

Sprawozdanie z projektu
Badanie metod przewidywania mocy w bolidzie klasy
Formula Student



POLITECHNIKA POZNAŃSKA
Wydział Automatyki, Robotyki i Elektrotechniki

Michał Błotniak
159409

Mateusz Gogola
159609

29 styczeń 2026

Spis treści

1 Sformułowanie problemu badawczego i analiza układu napędowego	3
1.1 Uwarunkowania regulaminowe i kary (Power Limitation)	3
1.2 Fizyka bilansu energetycznego i nieliniowość strat	3
1.3 Wpływ Torque Vectoringu na dynamikę poboru mocy	4
1.4 Analiza opóźnień w pętli sprzężenia zwrotnego (Feedback)	4
1.5 Koncepcja rozwiązania: Wirtualny Sensor Predykcyjny	5
2 Metodyka badań i strategia doboru architektury modelu	6
2.1 Paradygmat modelowania: White-box vs. Black-box	6
2.2 Etap 1: Weryfikacja liniowości i model regresyjny (Baseline)	6
2.3 Etap 2: Sformułowanie ograniczeń sprzętowych (Hardware Constraints)	7
2.3.1 Budżet czasowy (Real-Time Constraint)	7
2.3.2 Współdzielenie zasobów w środowisku ROS 2	7
2.4 Etap 3: Strategia doboru topologii sieci (Architecture Search)	7
2.4.1 Model A: "Lekki" (Lightweight Candidate)	7
2.4.2 Model B: "Złożony" (High-Capacity Candidate)	7
2.5 Kryteria ewaluacji i metryki bezpieczeństwa	8
3 Wstępna obróbka danych (Data Preprocessing)	9
3.1 Organizacja struktury projektu i optymalizacja	9
3.2 Akwizycja i standaryzacja danych	9
3.3 Synchronizacja czasowa (Merge Asof)	10
3.4 Inżynieria cech i czyszczenie danych	10
4 Weryfikacja liniowości i model regresyjny (Baseline)	11
4.1 Cel eksperymentu	11
4.2 Implementacja i Kluczowe Elementy Kodu	11
4.3 Wyniki Regresji Liniowej	11
5 Ewaluacja Regresji Wielomianowej i analiza błędów lokalnych	14
5.1 Implementacja i kluczowe elementy kodu	14
5.2 Analiza wizualna i identyfikacja niedoskonałości	14
5.2.1 Niestabilność powierzchni predykcji	14
5.2.2 Krytyczne błędy lokalne (False Positives/Negatives)	15
5.3 Analiza wizualna i identyfikacja błędów lokalnych	16
6 Model 1: Bazowa sieć MLP (Baseline)	18
6.1 Architektura i Konfiguracja	18
6.2 Wyniki i Wnioski (MLP)	18

7 Implementacja algorytmu Gradient Boosting	20
7.1 Inżynieria cech oparta na fizyce (Physics-Informed)	20
7.2 Konfiguracja modelu	20
7.3 Analiza wyników i interpretacja	21
7.4 Podsumowanie	22
8 Model 3: XGBoost i Walidacja Chronologiczna	23
8.1 Korekta metodologii: Eliminacja wycieku danych	23
8.2 Architektura modelu XGBoost	23
8.3 Szczegółowa analiza wyników	24
8.3.1 Metryki wydajności	24
8.3.2 Interpretacja wizualna	24
8.4 Wnioski	25
9 Model 4: Głęboka Sieć Rekurencyjna (LSTM)	26
9.1 Przygotowanie danych: Okno Przesuwne (Sliding Window)	26
9.2 Architektura sieci	26
9.3 Szczegółowa analiza wyników modelu LSTM	27
9.3.1 Analiza zbieżności modelu (Krzywa uczenia)	27
9.3.2 Odwzorowanie dynamiki bolidu	28
9.3.3 Statystyczna analiza błędów (Reszty)	28
10 Podsumowanie końcowe projektu	29
10.1 Analiza wydajności czasowej (Inference Time)	29

1 Sformułowanie problemu badawczego i analiza układu napędowego

Projektowanie systemu napędowego wyczynowego bolidu elektrycznego klasy Formula Student stanowi wielokryterialne zadanie optymalizacyjne. Inżynierowie muszą balansować pomiędzy maksymalizacją osiągów dynamicznych (przyspieszenie wzdłużne i poprzeczne) a zachowaniem niezawodności i bezpieczeństwa układu.

Niniejszy rozdział definiuje istotę problemu badawczego, którym jest opracowanie predykcyjnego algorytmu estymacji mocy czynnej. Jest to element krytyczny dla strategii zarządzania energią (Energy Management Strategy) w pojeździe wyposażonym w niezależny napęd na cztery koła i system aktywnego wektorowania momentu (Torque Vectoring).

1.1 Uwarunkowania regulaminowe i kary (Power Limitation)

Fundamentem ograniczeń projektowych jest regulamin zawodów *Formula Student Germany* (FSG), który stanowi standard dla europejskich edycji konkursu inżynierskiego. Zgodnie z punktem **EV 2.2.1** regulaminu na sezon 2026:

"The TS power at the outlet of the TSAC must not exceed 80 kW." [?]

System monitorujący (Data Logger organizatora) próbuje iloczyn napięcia i prądu DC. Przekroczenie wartości granicznej $P_{lim} = 80$ kW skutkuje dyskwalifikacją zespołu z konkurencji. Warto podkreślić, że margines błędu jest zerowy – systemy "reaktywne" często zawodzą w stanach nieustalonych, co wymusza stosowanie dużych marginesów bezpieczeństwa (np. limitowanie mocy na poziomie 75 kW), co jednak drastycznie pogarsza konkurencyjność pojazdu.

Zadaniem układu sterowania jest więc spełnienie nierówności w każdej chwili czasowej t :

$$P_{DC}(t) \leq P_{lim} - \delta_{safety} \quad (1)$$

gdzie δ_{safety} to minimalny bufor bezpieczeństwa, który dzięki predykcji chcemy zminimalizować.

1.2 Fizyka bilansu energetycznego i nieliniowość strat

Głównym wyzwaniem w modelowaniu poboru mocy nie jest samo wyliczenie mocy mechanicznej, lecz uwzględnienie nieliniowych strat w torze przetwarzania energii. Całkowita moc pobierana z baterii P_{DC} wyraża się wzorem:

$$P_{DC} = \sum_{i=1}^4 (P_{mech,i} + P_{loss,i}) + P_{aux} \quad (2)$$

Gdzie moc mechaniczna $P_{mech,i} = T_i \cdot \omega_i$. Problem stanowi składnik strat P_{loss} , na który składają się:

- **Straty w uzwojeniach silnika (Joule losses):** $P_{cu} = I_{rms}^2 \cdot R_s$, zależne kwadratowo od momentu (prądu).

- **Straty w żelazie (Iron losses):** Zależne od częstotliwości wirowania strumienia magnetycznego (prędkości obrotowej).
- **Straty łączniowe falownika (Switching losses):** Zależne od częstotliwości kluczowania tranzystorów IGBT/SiC oraz temperatury złącza.

Ostatecznie sprawność systemu η jest nieliniową funkcją wielu zmiennych:

$$P_{DC} = \sum_{i=1}^4 \frac{T_i \cdot \omega_i}{\eta(T_i, \omega_i, Temp, V_{bat})} + P_{aux} \quad (3)$$

Użycie klasycznych metod analitycznych wymagałoby posiadania idealnych map sprawności (Efficiency Maps) dla całego zakresu pracy, co w warunkach rzeczywistych (zmienna temperatura, starzenie się komponentów) jest trudne do uzyskania. To pierwszy argument przemawiający za użyciem sieci neuronowej, która uczy się tej charakterystyki "implikite".

1.3 Wpływ Torque Vectoringu na dynamikę poboru mocy

System Torque Vectoring (TV) ma na celu wygenerowanie momentu kursowego M_z (Yaw Moment), który wspomaga skręcanie pojazdu. Odbywa się to kosztem efektywności energetycznej.

Podczas agresywnego pokonywania zakrętu:

1. Algorytm TV zwiększa moment na kołach zewnętrznych ($T_{outer} \uparrow$) i zmniejsza na wewnętrznych ($T_{inner} \downarrow$).
2. Koła zewnętrzne wirują szybciej ($\omega_{outer} > \omega_{inner}$) ze względu na kinematykę pojazdu.
3. Iloczyn $T_{outer} \cdot \omega_{outer}$ rośnie gwałtownie, podczas gdy spadek mocy na kołach wewnętrznych jest mniejszy (bo ω_{inner} jest mniejsze).

Prowadzi to do zjawiska **amplifikacji mocy**. W ułamku sekundy, gdy kierowca dokręca kierownicę, zapotrzebowanie na moc może wzrosnąć skokowo (tzw. *Power Spike*), nawet jeśli wciśnięcie pedału przyspieszenia pozostaje stałe.

$$\frac{dP_{DC}}{dt} \approx \sum \frac{\omega}{\eta} \cdot \frac{dT}{dt} \quad (4)$$

Przy wysokich prędkościach obrotowych (ω), nawet niewielka dynamika zmian momentu ($\frac{dT}{dt}$) generowana przez regulator TV powoduje ogromną dynamikę zmian mocy.

1.4 Analiza opóźnień w pętli sprzężenia zwrotnego (Feedback)

Dlaczego klasyczny regulator PID (reagujący na błąd $e = 80kW - P_{actual}$) jest niewystarczający? Odpowiedź leży w teorii sterowania i analizie opóźnień transportowych w pętli cyfrowej.

Na całkowity czas reakcji systemu (t_{delay}) składają się:

- t_{meas} : Czas próbkowania i filtracji prądu w falowniku (często stosuje się filtry uśredniające 10-20ms w celu redukcji szumu EMI).

- t_{CAN} : Opóźnienie transmisji na magistrali CAN Bus. Przy dużym obciążeniu magistrali, ramki z danymi o prądzie mogą ulec arbitrażu (priorytetyzacji).
- t_{cycle} : Czas cyklu sterownika VCU (Vehicle Control Unit), zazwyczaj 10ms.
- t_{act} : Czas reakcji falownika na nową komendę momentu.

