МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ПАЛИВНО-МАСТИЛЬНИХ МАТЕРІАЛІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Інтелектуальні інформаційні системи»

Здобувач _____ Станіслав БЕЗОЩУК

«___»_____ 2025 р.

Керівник д-р техн. наук, професор _____ Ірина КАЛІНІНА

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Безощук Станіслав Сергійович
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Інтелектуальна система моніторингу паливно-мастильних матеріалів.

Керівник роботи: Калініна Ірина Олександрівна
професор кафедри ІІС, д-р техн. наук, професор,
(прізвище, ім’я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від «07» липня 2025 р. № 184.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні:
Програмний комплекс для інтелектуального моніторингу ПММ, що включає модель прогнозування витрат та модуль виявлення аномалій у споживанні палива.

4. Перелік питань, що підлягають розробці:

- аналіз сучасного стану проблеми моніторингу ПММ та огляд існуючих інтелектуальних рішень;

– дослідження та обґрунтування вибору методів машинного навчання для побудови нормативної моделі витрат палива;

– проведення експериментальних досліджень, навчання моделей та розробка алгоритму виявлення аномалій;

– проектування архітектури та програмна реалізація інтелектуальної системи з використанням мікросервісного підходу.

5. Перелік графічних матеріалів: презентація

Керівник роботи _____ _____Ірина КАЛІНІНА___
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ ____Станіслав БЕЗОЩУК__
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «28» червня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інтелектуальна система моніторингу паливно-мастильних матеріалів

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Отримання завдання на виконання КР	27.06.2025	30.06.2025	
2.	Аналіз предметної області та постановка задачі	01.07.2025	20.07.2025	
3.	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз публікацій, щодо прогнозування та моніторингу ПММ	21.07.2025	30.07.2025	
4.	Аналіз існуючих підходів та алгоритмів інтелектуального аналізу даних для моніторингу паливно-мастильних матеріалів.	01.09.2025	25.10.2025	
5.	Реалізація обраних технологій з аналізом отриманих результатів	26.10.2025	21.11.2025	
6.	Перший попередній захист КР на засіданні комісії кафедри	25.11.2025	26.11.2025	
7.	Корегування роботи за результатами попереднього захисту	27.11.2025	05.12.2025	
8.	Другий попередній захист КР на засіданні комісії кафедри	09.12.2025	09.12.2025	
9.	Доробка та остаточне оформлення КР	10.12.2025	12.12.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.12.2025	16.12.2025	

Керівник роботи _____ Ірина КАЛІНІНА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Станіслав БЕЗОЩУК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«08» липня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 601м ЧНУ ім. Петра Могили

Безощука Станіслава Сергійовича

на тему: “**ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ПАЛИВНО-МАСТИЛЬНИХ МАТЕРІАЛІВ**”

Дана робота досліджує перехід від традиційних до інтелектуальних систем моніторингу паливно-мастильних матеріалів (ПММ), які є критично важливими для промислових операцій. У ній висвітлюються основні проблеми, пов'язані з існуючими методами, включаючи низьку точність, відсутність даних у реальному часі та високі експлуатаційні витрати. Звіт деталізує концептуальну архітектуру інтелектуальних систем, що використовують Інтернет речей (IoT) та штучний інтелект/машинне навчання (AI/ML), а також описує ключові компоненти, такі як різноманітні датчики, модулі збору та попередньої обробки даних, комунікаційні протоколи та розподілені обчислювальні парадигми (периферійні та хмарні обчислення). Особлива увага приділяється застосуванню алгоритмів ML для прогнозного обслуговування, виявлення аномалій та оптимізації споживання ПММ. Також відбувається аналіз значних переваг, які надають ці системи, включаючи суттєву економію коштів, скорочення часу простою, підвищення безпеки та покращення екологічної відповідності. Нарешті, обговорюються виклики, пов'язані з впровадженням, підкреслюючи потребу в стратегічному плануванні та комплексній трансформації організації для досягнення довгострокового успіху.

Актуальність теми зумовлена зростаючими вимогами до ефективного та екологічно відповідального використання паливно-мастильних матеріалів, а також потребою в оперативному контролі їх споживання та стану в умовах автоматизованих і дистанційно керованих промислових процесів.

Метою дослідження є аналіз, розробка та обґрунтування інтелектуальної системи моніторингу ПММ на базі сучасних інформаційних технологій, таких як IoT, штучний інтелект та хмарні обчислення.

Об'єкт дослідження – процеси контролю, обліку та оптимізації використання паливно-мастильних матеріалів у промислових системах.

Предмет дослідження – архітектура, функціональні компоненти та алгоритми інтелектуальних систем моніторингу ПММ, засновані на використанні цифрових технологій.

КР містить 120 сторінок, 6 додатків, 17 рисунів, 5 таблиць і 50 посилань на джерела.

Ключові слова: інтелектуальна система, моніторинг витрат палива, морський транспорт, машинне навчання, прогнозування, виявлення аномалій, пояснюваний штучний інтелект (XAI), shap, мікросервісна архітектура.

ABSTRACT

to the qualification work by the student of the group 601m of Petro Mohyla Black Sea National University

Bezoshchuk Stanislav

“INTELLIGENT FUEL AND LUBRICANTS MONITORING SYSTEM”

This study explores the transition from traditional to intelligent monitoring systems for fuels and lubricants, which are critical for industrial operations. It highlights key challenges associated with existing methods, including low accuracy, lack of real-time data, and high operational costs. The report details the conceptual architecture of intelligent systems leveraging the Internet of Things (IoT) and Artificial Intelligence/Machine Learning (AI/ML), while outlining key components such as various sensors, data acquisition and preprocessing modules, communication protocols, and distributed computing paradigms (edge and cloud computing). Particular emphasis is placed on the application of ML algorithms for predictive maintenance, anomaly detection, and the optimization of fuel and lubricant consumption. The study also analyzes the significant benefits provided by these systems, including substantial cost savings, reduced downtime, enhanced safety, and improved environmental compliance. Finally, implementation challenges are discussed, highlighting the need for strategic planning and comprehensive organizational transformation to achieve long-term success.

The relevance of the topic stems from growing demands for the efficient and environmentally responsible use of fuels and lubricants, as well as the need for real-time monitoring of their consumption and condition within automated and remotely controlled industrial processes.

The objective of the research is the analysis, development, and justification of an intelligent fuel and lubricant monitoring system based on modern information technologies, such as IoT, Artificial Intelligence, and cloud computing.

The object of research is the processes of control, accounting, and optimization of fuel and lubricant usage in industrial systems.

The work contains 120 pages, 6 appendices, 17 figures, 5 tables and 50 sources in it.

Key words: INTELLIGENT SYSTEM, FUEL CONSUMPTION MONITORING, MARITIME TRANSPORT, MACHINE LEARNING, FORECASTING, ANOMALY DETECTION, EXPLAINABLE AI (XAI), SHAP, MICROSERVICE ARCHITECTURE.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ДРП – датчик рівня палива

ККД – коефіцієнт корисної дії

ПММ – паливно-мастильні матеріали

СУБД – система управління базами даних

ШІ – штучний інтелект

ACID – Atomicity, Consistency, Isolation, Durability

AI – Artificial Intelligence

API – Application Programming Interface

ASGI – Asynchronous Server Gateway Interface

CAN – Controller Area Network

CD – Continuous Deployment / Continuous Delivery

CI – Continuous Integration

CRUD – Create, Read, Update, Delete

CTE – Common Table Expressions

DI – Dependency Injection

DOM – Document Object Model

DTO – Data Transfer Object

EDA – Exploratory Data Analysis

EF Core – Entity Framework Core

FMS – Fleet Management System

GPS – Global Positioning System

GRU – Gated Recurrent Unit

HFO – Heavy Fuel Oil

HTTP – HyperText Transfer Protocol

IMO – International Maritime Organization

IoT – Internet of Things

JSON – JavaScript Object Notation

KPI – Key Performance Indicators

LINQ – Language Integrated Query

LSTM – Long Short-Term Memory

MAE – Mean Absolute Error

MDO/MGO – Marine Diesel Oil / Marine Gas Oil

ML – Machine Learning

MPA – Multi-Page Application

MVCC – Multi-Version Concurrency Control

OPEX – Operating Expenses

ORDBMS – Object-Relational Database Management System

ORM – Object-Relational Mapping

PGO – Profile-Guided Optimization

REST – Representational State Transfer

RMSE – Root Mean Squared Error

RNN – Recurrent Neural Network

ROI – Return on Investment

RPM – Revolutions Per Minute

SHAP – SHapley Additive exPlanations

SPA – Single Page Application

SQL – Structured Query Language

SVM – Support Vector Machine

UI – User Interface

XAI – Explainable Artificial Intelligence

ВСТУП

Зі зростанням ролі транспортної логістики у світовій економіці та посиленням екологічних стандартів питання ефективності використання енергоресурсів набувають особливої актуальності. У структурі операційних витрат транспортних компаній, зокрема у морських перевезеннях, вартість паливно-мастильних матеріалів (ПММ) є однією з найбільших статей витрат. Неефективне споживання палива завдає прямої фінансової шкоди бізнесу та водночас призводить до збільшення шкідливих викидів в атмосферу.

Існуючі методи контролю споживання ПММ мають певні обмеження. Багато традиційних підходів базуються на статичних нормативах, розрахованих на основі усереднених умов, що робить їх малоефективними для точної оцінки ефективності в реальних, динамічно змінюваних умовах експлуатації. Навіть сучасні телематичні системи, що збирають великі обсяги даних, здебільшого виконують функцію реєстрації та базової звітності, але не надають інструментів для глибокого інтелектуального аналізу причин перевитрат. Це підвищує ризик фінансових втрат через технічні несправності, неефективну експлуатацію або несанкціоновані дії.

Метою даної кваліфікаційної роботи є дослідження, розробка та експериментальна перевірка методів машинного навчання (Machine Learning, ML) для створення інтелектуальної системи моніторингу паливно-мастильних матеріалів. Об’єктом дослідження є процес споживання ПММ на транспортних засобах, а предметом – алгоритми ML, такі як Лінійна регресія (Linear Regression), Випадковий ліс (Random Forest) та Градієнтний бустинг (Gradient Boosting). Ці алгоритми дозволяють аналізувати сукупність технічних, експлуатаційних та зовнішніх факторів для побудови динамічної нормативної моделі споживання палива та подальшого виявлення аномальних відхилень.

У межах даної роботи проведено системний аналіз проблеми моніторингу ПММ, досліджено фактори, що впливають на паливну ефективність, та розглянуто обмеження існуючих систем контролю. Розроблено та обґрунтовано концептуальну архітектуру програмного комплексу, що поєднує аналітичне ядро на Python, сервер

бізнес-логіки на .NET та клієнтський інтерфейс на React. Проведено експериментальне порівняння обраних ML-алгоритмів на реальному наборі даних для визначення найбільш точної моделі. Отримані результати формують науково-технічну основу для практичної реалізації інтелектуальної системи моніторингу, здатної підвищити ефективність використання енергоресурсів, зменшити операційні витрати та мінімізувати негативний вплив на навколишнє середовище.

1 АНАЛІЗ ПРОБЛЕМИ МОНІТОРИНГУ ПАЛИВНО-МАСТИЛЬНИХ МАТЕРІАЛІВ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Фактори впливу на споживання палива та сучасний стан проблеми моніторингу

Споживання паливно-мастильних матеріалів (ПММ) є однією з найбільших статей операційних витрат у транспортній галузі, особливо у морських перевезеннях. Ефективність використання палива безпосередньо впливає не лише на економічні показники підприємств, але й на екологічну ситуацію, оскільки спалювання викопного палива є основним джерелом викидів парникових газів, таких як CO_2, та інших шкідливих речовин (SO_X, NO_X). У зв'язку з посиленням міжнародних екологічних стандартів, зокрема з боку Міжнародної морської організації (ІМО), питання оптимізації та точного моніторингу споживання палива набуває виняткової актуальності.

Процес споживання палива є складним, багатофакторним явищем, яке залежить від великої кількості взаємопов'язаних змінних. Для побудови ефективної системи моніторингу необхідно глибоко розуміти ці фактори (рис. 1.1), які можна умовно поділити на кілька основних груп.

1.1.1 Технічні та конструкційні фактори

Тип та архітектура судна: тоннаж, форма корпусу, тип рушійної установки (гвинти, водомети), наявність додаткового обладнання – все це визначає базовий рівень гідродинамічного опору та необхідну потужність для руху. Танкери та контейнеровози через свої розміри та масу споживають значно більше палива, ніж невеликі сервісні катери або риболовецькі траулери.

Тип та стан головного двигуна: ефективність (ККД) двигуна, його вік, технічний стан, своєчасність обслуговування та якість мастильних матеріалів є критичними параметрами. Зношення циліндро-поршневої групи, забруднення

паливних форсунок або несправності турбокомпресора можуть призводити до значного (10-15%) збільшення споживання палива.

Тип палива: різні види палива (важкий мазут – HFO, дизельне паливо – MDO/MGO) мають різну теплотворну здатність та в'язкість, що впливає на процес згоряння та, як наслідок, на питоме споживання.

1.1.2 Експлуатаційні фактори

Швидкість судна: залежність між швидкістю та споживанням палива є нелінійною, близькою до кубічної. Це означає, що незначне збільшення швидкості (наприклад, на 10%) може призвести до збільшення споживання палива на 30% і більше. Оптимізація швидкості (slow steaming) є одним з найефективніших методів економії палива.

Завантаженість судна та диферент: водотоннажність судна, розподіл вантажу, його осадка та диферент (поздовжній нахил) безпосередньо впливають на опір води та ефективність роботи гвинтів. Неоптимальний диферент може збільшити споживання палива на 2-5%.

Стиль управління (людський фактор): досвід та кваліфікація екіпажу, зокрема капітана та старшого механіка, відіграють важливу роль. Плавність маневрів, вибір оптимального режиму роботи двигуна, здатність ефективно використовувати навігаційні системи для прокладання маршруту – все це впливає на кінцеві витрати.

1.1.3 Зовнішні фактори (умови навколишнього середовища)

Погодні умови: вітер, хвилі, течії створюють додатковий опір руху судна, змушуючи двигун працювати з більшим навантаженням для підтримки заданої швидкості. Рейс у штормових умовах може призвести до збільшення споживання палива на 50% і більше порівняно з рейсом у спокійну погоду.

Маршрут та умови навігації: довжина маршруту, необхідність проходження вузьких проток, мілководдя, зон з інтенсивним трафіком – все це вимагає частих маневрів, змін швидкості та збільшує загальні витрати палива.

Сучасний стан проблеми моніторингу

Традиційні підходи до моніторингу споживання ПММ часто базуються на використанні статичних нормативів. Ці норми розраховуються на основі паспортних даних судна та усереднених умов експлуатації. Головним недоліком такого підходу є його нездатність враховувати динамічну взаємодію вищезазначених факторів. Статична норма не може адекватно оцінити, чи був перерасход у 15% під час шторму обґрунтованим, чи він є наслідком технічної несправності або несанкціонованих дій.

Сучасні телематичні та GPS-системи зробили крок уперед, дозволяючи збирати великі обсяги даних в режимі реального часу. Вони забезпечують точну реєстрацію маршруту, швидкості, рівня палива в баках та інших параметрів. Однак більшість комерційних систем обмежуються функціями реєстрації та базової звітності.

