wykonywanie miękkich usunięć z systemu, co ułatwi utrzymanie integralności danych.

Kolumny takie jak: `role`, `device_type`, `channel` itd. są ograniczone do predefiniowanych wartości, które w kodzie źródłowym programu występują jako typ enumeryczny. Pozwala to uniknąć błędnego wprowadzenia danych poprzez wyświetlenie odpowiedniego komunikatu błędu.



Rysunek 2.1: Diagram bazy danych aplikacji NetHelt
Źródło: Opracowanie własne w witrynie internetowej https://dbdiagram.io/

3. Model SI do wykrywania anomalii

3.1 Założenia

Jednym z kluczowych elementów systemu jest model sztucznej inteligencji przeznaczony do wykrywania anomalii w parametrach jakości połączenia sieciowego na wczesnym etapie ich występowania.

Na obecnym etapie implementacji model umożliwia analizę wyłącznie parametrów związanych z pomiarami czasu odpowiedzi (ping). Dane wejściowe mają postać zagregowanych statystyk obliczanych w stałych oknach czasowych, co pozwala na ograniczenie wpływu pojedynczych, losowych odchyleń oraz umożliwia podejmowanie decyzji w krótkim czasie po zakończeniu danego interwału.

Model wykorzystuje algorytm `Isolation Forest`, do którego przekazywane są przetworzone dane wejściowe. Przyjęto podejście uczenia nienadzorowanego, zakładające brak oznaczonych przykładów anomalii w zbiorze treningowym. Algorytm uczy się rozkładu obserwacji uznawanych za typowe, a istotne odchylenia od tego rozkładu klasyfikuje jako anomalie.

Wybór algorytmu `Isolation Forest` został podyktowany kilkoma czynnikami:

- efektywnością w wykrywaniu anomalii w danych nienadzorowanych, gdzie brak jest oznaczonych przykładów problematycznych obserwacji,
- zdolnością do radzenia sobie z wysoką zmiennością i niestabilnością w sygnale sieciowym, typową dla pomiarów czasu odpowiedzi (ping),
- skalowalnością dla dużych zbiorów danych oraz stosunkowo niskimi wymaganiami obliczeniowymi w porównaniu z innymi algorytmami wykrywania anomalii,
- prostotą interpretacji wyników – obserwacje zaklasyfikowane jako anomalie są łatwe do identyfikacji w analizie eksperckiej.

Dzięki tym właściwościom `Isolation Forest` jest szczególnie dobrze dopasowany do monitorowania jakości połączenia sieciowego w czasie rzeczywistym.

Ze względu na różnice w specyfikacji i przeznaczeniu w różnego typu urządzeniach sieciowych, model będzie musiał powstać w co najmniej 3 wariantach. Na początkowym etapie projektu zakłada się wytrenowanie modelu na 3 zestawach danych: zebranych z urządzeń podłączonych do sieci poprzez protokół WiFi, z urządzeń typu IoT oraz z urządzeń typu klient LAN (wliczając w to: komputery stacjonarne, serwery, oraz infrastrukturę sieciową - router, switch, AP).

Założono możliwość dalszego rozszerzenia modelu o dodatkowe parametry jakości połączenia oraz ponowne trenowanie modelu wraz ze zmianą charakterystyki środowiska sieciowego.

3.2 Użyte narzędzia

Model został zaimplementowany w języku Python (wersja 3.12) z wykorzystaniem odpowiednich bibliotek wspierających uczenie maszynowe i analizę danych. Wśród tych bibliotek znalazły się:

- **Pandas** – do wczytywania, przetwarzania i agregacji danych pomiarowych,
- **scikit-learn** – do implementacji algorytmu `Isolation Forest` oraz standaryzacji cech,
- **Matplotlib** oraz **Seaborn** – do wizualizacji wyników i analizy rozkładu danych

Środowisko projektu zostało zarządzane z wykorzystaniem narzędzia Poetry. Dodatkowo wykorzystana została biblioteka `requests` pozwalająca na komunikację z API aplikacji.

3.3 Trening modelu SI

3.3.1 Źródło danych

Dane wykorzystane w procesie trenowania modelu zostały podzielone na dwa zbiory: zbiór treningowy oraz zbiór testowy. Część danych pochodzi z rzeczywistego środowiska sieciowego, w którym cyklicznie wykonywano pomiary czasu odpowiedzi (ping), a następnie zapisywano je do plików tekstowych.

Pomiary zostały wykonane dla różnych typów urządzeń: komputera stacjonarnego (połączenie Ethernet), laptopa (WiFi), serwera, macierzy dyskowej, routera, przełącznika sieciowego, punktu dostępowego oraz urządzenia typu IoT. Pozwoliło to uzyskać zróżnicowany zbiór danych odzwierciedlający rzeczywiste warunki pracy sieci.