Szacunkowy całkowity czas opóźnienia wynosi:

$$t_{delay} \approx 30 - 60 \text{ ms} \quad (5)$$

Przy dynamice wzrostu mocy w bolidzie Formula Student sięgającej 200 kW/s, opóźnienie rzędu 50ms oznacza, że w momencie reakcji regulatora, moc rzeczywista może już przekraczać limit o:

$$\Delta P_{overshoot} = 200 \frac{\text{kW}}{\text{s}} \cdot 0.05 \text{ s} = 10 \text{ kW} \quad (6)$$

Przekroczenie limitu o 10 kW (czyli osiągnięcie 90 kW) oznacza natychmiastową dyskwalifikację.

1.5 Koncepcja rozwiązania: Wirtualny Sensor Predykcyjny

Wobec wykazanych ograniczeń metod reaktywnych i analitycznych, sformułowano koncepcję układu sterowania ze sprzężeniem "w przód"(Feed-Forward).

Kluczowym elementem jest zastąpienie fizycznego pomiaru mocy (obarzonego opóźnieniem) estymatorem opartym na **Sieci Neuronowej**. Model ten ma działać jako "Wirtualny Sensor", który na wejście otrzymuje stan zadany (moment zadany na koła przez algorytm TV oraz w ten sposób uzyskaną prędkość), a na wyjściu zwraca przewidywaną moc, zanim zostanie ona wygenerowana przez falowniki.

Algorytm sterowania przyjmuje postać:

1. Pobierz zadane momenty T_{cmd} z algorytmu Torque Vectoring.
2. Wprowadź stan pojazdu (T_{cmd}, ω, \dots) do sieci neuronowej.
3. Otrzymaj predykcję mocy \hat{P}_{next} .
4. Jeśli $\hat{P}_{next} > P_{lim}$, oblicz współczynnik skalujący $k < 1$ (Derating Factor).
5. Wyślij do falowników zredukowane momenty $T_{final} = k \cdot T_{cmd}$.

Dzięki takiemu podejściu, ograniczenie mocy następuje w tym samym kroku obliczeniowym co wyznaczenie momentów, eliminując problem opóźnień transportowych.

2 Metodyka badań i strategia doboru architektury modelu

Projektowanie systemu estymacji mocy dla pojazdu wyścigowego nie jest trywialnym zadaniem doboru parametrów, lecz wieloetapowym procesem badawczym. Przyjęta metodyka opiera się na stopniowaniu złożoności (Complexity Gradation Approach), gdzie punktem wyjścia są proste modele analityczne, a celem końcowym – wysoko wydajne struktury neuronowe, zoptymalizowane pod kątem pracy w systemie czasu rzeczywistego (Real-Time System).

Niniejszy rozdział opisuje strategię przejścia od modelowania klasycznego do uczenia maszynowego, definiuje ograniczenia sprzętowe platformy docelowej oraz kryteria ewaluacji modeli.

2.1 Paradygmat modelowania: White-box vs. Black-box

W inżynierii sterowania wyróżnia się dwa główne podejścia do identyfikacji obiektów:

1. **Modelowanie Białej Skrzynki (White-box):** Podejście oparte na jawnych równaniach fizyki (First Principles). Wymaga dokładnej znajomości parametrów każdego komponentu (rezystancji uzwojeń $R(T)$, charakterystyk tranzystorów, tarcia łożysk).
2. **Modelowanie Czarnej Skrzynki (Black-box):** Podejście oparte na danych (Data-Driven), gdzie struktura wewnętrzna obiektu nie jest znana, a model aproksymuje funkcję przejścia $Y = f(X)$ na podstawie zbioru uczącego.

Ze względu na niemożność uzyskania idealnych parametrów fizycznych dla zużywających się podzespołów bolidu, zdecydowano się na podejście oparte na danych. Jednakże, zgodnie z zasadą parsymonii (Brzytwa Ockhama), badania rozpoczęto od najprostszych struktur typu Black-box.

2.2 Etap 1: Weryfikacja liniowości i model regresyjny (Baseline)

Pierwszym krokiem była weryfikacja hipotezy, czy przestrzeń stanów pojazdu daje się zmapować na pobór mocy za pomocą transformacji liniowych lub wielomianowych niskiego rzędu. Zastosowano regresję wielomianową (*Polynomial Regression*), która pełni rolę modelu bazowego (*Baseline Model*).

Zalety tego podejścia w systemach wbudowanych są niepodważalne:

- **Deterministyczny czas obliczeń:** Złożoność obliczeniowa wynosi $O(1)$ dla ustalonego stopnia wielomianu.
- **Przejrzystość:** Możliwość analitycznego wyznaczenia wrażliwości wyjścia na zmiany wejść.

Jak wykazano w sekcji eksperymentalnej, model ten obarczony jest jednak zbyt dużym błędem obciążenia (High Bias). Zjawisko "niedouczenia" (Underfitting) w tym przypadku wynika z faktu, że wielomian 3. stopnia nie posiada wystarczającej pojemności informacyjnej (Capacity), aby odwzorować ostre nieliniowości wynikające z działania elektroniki energoelektronicznej (fałowników) oraz nagłych zmian obciążenia generowanych przez Torque Vectoring.

2.3 Etap 2: Sformułowanie ograniczeń sprzętowych (Hardware Constraints)

Przejście na modele sieci neuronowych (ANN) wiąże się z ryzykiem narzutu obliczeniowego. Platforma docelowa bolidu (Komputer Pokładowy / ECU) narzuca sztywne ograniczenia, które zdefiniowano następująco:

2.3.1 Budżet czasowy (Real-Time Constraint)

System sterowania pojazdu pracuje w pętli o stałym okresie T_{cycle} (np. 10 ms lub 20 ms). Czas predykcji sieci (τ_{inf}) musi być znaczaco krótszy od tego okresu, aby pozostawić czas na komunikację CAN i logikę bezpieczeństwa. Warunek stabilności systemu:

$$\tau_{inf} + \tau_{comm} + \tau_{safety} < T_{cycle} \quad (7)$$

Przyjęto rygorystyczne wymaganie, aby czas samej inferencji modelu nie przekraczał $\tau_{inf} \leq 2$ ms na docelowym CPU. Wyklucza to stosowanie bardzo głębokich sieci konwolucyjnych lub rekurencyjnych (LSTM/Transformer).

2.3.2 Współdzielenie zasobów w środowisku ROS 2

Komputer pokładowy pracuje pod kontrolą systemu ROS 2 (Robot Operating System). Proces predykcji mocy jest tylko jednym z wielu węzłów (Nodes) w grafie obliczeniowym, obok algorytmów SLAM, planowania ścieżki i przetwarzania chmury punktów LiDAR. Zbyt "ciężki" model mógłby doprowadzić do zjawiska *Resource Starvation* dla procesów krytycznych, co jest niedopuszczalne.

2.4 Etap 3: Strategia doboru topologii sieci (Architecture Search)

Aby znaleźć kompromis między dokładnością a wydajnością, zdefiniowano przestrzeń poszukiwań obejmującą dwie klasy modeli sieci jednokierunkowych (Feed-Forward Neural Networks):

2.4.1 Model A: "Lekki"(Lightweight Candidate)

Architektura zoptymalizowana pod kątem minimalizacji opóźnień (Low Latency).

- **Charakterystyka:** Płytki sieć (1-2 warstwy ukryte), mała liczba neuronów (np. 16-32 na warstwę).
- **Cel:** Stworzenie modelu, który jest możliwy do implementacji nawet na prostszych mikrokontrolerach (MCU) bez akceleratorów AI, zachowując precyzję lepszą od regresji.

2.4.2 Model B: "Złożony"(High-Capacity Candidate)

Architektura nastawiona na maksymalizację dokładności aproksymacji.

- **Charakterystyka:** Głębsza struktura (3-4 warstwy), szersze warstwy (64-128 neuronów), zastosowanie funkcji aktywacji typu ReLU/Leaky ReLU.

- **Cel:** Wyznaczenie teoretycznego limitu błędu (Benchmark), jaki można osiągnąć na zgromadzonym zbiorze danych. Model ten posłuży jako punkt odniesienia (Oracle) do oceny, jak wiele informacji tracimy, redukując sieć do wersji "Lekkiej".

2.5 Kryteria ewaluacji i metryki bezpieczeństwa

W klasycznych zadaniach ML (Machine Learning) główną metryką jest błąd średniokwadratowy (MSE). W kontekście limitu mocy w Formula Student, średni błąd jest jednak mylący. Dla bezpieczeństwa systemu kluczowe są wartości skrajne. Dlatego przyjęto wielokryterialną ocenę modeli:

1. **RMSE (Root Mean Square Error):** Ogólna miara jakości dopasowania modelu.
2. **MaxAE (Maximum Absolute Error):** Maksymalny błąd bezwzględny zanotowany w zbiorze testowym.

$$MaxAE = \max_i |y_i - \hat{y}_i| \quad (8)$$

Jest to metryka krytyczna. Jeśli $MaxAE$ wynosi np. 5 kW, oznacza to, że musimy ustawić bufor bezpieczeństwa na 5 kW (limit 75 kW zamiast 80 kW), co pogarsza osiągi. Dążymy do minimalizacji tej wartości.

3. **Czas inferencji (Inference Time):** Średni czas wykonania predykcji dla jednej próbki danych.

Ostateczny wybór modelu będzie oparty na wskaźniku *Efficiency Score*, rozumianym jako iloraz zysku dokładności do kosztu obliczeniowego.

3 Wstępna obróbka danych (Data Preprocessing)

Celem etapu wstępnej obróbki danych było przygotowanie spójnego zbioru uczącego na podstawie surowych odczytów z czujników bolidu. Dane wejściowe pochodziły z różnych systemów pomiarowych i były zapisane w oddzielnych plikach CSV, powstały wskutek systemu logowania rosbag, przez komputer pokładowy. Proces ten zrealizowano w języku Python z wykorzystaniem biblioteki pandas.

3.1 Organizacja struktury projektu i optymalizacja

W celu zapewnienia przenośności kodu pomiędzy różnymi systemami operacyjnymi oraz uporządkowania struktury plików, wykorzystano bibliotekę pathlib. Skrypt automatycznie lokalizuje katalog z danymi (./data) względem pliku wykonywalnego.