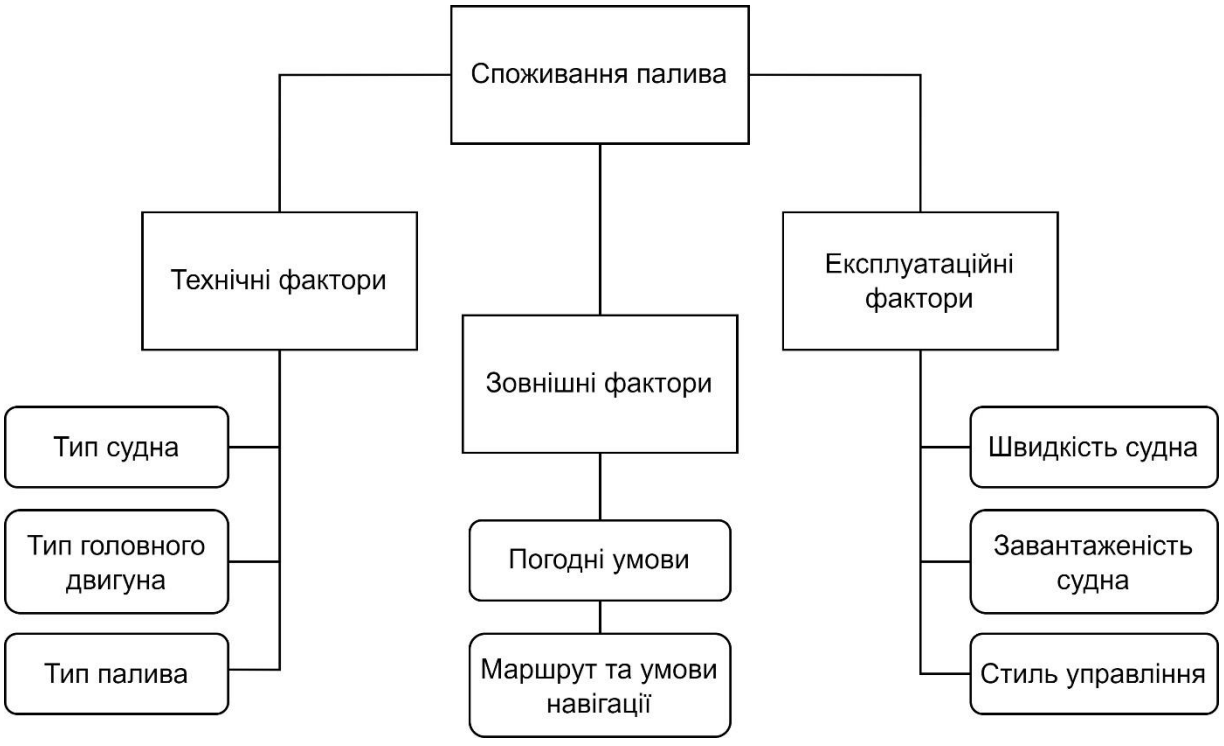

Рисунок 1.1 – Класифікація факторів, що впливають на споживання палива на морському транспорті

Вони можуть побудувати графік рівня палива і зафіксувати різке його падіння (потенційний злив), але не можуть відповісти на більш складне питання: "Чи спожило судно за цей рейс стільки палива, скільки повинно було спожити за цих конкретних умов?".

Таким чином, виникає розрив між здатністю збирати дані та здатністю інтерпретувати їх для прийняття управлінських рішень. Проблема сучасного моніторингу полягає не у відсутності даних, а у відсутності інструментів для їх інтелектуального аналізу. Необхідна система, яка могла б на основі історичних даних та методів машинного навчання створювати динамічну, адаптивну "норму" для кожного конкретного рейсу, враховуючи всю сукупність факторів. Порівняння фактичних показників з такою динамічною нормою і є основою для інтелектуального моніторингу, який дозволяє не просто фіксувати події, а проактивно виявляти аномалії, діагностувати проблеми та оптимізувати операційні процеси.

1.2 Способи контролю та обліку ПММ у транспортній галузі

Ефективне управління паливно-мастильними матеріалами неможливе без точних та надійних методів їх контролю та обліку. Історично ці методи пройшли значну еволюцію від ручних розрахунків до високотехнологічних інструментальних комплексів. Для повного розуміння існуючої проблеми необхідно проаналізувати ключові підходи, їхні можливості та фундаментальні обмеження. Загалом їх можна класифікувати на дві великі групи: традиційні та автоматизовані.

Основою традиційного підходу є нормативний метод (табл. 1.1), який тривалий час був стандартом у багатьох транспортних організаціях. Його суть полягає у порівнянні фактично витраченого палива, підтвердженого звітністю (чеки, шляхові листи), із заздалегідь встановленими нормами споживання для кожної моделі транспортного засобу. Для кожної одиниці техніки розраховується базова норма на 100 км пробігу, яка згодом коригується за допомогою системи

коефіцієнтів, що намагаються врахувати умови експлуатації: сезонність, рух у місті чи за його межами, роботу додаткового обладнання тощо.

Таблиця 1.1 – Переваги та недоліки нормативного методу

Переваги	Недоліки
Низька вартість впровадження, оскільки не потребує встановлення додаткового обладнання.	***Низька точність***: нормативи є усередненими і не здатні врахувати всю множину динамічних факторів, таких як реальний трафік, погодні умови, стиль водіння, технічний стан та якість дорожнього покриття.
Простота розрахунків, що вимагає лише ведення стандартної звітності.	***Висока вразливість до шахрайства***: метод створює можливості для махінацій, таких як завищення пробігу, надання фіктивних чеків або злив зекономленого палива.
	Відсутність оперативних даних: аналіз відбувається постфактум, що не дозволяє оперативно реагувати на перевитрати чи можливі крадіжки.

З розвитком GPS-технологій та мікроелектроніки з'явилися автоматизовані (інструментальні) методи, що дозволяють отримувати об'єктивні дані безпосередньо з транспортного засобу, мінімізуючи вплив людського фактора.

Найпоширенішим інструментальним методом (див. табл. 1.2) є встановлення датчиків рівня палива (ДРП). Це спеціальний сенсор, зазвичай ємнісного типу, що монтується в паливний бак і з високою точністю вимірює поточний об'єм палива. Підключений до GPS-трекера, датчик передає дані на сервер, де програмне забезпечення будує графік рівня палива. На цьому графіку чітко візуалізуються

ключові події: заправки (різкий підйом рівня) та несанкціоновані зливи (різке падіння рівня за відсутності роботи двигуна).

Таблиця 1.2 – Переваги та недоліки інструментального методу

Переваги	Недоліки
Прямий контроль об'єму палива в баку та ефективне виявлення фактів заправок і зливів.	Похибка вимірювань під час руху через коливання палива; наявність "мертвих зон" у баку.
	Не вимірює моментальне споживання палива двигуном, а лише зміну загального рівня.

Для отримання більш точних даних про фактичне споживання палива безпосередньо двигуном застосовуються проточні витратоміри (див. табл. 1.3). Ці пристрої встановлюються в розріз паливної магістралі, що веде до двигуна, і вимірюють об'єм палива, що проходить через них. Для дизельних двигунів, які мають зворотну магістраль ("обратку"), використовуються диференційні моделі, що розраховують різницю між об'ємом поданого та повернутого палива.

Таблиця 1.3 – Переваги та недоліки проточних витратомірів

Переваги	Недоліки
Дуже висока точність вимірювання фактично спожитого двигуном палива (похибка < 1%).	Висока вартість обладнання та монтажу, що вимагає втручання в паливну систему.
	Не контролює зливи з бака, оскільки вимірює лише те, що надійшло до двигуна.

Найбільш технологічно прогресивним методом отримання даних є підключення до цифрової CAN-шини сучасного транспортного засобу (табл. 1.4). CAN-шина є єдиною інформаційною мережею, що об'єднує електронні блоки автомобіля. Спеціальний зчитувач, підключений до шини, "прослуховує" стандартні повідомлення (за протоколами FMS, J1939) і витягує з них величезний масив даних: загальне спожите паливо за даними бортового комп'ютера, рівень палива, оберти та навантаження на двигун, швидкість, пробіг та багато іншого.

Таблиця 1.4 – Переваги та недоліки підключення до цифрової CAN-шини

Переваги	Недоліки
Надання розширеного контексту для аналізу (оберти, навантаження), що є критично важливим для інтелектуальних систем.	Не всі транспортні засоби оснащені CAN-шиною, або виробник може блокувати доступ до даних.
Отримання комплексних та точних даних від штатних систем автомобіля без встановлення додаткових датчиків.	Метод повністю залежить від точності штатних датчиків автомобіля.

Проведений аналіз показує, що кожен з існуючих методів має свої сильні сторони та суттєві обмеження. Традиційний нормативний підхід є застарілим та неточним. Інструментальні методи, хоча і надають об'єктивні дані, вирішують переважно задачу реєстрації фактів (заправки, зливи, поточний рівень), але не дають відповіді на головне питання менеджменту: наскільки ***ефективно*** було використано паливо за конкретних умов?

Проблема сучасної транспортної телематики змістилася з площини "як отримати дані?" до площини "як ці дані правильно інтерпретувати?". Навіть маючи точні дані з CAN-шини про спожиті 500 літрів палива, менеджер не може зробити

висновок, багато це чи мало для даного рейсу, що проходив у шторм зі зустрічним вітром. Таким чином, існує гостра потреба в розробці систем, які б могли синтезувати дані з різних джерел та створювати на їх основі динамічний еталон (норму), що і є завданням для інтелектуальних систем аналізу.

1.3 Огляд сучасних систем моніторингу та існуючих досліджень

Аналіз проблеми контролю паливно-мастильних матеріалів був би неповним без огляду існуючих програмних рішень, що представлені на ринку, а також наукових досліджень, спрямованих на вирішення цієї задачі за допомогою сучасних методів аналізу даних. Ці два напрямки, комерційний та науковий, хоч і переслідують схожу мету, часто використовують принципово різні підходи до її досягнення.

Сучасні комерційні системи моніторингу

Ринок транспортної телематики пропонує велику кількість програмних платформ, призначених для GPS-моніторингу та управління автопарком [1]. Такі системи є комплексними рішеннями, що агрегують дані, отримані від апаратних засобів (GPS-трекерів, датчиків), і надають користувачеві інтерфейс для їх візуалізації та аналізу. Серед провідних платформ на світовому та вітчизняному ринках можна виділити Wialon [2], UMT (Ukrtrans Telematics) [3], а також програмне забезпечення від виробників обладнання, як-от Teltonika [4].

Функціонал таких систем, як правило, є досить стандартизованим і включає:

– моніторинг у реальному часі: відображення місцезнаходження транспортних засобів на карті, їх швидкості та стану (рух, стоянка, робота двигуна);

– контроль палива: побудова графіків рівня палива на основі даних з ДРП або CAN-шини, автоматичне визначення та сповіщення про заправки та зливи;

– побудова звітів: генерація детальних звітів про пробіг, мотогодини, поїздки, стоянки, відвідування геозон та витрати палива за обраний період;

– контроль маршрутів та геозон: можливість створювати планові маршрути та географічні зони з налаштуванням сповіщень про їх відвідування або покидання.

Незважаючи на високий рівень деталізації даних, ключовим недоліком переважної більшості комерційних систем є їхня дескриптивна (описова) природа. Вони блискуче відповідають на питання "що сталося?" (наприклад, "був злив 50 літрів палива о 03:00") або "скільки було витрачено?" ("за рейс витрачено 500 літрів"). Однак вони не дають обґрунтованої відповіді на аналітичне питання: "чи є витрата у 500 літрів нормальною для цього конкретного рейсу за цих конкретних умов?". Розрахунок "норми" в таких системах часто зводиться до простого середнього показника (л/100 км) за минулий період, що не враховує нелінійну залежність від швидкості, завантаженості, погодних умов та інших факторів [5]. Таким чином, ці системи фіксують дані, але залишають їхню глибоку інтерпретацію та пошук прихованих неефективностей людині.

Існуючі наукові дослідження

На противагу комерційним системам, наукові дослідження останніх років все частіше зосереджуються на застосуванні методів машинного та глибокого навчання для вирішення проблеми аналізу паливної ефективності [6,7]. Цей підхід дозволяє перейти від простої реєстрації даних до їх предиктивного (прогнозного) аналізу. Основні напрямки досліджень можна згрупувати як наведено нижче.

1. Побудова моделей прогнозування споживання палива (регресійний аналіз). Це найбільш поширений напрямок. Дослідники використовують історичні дані телеметрії для навчання моделей, які здатні прогнозувати споживання палива на основі вхідних параметрів [8,9]. В опублікованих роботах для цього успішно застосовуються:

– класичні методи: множинна лінійна регресія [10], поліноміальна регресія;

– ансамблеві моделі: випадковий ліс (Random Forest) [11] та методи градієнтного бустингу (Gradient Boosting, XGBoost[12], LightGBM [13]). Ці моделі показують особливо високу точність, оскільки здатні вловлювати складні нелінійні залежності між десятками різних факторів [14,15];

– нейронні мережі: для аналізу даних у вигляді часових рядів (наприклад, похвилинних даних з CAN-шини) використовуються рекурентні нейронні мережі (RNN), зокрема архітектури LSTM [16] та GRU [17].

2. Цей напрямок безпосередньо пов'язаний із задачею моніторингу [18]. Основна ідея полягає у навчанні моделі на "нормальних" даних, щоб вона могла ідентифікувати викиди. Один із найефективніших підходів, що застосовується в дослідженнях, – це використання регресійної моделі для прогнозування норми[19]. Якщо різниця (залишок) між фактичним та прогнозованим значенням перевищує певний поріг, такий випадок позначається як аномалія. Також використовуються спеціалізовані алгоритми, такі як Isolation Forest [20] або One-Class SVM [21].

3. Аналіз стилю водіння (Driver Behavior Analysis). Ряд досліджень фокусується на аналізі впливу людського фактора [22]. За допомогою алгоритмів кластеризації (напр., K-Means [23]) дані про поїздки (різкі прискорення/гальмування, перевищення швидкості, робота двигуна на високих обертах) групуються для ідентифікації різних стилів водіння ("агресивний", "економний", "оптимальний"). Це дозволяє оцінити вплив кожного водія на споживання палива [24].

Аналіз ринку та наукових публікацій виявляє чіткий розрив: комерційні платформи надають потужні інструменти для збору та візуалізації даних, але мають обмежені аналітичні можливості. Водночас наукові дослідження демонструють високий потенціал методів машинного навчання для глибокого аналізу паливної ефективності, однак ці розробки рідко втілюються у готові, зручні для кінцевого користувача програмні продукти [25].

Таким чином, існує незаповнена ніша для створення інтелектуальної системи, яка б поєднувала надійність збору даних, характерну для комерційних систем, з аналітичною глибиною, продемонстрованою в наукових дослідженнях. Саме на вирішення цієї задачі – створення системи, що реалізує методологію моніторингу на основі динамічної, прогнозованої норми, – і спрямована дана кваліфікаційна робота.

1.4 Характеристика даних телеметрії та експлуатаційних параметрів

Фундаментом будь-якої інтелектуальної системи аналізу є дані. Якість, повнота та релевантність цих даних безпосередньо визначають точність та практичну цінність майбутньої моделі машинного навчання. У контексті моніторингу паливно-мастильних матеріалів основними джерелами інформації є дані телеметрії та інші експлуатаційні параметри, що фіксують різні аспекти роботи транспортного засобу.