Dodatkowo wygenerowano dane syntetyczne, symulujące skrajne przypadki, takie jak wysoka utrata pakietów lub niestabilny czas odpowiedzi. Dane te umożliwiły weryfikację zachowania modelu w kontrolowanych warunkach.

3.3.2 Przygotowanie i agregacja danych

Surowe dane zostały poddane wstępnemu przetworzeniu obejmującemu usunięcie nieprawidłowych rekordów, zbędnych kolumn oraz ujednolicenie formatu danych.

Następnie pomiary zostały zagregowane do stałych, jednominutowych okien czasowych. Dla każdego okna obliczono wybrane statystyki opisujące jakość połączenia. Kluczowe cechy wejściowe modelu stanowiły:

- **packet_loss_1m** – procent utraconych pakietów w danym oknie czasowym,
- **ping_std_1m** – odchylenie standardowe czasu odpowiedzi w danym oknie.

Rozważano również wykorzystanie cech `ping_avg_1m` (średni czas odpowiedzi w danym oknie czasowym) oraz `ping_diff_1m` (różnica między aktualnym, a poprzednim średnim czasem odpowiedzi w dwóch kolejnych oknach czasowych), jednak w dalszych eksperymentach skoncentrowano się na parametrach najlepiej opisujących niestabilność połączenia.

Fragment implementacji agregacji przedstawiono na listingu 3.1.

```
    for window, g in grouped:
        successful = g[g["success"] == 1]

        if len(successful) >= 3:
            ping_std = successful["ping_ms"].std()
        else:
            ping_std = float("nan")

        packet_loss = 1 - len(successful) / len(g)

        windows.append({
            "window_start": window,
            "device_group": g["device_group"].iloc[0],
            "ping_std_1m": round(ping_std, 3),
            "packet_loss_1m": round(packet_loss, 3)
        })
```

Listing 3.1: Agregacja danych do okien czasowych

3.3.3 Trening modelu

Przygotowany zbiór danych treningowych został wykorzystany do uczenia modelu opartego na algorytmie `Isolation Forest`. Przed rozpoczęciem procesu uczenia dane wejściowe zostały poddane standaryzacji z wykorzystaniem klasy `StandardScaler`.

Standaryzacja polega na przekształceniu każdej cechy do rozkładu o średniej równej 0 oraz odchyleniu standardowym równym 1, co zapobiega dominacji cech o większym zakresie wartości.

Proces przygotowania danych i trenowania modelu przedstawiono na listingu 3.2.

```
    df = pd.read_csv(csv_path)
    X = df[FEATURES]

    feature_scaler = StandardScaler()
    X_scaled = feature_scaler.fit_transform(X)

    model.fit(X_scaled)
```

Listing 3.2: Skalowanie danych i trening modelu

3.3.4 Parametry modelu

Model `Isolation Forest` został skonfigurowany z wykorzystaniem następujących parametrów:

- **n_estimators** – liczba drzew izolacyjnych budujących model,
- **contamination** – oczekiwany udział anomalii w zbiorze danych,
- **random_state** – parametr zapewniający powtarzalność wyników.

Parametr `contamination` wpływa bezpośrednio na próg decyzyjny modelu i określa, jaki odsetek obserwacji zostanie zaklasyfikowany jako anomalny. Wartość ta jest procentowa, podawana w zakresie 0-1. Wartość parametru `contamination` powinna być dobrana eksperymentalnie, tak aby zminimalizować liczbę fałszywych anomalii przy zachowaniu wysokiej wykrywalności istotnych odchyleń.

Parametr `n_estimators` odpowiada za stabilność działania modelu – większa liczba drzew zwiększa powtarzalność i precyzyjność wyników, kosztem wydłużenia czasu treningu, co jest spowodowane większą złożonością obliczeniową.

Konfigurację modelu przedstawiono na listingu 3.3.

```
    RANDOM_STATE = 42

    def create_model(contamination: float, n_estimators: int) ->
       IsolationForest:
    return IsolationForest(
        n_estimators=n_estimators,
        contamination=contamination,
        random_state=RANDOM_STATE,
    )
```

Listing 3.3: Inicjalizacja modelu Isolation Forest

Po zakończeniu procesu uczenia wytrenowany model wraz ze skalarem został zapisany do pliku w formacie `joblib`, co umożliwia jego późniejsze wykorzystanie w aplikacji bez konieczności ponownego trenowania.

3.4 Testowanie modelu

W celu jak najlepszego przystosowania modelu do realiów sieci komputerowej, należało przeprowadzić rozległe testy jakościowe. Przeprowadzone testy pomogły zdecydować o doborze odpowiednich cech danych wejściowych i parametrów modelu. Jest to szczególnie istotne, ponieważ nieodpowiednio dobrane dane mogą powodować efekt niewykrywania istotnych szczegółów, czyli np.: utraconych pakietów i nagłych różnic w czasie odpowiedzi.