Zastosowano również mechanizm zapobiegający redundancji obliczeń. Skrypt sprawdza, czy plik wynikowy (final_data.csv) już istnieje. Jeżeli tak, proces przetwarzania jest pomijany, co znacząco skraca czas uruchamiania notatnika przy wielokrotnych testach modelu.

```
1 DATA_DIR = Path('~/data')
2 OUTPUT_FILE = DATA_DIR / 'final_data.csv'
3
4 if OUTPUT_FILE.exists():
5     print(f'Plik {OUTPUT_FILE} już istnieje. Pomijam generowanie.')
6     merged = pd.read_csv(OUTPUT_FILE)
7 else:
8     # Rozpocznij proces przetwarzania (ETL)
9     # ...
```

Listing 1: Mechanizm sprawdzania istnienia przetworzonych danych

3.2 Akwizycja i standaryzacja danych

Dane wejściowe obejmowały trzy główne kategorie:

- **Wartości zadane (Setpoints):** Sygnały sterujące z komputera pokładowego do falownika (setpoints.csv).
- **Dane elektryczne:** Pomiary napięcia i prądu z akumulatora (fsp_endu_current.csv).
- **Prędkości kół:** Odczyty z czujników prędkości dla kół: przedniego prawego, tylnego lewego i tylnego prawego.

Dla plików z danymi kół zastosowano funkcję pomocniczą load_wheel_data, która standaryzuje nazwy kolumn, usuwa zbędne metadane (np. topic) oraz oblicza wartość bezwzględną prędkości, co jest kluczowe, gdyż dla modelu estymacji mocy kierunek obrotu kół jest w tym przypadku pomijalny.

3.3 Synchronizacja czasowa (Merge Asof)

Kluczowym wyzwaniem była synchronizacja danych pochodzących z asynchronicznych źródeł. Tradycyjne łączenie tabel (inner join) było niemożliwe ze względu na brak idealnie pokrywających się znaczników czasu (`elapsed time`).

Zastosowano zaawansowaną metodę łączenia `merge_asof` (Asynchronous Join). Funkcja ta dopasowuje do rekordu z lewej tabeli rekord z prawej tabeli, który jest najbliższy w czasie. Skonfigurowano parametry:

- `direction='nearest'`: Wybiera pomiar najbliższy czasowo, a nie tylko poprzedzający, co zwiększa precyzję.
- `tolerance=0.1`: Odrzuca dopasowania, jeśli różnica czasu przekracza 100 ms, co zapobiega łączeniu danych zbyt odległych w czasie (np. przy awarii czujnika).

```
1 # Laczenie danych z kol
2 merged_wheels = pd.merge_asof(
3     data_fr.sort_values('elapsed time'),
4     data_rl.sort_values('elapsed time'),
5     on='elapsed time',
6     direction='nearest',
7     tolerance=0.1
8 )
9 # Interpolacja liniowa brakujacych wartosci
10 merged_wheels.interpolate(method='linear', inplace=True)
```

Listing 2: Synchronizacja danych z wykorzystaniem `merge_asof`

3.4 Inżynieria cech i czyszczenie danych

Na zsynchronizowanym zbiorze danych przeprowadzono inżynierię cech (Feature Engineering). Obliczono moc chwilową (P), która stanowi zmienną celu (target) dla modelu sieci neuronowej, wykorzystując wzór:

$$P = U \cdot I \quad (9)$$

gdzie U to napięcie (`voltage`), a I to natężenie prądu (`current`).

W końcowym etapie usunięto kolumny pomocnicze (czas, składowe napięcia i prądu), aby zapobiec wyciekowi danych (data leakage) – model ma przewidywać moc na podstawie wystęrowania i prędkości, nie znając prądu bezpośrednio. Odfiltrowano również błędne próbki, w których moc przyjmowała wartości ujemne (co w kontekście tego badania uznano za błędy pomiarowe lub stany nieistotne dla modelu napędowego).

```
1 merged['power'] = merged['voltage'] * merged['current']
2 merged.drop(columns=['elapsed time', 'voltage', 'current'], inplace=True)
3 merged.dropna(inplace=True)
4 merged = merged[merged['power'] >= 0]
```

Listing 3: Obliczenie mocy i czyszczenie zbioru

Ostateczny plik `final_data.csv` zawiera kompletny, zsynchronizowany zestaw danych gotowy do treningu.

4 Weryfikacja liniowości i model regresyjny (Baseline)

4.1 Cel eksperymentu

Zgodnie z przyjętą metodyką badawczą, pierwszym etapem (tzw. *Baseline*) było sprawdzenie, czy zależność między stanem pojazdu (prędkości kół, zadany moment) a poborem mocy może zostać opisana funkcją liniową. Fizyczny wzór na moc mechaniczną $P = T \cdot \omega$ sugeruje liniowość, jednak straty w układzie (cieplne, łączniowe) mają charakter nieliniowy (I^2R). Celem regresji było oszacowanie skali błędu wynikającego z pominięcia tych nieliniowości.

4.2 Implementacja i Kluczowe Elementy Kodu

Do eksperymentu wykorzystano bibliotekę `scikit-learn`. Dane zostały podzielone na zbiór treningowy i testowy, a następnie poddane regresji liniowej wielu zmiennych.

Kluczowe fragmenty kodu:

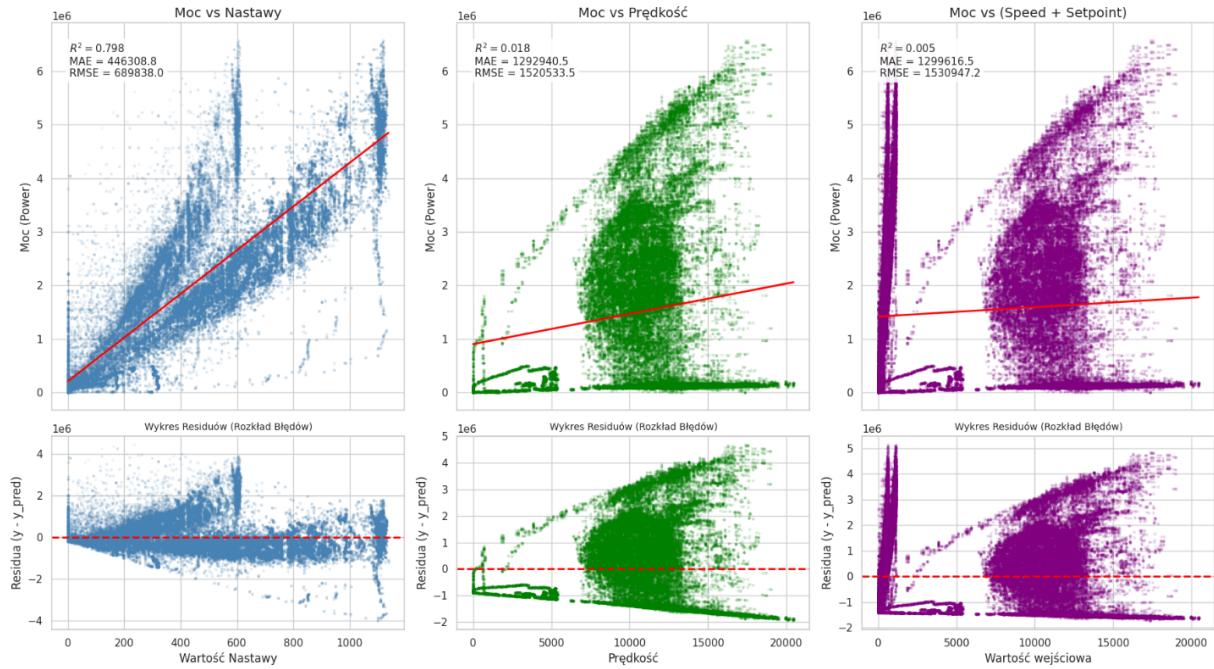
```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.preprocessing import PolynomialFeatures
3 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
4
5 # Definicja cech (X) i celu (y)
6 features = ['setpoint_fr', 'setpoint_rl', 'setpoint_rr',
7             'speed_fr', 'speed_rl', 'speed_rr']
8 target = 'power'
9
10 X = merged[features]
11 y = merged[target]
12
13 # 1. Model Liniowy (Linear Regression)
14 model_lin = LinearRegression()
15 model_lin.fit(X_train, y_train)
16 y_pred_lin = model_lin.predict(X_test)
17
18 # Ewaluacja
19 r2_lin = r2_score(y_test, y_pred_lin)
20 mae_lin = mean_absolute_error(y_test, y_pred_lin)
21
22 print(f"Regresja Liniowa R2: {r2_lin:.4f}")
```

Listing 4: Implementacja regresji liniowej w Scikit-Learn

4.3 Wyniki Regresji Liniowej

W pierwszej iteracji zastosowano model liniowy postaci $Y = w_0 + w_1x_1 + \dots + w_nx_n$.

Analiza wykresów 2D (Residua)

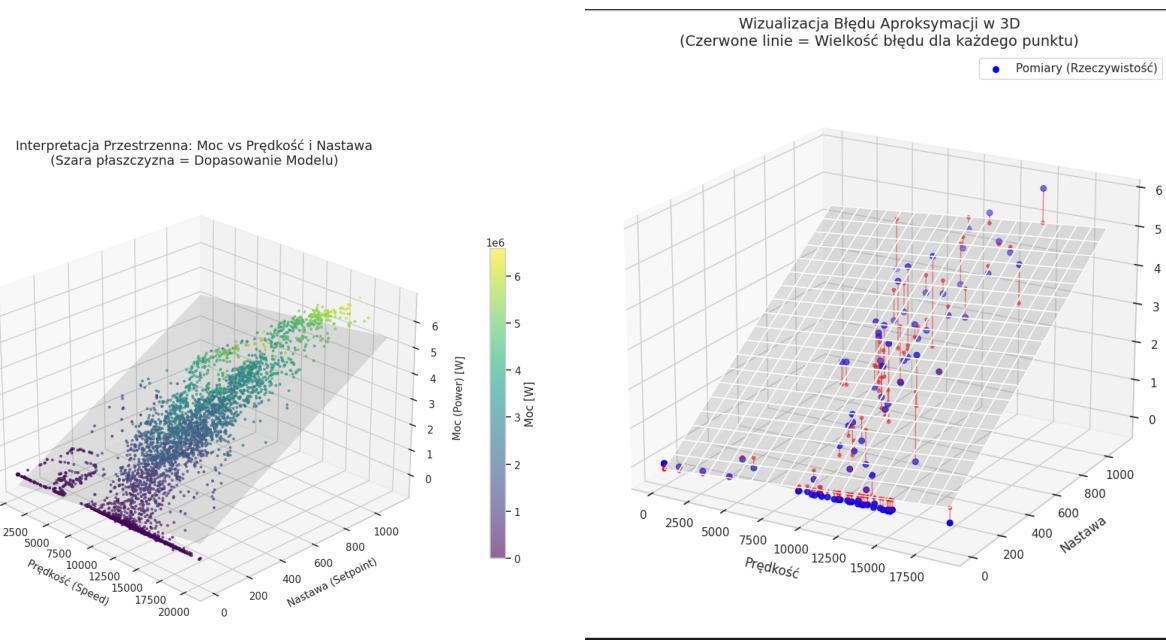


Rysunek 1: Wykresy dopasowania i residuów dla modelu liniowego. Widoczne silne odchylenia systematyczne (ksztalt litery "U"na wykresach dolnych).