Дані телеметрії за своєю природою є часовими рядами – послідовністю вимірювань, отриманих через певні проміжки часу. Гранулярність цих даних може варіюватися від кількох секунд (для систем, що працюють з CAN-шиною) до кількох хвилин (для стандартних GPS-трекерів). Основні параметри, що збираються сучасними телематичними системами, наведено нижче.

1. Просторово-часові дані:

– мітка часу (Timestamp): точний час фіксації показників, є ключем для синхронізації всіх даних;

– географічні координати (Latitude, Longitude): дозволяють відстежувати маршрут, розраховувати пробіг та аналізувати швидкісні режими на різних ділянках шляху;

– швидкість (Speed): один із найважливіших параметрів, що має нелінійний вплив на споживання палива;

– напрямок руху (Heading/Azimuth): використовується для візуалізації треку на карті.

2. Дані про стан транспортного засобу:

– стан запалювання (Ignition Status): бінарний показник (увімкнено/вимкнено), що дозволяє чітко розділяти час роботи двигуна (мотогодини) та час простою;

– пробіг (Odometer): загальний пробіг, що фіксується лічильником автомобіля;

– рівень палива (Fuel Level): дані, що надходять від штатного або додатково встановленого датчика рівня палива. Аналіз динаміки цього показника дозволяє фіксувати заправки та зливи.

3. Розширені дані з CAN-шини (за наявності):

– оберти двигуна (Engine RPM): характеризує режим роботи двигуна;

– навантаження на двигун (Engine Load): показує у відсотках, наскільки інтенсивно працює двигун у даний момент;

– температура охолоджуючої рідини (Coolant Temperature): дозволяє визначити, чи прогрітий двигун до робочої температури, що впливає на ефективність;

– моментальне споживання палива (Instant Fuel Consumption): показник, що розраховується бортовим комп'ютером (напр., в л/год);

– загальне спожите паливо (Total Fuel Consumed): лічильник, що акумулює дані про спожите паливо з моменту виробництва автомобіля.

Характеристика набору даних, що використовується у роботі

Набір даних, наданий для проведення даного кваліфікаційної роботи, є прикладом агрегованих експлуатаційних звітів. На відміну від високочастотних телеметричних даних, де кожен запис відповідає моменту часу, тут кожен запис представляє собою підсумок роботи конкретного судна за один календарний місяць. Така структура даних визначає специфіку задачі: замість моніторингу в реальному часі, фокус зміщується на стратегічний моніторинг та аналіз ефективності за тривалі періоди.

Набір даних містить наступні ключові атрибути (ознаки):

– ідентифікаційні: ***ship_id*** (унікальний ідентифікатор судна), ***route_id*** (ідентифікатор типового маршруту);

– часові: ***month*** (місяць, за який наведено дані);

– категоріальні: ***ship_type*** (тип судна), ***fuel_type*** (тип палива), ***weather_conditions*** (переважаючі погодні умови за місяць). Ці ознаки дозволяють моделі враховувати базові відмінності між об'єктами та умовами їх експлуатації;

– числові: ***distance***: сумарна пройдена дистанція за місяць. ***engine_efficiency***: усереднений показник ефективності двигуна, що може бути розрахований на основі внутрішніх діагностичних систем. ***fuel_consumption***: цільова змінна дослідження. Це загальний об'єм палива, фактично спожитий судном за звітний місяць. ***CO2_emissions***: загальні викиди CO_2, що є похідною величиною від спожитого палива.

Даний набір даних є надзвичайно цінним, оскільки він містить реальні, а не синтетичні показники. Водночас його агрегована природа накладає певні обмеження: неможливо проаналізувати вплив моментальних змін (наприклад, різкого прискорення) або відстежити короткочасні події, такі як злив палива. Проте, він ідеально підходить для побудови моделі, що прогнозує місячне нормативне споживання, та подальшого використання цієї моделі для виявлення суден або періодів зі стабільно завищеними витратами, що свідчить про системні проблеми – технічні або організаційні. Таким чином, задача моніторингу в рамках даної роботи фокусується на виявленні довгострокових аномалій та патернів неефективності.

1.5 Постановка задачі дослідження

Проведений аналіз сучасного стану проблеми моніторингу паливно-мастильних матеріалів виявив ключову суперечність: з одного боку, існують потужні комерційні системи, здатні збирати великі обсяги телеметричних даних, а з іншого – наукові дослідження демонструють високу ефективність методів машинного навчання для глибокого аналізу цих даних. Проте, між цими двома напрямками існує значний розрив: комерційні продукти здебільшого обмежуються функціями реєстрації та описової звітності, тоді як наукові розробки рідко втілюються у готові до використання програмні рішення.

Традиційні методи контролю, що базуються на статичних нормативах, є неточними та не враховують динамічну природу експлуатаційних та зовнішніх факторів, що створює можливості для неефективного використання ресурсів та зловживань. Сучасні інструментальні засоби, хоча й надають об'єктивні дані, не

дають відповіді на головне управлінське питання: чи є зафіксована витрата палива обґрунтованою та оптимальною для конкретних умов рейсу?

Виходячи з цього, виникає необхідність у розробці інтелектуальної системи, яка б перейшла від парадигми простої реєстрації даних до парадигми проактивного моніторингу ефективності.

Таким чином, метою даного дослідження є розробка та експериментальна перевірка методології побудови інтелектуальної системи для моніторингу споживання ПММ, що базується на створенні динамічної нормативної моделі за допомогою методів машинного навчання та подальшому аналізі відхилень фактичних показників від прогнозованих.

Для досягнення поставленої мети необхідно вирішити наступні завдання дослідження:

– провести системний аналіз предметної області, виявити ключові фактори, що впливають на споживання ПММ, та дослідити обмеження існуючих методів і систем контролю;

– обґрунтувати вибір методів машинного навчання для побудови регресійної моделі прогнозування споживання палива, проаналізувавши їхні теоретичні основи, переваги та недоліки в контексті поставленої задачі;

– розробити методику підготовки даних для моделювання, що включає етапи очищення, кодування, інжинірингу ознак та масштабування на прикладі наданого набору реальних експлуатаційних даних;

– провести експериментальне дослідження обраних алгоритмів машинного навчання (лінійна регресія, випадковий ліс, градієнтний бустинг), навчити та протестувати моделі на підготовленому наборі даних;

– виконати порівняльний аналіз отриманих результатів на основі обраних метрик якості (MAE, RMSE, R^2) та обрати оптимальну модель для використання в ядрі системи моніторингу;

– розробити та описати архітектуру програмного прототипу, що реалізує запропоновану методологію, та продемонструвати її роботу на конкретному прикладі, показавши, як система виявляє аномальні відхилення.

Об'єктом дослідження є процес споживання паливно-мастильних матеріалів морськими суднами. Предметом дослідження є моделі та методи машинного навчання для побудови нормативних прогнозів та їх застосування для задач моніторингу та виявлення аномалій.

Висновки до розділу 1

У даному розділі було проведено комплексний аналіз проблеми моніторингу паливно-мастильних матеріалів у транспортній галузі, що дозволило сформулювати основні висновки та обґрунтувати актуальність і напрямок подальшого дослідження.

1. Встановлено, що ефективність споживання палива є критичним фактором, який визначає як економічні показники транспортних підприємств, так і їхній вплив на навколишнє середовище. Процес споживання палива є складним явищем, що залежить від великої кількості технічних, експлуатаційних та зовнішніх факторів, таких як тип судна, його швидкість, завантаженість та погодні умови.

2. Проаналізовано існуючі методи контролю та обліку ПММ. Виявлено, що традиційні нормативні методи є неточними, усередненими та вразливими до зловживань. Сучасні автоматизовані системи (на базі ДРП, витратомірів, CAN-шини) ефективно вирішують задачу збору об'єктивних даних, однак їхній функціонал здебільшого обмежується дескриптивним аналізом – реєстрацією фактів та побудовою звітів, а не глибокою аналітикою.

3. Проведено огляд комерційних платформ моніторингу та наукових досліджень. Встановлено наявність розриву між ринком та наукою: комерційні продукти мають обмежені інтелектуальні можливості, тоді як наукові роботи, що

демонструють високий потенціал методів машинного навчання для прогнозування та аналізу аномалій, рідко втілюються у готові до використання програмні рішення.

4. На основі аналізу предметної області та існуючих проблем була сформульована головна задача дослідження: розробка та дослідження інтелектуальної системи, яка б виконувала функцію моніторингу ефективності не шляхом порівняння з застарілими статичними нормативами, а шляхом аналізу відхилень фактичних показників від динамічної норми. Ця норма повинна генеруватися моделлю машинного навчання в режимі реального часу для кожних конкретних умов експлуатації.

Таким чином, результати, отримані в першому розділі, підтверджують високу актуальність обраної теми та дозволяють перейти до наступного етапу роботи – розробки концептуальної архітектури такої системи та вибору математичного апарату для її реалізації.

2. СТРУКТУРА ТА МЕТОДИ ПОБУДОВИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ ПММ

2.1 Концептуальна архітектура інтелектуальної системи моніторингу "OptiFuel"

На основі аналізу проблеми та постановки задачі дослідження, проведених у першому розділі, було розроблено концептуальну архітектуру програмного комплексу "OptiFuel". Головною метою при проектуванні було створення гнучкої, масштабованої та легко підтримуваної системи, здатної реалізувати методологію інтелектуального моніторингу шляхом аналізу відхилень фактичних показників від динамічно генерованої норми.

Основний принцип роботи системи полягає у взаємодії трьох логічних рівнів: представлення даних (клієнтський інтерфейс), обробки бізнес-логіки (основний сервер) та аналітичного ядра (сервіс машинного навчання). Для реалізації такої структури було обрано мікросервісну архітектуру. Цей підхід дозволяє розробляти, розгортати та масштабувати кожен компонент системи незалежно, використовуючи для кожного з них найбільш відповідний технологічний стек.

Переваги обраної архітектури в контексті даної задачі:

– технологічна гнучкість: можливість використовувати Python, який є стандартом де-факто для задач машинного навчання, для аналітичного ядра, та C# .NET, що є потужною платформою для створення надійних та високопродуктивних бекенд-сервісів, для бізнес-логіки;

– розділення відповідальностей: кожен сервіс має чітко визначену, єдину відповідальність. Це спрощує розробку, тестування та подальшу підтримку системи;

– незалежне масштабування: у випадку високого навантаження, наприклад, на сервіс прогнозування, можна збільшити кількість його екземплярів, не зачіпаючи інші компоненти системи.

Загальна архітектура системи "OptiFuel" складається з трьох основних компонентів та бази даних, взаємодія яких представлена на рис. 2.1.

Характеристика компонентів системи

Клієнтський застосунок (Frontend). Рівень представлення, реалізований як односторінковий додаток (SPA) на базі бібліотеки React та фреймворку компонентів Mantine UI. Його головними завданнями є:

– надання користувачеві інтерактивного інтерфейсу для введення параметрів рейсу;

– валідація введених даних на стороні клієнта;

– відправка запитів на Backend API та отримання результатів;

– візуалізація отриманих прогнозів та результатів моніторингу (зокрема, розрахунок та відображення відхилень).

Основний сервер додатку (Backend API). Рівень бізнес-логіки, розроблений на платформі C# .NET з використанням ASP.NET Core Web API. Цей компонент є центральною ланкою, що оркеструє роботу всієї системи. Його відповідальність включає:

– обробку HTTP-запитів від клієнтського застосунку;

– серверну валідацію вхідних даних для забезпечення безпеки та цілісності;

– формування запиту та звернення до сервісу машинного навчання для отримання прогнозу;

– взаємодію з базою даних PostgreSQL за допомогою Entity Framework Core для збереження історії всіх операцій (вхідних параметрів та отриманих прогнозів);

– надання API для отримання історичних даних (наприклад, для сторінки "Історія операцій").

Сервіс машинного навчання (ML Service). Аналітичне ядро системи, реалізоване на Python з використанням фреймворку FastAPI. Цей мікросервіс інкапсулює всю логіку, пов'язану з машинним навчанням. Його завдання:

– при старті завантажувати навчену модель (наприклад, Gradient Boosting) та об'єкт для масштабування даних (StandardScaler) зі збережених артефактів;

– надавати єдиний API-ендпоінт (/predict), що приймає параметри рейсу у форматі JSON;

– виконувати передпроцесинг вхідних даних у реальному часі, приводячи їх до формату, ідентичного тому, що використовувався при навчанні моделі;

– застосовувати модель для розрахунку прогнозованого споживання палива;

– повертати отриманий прогноз основному серверу додатку.

Логічна послідовність обробки запиту (Data Flow)

Процес отримання прогнозу в системі "OptiFuel" проходить наступні етапи.

1. Користувач заповнює форму на Frontend-додатку та натискає кнопку розрахунку.
2. React-додаток відправляє валідовані дані у вигляді JSON-об'єкта на POST /api/prediction ендпоінт Backend API.
3. .NET-сервер отримує запит, проводить повторну серверну валідацію та формує запит до ML-сервісу.
4. .NET-сервер відправляє HTTP-запит на POST /predict ендпоінт Python-сервісу.
5. Python-сервіс обробляє дані, застосовує модель та повертає прогноз у відповіді.
6. .NET-сервер отримує прогноз, створює новий запис в таблиці PredictionHistories у базі даних PostgreSQL та зберігає його.
7. .NET-сервер відправляє фінальну відповідь з прогнозом назад на Frontend.
8. React-додаток отримує відповідь та відображає результат користувачеві.

Така архітектура є сучасною, надійною та повністю відповідає поставленим у роботі завданням, забезпечуючи чітке розділення логіки та можливість подальшого розвитку кожного з компонентів системи.

2.2 Огляд та обґрунтування вибору алгоритмів машинного навчання

Вирішення задачі прогнозування споживання палива, яка за своєю суттю є задачею регресії з множиною вхідних параметрів, вимагає застосування відповідного математичного апарату. Методи машинного навчання (Machine Learning, ML) надають потужний інструментарій для виявлення складних, нелінійних залежностей у даних, що є неможливим при використанні традиційних нормативних підходів.

Для проведення експериментального дослідження було обрано три алгоритми, що представляють різні класи моделей за складністю та принципом роботи: лінійна регресія як базовий метод, а також два потужні ансамблеві методи – випадковий ліс та градієнтний бустинг. Такий вибір дозволяє не лише побудувати точну прогнозну модель, але й оцінити приріст якості, який дають більш складні алгоритми порівняно з простою базовою моделлю (baseline).

2.2.1 Лінійна регресія (Linear Regression)

Лінійна регресія є одним з найбільш фундаментальних та інтерпретованих алгоритмів у статистиці та машинному навчанні. Її основна ідея полягає у моделюванні лінійної залежності між незалежними змінними (ознаками) та залежною змінною (цільовим показником).

Принцип роботи. Модель намагається знайти оптимальні коефіцієнти (ваги) для кожної ознаки, щоб сума ознак, помножених на їхні ваги, найкращим чином апроксимувала цільову змінну. Математично, модель множинної лінійної регресії описується рівнянням:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon, \quad (2.1)$$

де y – прогнозоване значення (споживання палива);

$x_1, x_2, \dots, x_n$ – значення ознак (дистанція, ефективність двигуна тощо);

β_0 – вільний член (зсув), значення прогнозу, коли всі ознаки дорівнюють нулю;

$\beta_1, \beta_2, \ldots, \beta_n$ – коефіцієнти регресії, що показують, наскільки зміниться y при зміні відповідальної ознаки x_i на одиницю;

ε – похибка моделі (різниця між реальними та прогнозованим значенням).