Wszystkie testy opisane w niniejszej pracy przedstawiono dla modelu trenowanego na danych z urządzeń typu klient LAN. Analogiczne testy przeprowadzono również dla modeli wytrenowanych do urządzeń typu WiFi oraz IoT, uzyskując spójne wnioski, przy czym charakterystyka anomalii różni się w zależności od typu urządzenia.

Poza dobieraniem odpowiednich cech, należało również dobrać jak najbardziej dopasowane parametry. Wśród nich najistotniejsze były: `contamination` oraz `n_estimators`, które opisane zostały w sekcji 3.3.4.

3.4.1 Założenia dotyczące testowania modelu

Aby uzyskać jak najlepiej dopasowany model, przetestowano różne kombinacje cech i parametrów, w celu znalezienia najlepszej z tych konfiguracji.

Jakość algorytmu była mierzona na podstawie analizy histogramu, wykresu skrzypcowego oraz dwóch tabel zawierających najważniejsze dane wynikowe.

Poza odczytami wykresów i tabeli posłużono się również wiedzą ekspercką, polegającą na odczycie danych wejściowych i wyjściowych, a następnie manualnej analizie zawartych w nich informacji.

Z przyczyn braku danych etykietowanych, nie można było posłużyć się bardziej precyzyjnymi metodami analizy wyników, która wprost wskazałaby na procentową jakość modelu.

Test każdej z kombinacji odbywał się na dwóch, takich samych dla każdej konfiguracji, zbiorów danych. Pierwszy zbiór zawierał dane rzeczywiste, zebrane w sposób naturalny w istniejącej i działającej sieci komputerowej, składającej się z dwóch podsieci i ok. 300 urządzeń. Drugi zestaw danych, to dane spreparowane, wygenerowane przez specjalnie stworzony do tego celu program.

3.4.2 Plan testów

Faza testowa, została podzielona na dwa etapy. W pierwszym szukany był odpowiedni zestaw cech wejściowych. W drugim etapie, korzystając z najlepszych cech, uzyskanych w fazie pierwszej, korygowane zostały parametry modelu, aby znaleźć odpowiedni środek ciężkości między jakością a wydajnością.

W testach nie uwzględniono np. kombinacji bez cechy `packet_loss_1m`, ponieważ odgrywa

ona kluczową rolę i jest najmocniejszym kandydatem wskazującym na prawdopodobne anomalie w sieci.

Mając na uwadze powyższe względy, powstała macierz testów, zaprezentowana w tabeli 3.1.

Tabela 3.1: Konfiguracje parametrów w poszczególnych fazach eksperymentu

Lp	Cechy	n_estimators	contamination
	FAZA 1		
1	ping_std_1m, packet_loss_1m	200	0.015
2	ping_avg_1m, packet_loss_1m	200	0.015
3	ping_diff_1m, packet_loss_1m	200	0.015
4	ping_diff_1m, ping_std_1m, ping_avg_1m, packet_loss_1m	200	0.015
	FAZA 2		
1	ping_std_1m, packet_loss_1m	100	0.015
2	ping_std_1m, packet_loss_1m	300	0.015
3	ping_std_1m, packet_loss_1m	200	0.010
4	ping_std_1m, packet_loss_1m	200	0.020

Źródło: opracowanie własne.

3.4.3 Wyniki testów

Testy przeprowadzono na dwóch zestawach danych: generowanych oraz rzeczywistych z serwera. W fazie 1 porównywano różne kombinacje cech wejściowych. Mimo, że najlepszy wynik wskazywał wariant 4 - kombinacja wszystkich cech, to jednak ręczna analiza wyników okazała inny rezultat. Najlepiej wypadł zestaw `ping_std_1m` i `packet_loss_1m` – generowane dane wykrywały anomalie prawidłowo, bez wyników fałszywie pozytywnych, choć pojedyncze zgubione pakiety przy niskim odchyleniu standardowym czasami nie były wykrywane. Dane serwera wskazały, że w tej konfiguracji niskie odchylenia są traktowane jako normalne, a istotne anomalie zostały rozpoznane, co potwierdzają wartości procentu anomalii i separacji w tabelach 3.2 i 3.3. Pozostałe zestawy cech wykazywały niską wykrywalność (zestawy 2 i 3) lub nadmiar fałszywych alarmów (zestaw 4).

W fazie 2 testowano różne parametry `n_estimators` i `contamination`. Wariant z 200 drzew i `contamination`=0.015, który był wartością bazową w fazie pierwszej, zapewniał najlepszy kompromis – na danych generowanych wykrywał niemal wszystkie anomalie bez fałszywych alarmów, natomiast dane serwera wskazywały na nadmiar wyników fałszywie pozytywnych przy innych konfiguracjach. Warianty z niską wykrywalnością (contamination 0.01) lub nadmiernymi fałszywymi alarmami (contamination 0.02) zostały odrzucone.