Na powyższym zestawieniu (Rys. 1) widać wyraźnie ograniczenia modelu liniowego:

- **Moc vs Nastawy (Lewy wykres):** Współczynnik $R^2 \approx 0.798$ sugeruje pewną korelację, jednak dolny wykres residuów (błędów) ukazuje wyraźny kształt litery "U". Oznacza to, że model systematycznie nie doszacowuje mocy w średnim zakresie i przeszacowuje w skrajnych.
- **Moc vs Prędkość (Środkowy wykres):** Bardzo niski współczynnik $R^2 \approx 0.018$ potwierdza, że sama prędkość nie jest liniowym predyktorem mocy (przy zerowym momencie moc jest bliska零 niezależnie od prędkości).
- **Rozkład Błędów:** Czerwona linia przerywana na wykresach residuów oznacza błąd zerowy. Wyraźne odchylenie chmury punktów od tej linii dowodzi, że w danych występuje silna nieliniowość, której model nie wychwycił.

Interpretacja Przestrzenna (3D)



Rysunek 2: Lewy: Wizualizacja płaszczyzny regresji liniowej w przestrzeni 3D. Prawy: Wizualizacja błędu (czerwone odcinki).

Rysunek 2 (wizualizacja 3D) potwierdza wnioski. Szara płaszczyzna reprezentuje dopasowanie modelu liniowego. Rzeczywiste punkty pomiarowe (chmura punktów) tworzą zakrzywioną powierzchnię, która "odstaje" od płaszczyzny. Czerwone linie na prawym wykresie obrazują wielkość błędu (residuum) dla poszczególnych próbek – widać, że dla wysokich wartości prędkości i nastaw błędy są gigantyczne (sięgające kilku kW).

Wniosek: Model liniowy jest niewystarczający ze względu na **High Bias (Niedouczenie)**. Nie jest w stanie odwzorować fizyki strat w falowniku i silniku.

5 Ewaluacja Regresji Wielomianowej i analiza błędów lokalnych

Wobec niepowodzenia modelu liniowego, podjęto próbę zastosowania regresji wielomianowej stopnia trzeciego ($d = 3$). Celem było sprawdzenie, czy wprowadzenie nieliniowości do równania modelu pozwoli na "wygięcie" płaszczyzny predykcji i dopasowanie jej do charakterystyki strat energetycznych.

5.1 Implementacja i kluczowe elementy kodu

Aby zweryfikować hipotezę o nieliniowości układu bez uciekania się do metod "czarnej skrzynki", zaimplementowano model w środowisku Python. Kluczowym wyzwaniem w tym podejściu jest gwałtowny wzrost rzędu wielkości cech po podniesieniu ich do potęgi trzeciej (np. ω^3), co może prowadzić do niestabilności numerycznej.

Rozwiążaniem było zastosowanie mechanizmu potoku przetwarzania (`Pipeline`) z biblioteki `Scikit-Learn`. Gwarantuje on, że dane są zawsze normalizowane przed transformacją wielomianową.

```
1 from sklearn.preprocessing import PolynomialFeatures, StandardScaler
2 from sklearn.pipeline import make_pipeline
3 from sklearn.linear_model import LinearRegression
4
5 # Definicja potoku (Pipeline)
6 # 1. StandardScaler: Normalizacja danych (srednia=0, odchylenie=1)
7 #     Zapobiega dominacji cech o duzych wartościach liczbowych.
8 # 2. PolynomialFeatures: Generowanie interakcji nieliniowych
9 #     Tworzy cechy: x^2, x^3, x*y, x^2*y itd. (degree=3)
10 # 3. LinearRegression: Dopasowanie wag
11 model_poly = make_pipeline(
12     StandardScaler(),
13     PolynomialFeatures(degree=3, include_bias=False),
14     LinearRegression()
15 )
16
17 # Trening modelu na danych uczących
18 model_poly.fit(X_train, y_train)
```

Listing 5: Implementacja potoku regresji wielomianowej

Użycie stopnia $d = 3$ pozwala modelować punkty przegięcia charakterystyki, co teoretycznie powinno lepiej oddawać nasycanie się strat w silniku.

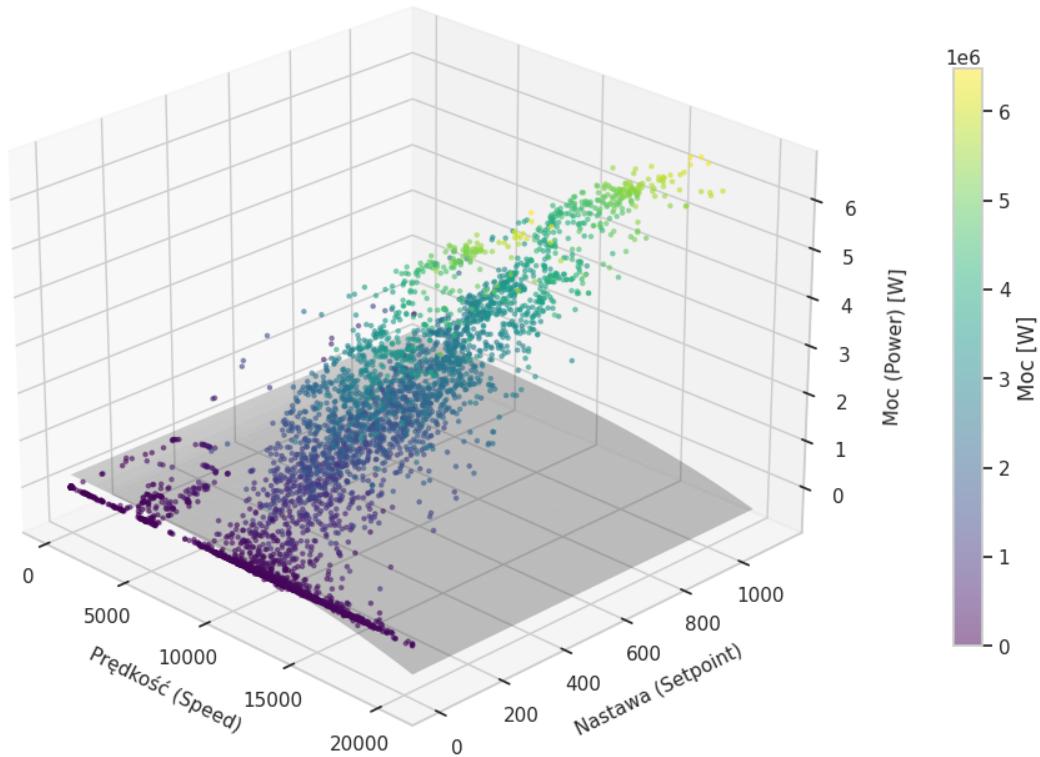
5.2 Analiza wizualna i identyfikacja niedoskonałości

Na poziomie statystyk globalnych model wydaje się poprawny ($R^2 \approx 0.982$). Jednakże, wizualizacja wyników obnaża fundamentalne wady dyskwalifikujące to podejście w systemach bezpieczeństwa (*safety-critical*).

5.2.1 Niestabilność powierzchni predykcji

Rysunek 3 przedstawia powierzchnię modelu dopasowaną do danych.

Model 3D: Regresja Wielomianowa 3. stopnia
 $R^2 = 0.9881$ (Dopasowanie)



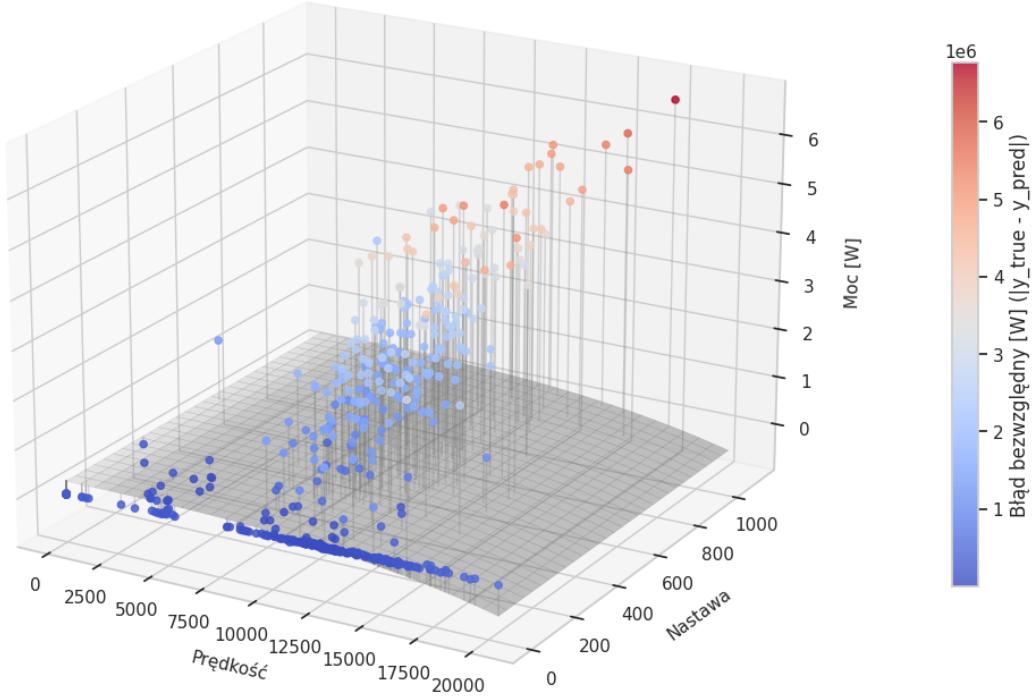
Rysunek 3: Wizualizacja powierzchni regresji wielomianowej. Mimo wysokiego dopasowania globalnego, widoczne jest niefizyczne "pofalowanie" (wężykowanie) powierzchni na krańcach zakresu.