Навчання моделі полягає у знаходженні таких коефіцієнтів β, при яких сума квадратів похибок (метод найменших квадратів, Ordinary Least Squares) на навчальних даних є мінімальною.

Обґрунтування вибору. У даному дослідженні лінійна регресія використовується як базова модель (baseline). Вона дозволяє встановити початковий рівень точності, з яким будуть порівнюватися більш складні моделі. Її головна перевага – висока інтерпретованість: аналізуючи коефіцієнти β, можна оцінити, який фактор і наскільки сильно (за лінійною моделлю) впливає на споживання палива. Водночас головним недоліком є припущення про лінійність залежностей, що в реальних фізичних процесах, як-от споживання палива, майже ніколи не виконується, а тому очікується, що точність цієї моделі буде найнижчою.

2.2.2 Випадковий ліс (Random Forest)

Випадковий ліс – це ансамблевий метод машинного навчання, що належить до класу моделей, які використовують "мудрість натовпу" для підвищення точності. Замість побудови однієї складної моделі, він створює велику кількість простих моделей (дерев рішень) і усереднює їхні прогнози.

Принцип роботи. Алгоритм випадкового лісу для задачі регресії працює наступним чином:

– створення підвибірок (Bagging): з вихідного навчального набору даних створюється N нових підвибірок за допомогою методу бутстрепу (випадковий вибір з поверненням). Кожна підвибірка має такий самий розмір, як і оригінальна, але деякі об'єкти в ній можуть повторюватися, а деякі – бути відсутніми;

– побудова дерев рішень: на кожній підвибірці навчається окреме дерево рішень. При побудові кожного дерева вводиться додатковий елемент випадковості: при розбитті кожного вузла розглядається не весь набір ознак, а лише їх випадкова

підмножина. Це робить дерева більш різноманітними та менш скорельованими між собою;

– усереднення результатів: для нового об'єкта кожне з N дерев робить свій прогноз. Фінальним прогнозом випадкового лісу є середнє арифметичне всіх індивідуальних прогнозів.

Обґрунтування вибору. Випадковий ліс є надзвичайно популярним та ефективним алгоритмом, що добре підходить для даної задачі. Його ключова перевага – здатність моделювати складні, нелінійні залежності між ознаками без необхідності їх явної специфікації. Завдяки усередненню прогнозів від великої кількості дерев, він є дуже стійким до перенавчання. Крім того, алгоритм надає вбудований механізм оцінки важливості ознак (Feature Importance), що дозволяє проаналізувати, які саме фактори мали найбільший вплив на прогнози моделі.

2.2.3 Градієнтний бустинг (Gradient Boosting)

Градієнтний бустинг, як і випадковий ліс, є ансамблевим методом, що використовує дерева рішень, проте його філософія кардинально інша. Замість паралельної побудови незалежних дерев, градієнтний бустинг будує їх послідовно, де кожне наступне дерево намагається виправити помилки попереднього.

Принцип роботи. Процес є ітеративним:

а) ініціалізація: модель починається з простого прогнозу, зазвичай середнього значення цільової змінної для всіх об'єктів;

б) ітеративна побудова: на кожному наступному кроці (ітерації):

1) Розраховуються помилки (залишки) поточної ансамблевої моделі – різниця між фактичними та прогнозованими значеннями.

2) Навчається нове, просте дерево рішень, яке намагається спрогнозувати не саму цільову змінну, а ці помилки.

3) Нове дерево додається до ансамблю з певним невеликим коефіцієнтом (темпом навчання, learning rate), тим самим поступово "коригуючи" загальний прогноз у правильному напрямку.

в) фінальний прогноз: процес повторюється задану кількість разів. Фінальний прогноз для нового об'єкта є сумою початкового прогнозу та прогнозів усіх "дерев-коректорів", помножених на темп навчання.

Обґрунтування вибору

Алгоритми на основі градієнтного бустингу (зокрема, його реалізації GradientBoostingRegressor, XGBoost, LightGBM) часто демонструють найвищу точність у змаганнях з машинного навчання та в промислових задачах. Цей метод був обраний для дослідження як кандидат на найбільш точну модель. Його здатність послідовно концентруватися на складних для прогнозування випадках дозволяє досягти кращих результатів, ніж у випадкового лісу. Як і Random Forest, він також дозволяє оцінити важливість ознак, що є цінним для аналізу результатів. Однак він є більш чутливим до налаштування гіперпараметрів (кількість дерев, темп навчання, глибина дерев) і схильний до перенавчання, якщо їх підібрати некоректно.

Таким чином, обраний набір алгоритмів дозволяє провести всебічне дослідження: від простої інтерпретованої моделі до складних, високоточних ансамблів, що є необхідним для побудови надійного аналітичного ядра системи "OptiFuel".

2.3 Методологія підготовки даних та проведення експерименту

Якість та надійність моделей машинного навчання безпосередньо залежать від якості даних, на яких вони навчаються, а також від коректно організованого процесу експериментальних досліджень. У даному підрозділі детально описано методологію, що застосовувалася для підготовки вихідного набору даних та проведення експерименту з порівняння алгоритмів.

2.3.1 Опис та аналіз вихідного набору даних

Для проведення дослідження було використано набір реальних експлуатаційних даних, що містить агреговану інформацію про роботу морських

суден за звітні періоди (місяці). Набір даних складається з 1440 записів та 10 атрибутів, які було детально охарактеризовано у підрозділі 1.4.

Первинний аналіз даних показав, що набір є повним, тобто не містить пропущених значень, що значно спрощує етап попередньої обробки. Атрибути представлені різними типами даних: числовими (наприклад, distance, engine_efficiency) та категоріальними (наприклад, ship_type, weather_conditions). Цільовою змінною для прогнозування є fuel_consumption – загальне споживання палива.

2.3.2 Етапи передпроцесингу даних

Моделі машинного навчання працюють з числовими даними у векторному представленні, тому перед початком моделювання необхідно було виконати низку перетворень, щоб привести вихідний набір даних до відповідного формату. Процес передпроцесингу було організовано у вигляді послідовного конвеєра (pipeline) з наступних єтапів.

1. Кодування категоріальних ознак. Оскільки алгоритми не можуть працювати з текстовими даними, було застосовано два методи кодування:

– порядкове кодування (Ordinal Encoding): застосовано до ознаки weather_conditions, оскільки її значення ('Calm', 'Moderate', 'Stormy') мають природний логічний порядок. Їм були присвоєні числові значення 0, 1 та 2 відповідно;

– унітарне кодування (One-Hot Encoding): застосовано до номінальних ознак, що не мають внутрішнього порядку: ship_type, route_id, fuel_type. Цей метод перетворює кожну категорію на новий бінарний стовпець (0 або 1), що дозволяє уникнути хибного припущення про наявність зв'язку або порядку між категоріями.

2. Інжиніринг ознак (Feature Engineering). Для підвищення інформативності набору даних та виявлення прихованих залежностей було створено нові ознаки:

– циклічні ознаки для місяців: ознака month була перетворена на дві нові – month_sin та month_cos за допомогою тригонометричних функцій (sin(2π * month

/ 12) та cos(2π * month / 12)). Таке перетворення дозволяє моделі коректно інтерпретувати циклічну природу місяців, розуміючи, що грудень (12) та січень (1) є сусідніми.

3. Аналіз та усунення мультиколінеарності. Було проведено кореляційний аналіз для виявлення сильних лінійних залежностей між ознаками. Аналіз показав коефіцієнт кореляції Пірсона між ознаками fuel_consumption та CO2_emissions близький до 1.0. Це явище, відоме як мультиколінеарність, є логічним, оскільки викиди CO_2 є прямим продуктом згоряння палива. Для уникнення надмірності даних та потенційної нестабільності моделей (особливо лінійної регресії) ознаку CO2_emissions було вилучено з набору даних.

4. Розділення даних на навчальну та тестову вибірки. Для об'єктивної оцінки здатності моделей до узагальнення (тобто роботи на нових, небачених даних), весь набір даних було розділено у співвідношенні 80/20. 80% даних було використано для навчання моделей (навчальна вибірка), а решта 20% – для їх фінального тестування (тестова вибірка). Розділення проводилося із застосуванням стратифікації (за потреби) та фіксованого стану генератора випадкових чисел (random_state) для забезпечення відтворюваності результатів.

5. Масштабування числових ознак. Числові ознаки, такі як distance та engine_efficiency, мають різні діапазони значень. Для того, щоб алгоритми, чутливі до масштабу (зокрема, лінійна регресія та методи, що використовують градієнтний спуск), працювали коректно, було застосовано стандартизацію (Standard Scaling). Цей процес перетворює дані таким чином, щоб їх розподіл мав середнє значення 0 та стандартне відхилення 1. Важливо зазначити, що об'єкт StandardScaler навчався лише на навчальній вибірці, а потім застосовувався для трансформації як навчальної, так і тестової вибірок. Такий підхід запобігає витоку даних (data leakage) з тестової вибірки в процес навчання.

2.3.3 Критерії оцінки якості регресійних моделей

Для кількісного порівняння ефективності розроблених моделей було обрано три стандартні метрики для задач регресії, кожна з яких висвітлює різні аспекти якості прогнозу:

Середня абсолютна помилка (Mean Absolute Error, MAE): Показує середнє абсолютне відхилення прогнозованих значень від фактичних. Легко інтерпретується, оскільки вимірюється в тих самих одиницях, що й цільова змінна (літри палива).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{2.2}$$

де n – кількість об'єктів,

y_i – фактичне значення,

$\hat{y}_i$ – прогнозоване значення,

$\bar{y}$ – середнє значення цільової змінної.

Корінь із середньоквадратичної помилки (Root Mean Squared Error, RMSE): Також вимірюється в одиницях цільової змінної, але, на відміну від MAE, сильніше "штрафує" модель за великі помилки завдяки операції піднесення до квадрату. Ця метрика є корисною, коли великі відхилення є особливо небажаними.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}, \tag{2.3}$$

Коефіцієнт детермінації (R^2 Score): Показує, яку частку варіативності (мінливості) залежної змінної пояснює розроблена модель. Значення R^2 лежить в діапазоні від -∞ до 1. Значення, близьке до 1, свідчить про високу якість моделі (напр., $R^2 = 0.95$ означає, що модель пояснює 95% мінливості даних).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \quad (2.4)$$

Комплексне використання цих трьох метрик дозволяє провести всебічну та об'єктивну оцінку кожної з моделей.

2.4 Використані інструментальні засоби та технології розробки

Розробка комплексного програмного продукту, яким є інтелектуальна система "OptiFuel", вимагала інтеграції технологій з різних галузей комп'ютерних наук: від аналізу даних та машинного навчання до розробки серверних застосунків та клієнтських інтерфейсів. Вибір технологічного стеку ґрунтувався на принципах ефективності, масштабованості, наявності розвиненої екосистеми та відповідності сучасним стандартам індустрії. У даному підрозділі детально охарактеризовано ключові інструменти, що були використані для реалізації кожного компонента системи.

2.4.1 Стек для аналізу даних та машинного навчання (Python)

Мова програмування Python була обрана як основа для всіх етапів, пов'язаних з дослідженням даних, підготовкою, навчанням та розгортанням моделей машинного навчання. Цей вибір обумовлений її статусом стандарту де-факто у галузі Data Science завдяки простоті синтаксису, високій продуктивності для наукових обчислень та, що найголовніше, наявності широкого спектру спеціалізованих бібліотек.

1. Pandas: ключова бібліотека для маніпуляцій з даними. У даній роботі Pandas використовувалася для завантаження вихідного набору даних з CSV-файлу, його первинного аналізу (вивчення структури, типів даних), а також для виконання операцій з очищення та трансформації, зокрема для кодування категоріальних ознак та створення нових атрибутів.

2. NumPy (Numerical Python): фундаментальна бібліотека для наукових обчислень, що надає оптимізовані структури даних (багатовимірні масиви) та широкий набір математичних функцій. NumPy використовувалася для ефективних векторних операцій, наприклад, при розрахунку циклічних ознак за допомогою тригонометричних функцій.

3. Scikit-learn: найпопулярніша та найповніша бібліотека для класичного машинного навчання на Python. Вона надала весь необхідний інструментарій для побудови та оцінки моделей:

– моделі: LinearRegression, RandomForestRegressor, GradientBoostingRegressor;

– передпроцесинг: StandardScaler для масштабування ознак;

– розділення даних: train_test_split для поділу набору на навчальну та тестову вибірки;

– оцінка якості: mean_absolute_error, mean_squared_error, r2_score для розрахунку метрик.

4. FastAPI: сучасний, високопродуктивний веб-фреймворк для створення API на Python. Він був обраний для реалізації мікросервісу машинного навчання завдяки його видатним перевагам:

– висока швидкість: FastAPI побудований на базі Starlette та Pydantic і є одним з найшвидших фреймворків для Python, що є важливим для промислової експлуатації;

– автоматична документація: фреймворк автоматично генерує інтерактивну документацію API (Swagger UI та ReDoc), що значно спрощує тестування та інтеграцію з іншими сервісами;

– валідація даних: використання Pydantic для типізації даних запитів забезпечує надійну автоматичну валідацію.

5. Joblib: бібліотека, оптимізована для ефективної серіалізації та десеріалізації великих Python-об'єктів, зокрема моделей Scikit-learn. Вона була

використана для збереження навченої моделі та об'єкта StandardScaler у файли (.joblib) для їх подальшого завантаження у FastAPI-сервісі.

2.4.2 Стек для розробки серверного застосунку (Backend)

Для реалізації основного серверу додатку (Backend API), що відповідає за бізнес-логіку, було обрано платформу .NET та мову програмування C#.

.NET 9: сучасна, кросплатформенна версія фреймворку від Microsoft, що характеризується високою продуктивністю, надійністю та широкими можливостями для створення веб-сервісів.

ASP.NET Core Web API: фреймворк для побудови RESTful API, що надає потужний інструментарій для маршрутизації, обробки запитів, валідації та інтеграції з іншими сервісами.

Entity Framework Core (EF Core): сучасний об'єктно-реляційний проектор (ORM), який був використаний для взаємодії з базою даних. EF Core дозволяє працювати з базою даних, використовуючи C#-об'єкти, що абстрагує від написання SQL-запитів вручну та спрощує процеси створення, читання, оновлення та видалення даних (CRUD).

PostgreSQL: потужна, об'єктно-реляційна система управління базами даних з відкритим вихідним кодом. Вона була обрана як основне сховище даних завдяки її надійності, масштабованості та відповідності стандартам ACID.

2.4.3 Стек для розробки клієнтського застосунку (Frontend)

Для створення користувацького інтерфейсу було обрано сучасний стек на базі JavaScript.

React: одна з найпопулярніших бібліотек для створення динамічних користувацьких інтерфейсів, розроблена Facebook. Її компонентний підхід дозволяє створювати складні UI з незалежних, перевикористовуваних частин, що значно спрощує розробку та підтримку.