Tabela 3.2: Wyniki eksperymentów – wygenerowany klient - czas odpowiedzi

Wariant	n_estimators	contamination	% anomalii	separacja
FAZA 1 – porównanie cech				
ping_std_1m, packet_loss_1m	200	0.015	6.005	0.00132
ping_avg_1m, packet_loss_1m	200	0.015	7.046	0.00022
ping_diff_1m, packet_loss_1m	200	0.015	5.970	0.00052
ping_avg_1m, ping_std_1m, packet_loss_1m, ping_diff_1m	200	0.015	7.289	0.00150
FAZA 2 – zmiana parametrów				
100 drzew	100	0.015	6.213	0.00141
300 drzew	300	0.015	6.144	0.00145
contamination 0.01	200	0.010	5.276	0.00013
contamination 0.02	200	0.020	7.567	0.00037

Źródło: opracowanie własne.

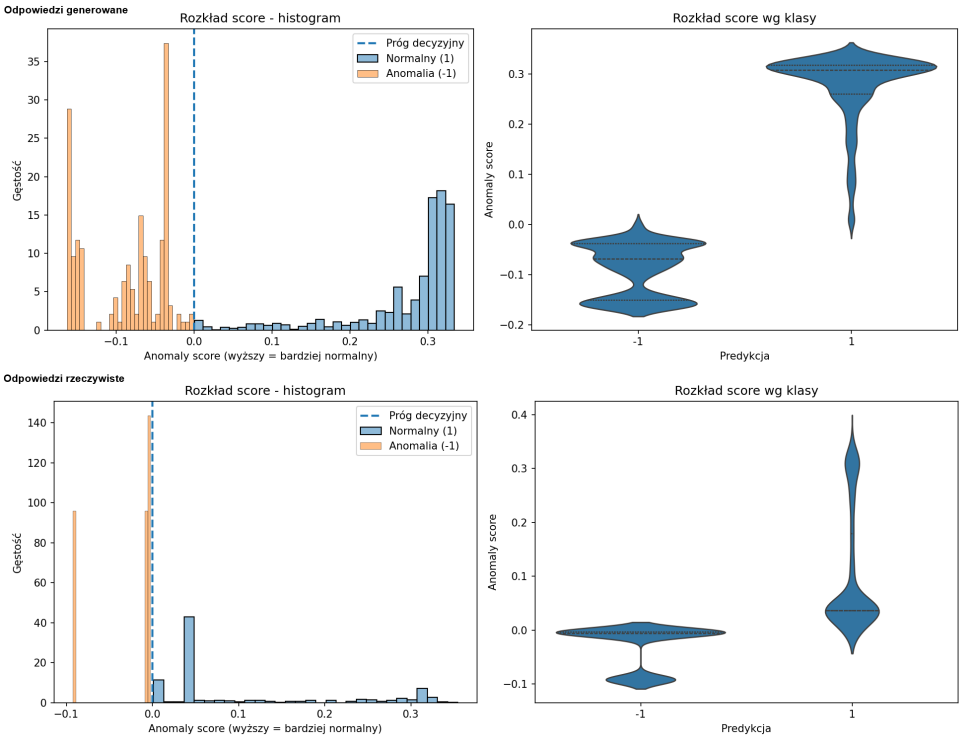
Tabela 3.3: Wyniki eksperymentów – serwer - czas odpowiedzi

Wariant	n_estimators	contamination	% anomalii	separacja
FAZA 1 – porównanie cech				
ping_std_1m, packet_loss_1m	200	0.015	1.118	0.00382
ping_avg_1m, packet_loss_1m	200	0.015	0.000	–
ping_diff_1m, packet_loss_1m	200	0.015	0.319	0.03637
ping_avg_1m, ping_std_1m, packet_loss_1m, ping_diff_1m	200	0.015	0.799	0.01630
FAZA 2 – zmiana parametrów				
100 drzew	100	0.015	9.425	0.00331
300 drzew	300	0.015	1.118	0.00507
contamination 0.01	200	0.010	0.319	0.07925
contamination 0.02	200	0.020	14.537	0.00374