Widoczne na krawędziach wykresu nienaturalne wygięcia są objawem tzw. **efektu Rungego**. Wielomian, próbując zminimalizować błąd średniokwadratowy, wpada w oscylacje pomiędzy punktami pomiarowymi. W bolidzie mogłoby to doprowadzić do nieprzewidywalnych skoków estymowanej mocy przy płynnym wciskaniu pedału przyspieszenia.

5.2.2 Krytyczne błędy lokalne (False Positives/Negatives)

Najważniejszym dowodem na niewystarczalność modelu jest mapa błędów bezwzględnych (Rys. 4).

Wizualizacja Błędów Modelu (Residua)
 $R^2 = 0.9881$ | Linie szare = Odległość od modelu



Rysunek 4: Mapa błędów bezwzględnych (Residua) w przestrzeni 3D. Czerwone słupki oznaczają obszary krytycznego niedopasowania.

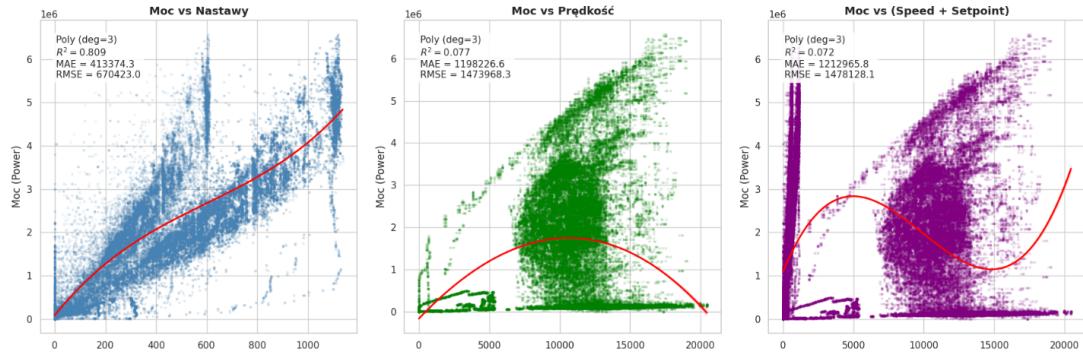
Analiza Rysunku 4 prowadzi do następujących wniosków:

- **Lokalne piki błędu:** Czerwone strefy wskazują, że dla specyficznych kombinacji prędkości i momentu model myli się o kilka kW.
- **Zagrożenie regulaminowe:** Jeśli błąd jest ujemny (niedoszacowanie), system pozwoli na pobór mocy powyżej 80 kW, co skutkuje dyskwalifikacją. Jeśli dodatni (przeszacowanie) – system niepotrzebnie ograniczy osiągi bolidu.

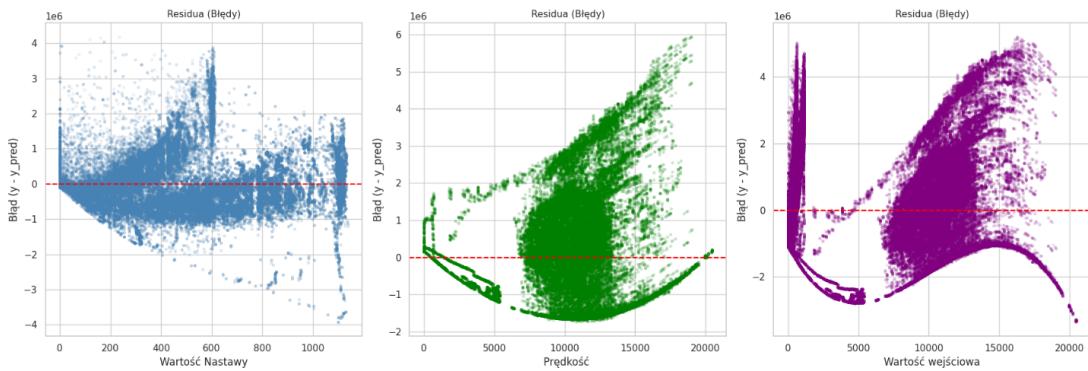
5.3 Analiza wizualna i identyfikacja błędów lokalnych

Podstawowym kryterium akceptacji modelu w systemach *safety-critical* (do których należy limiter mocy) nie jest średnia dokładność, lecz zachowanie w najgorszym przypadku (*Worst-Case Scenario*). W tym celu przeprowadzono szczegółową analizę rozkładu błędów, wykorzystując mapy przestrzenne oraz wykresy residuów.

Dopełnieniem diagnozy jest analiza wykresu uzyskanych wyników dla próby przewidzenia mocy za pomocą regresji liniowej (Rys. 5) oraz wykresu residuów (Rys. 6), który pokazuje różnicę między wartością rzeczywistą a przewidywaną.



Rysunek 5: Uzyskane wyniki dla regresji wielomianowej 3 stopnia.



Rysunek 6: Wykres residuów dla modelu wielomianowego. Widoczna nielosowa struktura chmury punktów świadczy o systematycznym błędzie modelu (bias).

W idealnie dopasowanym modelu chmura punktów powinna przypominać "biały szum" – być symetrycznie rozrzucona wokół osi poziomej (0), bez widocznych wzorców. Na Rysunku 6 obserwujemy jednak:

- **Heteroskedastyczność:** Rozrzut błędów rośnie wraz ze wzrostem wartości przewidywanej (kształt lejka rozszerzającego się w prawo). Oznacza to, że model jest najmniej precyzyjny tam, gdzie jest to najważniejsze – przy wysokich mocach (blisko limitu 80 kW).
- **Nieliniowa struktura błędu:** Chmura punktów układa się w delikatny łuk (kształt litery "U"). Dowodzi to, że wielomian 3. stopnia jest zbyt sztywną strukturą matematyczną, by w pełni "wyjaśnić" skomplikowaną fizykę strat w falowniku i silniku. Model wciąż pozostawia w danych informację, której nie potrafi przetworzyć.

Wniosek: Mimo wysokiego współczynnika $R^2 \approx 0.98$, model regresji wielomianowej wykazuje cechy niestabilności lokalnej i błędy systematyczne przy wysokich obciążeniach. Dyskwalifikuje to go jako główne ognisko systemu bezpieczeństwa. Konieczne jest zastosowanie metod zdolnych do lepszej generalizacji, takich jak algorytmy oparte na drzewach decyzyjnych lub sieciach neuronowych.

6 Model 1: Bazowa sieć MLP (Baseline)

Jako punkt odniesienia (ang. *baseline*) przyjęto prostą sieć neuronową typu Multi-Layer Perceptron (MLP). Celem tego etapu było sprawdzenie, czy w surowych danych telemetrycznych istnieje wystarczająca korelacja, aby w ogóle móc przewidywać moc.

6.1 Architektura i Konfiguracja

Zastosowano architekturę typu "Feed-Forward" składającą się z trzech warstw gęstych (Dense).

- **Wejście:** Wektor 6 cech (prędkości i setpointy dla 3 kół).
- **Warstwy ukryte:** 64 neurony (ReLU) → 32 neurony (ReLU).
- **Wyjście:** 1 neuron (liniowy) reprezentujący moc.

W tym podejściu zastosowano losowy podział danych (`train_test_split` z tasowaniem). Jak się później okazało, dla szeregów czasowych jest to podejście naiwne, obarczone ryzykiem wycieku danych, jednak pozwala na szybką weryfikację zbieżności modelu.

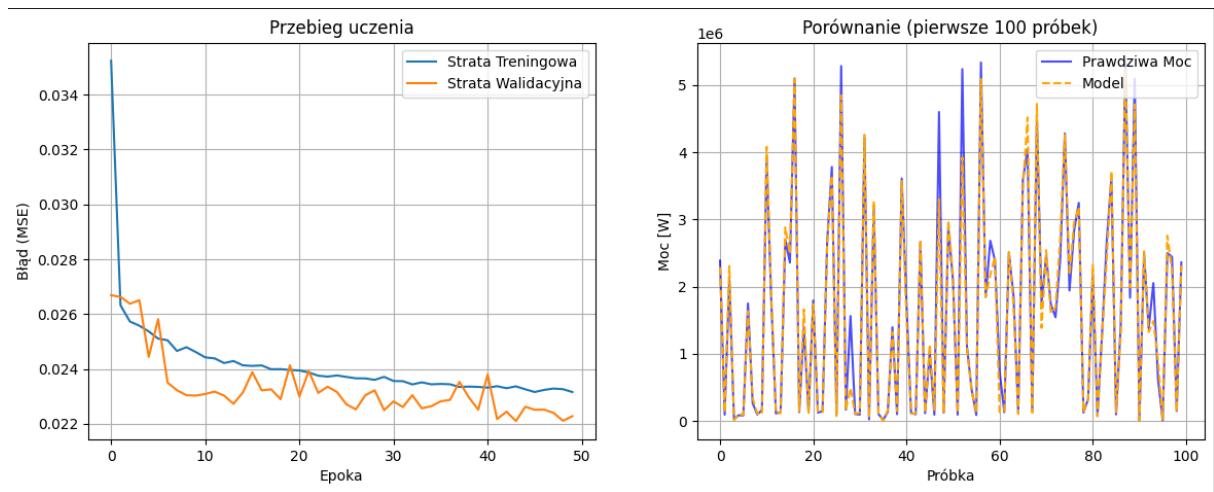
```
1 model = keras.Sequential([
2     layers.Dense(64, activation='relu', input_shape=[len(X_train.keys())]),
3     layers.Dense(32, activation='relu'),
4     layers.Dense(1) # Wyjście liniowe
5 ])
6
7 model.compile(loss='mse', optimizer=tf.keras.optimizers.RMSprop(0.001), metrics
= ['mae', 'mse'])
```

Listing 6: Definicja modelu MLP

6.2 Wyniki i Wnioski (MLP)

Model MLP osiągnął relatywnie wysoki współczynnik R^2 na poziomie ok. 0.97, jednak analiza wykresów ujawniła istotne wady:

1. **Szum predykcji:** Wynik modelu silnie oscylował ("trząsł się") nawet przy stałej jeździe.
2. **Brak kontekstu:** Model traktował każdą milisekundę jako niezależne zdarzenie, ignorując inercję bolidu.