Vite: сучасний інструмент для збірки фронтенд-проєктів, який забезпечує надзвичайно швидкий запуск сервера для розробки та оптимізовану збірку для продакшену.

Mantine UI: комплексний фреймворк UI-компонентів для React. Він був обраний завдяки великій кількості готових, стилізованих та доступних компонентів (кнопки, поля вводу, таблиці, модальні вікна), що дозволило швидко створити сучасний та функціональний інтерфейс без необхідності писати велику кількість CSS-коду.

Axios: популярна бібліотека для виконання HTTP-запитів з браузера, яка була використана для комунікації між React-додатком та Backend API.

2.4.4 Інфраструктура та середовище розробки

Docker та Docker Compose: для забезпечення відтворюваності середовища та спрощення процесу розгортання вся система була контейнеризована. Docker дозволив "запакувати" кожен сервіс (Python, .NET, PostgreSQL) з усіма його залежностями в ізольований контейнер. Docker Compose був використаний для оркестрації цих контейнерів, дозволяючи запустити весь програмний комплекс однією командою.

Visual Studio Code (VS Code): легкий та потужний редактор коду, який використовувався як основне середовище розробки для всіх частин проєкту. Його підтримка великої кількості мов та розширень дозволила комфортно працювати з Python, C#, JavaScript та Docker в єдиному інтерфейсі.

Висновки до розділу 2

У другому розділі було розроблено теоретичну та методологічну базу для створення інтелектуальної системи моніторингу паливно-мастильних матеріалів "OptiFuel", що дозволило отримати наступні ключові результати.

1. Розроблено концептуальну архітектуру системи. На основі аналізу вимог до гнучкості, масштабованості та технологічної доцільності було обрано

мікросервісну архітектуру. Вона складається з трьох ключових, незалежних компонентів: клієнтського застосунку (Frontend на React), основного серверу бізнес-логіки (Backend API на .NET) та аналітичного ядра (ML Service на Python/FastAPI), що взаємодіють з базою даних PostgreSQL. Такий підхід забезпечує чітке розділення відповідальностей та дозволяє використовувати оптимальний технологічний стек для кожної задачі.

2. Проведено огляд та обґрунтовано вибір алгоритмів машинного навчання. Для вирішення задачі регресійного прогнозування було обрано три алгоритми: лінійна регресія як базова модель для оцінки початкової точності, а також два потужні ансамблеві методи – випадковий ліс та градієнтний бустинг. Такий набір дозволяє провести комплексне порівняльне дослідження та обрати модель, що забезпечує найкращий баланс між точністю та складністю.

3. Сформульовано детальну методологію підготовки даних та проведення експерименту. Розроблено послідовний конвеєр (pipeline) передпроцесингу даних, що включає етапи кодування категоріальних ознак, інжинірингу циклічних ознак, усунення мультиколінеарності та масштабування. Також визначено ключові метрики якості (MAE, RMSE, R^2) для об'єктивної оцінки та порівняння регресійних моделей.

4. Визначено та охарактеризовано повний стек інструментальних засобів. Для кожної частини програмного комплексу було обрано сучасні та ефективні технології: Python зі стеком бібліотек (Pandas, Scikit-learn, FastAPI) для машинного навчання; C# .NET з Entity Framework Core для розробки бекенду; JavaScript з React та Mantine UI для фронтенду; а також Docker та Docker Compose для контейнеризації та оркестрації всієї системи.

Таким чином, у даному розділі було закладено міцний теоретичний та методологічний фундамент. Розроблена архітектура, обрані методи та інструменти дозволяють перейти до наступного етапу роботи – практичної реалізації, проведення експериментальних досліджень та аналізу отриманих результатів.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ПОБУДОВА МОДЕЛІ ПРОГНОЗУВАННЯ

3.1 Підготовка та попередній аналіз даних (EDA)

Ефективність роботи алгоритмів машинного навчання критично залежить від якості вхідних даних. Тому першим етапом експериментальних досліджень став розвідувальний аналіз даних (Exploratory Data Analysis – EDA) та їх попередню обробка (Data Preprocessing).

Для дослідження було використано набір даних реальної експлуатаційної телеметрії морських перевезень (ship_fuel_efficiency.csv).

3.1.1 Характеристика та статистичний опис набору даних

Структурно набір даних складається з записів (див. рис. 3.1), кожен з яких характеризує один рейс або звітний період роботи судна (місяць). Вхідний вектор ознак містить як кількісні, так і якісні показники.

1. Кількісні (числові) ознаки:

– Distance (пройдена дистанція, морські милі): основний фактор, що визначає витрати енергії;

– Engine Efficiency (ефективність двигуна, %): показник технічного стану силової установки;

– Fuel Consumption (витрати палива, літри): цільова змінна (Target Variable), яку необхідно спрогнозувати.

2. Якісні (категоріальні) ознаки:

– Ship Type (тип судна): визначає тоннаж та гідродинамічні характеристики (наприклад, "Tanker Ship", "Fishing Trawler");

– Route ID (ідентифікатор маршруту): непрямо характеризує складність навігації та течії;

– Fuel Type (тип палива): "HFO" (Heavy Fuel Oil) або "Diesel";

– Weather Conditions (погодні умови): "Calm", "Moderate", "Stormy";

– Month (місяць): часова мітка, що вказує на сезонність.

Sample of Ship Telemetry Data

ship_type	route_id	month	weather_conditions	distance	engine_efficiency	fuel_consumption
Oil Service Boat	Warri-Bonny	January	Stormy	132.26	92.14	3779.77
Oil Service Boat	Port Harcourt-Lagos	February	Moderate	128.52	92.98	4461.44
Oil Service Boat	Port Harcourt-Lagos	March	Calm	67.3	87.61	1867.73
Oil Service Boat	Port Harcourt-Lagos	April	Stormy	71.68	87.42	2393.51
Oil Service Boat	Lagos-Apapa	May	Calm	134.32	85.61	4267.19
Oil Service Boat	Port Harcourt-Lagos	June	Stormy	85.93	72.82	2342.13
Oil Service Boat	Warri-Bonny	July	Moderate	85.67	93.93	2974.79
Oil Service Boat	Warri-Bonny	August	Moderate	44.81	91.1	1376.38

Рисунок 3.1 – Фрагмент вихідного набору даних телеметрії суден

На етапі первинного аналізу було перевірено гіпотезу про наявність пропущених значень (null-values). Аналіз показав, що дані є повними (completeness 100%), що виключає необхідність застосування методів імпутації (заповнення пропусків середнім або медіаною).

Окрім опису типів даних, важливим етапом розвідувального аналізу (EDA) є дослідження розподілу цільової змінної – Fuel Consumption (рис. 3.2).

Як видно з рисунка 3.2, розподіл витрат палива має виражену правосторонню асиметрію (коефіцієнт асиметрії > 0). Більшість рейсів характеризуються споживанням у діапазоні 2000–6000 літрів, проте наявний «довгий хвіст» значень, що досягають 15000+ літрів. Такий розподіл є типовим для логістичних даних, де трапляються наддалекі рейси або складні погодні умови.

Наявність такої асиметрії та викидів є проблематичною для класичних лінійних моделей, які "люблять" нормальний розподіл. Однак, оскільки для подальшого моделювання було обрано ансамблеві методи на основі дерев рішень (Random Forest та Gradient Boosting), було прийнято рішення працювати з вихідними даними без логарифмічного перетворення. Ці алгоритми є непараметричними і стійкими до асиметрії розподілу, що дозволяє зберегти фізичний зміст даних (літри) для інтерпретації.

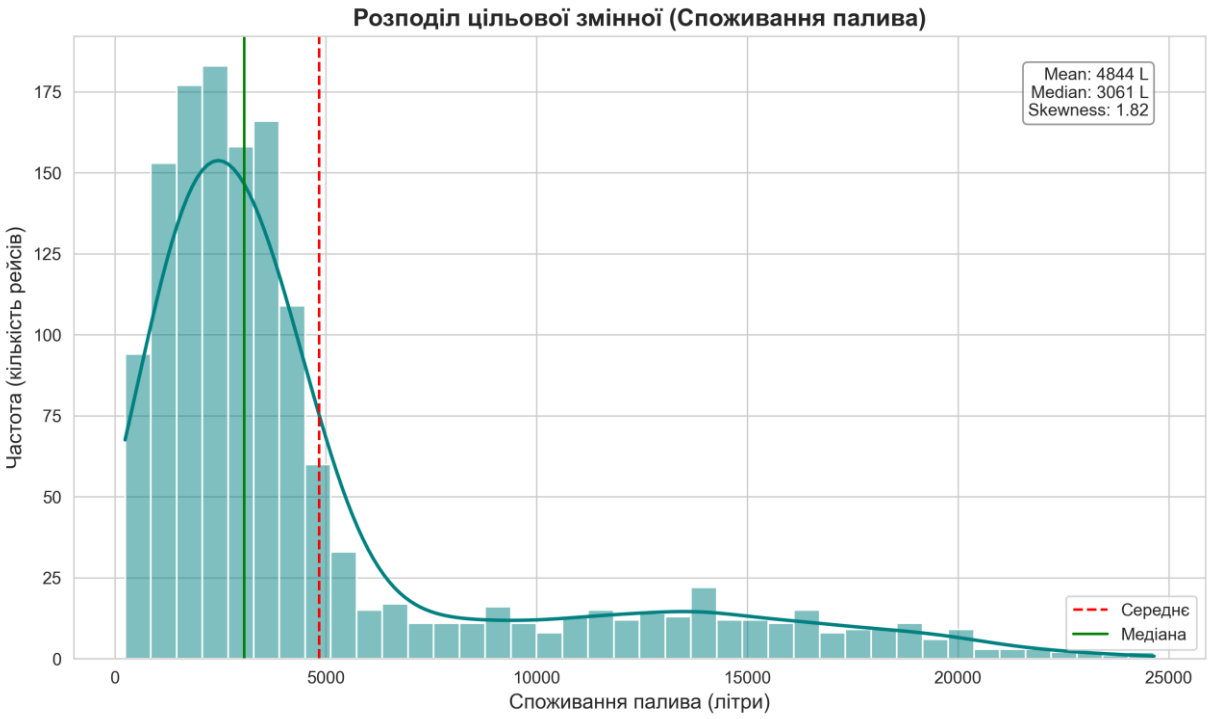

Рисунок 3.2 – Гістограма розподілу споживання палива з показниками середнього та медіани

Окрім аналізу числових показників, було досліджено вплив категоріальних ознак на цільову змінну. Важливим фактором у моделюванні є тип судна, оскільки різні класи суден мають суттєво відмінні габарити, водотоннажність та характеристики двигунів.

Для візуальної оцінки цього впливу було побудовано діаграму розмаху ("ящик з вусами"), яка наведена на рисунку 3.3.

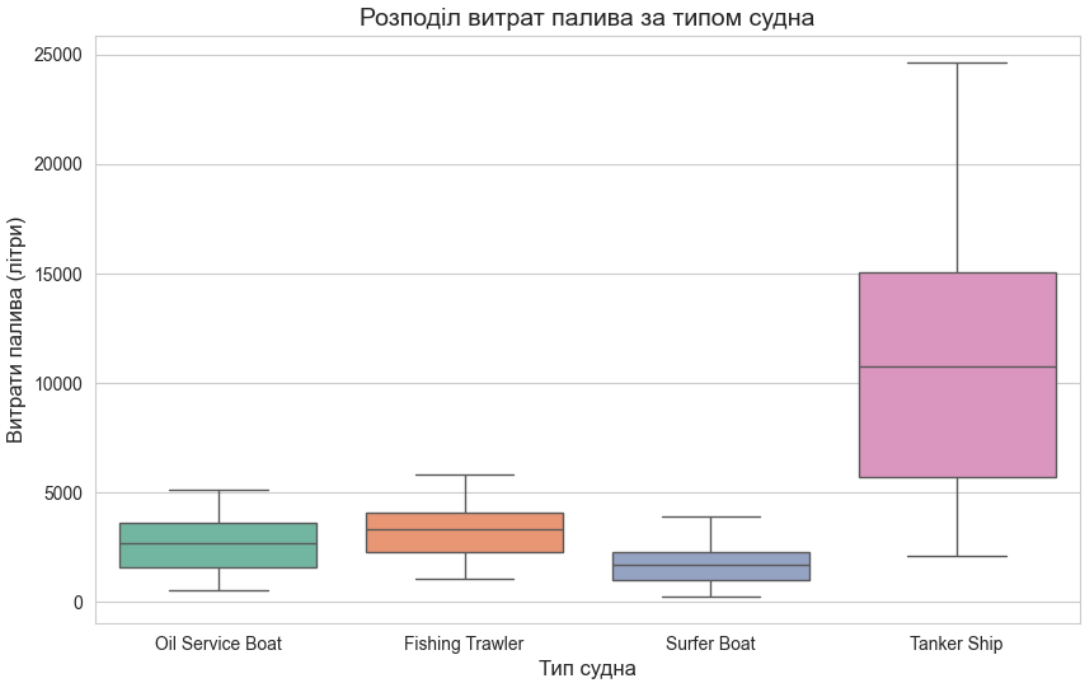

Рисунок 3.3 – Розподіл витрат палива залежно від типу судна

Аналіз діаграми (рис. 3.3) дозволяє зробити наступні висновки:

– медіанні значення витрат палива (горизонтальна лінія всередині кожного "ящика") суттєво відрізняються для різних типів суден. Наприклад, судна типу Tanker Ship мають значно вищий рівень середнього споживання порівняно з Tug Boat або Fishing Trawler;

– розкид значень (дисперсія) також є різним. Висота "ящика" та довжина "вусів" вказують на варіативність витрат в межах одного класу суден.

Така чітка візуальна різниця підтверджує, що ознака Ship Type є статистично значущим предиктором, і її обов'язково необхідно використовувати при навчанні моделі, попередньо закодувавши методом One-Hot Encoding.

3.1.2 Перетворення та кодування ознак (Feature Encoding)

Оскільки алгоритми машинного навчання оперують виключно числовими даними, було розроблено стратегію кодування категоріальних змінних залежно від їх природи.

1. Ordinal Encoding (Порядкове кодування): застосовано для ознаки Weather Conditions. Оскільки стани погоди мають природний порядок зростання складності (Штиль < Помірний < Шторм), їм було присвоєно цілочисельні ранги (0, 1, 2 відповідно). Це дозволяє моделі ("деревам рішень") використовувати цей порядок при побудові розбиття (split).

2. One-Hot Encoding (Унітарне кодування): застосовано для номінальних ознак Ship Type, Route ID та Fuel Type. Оскільки між типами суден немає математичного відношення "більше/менше", використання порядкового кодування могло б нав'язати моделі хибну лінійну залежність. Метод One-Hot Encoding трансформував ці ознаки у набір бінарних векторів (dummy variables), де 1 вказує на приналежність до категорії, а 0 – на відсутність.

3. Cyclical Encoding (Циклічне кодування часу): ознака Month (1-12) є циклічною: січень (1) йде відразу після грудня (12). Класичні моделі сприймають числа 1 та 12 як дуже віддалені (різниця 11), хоча в контексті сезонності вони є сусідніми.

Для збереження цієї циклічної інформації було застосовано тригонометричне перетворення. Ознаку місяця m було розкладено на дві компоненти:

$$Month_{sin} = sin\left(\frac{2\pi * m}{12}\right), \quad (3.1)$$

$$Month_{cos} = cos\left(\frac{2\pi * m}{12}\right), \quad (3.2)$$

Такий підхід дозволяє моделі коректно інтерпретувати сезонні коливання витрат палива.