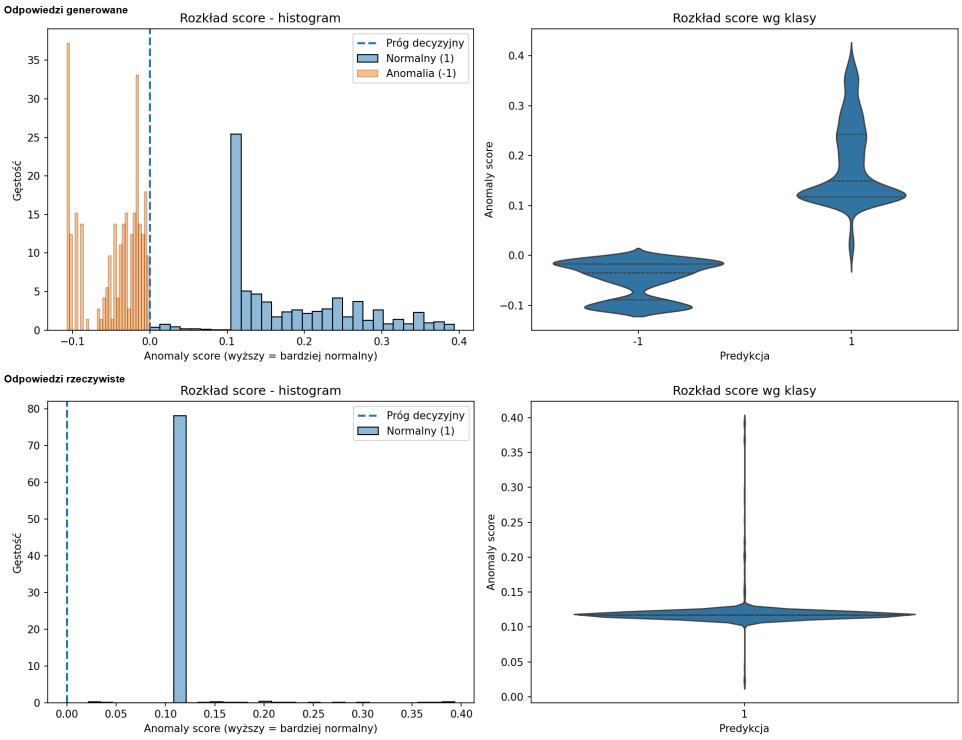
Źródło: opracowanie własne.

Dodatkowym elementem podlegającym analizie sa wykresy, przedstawione poniżej. Ukazują one wizualizację wyników, gdzie dane mniejsze od 0 wskazują na prawdopodobną anomalię, a te większe od 0 na wyniki w granicach normy.

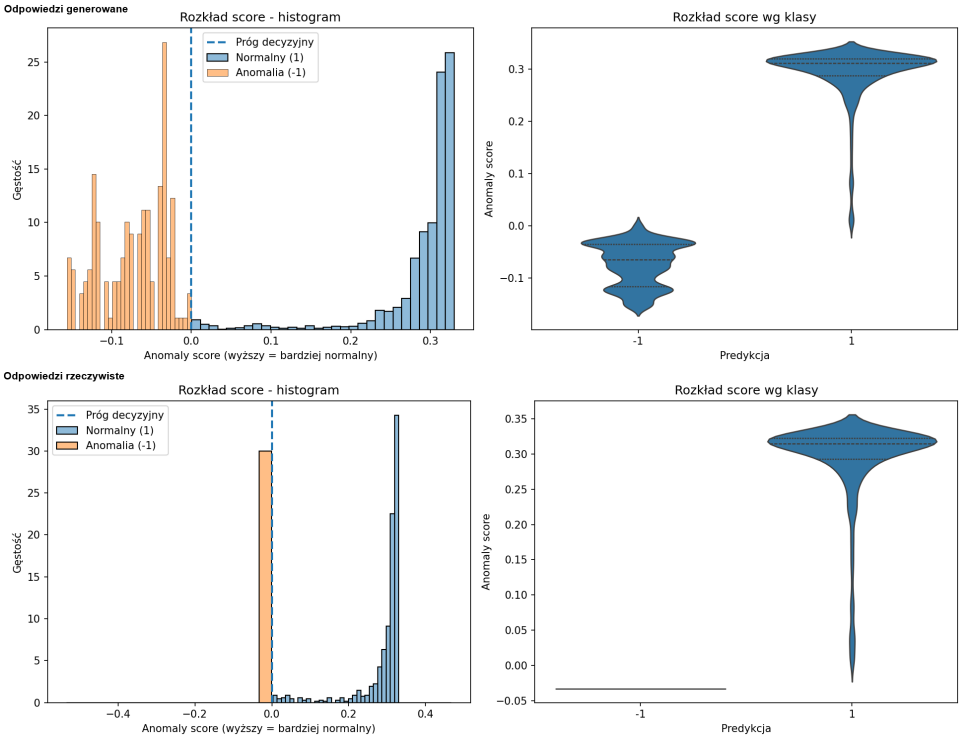
Dzięki analizie wykresów, jak i kolumny określającą separację w tabelach 3.2 i 3.3, można zauważyć problematyczną stronę aktualnego modelu i miejsce do dalszego udoskonalania. Widać tutaj, że granica między anomalią, a wynikiem oznaczonym jako normalny, bywa cienka. Oznacza to, że skrajne wyniki są wykrywane ze znaczną pewnością, lecz rezultaty pośrednie są położone blisko siebie.