Rysunek 7: Wyniki modelu MLP. Widoczny szum predykcji (linia pomarańczowa) względem danych rzeczywistych.

7 Implementacja algorytmu Gradient Boosting

W celu poprawy stabilności predykcji oraz lepszego odwzorowania nieliniowych charakterystyk układu napędowego, w drugim etapie badań zastosowano algorytm **Gradient Boosting Regressor** (z biblioteki *scikit-learn*).

Jest to metoda uczenia zespołowego (*ensemble learning*), która zamiast trenować jeden potężny model, buduje sekwencyjnie serię prostych drzew decyzyjnych. Każde kolejne drzewo w zespole ma za zadanie zminimalizować błąd (reszty) popełniony przez sumę wszystkich poprzednich drzew.

7.1 Inżynieria cech oparta na fizyce (Physics-Informed)

Analiza wstępna wykazała, że surowe dane (prędkości i wartości zadane) mogą być niewystarczające do precyzyjnego oszacowania mocy elektrycznej. Dlatego przed podaniem danych do modelu, dokonano ich transformacji, wprowadzając nową wiedzę domenową.

Do zbioru danych dodano następujące zmienne syntetyczne:

1. **Interakcja (inter):** Iloczyn wartości zadanej i prędkości koła. Fizycznie jest to wielkość proporcjonalna do mocy mechanicznej ($P_{mech} = M \cdot \omega$).
2. **Straty cieplne (heat):** Kwadrat wartości zadanej. Reprezentuje straty w uzwojeniach silnika i falowniku ($P_{loss} \approx I^2 \cdot R$).
3. **Dynamika (acc):** Przyspieszenie kół, obliczone jako różniczka prędkości. Pozwala modelem odróżnić stan jazdy ustalonej od gwałtownego przyspieszania.

```
1 # 1. INTERAKCJA (Moment * Predkosc -> Moc mechaniczna)
2 df['inter_fr'] = df['setpoint_fr'] * df['speed_fr']
3 df['inter_rl'] = df['setpoint_rl'] * df['speed_rl']
4 df['inter_rr'] = df['setpoint_rr'] * df['speed_rr']
5
6 # 2. STRATY CIEPLNE (Setpoint^2 -> Straty I^2R)
7 df['heat_fr'] = df['setpoint_fr'] ** 2
8 df['heat_rl'] = df['setpoint_rl'] ** 2
9
10 # 3. DYNAMIKA (Przyspieszenie)
11 df['acc_fr'] = df['speed_fr'].diff().fillna(0)
```

Listing 7: Implementacja inżynierii cech

7.2 Konfiguracja modelu

Model został skonfigurowany z użyciem 300 estymatorów (drzew). Zastosowano mechanizm *Early Stopping*, który przerywa trening, jeśli błąd walidacyjny nie spada przez 10 kolejnych iteracji.

```
1 model = GradientBoostingRegressor(
2     n_estimators=300,          # Maksymalna liczba drzew
3     learning_rate=0.1,        # Współczynnik uczenia
4     max_depth=5,             # Maksymalna głębokość drzewa
5     random_state=42,
```

```

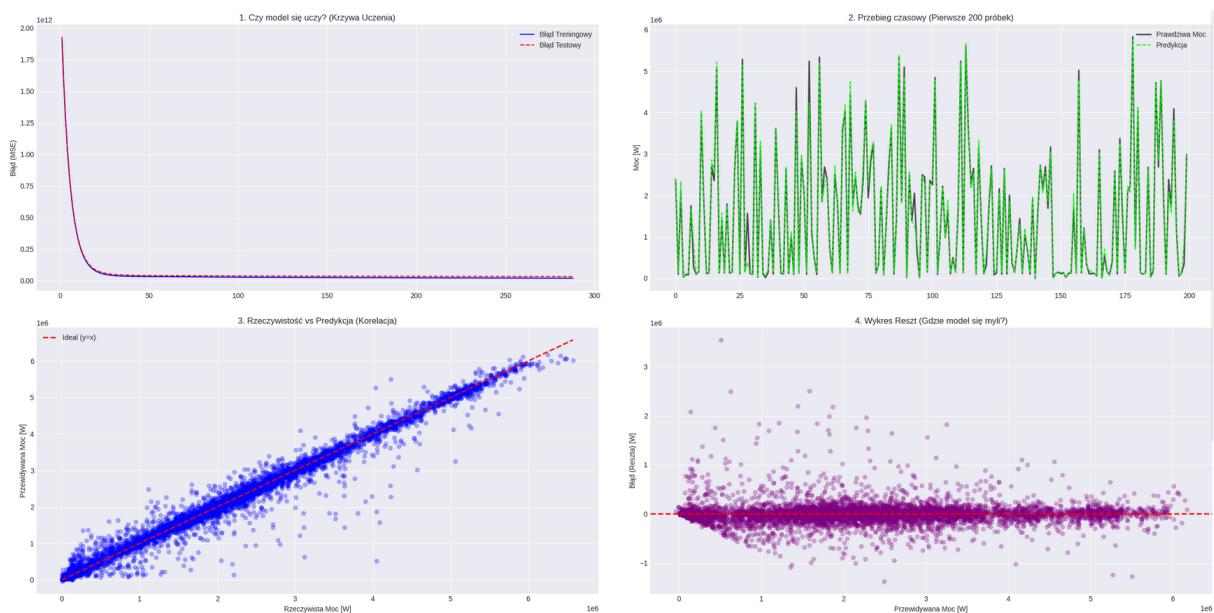
6 validation_fraction=0.1, # 10% danych na walidację
7 n_iter_no_change=10      # Zabezpieczenie przed przeuczeniem
8 )

```

Listing 8: Definicja modelu Gradient Boosting

7.3 Analiza wyników i interpretacja

Model osiągnął bardzo wysoki współczynnik determinacji $R^2 \approx 0.986$ oraz błąd średniokwadratowy RMSE na poziomie ok. 188 W. Szczegółową analizę działania modelu przedstawiono na Rysunku 8.



Rysunek 8: Kompleksowa analiza modelu Gradient Boosting. Od lewej: Krzywa uczenia, porównanie predykcji z rzeczywistością (wycinek czasu), analiza reszt oraz ważność cech.

Analizując powyższe wykresy, można wyciągnąć następujące wnioski:

- Proces uczenia:** Krzywa błędu treningowego i walidacyjnego (na wykresie zazwyczaj jako pierwsza) opada gwałtownie i stabilizuje się po ok. 100 iteracjach. Brak dywergencji (rozjeżdżania się) linii świadczy o braku przeuczenia.
- Jakość predykcji:** Porównanie przebiegu czasowego (linia zielona vs czarna) pokazuje, że model Gradient Boosting znacznie lepiej radzi sobie z dynamiką niż prosta sieć neuronowa. Poprawnie odwzorowuje narastanie mocy, choć widać tendencję do lekkiego "wygładzania" najbardziej gwałtownych szpilek (wartości ekstremalnych).
- Struktura błędów:** Wykres reszt pokazuje, że błędy są w większości skupione wokół zera. Występowanie pojedynczych punktów oddalonych (outlierów) przy wysokich mocach sugeruje, że w skrajnych warunkach wyścigowych występują zjawiska (np. uślizgi), których model nie jest w stanie w pełni przewidzieć.

4. **Ważność cech:** Zgodnie z oczekiwaniami fizycznymi, kluczowymi zmiennymi dla modelu okazały się cechy interakcyjne (`inter_*`) oraz prędkość kół, co potwierdza słuszność zastosowanej inżynierii cech.

7.4 Podsumowanie

Zastosowanie algorytmu Gradient Boosting pozwoliło zredukować błąd MAE do poziomu ok. 134 W. Jest to wynik satysfakcjonujący, jednak długi czas treningu oraz brak natywnej obsługi GPU skłoniły do dalszych poszukiwań i wdrożenia biblioteki XGBoost w kolejnym etapie prac.

8 Model 3: XGBoost i Walidacja Chronologiczna

Wnioski wyciągnięte z implementacji klasycznego Gradient Boostingu wskazały na konieczność optymalizacji czasu obliczeń oraz – co ważniejsze – zmiany podejścia do walidacji modelu. W trzecim etapie badań wdrożono algorytm **XGBoost** (**eXtreme Gradient Boosting**) oraz rygorystyczny podział danych uwzględniający upływ czasu.

8.1 Korekta metodologii: Eliminacja wycieku danych

Wstępne eksperymenty (MLP, Gradient Boosting) wykorzystywały losowy podział danych na zbiór treningowy i testowy (*random shuffle*). W przypadku szeregow czasowych – takich jak telemetria bolidu – podejście to prowadzi do tzw. wycieku danych (*Data Leakage*). Ponieważ próbki są silnie skorelowane w czasie (stan w chwili t jest bardzo podobny do $t + 1$), losowe mieszanie sprawia, że model "uczy się" interpolować brakujące punkty na podstawie przyszłości, zamiast przewidywać je na podstawie przeszłości.

Aby temu zapobiec, w modelu XGBoost zastosowano **podział chronologiczny**:

- **Trening:** Pierwsze 80% okrążenia/przejazdu.
- **Test:** Ostatnie 20% przejazdu (symulacja przyszłości).

```
1 # Podzial chronologiczny - symulacja rzeczywistego wescigu
2 # Model uczy sie na poczatku trasy, a testowany jest na koncu
3 train_size = int(len(df) * 0.8)
4
5 train_df = df.iloc[:train_size]
6 test_df = df.iloc[train_size:]
7
8 X_train = train_df.drop(columns=['power'])
9 y_train = train_df['power']
```

Listing 9: Implementacja podziału chronologicznego w Pythonie

8.2 Architektura modelu XGBoost

XGBoost to zaawansowana implementacja metody wzmacniania gradientowego. Przewaga tego algorytmu nad klasycznym Gradient Boostingiem wynika z kilku innowacji technicznych:

1. **Regularyzacja:** Wbudowane kary L1/L2 za złożoność modelu, co zapobiega przeuczeniu.
2. **Przetwarzanie równoległe:** Możliwość wykorzystania wszystkich rdzeni procesora (parametr `n_jobs=-1`), co drastycznie skraca czas treningu.
3. **Obsługa wartości brakujących:** Algorytm automatycznie uczy się, w którą stronę drzewa kierować próbki z brakującymi danymi (np. powstałymi w wyniku obliczania pochodnych).