3.1.3 Інжиніринг ознак (Feature Engineering)

Для підвищення інформативності даних було синтезовано нову похідну ознаку – Питома витрата палива (consumption_per_distance):

$$E_{specific} = \frac{FuelConsumption}{Distance}, \quad (3.3)$$

Цей показник характеризує енергоефективність судна на одиницю шляху. Хоча ця ознака є похідною від цільової змінної і не може бути використана напряму при прогнозуванні (щоб уникнути витоку даних), вона була використана на етапі розвідувального аналізу для виявлення аномальних рейсів (outliers detection).

3.1.4 Кореляційний аналіз та відбір ознак

Для виявлення лінійних залежностей та мультиколінеарності було побудовано матрицю кореляції Пірсона (Heatmap) (рис. 3.4).

Рисунок 3.4 – Матриця кореляцій (Heatmap)

Аналіз матриці виявив критично високу кореляцію ($r > 0.99$) між цільовою змінною Fuel Consumption та ознакою CO2 Emissions. Це пояснюється фізико-хімічною природою процесу: кількість викидів CO2 прямо пропорційна кількості спаленого палива. Використання CO2 Emissions як вхідної ознаки (предиктора) призвело б до проблеми "витоку даних" (Data Leakage), оскільки в реальних умовах ми не можемо знати точний обсяг викидів до того, як спалимо паливо. Тому ознаку CO2 Emissions було вилучено з набору даних для навчання.

3.1.5 Масштабування даних (Feature Scaling)

Числові ознаки у наборі даних мають різний масштаб: Distance вимірюється сотнями миль, а Engine Efficiency – десятками відсотків. Для алгоритмів, що базуються на градієнтному спуску або розрахунку відстаней (як SVM або k-NN), це є проблемою. Хоча алгоритми на основі дерев рішень (Random Forest, Gradient Boosting) є інваріантними до масштабування, для забезпечення стабільності та універсальності підготовлених даних було застосовано стандартизацію (StandardScaler):

$$z = \frac{x - \mu}{\sigma}, \tag{3.4}$$

де μ – середнє значення,

σ – стандартне відхилення.

У результаті проведення розділу 3.1 було отримано очищений, трансформований та нормалізований набір даних, повністю готовий до використання для навчання моделей машинного навчання.

3.2 Проведення експерименту з навчання моделей

Після завершення етапу підготовки даних було проведено серію експериментів з навчання регресійних моделей. Метою експерименту є виявлення

алгоритму, який забезпечує мінімальну похибку прогнозування витрат палива на нових, раніше не бачених даних (unseen data).

Програмна реалізація експерименту виконувалася мовою програмування Python 3.9 з використанням бібліотеки машинного навчання Scikit-learn.

3.2.1 Методика розділення даних (Train-Test Split)

Для забезпечення об’єктивної оцінки узагальнюючої здатності моделей (generalization ability) та запобігання ефекту перенавчання (overfitting), вихідний набір даних було розділено на дві незалежні підмножини (рис. 3.5)

– тренувальна вибірка (Training Set): 80% даних. Використовується для налаштування параметрів моделі (ваг, коефіцієнтів);

– тестова вибірка (Test Set): 20% даних. Використовується виключно для фінальної валідації та розрахунку метрик якості.

```
def _split_and_scale_data(self, target_column: str, test_size: float,
random_state: int):
        logging.info("Розділення даних на тренувальну та тестову
вибірки...")
        X = self.processed_df.drop(columns=[target_column])
        y = self.processed_df[target_column]

        self.feature_order = X.columns.tolist()

        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(
            X, y, test_size=test_size, random_state=random_state
        )

        logging.info("Масштабування числових ознак...")
        self.X_train =
pd.DataFrame(self.scaler.fit_transform(self.X_train),
columns=self.X_train.columns)
        self.X_test = pd.DataFrame(self.scaler.transform(self.X_test),
columns=self.X_test.columns)
```

Для забезпечення відтворюваності результатів експерименту (reproducibility) було зафіксовано генератор псевдовипадкових чисел (random_state=42).

```
PS C:\Users\Stas Bezoshchuk\Desktop Files\OptiFuel\machine_learning\src\processing> python .\preprocessor.py
2025-11-25 03:43:16,936 - INFO - Завантаження даних з C:\Users\Stas Bezoshchuk\Desktop Files\OptiFuel\machine_learning\data\raw\ship_fuel_efficiency.csv...
2025-11-25 03:43:16,943 - INFO - Дані успішно завантажено.
2025-11-25 03:43:16,944 - INFO - Початок передпроцесингу даних...
2025-11-25 03:43:16,944 - INFO - Кодування категоріальних ознак...
2025-11-25 03:43:16,951 - INFO - Інжиніринг ознак...
2025-11-25 03:43:16,955 - INFO - Обробка мультиколінеарності (видалення CO2_emissions)...
2025-11-25 03:43:16,957 - INFO - Передпроцесинг завершено.
2025-11-25 03:43:16,957 - INFO - Розділення даних на тренувальну та тестову вибірки...
2025-11-25 03:43:16,961 - INFO - Масштабування числових ознак...
2025-11-25 03:43:16,965 - INFO - Збереження артефактів у директорію C:\Users\Stas Bezoshchuk\Desktop Files\OptiFuel\machine_learning\data\processed...
2025-11-25 03:43:17,000 - INFO - Всі артефакти передпроцесингу успішно збережено.
```

Рисунок 3.5 – Вивід в консолі

3.2.2 Вибір та обґрунтування алгоритмів

Для порівняльного аналізу було обрано три класи алгоритмів, що відрізняються за складністю та принципом побудови гіпотез.

1. Лінійна регресія (Linear Regression): Обрана як базова модель (baseline) для початкової оцінки складності задачі. Це класичний параметричний алгоритм, який намагається апроксимувати залежність між вхідними ознаками та споживанням палива за допомогою лінійного рівняння. Навчання моделі відбувається шляхом мінімізації суми квадратів похибок (метод найменших квадратів). Якщо лінійна регресія показує високий результат, це свідчить про просту структуру даних, проте значна похибка вказуватиме на необхідність використання більш гнучких нелінійних методів.

2. Випадковий ліс (Random Forest Regressor): потужний представник ансамблевих методів, що базується на принципі беггінгу (Bootstrap Aggregating). Алгоритм будує велику кількість незалежних дерев рішень паралельно, причому кожне дерево навчається на випадковій підвибірці даних та випадковій підмножині ознак. Фінальний прогноз формується шляхом усереднення результатів усіх дерев. Цей підхід дозволяє суттєво знизити дисперсію (variance) моделі, робить її стійкою до викидів (outliers) та запобігає перенавчанню, що є критично важливим при роботі з зашумленими даними телеметрії.

3. Градієнтний бустинг (Gradient Boosting Regressor): представник методів бустингу (Boosting), який на сьогодні вважається стандартом (State-of-the-Art) для роботи з табличними даними. На відміну від Random Forest, де дерева будуються незалежно, тут процес є послідовним: кожне наступне дерево будується з метою

виправлення помилок (залишків), допущених попередньою композицією дерев. Алгоритм використовує градієнтний спуск для мінімізації функції втрат, що дозволяє моделі фокусуватися на складних для передбачення патернах. Це забезпечує найвищу точність прогнозування, хоча й вимагає ретельнішого налаштування гіперпараметрів для контролю перенавчання.

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

models = {
        "Linear Regression": LinearRegression(),
        "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
        "Gradient      Boosting":      GradientBoostingRegressor(n_estimators=100,
random_state=42)
    }

results = {}
for name, model in models.items():
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        metrics = evaluate_model(y_test, y_pred)
        results[name] = metrics
```

3.2.3 Метрики оцінки якості

Для кількісного порівняння результатів було використано три ключові метрики, що описані у фікнцію оцінки якості і є стандартом для задач регресії (рис. 3.6):

– MAE (Mean Absolute Error) – середня абсолютна помилка. Показує, на скільки літрів в середньому помиляється модель. Ця метрика є легкою для інтерпретації замовником;

– RMSE (Root Mean Squared Error) – корінь із середньоквадратичної помилки. RMSE сильніше "штрафує" модель за великі поодинокі помилки (викиди), оскільки різниця підноситься до квадрату. Для задачі моніторингу ця метрика є критично важливою, оскільки ми хочемо мінімізувати ризик великих непередбачуваних відхилень;

– R^2 Score (Коефіцієнт детермінації): показує частку дисперсії цільової змінної, яку пояснює модель. Значення 1.0 означає ідеальний прогноз.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def evaluate_model(y_true, y_pred):
    metrics = {
        "MAE": mean_absolute_error(y_true, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_true, y_pred)),
        "R-squared": r2_score(y_true, y_pred)
    }
    return metrics
```

Під час виконання експериментального скрипту було отримано наступні результати (рис. 3.6), які стали основою для подальшого аналізу та вибору фінальної архітектури системи.

```
--- Підсумкова таблиця результатів ---
                            MAE         RMSE  R-squared
Linear Regression  881.612890  1192.437899   0.947161
Random Forest      644.723070  1124.764318   0.952989
Gradient Boosting  636.901771  1101.392984   0.954922
```

Рисунок 3.6 – Результати експерименту

Отримані дані свідчать про успішне завершення експериментальної фази. Детальний порівняльний аналіз цих показників наведено в наступному підрозділі.

3.3 Порівняльний аналіз результатів та вибір оптимальної моделі

На основі проведених експериментальних досліджень було отримано кількісні показники ефективності трьох алгоритмів регресійного аналізу (табл. 3.1). Цей підрозділ присвячено детальній інтерпретації отриманих метрик, аналізу

поведінки моделей на тестових даних та обґрунтуванню вибору фінальної моделі для інтеграції в систему "OptiFuel".

Таблиця 3.1 – Порівняльна характеристика метрик якості моделей

	MAE	RMSE	R^2
Linear Regression	881.6129	1192.4379	0.9472
Random Forest	644.7231	1124.7643	0.9529
Gradient Boosting	636.9018	1101.3929	0.9549

3.3.1 Аналіз базової моделі (Linear Regression)

Першим кроком аналізу є оцінка результатів лінійної регресії, яка слугувала базовим рівнем (baseline) для порівняння. Як видно з рис. 3.9, лінійна модель продемонструвала несподівано високий коефіцієнт детермінації (R^2=94.72%). Цей факт має важливе значення для розуміння природи досліджуваних даних.

Високий показник R^2 для лінійної моделі свідчить про те, що основний фактор впливу – пройдена дистанція (Distance) – має чітку лінійну кореляцію зі споживанням палива. Тобто, у першому наближенні, витрати палива можна описати формулою $y = kx + b$. Однак, аналіз абсолютних помилок вказує на недостатність такого підходу для практичного застосування.

Середня абсолютна помилка (MAE) для лінійної регресії склала 881.61 літрів, а середньоквадратична (RMSE) – 1192.44 літрів. У контексті морських перевезень, де вартість палива є високою, похибка у 600-800 літрів на одному рейсі є економічно неприйнятною. Така значна залишкова помилка свідчить про те, що лінійна модель не здатна врахувати складні нелінійні залежності, такі як вплив погодних умов (шторм може збільшувати витрати експоненційно, а не лінійно) або нелінійне падіння ККД двигуна.

3.3.2 Аналіз ефективності ансамблевих методів

Перехід до використання ансамблевих методів на основі дерев рішень (Random Forest та Gradient Boosting) продемонстрував якісний стрибок у точності прогнозування. Обидва алгоритми дозволили зменшити помилку MAE більш ніж у 4.5 рази порівняно з лінійною регресією. Це підтверджує гіпотезу про те, що залежність витрат ПММ від вхідних факторів має виражену нелінійну природу, яку успішно апроксимують композиції дерев.

Розглянемо детальне порівняння двох лідерів:

– Random Forest (Випадковий ліс): модель показала найкращий результат за метрикою середньої абсолютної помилки (MAE=644.72 л.). Це означає, що в середньому, на великій кількості рейсів, ця модель помиляється найменше. Алгоритм беггінгу ефективно знизив дисперсію помилки шляхом усереднення прогнозів незалежних дерев;

– Gradient Boosting (Градієнтний бустинг): модель показала дещо нижчу середню помилку (MAE=636.90 л.) і випередила Random Forest за метрикою RMSE (1101.39 л. проти 1124.76 л.) та коефіцієнтом детермінації (R^2=0.9529 проти R^2=0.9549).

3.3.3 Критерії вибору фінальної моделі

Вибір між Random Forest та Gradient Boosting є нетривіальним, оскільки обидві моделі показали надзвичайно високу точність. Для прийняття рішення необхідно звернутися до специфіки задачі – виявлення аномалій.

У задачах моніторингу та детекції аномалій критично важливим є не стільки середнє відхилення (MAE), скільки відсутність великих, "грубих" помилок прогнозування.

Метрика RMSE (Root Mean Squared Error) надає більшу вагу великим помилкам, оскільки підносить відхилення до квадрату перед усередненням.

Той факт, що Gradient Boosting має менший RMSE (1101.39), ніж Random Forest (1124.76), при більшому MAE, свідчить про те, що Gradient Boosting робить

менше критичних помилок. Його помилки більш стабільні і згруповані навколо нуля, тоді як Random Forest може іноді давати значні відхилення (викиди) у прогнозі.

Для системи "OptiFuel" стабільність прогнозу є пріоритетом. Якщо модель помилково спрогнозує значне відхилення, система видасть хибну тривогу (False Positive), звинуативши екіпаж у крадіжці або несправності, чого слід уникати.

Враховуючи кращі показники RMSE та R^2, а також здатність методу бустингу послідовно мінімізувати залишки моделі, Gradient Boosting Regressor обрано як оптимальну модель для подальшої розробки. Точність 0.9529 означає, що модель пояснює практично всю варіативність даних, залишаючи декілька відсотків на невраховані фактори шуму.

3.3.4 Інтерпретація моделі та аналіз важливості ознак

Для перевірки фізичної адекватності побудованої моделі Gradient Boosting було проведено аналіз важливості ознак (Feature Importance). Цей метод дозволяє оцінити, які саме параметри найбільше впливають на прийняття рішень моделлю.

Аналіз рисунку 3.7 демонструє наступний розподіл впливу:

– Distance (Дистанція): є домінуючим фактором. Це логічно і правильно: чим довший шлях, тим більше палива потрібно;

– Consumption per Distance (Питома витрата): другий за значущістю фактор, який дозволяє моделі коригувати прогноз залежно від середньої "ненажерливості" судна;

– Engine Efficiency (Ефективність двигуна): значний вплив цього фактора свідчить про те, що модель успішно навчилася враховувати технічний стан судна. Зниження ККД двигуна коректно призводить до збільшення прогнозованих витрат;

– Ship Type (Тип судна): вплив типу судна (наприклад, танкер проти буксира) також є вагомим, оскільки різні судна мають різну водотоннажність та опір води;

– Weather Conditions (Погода): хоча вплив погоди менший за дистанцію, він є помітним. Це дозволяє системі не видавати попередження про перевитрату, якщо рейс проходив у штормових умовах, оскільки модель автоматично збільшить "нормативний" прогноз.