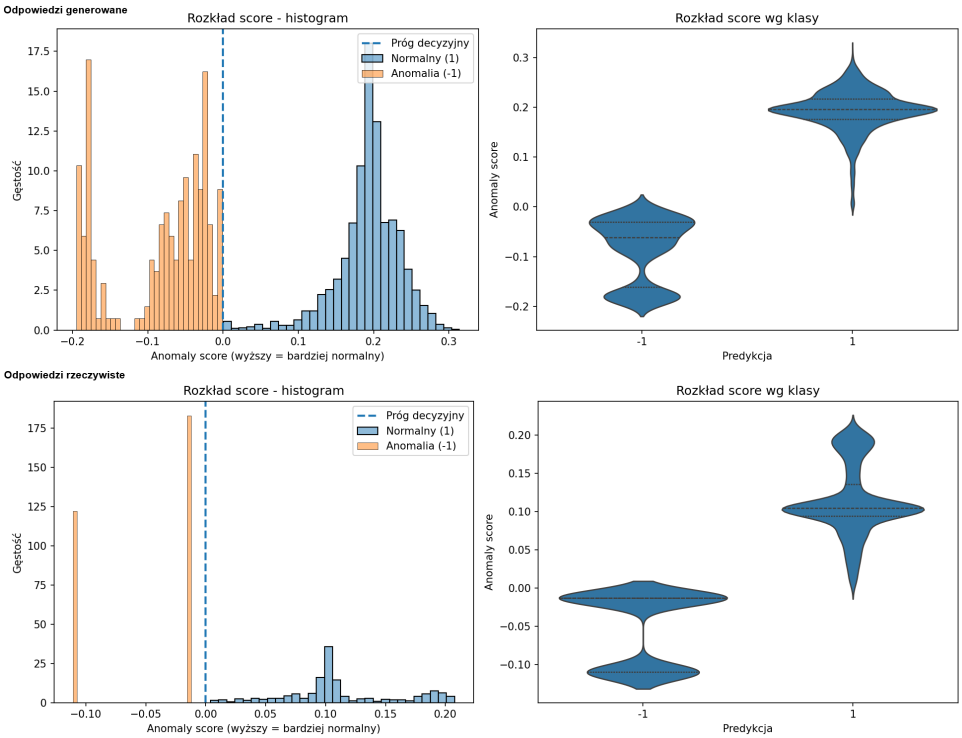
Rysunek 3.1: Wykresy rezultatów wariantu 1. fazy 1.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



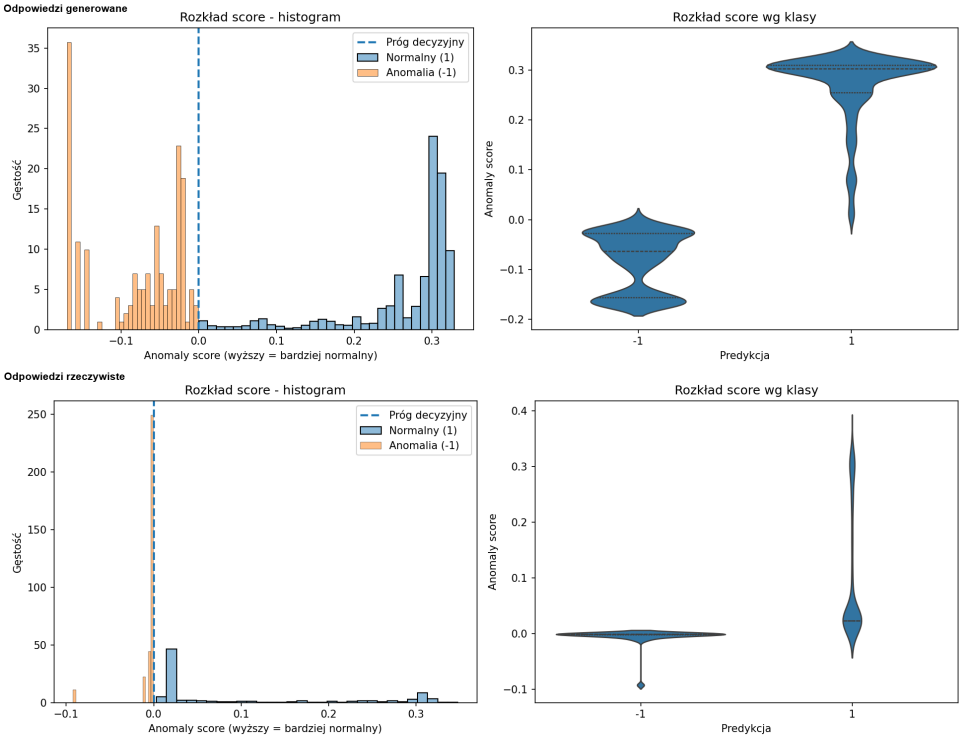
Rysunek 3.2: Wykresy rezultatów wariantu 2. fazy 1.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