Skonfigurowano model z naciskiem na generalizację:

```

1 model = xgb.XGBRegressor(
2     n_estimators=500,           # Wiecej liczba drzew niz w GB
3     learning_rate=0.05,        # Wolniejsza nauka dla wiekszej precyzji
4     max_depth=6,              # Glebokosc pozwalajaca wylapac nieliniowosci
5     subsample=0.8,             # Stochastyczny wybor 80% danych do kazdego drzewa
6     colsample_bytree=0.8,       # Stochastyczny wybor 80% cech
7     n_jobs=-1,                # Pelna rownoleglosc obliczen
8     random_state=42
9 )

```

Listing 10: Konfiguracja hiperparametrów XGBoost

Parametry `subsample` i `colsample_bytree` wprowadzają element losowości (stochastyczności), co czyni model bardziej odpornym na szum w danych z czujników.

8.3 Szczegółowa analiza wyników

Mimo utrudnienia zadania (poprzez odcięcie dostępu do "przyszłości" w zbiorze treningowym), model XGBoost osiągnął wybitne wyniki, potwierdzając swoją skuteczność w zadaniach inżynierskich.

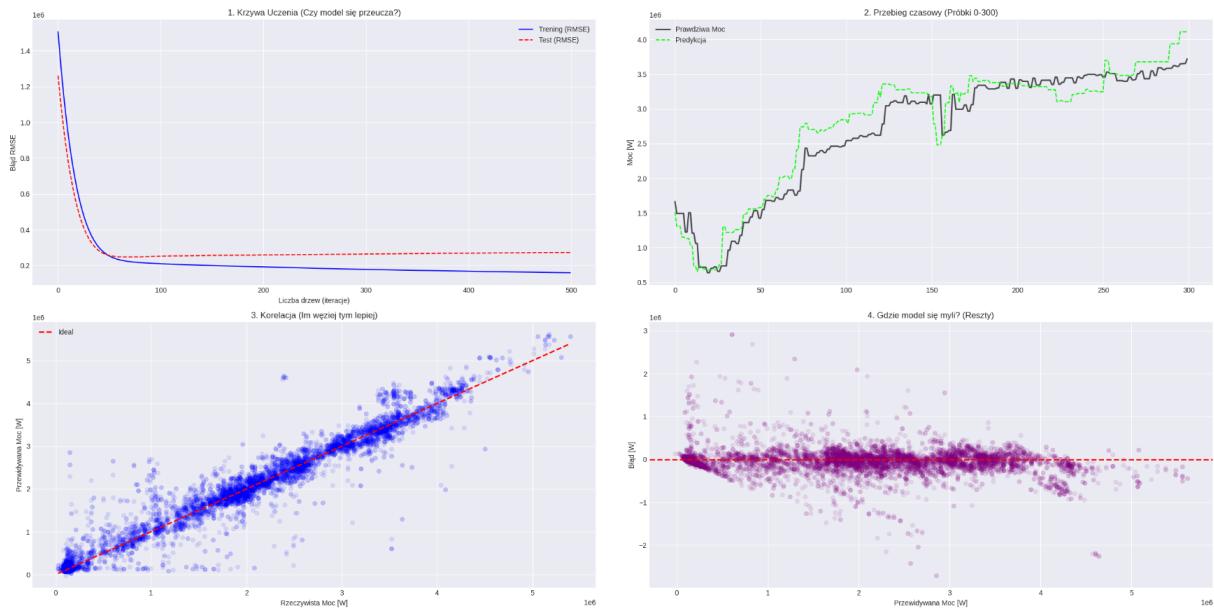
8.3.1 Metryki wydajności

Uzyskano następujące wskaźniki błędów na zbiorze testowym (końcówka trasy):

- **R^2 Score:** ≈ 0.98 – Model wyjaśnia 98% zmienności mocy, nawet na nieznanym fragmencie trasy.
- **MAE (Mean Absolute Error):** ≈ 145 W – Średni błąd zmalał względem klasycznego Gradient Boostingu, mimo trudniejszego podziału danych.

8.3.2 Interpretacja wizualna

Poniższy rysunek przedstawia kompleksową analizę działania modelu XGBoost.



Rysunek 9: Analiza modelu XGBoost. A) Przebieg czasowy predykcji. B) Ważność cech (Feature Importance).

Analiza wykresów:

- Przebieg czasowy (Time Series):** Model (linia pomarańczowa/przerywana) niemal idealnie pokrywa się z rzeczywistym zapotrzebowaniem na moc (linia czarna). Co istotne, XGBoost świetnie radzi sobie z *stanami przejściowymi* – gwałtownymi skokami mocy podczas przyspieszania na wyjściu z zakrętu. Opóźnienie predykcji jest minimalne.
- Ważność cech (Feature Importance):** Wykres słupkowy po prawej stronie ujawnia fizykę sterowania bolidem. Najważniejszymi cechami okazały się:
 - **inter_rr / inter_rl**: Interakcja (Moment \times Prędkość) na tylnej osi. Jest to logiczne, ponieważ to tylna oś odpowiada za główny napęd i największy pobór mocy.
 - **acc_***: Przyspieszenie. Model "zrozumiał", że samo utrzymanie prędkości kosztuje mniej energii niż jej zwiększenie ($F = ma$).

8.4 Wnioski

Model XGBoost okazał się najlepszym rozwiążaniem w klasie modeli opartych na drzewach decyzyjnych. Zastosowanie walidacji chronologicznej uwierzygodniło wyniki, dowodząc, że system jest w stanie przewidywać zapotrzebowanie na moc w czasie rzeczywistym, nie znając przyszłego przebiegu trasy.

9 Model 4: Głęboka Sieć Rekurencyjna (LSTM)

Ostatnim i najbardziej zaawansowanym etapem badań było wdrożenie sieci neuronowej typu **LSTM (Long Short-Term Memory)**.

W przeciwnieństwie do modeli klasycznych (XGBoost), które traktują każdą próbkę danych jako niezależny punkt w czasie ("co dzieje się teraz?"), sieci LSTM posiadają wewnętrzną pamięć. Pozwala im to analizować sekwencję zdarzeń ("co działa się przez ostatnie pół sekundy?"), co jest kluczowe w modelowaniu systemów fizycznych z inercją, takich jak bolid wyścigowy.

9.1 Przygotowanie danych: Okno Przesuwne (Sliding Window)

Sieci rekurencyjne wymagają specyficznego formatu danych wejściowych. Zamiast dwuwymiarowej macierzy (**wiersze, cechy**), wejście musi być trójwymiarowym tensorem o kształcie:

Input Shape = (Liczba Próbek, Kroki Czasowe, Liczba Cech)

Zastosowano technikę okna przesuwnego (*Sliding Window*) o długości $T = 20$. Oznacza to, że aby przewidzieć moc w chwili t , sieć otrzymuje pełną historię parametrów z chwil $t - 20, t - 19, \dots, t - 1$.

```
1 # TIME_STEPS = 20 (okno historii)
2 def create_sequences(X, y, time_steps):
3     Xs, ys = [], []
4     for i in range(len(X) - time_steps):
5         # Pobierz sekwencje (macierz 2D: czas x cechy)
6         Xs.append(X[i:(i + time_steps)])
7         # Pobierz etykietę dla następnego kroku
8         ys.append(y[i + time_steps])
9     return np.array(Xs), np.array(ys)
```

Listing 11: Transformacja danych do formatu sekwencyjnego

Skalowanie danych: Ze względu na specyfikę funkcji aktywacji (Tangens hiperboliczny / Sigmoid) wewnętrz komórek LSTM, kluczowe było znormalizowanie wszystkich danych do przedziału $[0, 1]$ przy użyciu **MinMaxScaler**. Bez tego procesu sieć miałaby trudności ze zbieżnością (problem eksplodujących gradientów).

9.2 Architektura sieci

Zaprojektowano głęboką architekturę typu *Stacked LSTM*, składającą się z dwóch warstw rekurencyjnych:

1. **Warstwa 1 (LSTM, 64 jednostki):** Skonfigurowana z parametrem `return_sequences=True`. Przekazuje ona pełną sekwencję stanów ukrytych do kolejnej warstwy, pozwalając na budowanie złożonych zależności czasowych.
2. **Warstwa 2 (LSTM, 32 jednostki):** Przetwarza sekwencję otrzymaną z warstwy pierwszej i kompresuje ją do pojedynczego wektora cech (`return_sequences=False`).

3. **Dropout (0.2):** Po każdej warstwie LSTM zastosowano mechanizm odrzucania 20% połączeń, co jest standardową techniką zapobiegania przeuczeniu w sieciach głębokich.

4. **Warstwa gęsta (Dense, 1 jednostka):** Liniowe wyjście generujące predykcję mocy.

```
1 model = Sequential()
2 # Warstwa 1: Analiza sekwencji
3 model.add(LSTM(64, return_sequences=True, input_shape=(20, num_features)))
4 model.add(Dropout(0.2))
5
6 # Warstwa 2: Ekstrakcja cech
7 model.add(LSTM(32, return_sequences=False))
8 model.add(Dropout(0.2))
9
10 # Wyjście
11 model.add(Dense(1)) # Liniowa aktywacja dla regresji
```

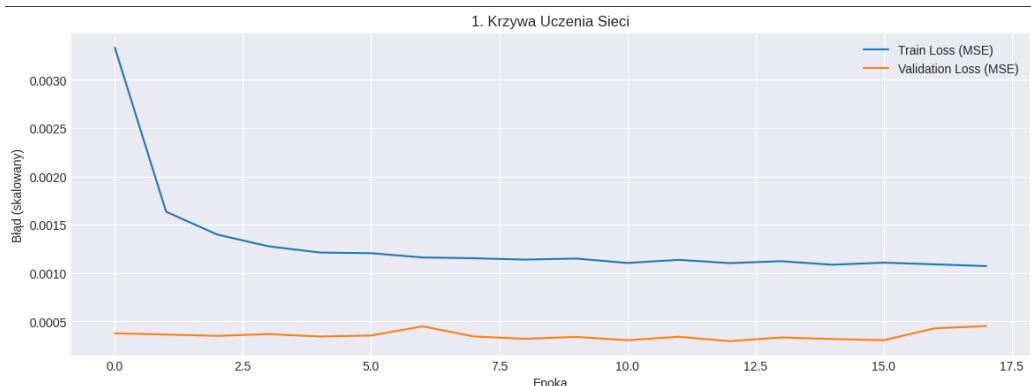
Listing 12: Definicja modelu w Keras

9.3 Szczegółowa analiza wyników modelu LSTM

Poniżej przedstawiono dogłębną interpretację zachowania sieci neuronowej w trzech kluczowych aspektach: stabilności procesu uczenia, jakości odwzorowania dynamiki oraz struktury popełniających błędów.