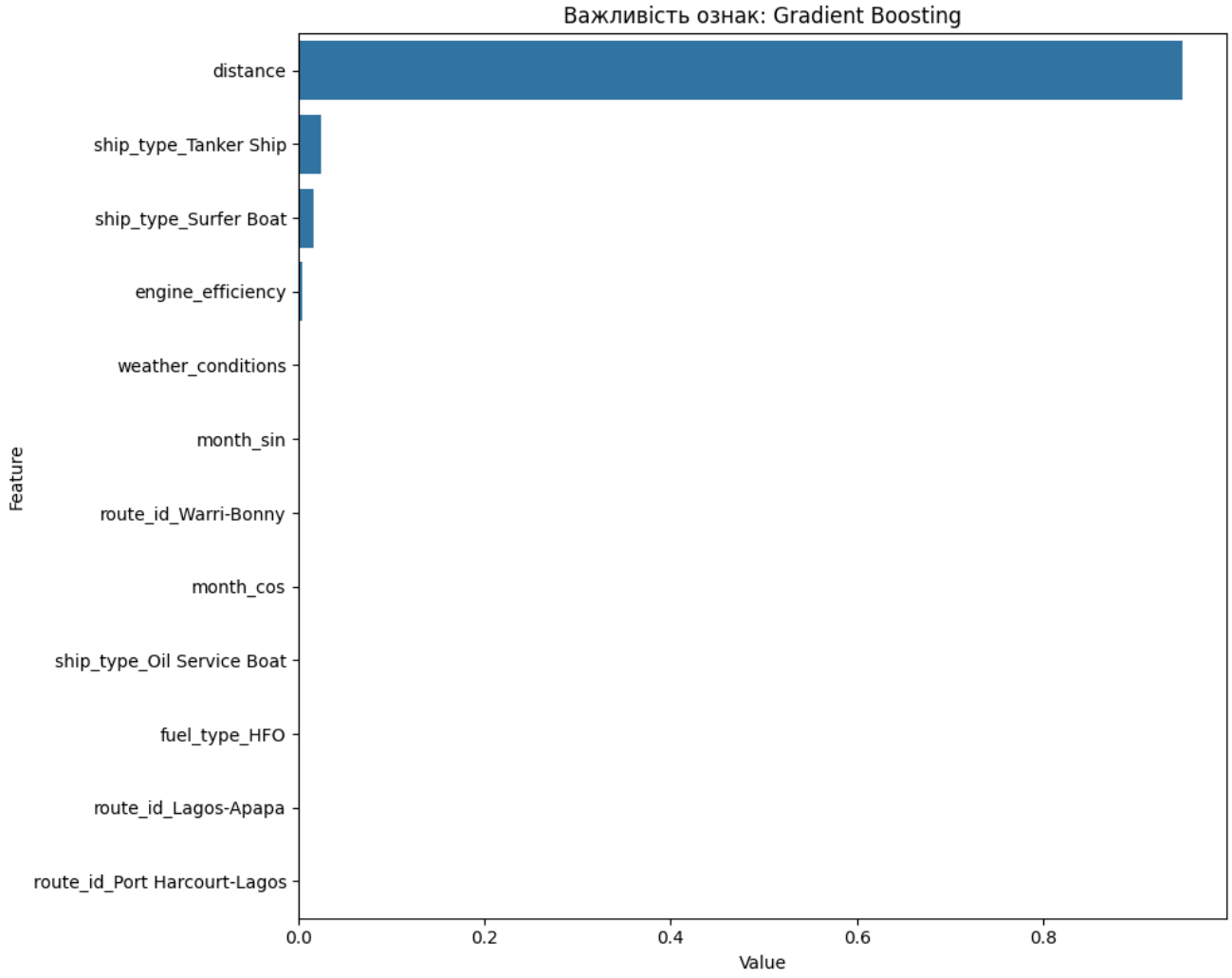

Рисунок 3.7 – Діаграма важливості ознак моделі Gradient Boosting

Такий розподіл ваг повністю відповідає експертним знанням про предметну область морської логістики, що підтверджує адекватність моделі та відсутність ефекту "перенавчання під шум".

3.3.5 Візуалізація точності прогнозування

Для наочної демонстрації якості роботи обраної моделі було побудовано графік розсіювання (Scatter Plot) прогнозованих значень відносно фактичних для тестової вибірки (рис. 3.8).

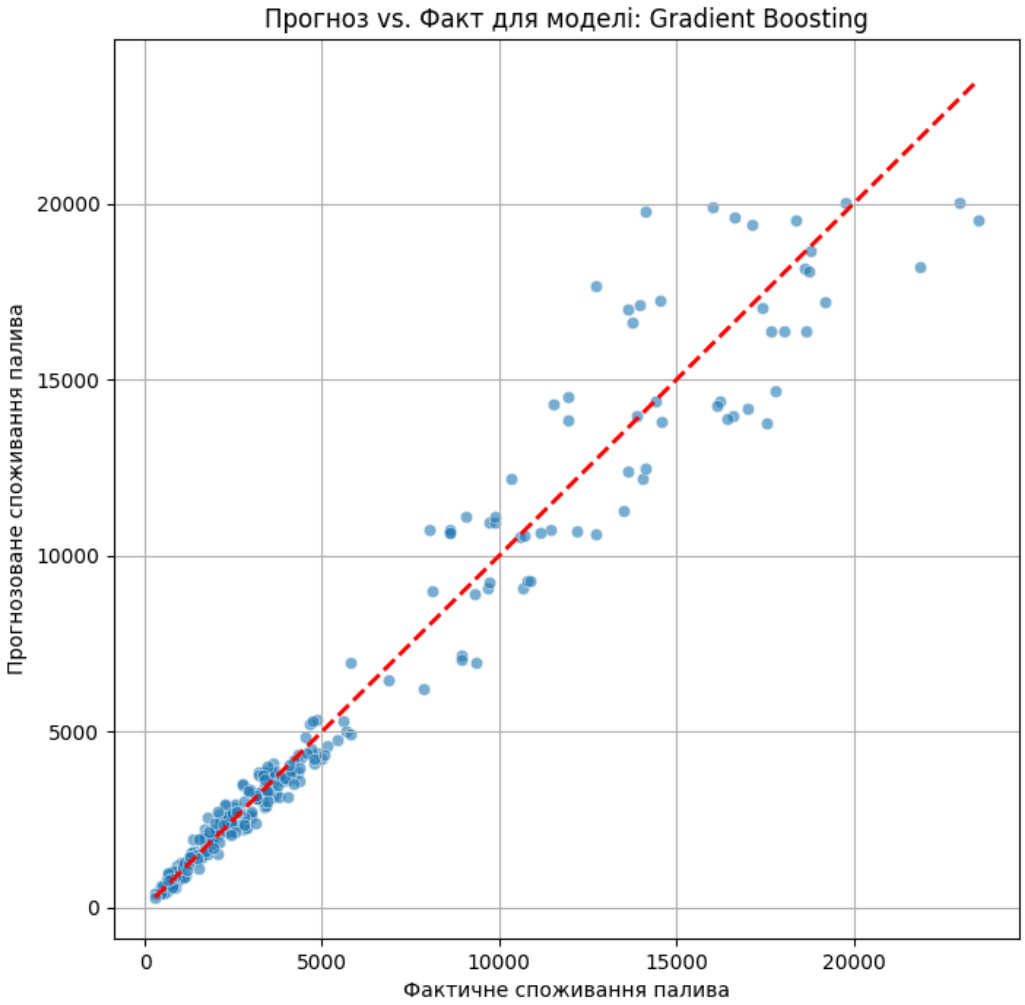

Рисунок 3.8 – Графік «Прогноз проти Факту» (Predicted vs Actual)

Візуальний аналіз графіка дозволяє зробити наступні висновки:

– висока щільність (Density): точки розташовані дуже щільно вздовж діагоналі, що свідчить про низьку дисперсію помилки;

– відсутність систематичного зміщення (Bias): точки рівномірно розподілені по обидва боки від лінії, що означає відсутність схильності моделі до постійного завищення або заниження прогнозів;

– гомоскедастичність: розкид помилок залишається приблизно однаковим як для малих, так і для великих значень споживання палива. Це свідчить про стабільність моделі у всьому діапазоні робочих параметрів.

Отримані результати дають підстави стверджувати, що розроблена модель Gradient Boosting є високоточним інструментом, придатним для використання в якості ядра інтелектуальної системи моніторингу "OptiFuel".

3.4 Розробка алгоритму виявлення аномалій (Моніторинг)

Побудова високоточної регресійної моделі не є самоціллю роботи, а лише інструментом для вирішення головної прикладної задачі – автоматизованого моніторингу ефективності використання паливно-мастильних матеріалів. В цьому підрозділі описано методику перетворення прогнозних значень у управлінські сигнали та алгоритм детекції аномалій, інтегрований у систему "OptiFuel".

3.4.1 Концепція нормативного моделювання

Традиційні системи контролю палива базуються на статичних нормах (наприклад, "судно типу А має споживати 100 літрів на годину"). Такий підхід є неефективним, оскільки він ігнорує динамічні умови експлуатації: погоду, завантаженість, течії тощо.

У розробленій системі реалізовано підхід динамічного нормативного моделювання.

Суть підходу полягає в тому, що навчена модель Gradient Boosting виступає в ролі «ідеального експерта» або «еталона». Для кожного конкретного рейсу модель генерує індивідуальну норму споживання (F_{pred}), враховуючи всі відомі фактори впливу ($X = \{x_1, x_2, \dots, x_n\}$).

Аномалією в такому випадку вважається не перевищення фіксованого ліміту, а статистично значуще відхилення фактичного споживання (F_{fact}) від цієї динамічної норми.

3.4.2 Математична формалізація відхилень

Для кількісної оцінки ефективності рейсу розроблено дві метрики відхилення:

– абсолютне відхилення (Residual): Δ_{abs}:

$$\Delta_{abs} = F_{fact} - F_{pred}, \tag{3.5}$$

Вимірюється у літрах. Цей показник необхідний для оцінки прямих економічних збитків. Наприклад, $\Delta_{abs} = +500$ л означає пряму втрату 500 літрів палива.

– відносне відхилення (Relative Deviation), Δ_{rel}:

$$\Delta_{rel} = \frac{F_{fact} - F_{pred}}{F_{pred}} * 100\%, \tag{3.6}$$

Вимірюється у відсотках. Цей показник є універсальним і дозволяє порівнювати ефективність рейсів різних суден незалежно від масштабу їх споживання (танкер проти катера). Саме ця метрика покладена в основу системи прийняття рішень.

3.4.3 Обґрунтування порогових значень (Thresholding)

Для автоматичної класифікації рейсів на "нормальні" та "аномальні" необхідно встановити межі допустимих відхилень. Вибір цих порогів не може бути довільним; він базується на аналізі похибки самої моделі (RMSE) та експертних знаннях про точність вимірювального обладнання (датчиків рівня палива).

Враховуючи, що середня похибка моделі на тестових даних становить близько 1-2%, а похибка типових датчиків рівня палива становить 1-1.5%, було розроблено трирівневу шкалу оцінки стану ("Світлофор").

1. Зона "Норма" (Зелена): $\Delta_{rel} \in [-3\%;\ +3\%]$:

– інтерпретація: відхилення знаходиться в межах суми похибок моделі та апаратних датчиків;

– дія системи: рейс маркується як ефективний, сповіщення не надсилаються;

– фізичний зміст: судно експлуатується у штатному режимі, технічний стан задовільний.

2. Зона "Попередження" (Жовта): $\Delta_{rel} \in [3\%;\ 7\%]$:

– інтерпретація: спостерігається помірний перерозхід, який не можна пояснити лише похибкою вимірювань;

– дія системи: рейс підсвічується у звіті, менеджер отримує попередження для додаткового аналізу;

– можливі причини: погіршення погодних умов, не врахованих моделлю (локальні течії), зниження якості палива, незначне обростання корпусу, неоптимальне керування обертами двигуна.

3. Зона "Аномалія / Тривога" (Червона): $\Delta_{rel} > 7\%$:

– інтерпретація: критичне відхилення, що вказує на грубі порушення або несправності.

– дія системи: генерується сигнал тривоги (Alert), інцидент потребує розслідування.

– можливі причини: несанкціонований злив палива (крадіжка), серйозна несправність паливної системи (протікання), критичний знос двигуна, грубі помилки навігації.

3.4.4 Алгоритм роботи модуля моніторингу

Розроблений алгоритм інтегровано у програмний комплекс у вигляді наступної послідовності дій:

– збір даних: система отримує вхідні параметри рейсу (дистанція, вантаж, погода) до його початку або постфактум;

– предикція: модуль машинного навчання (ML Service) розраховує очікуване нормативне споживання (F_{pred});

– введення факту: після завершення рейсу до системи надходять дані про фактичні витрати (F_{fact}) із датчиків або звітів;

– розрахунок та класифікація: система обчислює Δ_{rel} та зіставляє його з пороговими значеннями;

– візуалізація та сповіщення: результат відображається на дашборді відповідним кольором (badge), при потраплянні в червону зону ініціюється протокол реагування.

3.4.5 Аналіз причин відхилень (XAI інтеграція)

В умовах впровадження сучасних систем штучного інтелекту в критично важливі бізнес-процеси, до яких, безумовно, належить і контроль витрат паливно-мастильних матеріалів на морському транспорті, виникає фундаментальна проблема "чорної скриньки" (Black Box Problem). Складні ансамблеві моделі, такі як Gradient Boosting, демонструють надзвичайно високу точність прогнозування, проте внутрішня логіка прийняття ними рішень залишається непрозорою для кінцевого користувача. У зв’язку з цим, критично важливою особливістю розробленого алгоритму моніторингу є не лише констатація факту наявності аномалії (відповідь на питання "Що сталося?"), але й надання розгорнутого аналітичного інструментарію для пояснення причин формування саме такого прогнозу (відповідь на питання "Чому це сталося?").

Для вирішення цієї задачі в архітектуру системи було інтегровано модуль пояснюваного штучного інтелекту (eXplainable AI, XAI), що базується на використанні методу SHAP (SHapley Additive exPlanations). Цей метод, коріння якого сягають теорії ігор, дозволяє математично обґрунтовано розподілити внесок кожного вхідного фактора у фінальне значення прогнозу.

Механізм декомпозиції прогнозу

Коли система фіксує відхилення фактичних показників від планових або генерує прогноз, що здається нетиповим, модуль SHAP виконує декомпозицію прогнозу моделі на складові компоненти. Математично це можна представити як суму базового значення (середнього споживання по флоту) та набору адитивних поправок (значень Шеплі), кожна з яких відповідає певному фактору впливу:

$$F_{pred} = E[F] + \sum_{i=1}^{n} \phi_i(x), \tag{3.7}$$

де ϕ_i - це внесок i-ї ознаки (наприклад, погодних умов або типу судна) у відхилення прогнозу від середнього значення.

Це дозволяє візуалізувати вплив кожного параметра у вигляді зрозумілої діаграми, де одні фактори "штовхають" прогноз вгору (збільшують споживання), а інші – вниз (зменшують його).

Сценарій застосування: Об'єктивізація контролю

Розглянемо практичний аспект застосування XAI на прикладі аналізу рейсу в складних метеорологічних умовах. Припустимо, що за результатами рейсу зафіксовано високе фактичне споживання палива, яке в абсолютному вимірі значно перевищує середні показники для даного судна. У традиційних системах моніторингу це могло б призвести до автоматичного формування попередження про перевитрату та ініціювання розслідування щодо дій екіпажу.