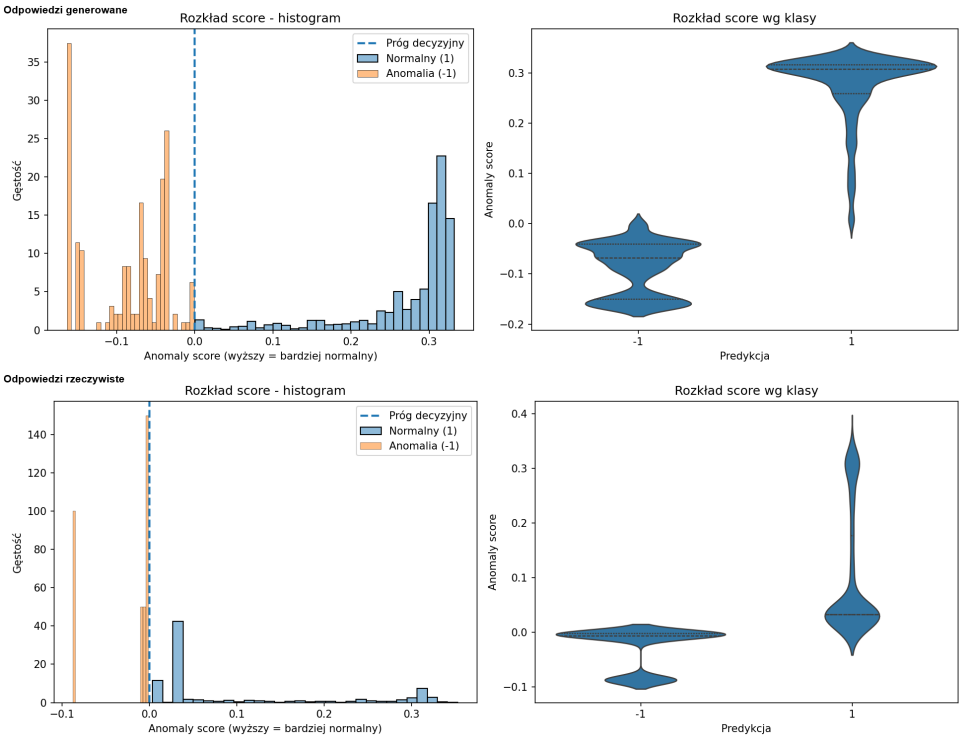
Rysunek 3.3: Wykresy rezultatów wariantu 3. fazy 1.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



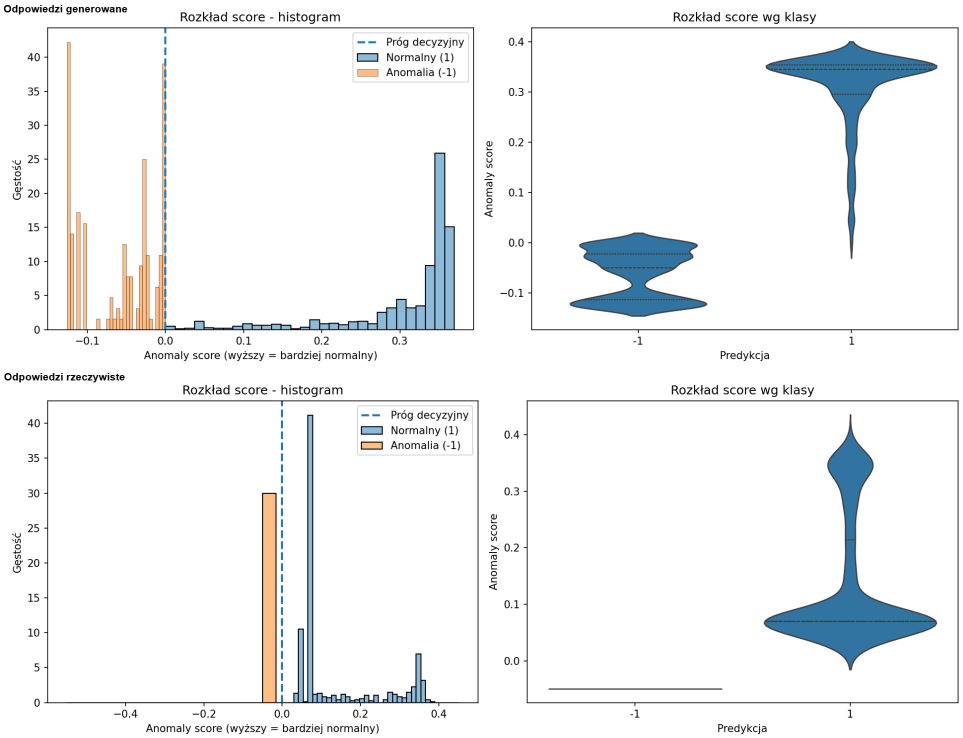
Rysunek 3.4: Wykresy rezultatów wariantu 4. fazy 1.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