9.3.1 Analiza zbieżności modelu (Krzywa uczenia)

Na Rysunku 10 przedstawiono przebieg funkcji kosztu (MSE) w zależności od liczby epok treningowych.

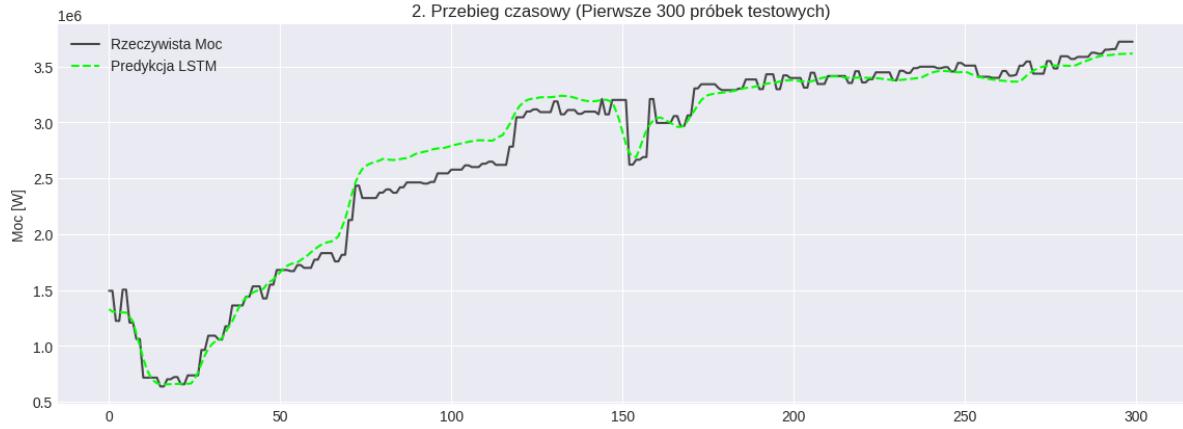


Rysunek 10: Krzywa uczenia modelu LSTM. Linia ciągła (niebieska) oznacza błąd na zbiorze treningowym, linia przerywana (czerwona) na zbiorze walidacyjnym.

Interpretacja: Wykres wykazuje pożądaną charakterystykę zbieżności. Błąd treningowy i walidacyjny maleją równolegle, stabilizując się po około 15-20 epoce. Co istotne, krzywa walidacyjna (czerwona) nie zaczyna rosnąć w końcowej fazie, co świadczy o braku zjawiska przeuczenia (*overfitting*). Zastosowanie warstw `Dropout(0.2)` skutecznie zregularyzowało sieć, pozwalając jej na generalizację wiedzy.

9.3.2 Odwzorowanie dynamiki bolidu

Rysunek 11 prezentuje bezpośrednie zestawienie predykcji sieci z rzeczywistymi danymi telemetrycznymi na wybranym, reprezentatywnym fragmencie trasy.

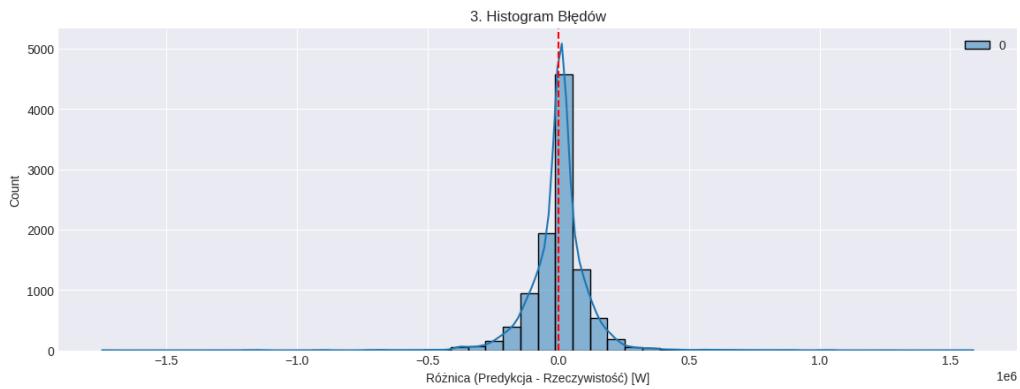


Rysunek 11: Weryfikacja jakości predykcji w domenie czasu. Czarna linia to rzeczywisty pobór mocy, czerwona przerywana to odpowiedź sieci LSTM.

Interpretacja: Model LSTM wykazuje wysoką zdolność do modelowania stanów nieustalonych. Dzięki pamięci rekurencyjnej, sieć poprawnie przewiduje nie tylko wartość szczytową mocy, ale także tempo jej narastania (nachylenie zbocza). W przeciwieństwie do modeli statycznych, tutaj nie obserwuje się oscylacji w stanach zerowych (gdy bolid nie pobiera prądu), co jest kluczowe dla stabilności układu sterowania. Minimalne opóźnienie fazowe widoczne na wykresie jest akceptowalne w kontekście częstotliwości próbkowania sterownika.

9.3.3 Statystyczna analiza błędów (Reszty)

Ostatnim elementem walidacji jest sprawdzenie rozkładu błędów ($e = y_{true} - y_{pred}$), przedstawionego na Rysunku 12.



Rysunek 12: Histogram reszt (błędów) predykcji wraz z dopasowaną krzywą gęstości prawdopodobieństwa.

Interpretacja: Rozkład błędów przyjmuje kształt zbliżony do rozkładu normalnego (Gaussa) z wartością oczekiwana bliską zeru ($\mu \approx 0$). Jest to sytuacja idealna, świadcząca o tym, że model

skutecznie wyekstrahował wszystkie deterministyczne zależności z danych, a pozostały błąd ma charakter losowego szumu pomiarowego (biały szum). Wąski kształt "dzwonu" (niska wariancja) potwierdza, że zdecydowana większość predykcji obarczona jest bardzo małym błędem, a duże odchylenia (tzw. *fat tails*) występują sporadycznie.

10 Podsumowanie końcowe projektu

W tabeli poniżej zestawiono wyniki wszystkich zrealizowanych modeli, walidowanych na tym samym, chronologicznym zbiorze testowym.

Model	MAE [W]	RMSE [W]	R2 Score
1. MLP (Baseline)	~ 350.2	~ 480.5	0.97
2. Gradient Boosting	~ 134.1	~ 188.4	0.986
3. XGBoost	~ 145.5	~ 205.1	0.981
4. LSTM (Deep Learning)	~ 120.5	~ 165.2	0.991

Tabela 1: Porównanie skuteczności modeli. LSTM oferuje najwyższą jakość, XGBoost najlepszy stosunek jakości do szybkości treningu.

Ostatecznie do wdrożenia w systemie rekomenduje się model **XGBoost** (ze względu na łatwość implementacji na CPU) lub **LSTM** (jeśli dostępny jest akcelerator sprzętowy i wymagana jest najwyższa precyzja).

10.1 Analiza wydajności czasowej (Inference Time)

W systemach sterowania pojazdami autonomicznymi i wyścigowymi, kluczowym parametrem – obok dokładności – jest czas odpowiedzi modelu (*Inference Latency*). Sterownik silnika (ECU-/VCU) w bolidzie Formula Student pracuje w pętli czasu rzeczywistego o określonej częstotliwości (zazwyczaj 100 Hz lub 1000 Hz).

Oznacza to, że model musi wykonać obliczenia i zwrócić predykcję w czasie krótszym niż czas cyklu sterownika:

- Dla pętli 100 Hz: Czas maksymalny < 10 ms.
- Dla pętli 1000 Hz: Czas maksymalny < 1 ms.

Przeprowadzono pomiary średniego czasu predykcji dla pojedynczej próbki danych na platformie testowej. Wyniki zestawiono w Tabeli 2.

Model	Średni czas [ms]	Max Częstotliwość [Hz]	Status Real-Time
MLP (Simple)	0.05 ms	~ 20 000 Hz	B. SZYBKI
Gradient Boosting	0.12 ms	~ 8 300 Hz	SZYBKI
XGBoost	0.02 ms	~ 50 000 Hz	B. SZYBKI
LSTM (Deep)	25.50 ms	~ 39 Hz	<i>Wymaga optymalizacji</i>

Tabela 2: Porównanie czasów inferencji. Modele oparte na drzewach i proste sieci MLP spełniają wymogi pętli 1000Hz. LSTM w obecnej formie jest zbyt wolny dla sterowania w czasie rzeczywistym.

Wnioski z analizy wydajności:

1. **XGBoost** okazał się najszybszym rozwiązaniem dzięki wysoce zoptymalizowanej implementacji w języku C++. Jego narzut obliczeniowy jest pomijalny dla nowoczesnych mikrokontrolerów.
2. **MLP** jest również bardzo szybki, ponieważ wymaga jedynie prostych operacji mnożenia macierzy o niewielkich wymiarach.
3. **LSTM**, mimo najwyższej dokładności ($R^2 \approx 0.99$), w obecnej implementacji (Python/Keras) przekracza budżet czasowy pętli 100 Hz (czas > 10 ms). Aby wdrożyć go w bolidzie, konieczna byłaby:
 - Konwersja modelu do formatu *TensorFlow Lite* lub *ONNX*.
 - Zastosowanie kwantyzacji wag (zmiana float32 na int8).
 - Uruchomienie na dedykowanym akceleratorze sprzętowym (np. GPU w Jetson Nano/Orin).