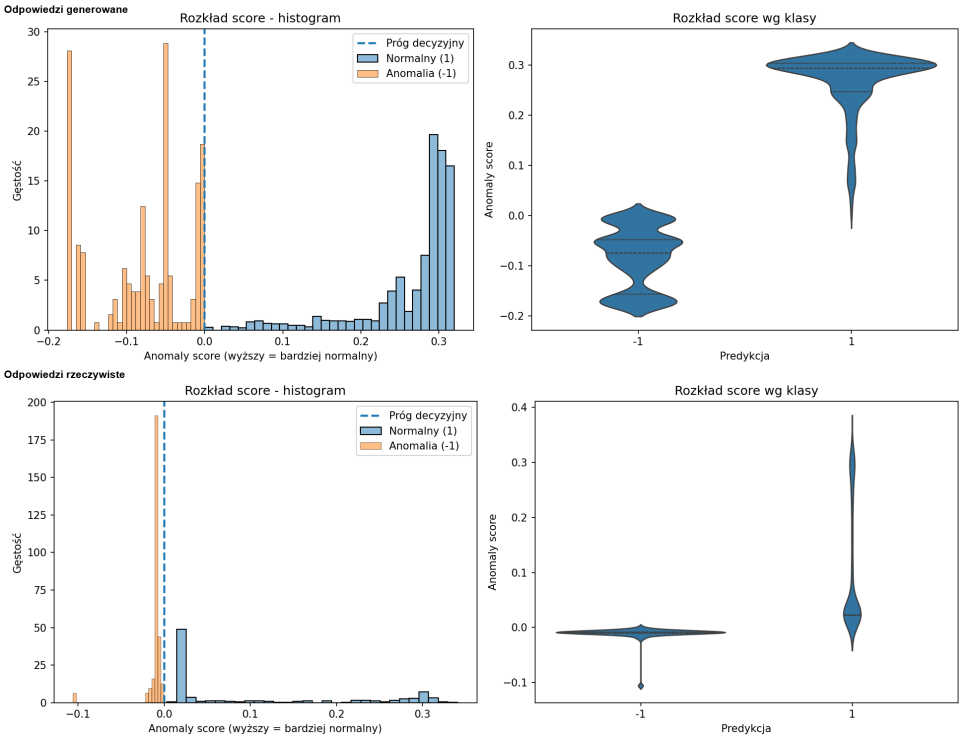
Rysunek 3.5: Wykresy rezultatów wariantu 1. fazy 2.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



Rysunek 3.6: Wykresy rezultatów wariantu 2. fazy 2.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



Rysunek 3.7: Wykresy rezultatów wariantu 3. fazy 2.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)



Rysunek 3.8: Wykresy rezultatów wariantu 4. fazy 2.
Źródło: opracowanie własne (Python, biblioteki matplotlib i seaborn)

Podsumowując, zarówno analiza ekspercka, jak i dane tabelaryczne potwierdzają, że optymalnym wyborem jest model z cechami `ping_std_1m` i `packet_loss_1m`, z parametrami `n_estimators`=200 i `contamination`=0.015. Konfiguracja ta zapewnia równowagę między skutecznością wykrywania rzeczywistych anomalii a ograniczeniem wyników fałszywie pozytywnych. Należy jednak mieć na uwadze, że żaden z testowanych wariantów nie pozwoli uzyskać 100% skuteczności i mimo wysokiej jakości wytrenowanego modelu, mogą zdarzać się wyniki fałszywie pozytywne, lub pozytywnie fałszywe.

3.5 Podsumowanie testów

Analiza testów jakościowych wykazała, że dobór odpowiednich cech wejściowych (`ping_std_1m`, `packet_loss_1m`) oraz parametrów modelu (`n_estimators`=200, `contamination`=0.015) pozwala na skuteczne wykrywanie anomalii w danych z urządzeń typu klient LAN. Podobne testy dla urządzeń WiFi i IoT wskazały, że model wymaga indywidualnego dopasowania parametrów ze względu na różną charakterystykę sygnału i częstotliwość występowania anomalii. Otrzymane wyniki pokazują, że algorytm `Isolation Forest` jest efektywny w analizie niestabilności sieci oraz pozwala ograniczyć wpływ fałszywych pozytywów w większości scenariuszy testowych.

Spis rysunków

Spis tabel

Spis listingów

Bibliografia

[1] S. Sinha, *State of IoT 2025: Number of connected IoT devices growing 14% to 21.1 billion globally*, 2025, `https://iot-analytics.com/number-connected-iot-devices/` [dostęp: 07.12.2025].