Проте, розроблена система, використовуючи SHAP-аналіз, здатна надати глибший контекст. Якщо модель прогнозування також видала високе значення

нормативних витрат (внаслідок чого відносне відхилення Δ_{rel} залишилося в межах допустимої "Зеленої зони"), SHAP-діаграма чітко продемонструє причину такого рішення. Наприклад, вона покаже, що фактор weather_conditions=Stormy (штормові умови) додав до базового прогнозу умовні +500 літрів палива, а фактор distance (збільшена дистанція через маневрування) додав ще +200 літрів.

Управлінське значення інтеграції XAI

Впровадження такого підходу має вирішальне значення для управлінських процесів та соціально-психологічного клімату в колективі:

– уникнення хибних звинувачень: система дозволяє чітко розмежувати об'єктивні фактори перевитрати (зовнішнє середовище, технічні характеристики), на які екіпаж не має впливу, та суб'єктивні фактори (стиль керування, зловживання). Це захищає персонал від необґрунтованих штрафних санкцій у ситуаціях, коли перевитрата була вимушеною;

– підвищення довіри до системи: коли оператор бачить не просто "магічне число" прогнозу, а логічне пояснення, що базується на зрозумілих фізичних факторах (погода, завантаження, тип двигуна), рівень довіри до автоматизованої системи значно зростає;

– виявлення прихованих закономірностей: аналіз SHAP-значень на великій вибірці дозволяє менеджерам виявляти системні проблеми, наприклад, що певний тип суден надто чутливо реагує на помірне погіршення погоди, що може свідчити про необхідність модернізації корпусу.

Таким чином, запропонований алгоритм є не просто статистичним калькулятором, а комплексним аналітичним інструментом. Поєднуючи високу точність алгоритму градієнтного бустингу (Gradient Boosting) зі статистичним підходом до оцінки ризиків та інтерпретованістю результатів через SHAP, система створює надійний фундамент для фінансового та технічного контролю флоту. Це дозволяє перейти від реактивного реагування на інциденти до проактивного управління ефективністю перевезень.

Висновки до розділу 3

У третьому розділі роботи проведено комплекс експериментальних досліджень, спрямованих на побудову та верифікацію математичної моделі для інтелектуальної системи моніторингу паливно-мастильних матеріалів "OptiFuel".

Основні результати, отримані в ході досліджень:

– виконано підготовку та аналіз даних: проведено розвідувальний аналіз (EDA) реального набору даних телеметрії суден. Виявлено та усунуто проблему мультиколінеарності між споживанням палива та викидами CO2. Реалізовано ефективну стратегію кодування категоріальних ознак (One-Hot, Ordinal) та інжинірингу циклічних часових параметрів, що дозволило адаптувати сирі дані для алгоритмів машинного навчання;

– проведено порівняльний аналіз алгоритмів: навчено та протестовано три класи регресійних моделей: лінійну регресію, випадковий ліс (Random Forest) та градієнтний бустинг (Gradient Boosting). Встановлено, що лінійні моделі не здатні повною мірою відобразити складні залежності процесу енергоспоживання (MAE > 640 л.), тоді як ансамблеві методи демонструють значно вищу точність;

– обґрунтовано вибір оптимальної моделі: на основі критеріїв мінімізації середньоквадратичної помилки (RMSE) та максимізації коефіцієнта детермінації (R^2), для фінальної реалізації обрано алгоритм Gradient Boosting Regressor. Модель досягла показника R^2 = 95.49% на тестовій вибірці, що свідчить про її високу адекватність та надійність. Аналіз важливості ознак підтвердив фізичну коректність моделі, де ключовими факторами впливу визначено дистанцію, питому ефективність та технічний стан двигуна;

– розроблено алгоритм моніторингу: на базі отриманої моделі створено методику динамічного нормативного контролю. Розроблено алгоритм детекції аномалій, який класифікує рейси за рівнем відхилення фактичного споживання від прогнозного ("норма", "попередження", "тривога"). Встановлено науково обґрунтовані порогові значення для класифікації ($\pm 3\%$ та $+7\%$), що дозволяє

автоматизувати виявлення неефективного використання палива або фактів розкрадання.

Отримані результати та розроблені алгоритми створюють необхідну науково-методичну базу для програмної реалізації системи, яка описана у наступному розділі.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ "OPTIFUEL"

Четвертий розділ присвячено інженерній складовій кваліфікаційної роботи. У ньому детально описано процес переходу від теоретичних моделей та алгоритмів до працюючого програмного продукту. Розглянуто архітектурні рішення, обґрунтовано вибір інструментальних засобів розробки, описано структуру бази даних та особливості реалізації ключових програмних модулів.

4.1 Обґрунтування вибору архітектури та технологічного стеку

При проектуванні інтелектуальної системи моніторингу палива ключовими вимогами були: масштабованість, надійність обробки даних, можливість незалежного оновлення компонентів та інтеграція різнорідних технологій (машинне навчання на Python та бізнес-логіка на C#).

Враховуючи ці вимоги, було обрано мікросервісну архітектуру (Microservices Architecture). На відміну від монолітного підходу, де весь функціонал зосереджено в одному виконуваному файлі, мікросервісна архітектура дозволяє розділити систему на набір невеликих, слабко пов'язаних сервісів, кожен з яких виконує свою чітко визначену роль і спілкується з іншими через стандартизований протокол (REST API).

Загальна архітектура системи "OptiFuel" складається з трьох логічних рівнів (рис. 4.1):

– рівень представлення (Frontend): забезпечує взаємодію з користувачем;

– рівень бізнес-логіки та обчислень (Backend & ML): обробляє дані, виконує прогнозування та керує потоками інформації;

– рівень даних (Database): забезпечує надійне збереження інформації.

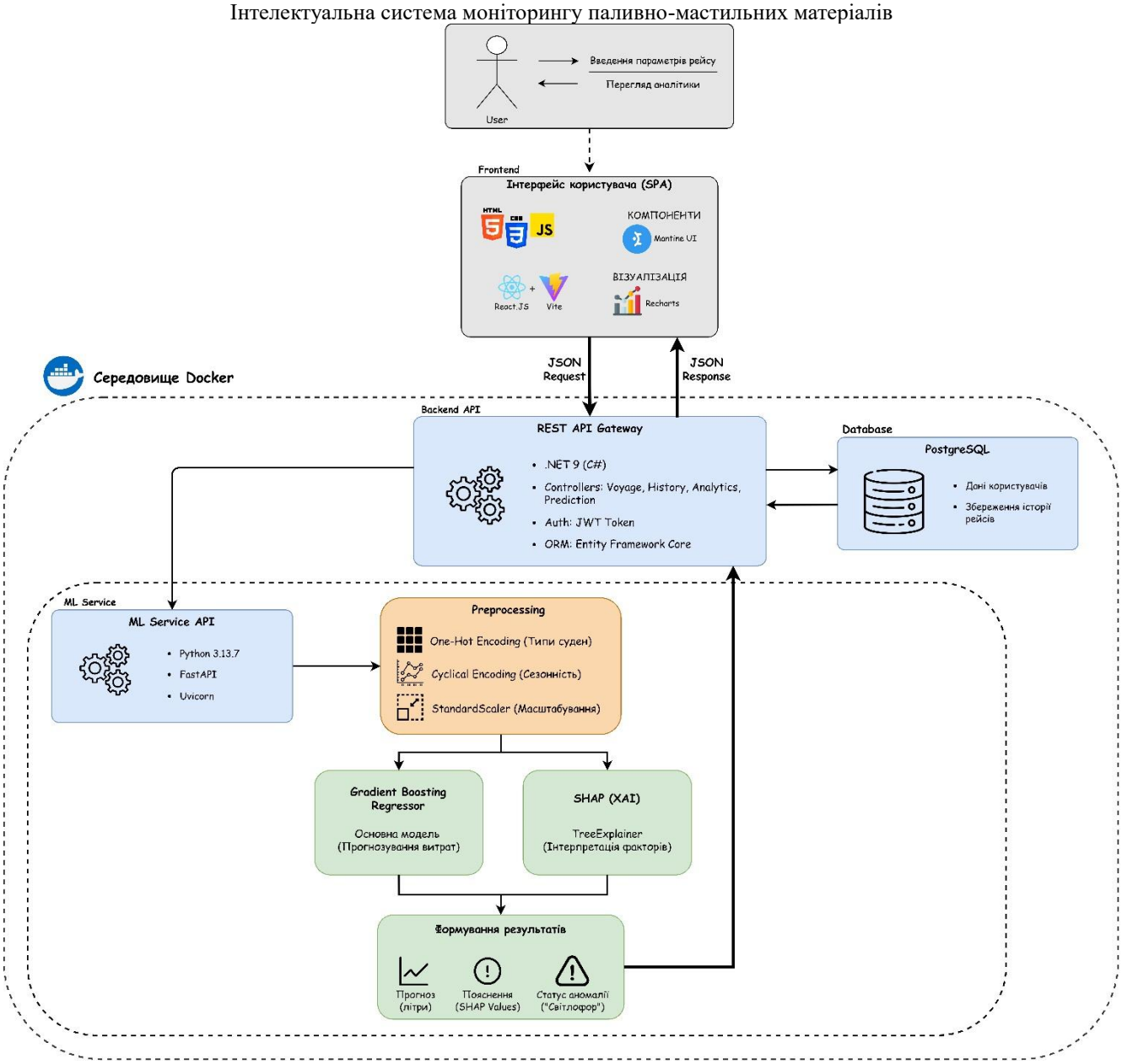

Рисунок 4.1 – Структурна схема системи "OptiFuel"

Далі описано детальне обґрунтування вибору технологій для кожного з компонентів.

4.1.1 Серверна частина бізнес-логіки (Backend API)

В якості технологічного ядра системи, що відповідає за реалізацію бізнес-логіки, авторизацію користувачів, валідацію вхідних даних та взаємодію з базою даних, було обрано новітню платформу .NET 9 (мова програмування C# 13). Цей вибір є стратегічним рішенням, обумовленим необхідністю побудови

високопродуктивної, масштабованої та безпечної системи корпоративного рівня, що відповідає сучасним стандартам розробки Cloud-Native застосунків.

Вибір саме цієї технологічної платформи базується на наступних ключових факторах:

– екстремальна продуктивність та оптимізація ресурсів: .NET 9 представляє собою еволюційний розвиток платформи, що демонструє значний приріст швидкодії порівняно з попередніми версіями (LTS-релізами). Завдяки вдосконаленому механізму динамічної профільної оптимізації (Dynamic PGO), який увімкнено за замовчуванням, та оновленому JIT-компілятору, система здатна адаптуватися до реального навантаження "на льоту". Це дозволяє обробляти тисячі запитів на секунду з мінімальними затримками (low latency) та меншим споживанням оперативної пам'яті, що є критичним для мікросервісної архітектури. Підтримка асинхронної моделі програмування (async/await) дозволяє ефективно використовувати системні ресурси при виконанні операцій вводу-виводу (I/O), таких як запити до бази даних або звернення до зовнішнього ML-сервісу, не блокуючи потоки виконання;

– орієнтація на хмарні середовища (Cloud-Native) та контейнеризацію: .NET 9 був спеціально оптимізований для роботи в середовищі Linux та контейнерах. Це є критично важливим для обраної стратегії розгортання "OptiFuel". Використання легковажних базових образів Docker для .NET 9 дозволяє значно зменшити розмір кінцевого артефакту розгортання та пришвидшити "холодний старт" мікросервісів, що спрощує масштабування та оркестрацію системи;

– сувора статична типізація мови C# 13: використання останньої версії мови C# надає доступ до сучасних синтаксичних конструкцій, що підвищують безпеку та читабельність коду. На відміну від інтерпретованих мов з динамічною типізацією (наприклад, JavaScript або Python), C# забезпечує сувору перевірку типів даних ще на етапі компіляції. Це дозволяє виявляти та усувати більшість потенційних помилок (таких як некоректна передача параметрів або порушення контрактів даних) ще до запуску програми. Для фінансових та логістичних систем, якою є

"OptiFuel", де точність розрахунків витрат палива є пріоритетом, така надійність є критичною вимогою;

– Entity Framework Core 9 (EF Core 9): використання новітньої версії цього ORM-фреймворку (Object-Relational Mapping) дозволило досягти нового рівня продуктивності при роботі з даними. EF Core 9 містить значні покращення у трансляції запитів LINQ в SQL, що дозволяє виконувати складні аналітичні вибірки швидше та ефективніше. ***Безпека***: Фреймворк автоматично використовує параметризацію запитів, що забезпечує вбудований захист від атак типу SQL Injection. ***Гнучкість***: Підхід Code-First дозволяє керувати схемою бази даних безпосередньо з коду, автоматично генеруючи скрипти міграції, що спрощує підтримку життєвого циклу застосунку;

– вбудована система впровадження залежностей (Dependency Injection): .NET 9 має вдосконалений вбудований DI-контейнер, що є стандартом для побудови слабко пов'язаних архітектур. Це дозволило легко інтегрувати різні сервіси (сервіс логування, сервіс доступу до даних, HTTP-клієнти) та забезпечити високу тестопридатність (testability) компонентів системи через підміну реалізацій на етапі тестування.

4.1.2 Сервіс інтелектуального аналізу (ML Service)

Ось розширена та деталізована версія опису для підрозділу, що стосується Python та FastAPI. Я додав більше технічного контексту про стандарти, продуктивність та архітектурні переваги, щоб текст виглядав вагомо та науково.

Для програмної реалізації модуля інтелектуального аналізу даних, прогнозування та детекції аномалій безальтернативним вибором стала мова програмування Python (версія 3.12). На сьогоднішній день Python є визнаним де-факто стандартом в індустрії Data Science та Machine Learning. Цей вибір обумовлений наявністю найпотужнішої екосистеми спеціалізованих бібліотек, які були використані в роботі:

– Pandas та NumPy: для ефективної маніпуляції табличними даними, векторизації обчислень та виконання операцій лінійної алгебри;

– Scikit-learn та XGBoost: для реалізації алгоритмів машинного навчання, зокрема градієнтного бустингу, який показав найкращі результати точності;

– SHAP (SHapley Additive exPlanations): для реалізації модуля пояснюваного штучного інтелекту (XAI), що є критичним для інтерпретації результатів моніторингу.

В якості веб-фреймворку для інкапсуляції розробленої ML-моделі у високопродуктивний REST API було обрано FastAPI. Це сучасний, швидкий (високопродуктивний) веб-фреймворк для створення API на Python 3.8+, що базується на стандартних підказках типу (type hints).

Вибір FastAPI обумовлений наступними архітектурними перевагами:

– висока продуктивність та асинхронність (ASGI): FastAPI побудований на базі стандарту ASGI (Asynchronous Server Gateway Interface) та використовує під капотом Starlette для веб-частини та Uvicorn як сервер. Це дозволяє реалізувати повноцінну асинхронну обробку запитів (async/await). Така архітектура дозволяє сервісу ефективно обробляти велику кількість одночасних HTTP-запитів на прогнозування без блокування основного потоку виконання (non-blocking I/O), що забезпечує пропускну здатність, порівнянну з сервісами на Go або NodeJS;

– сувора валідація даних та управління схемами (Pydantic): інтеграція з бібліотекою Pydantic забезпечує автоматичну валідацію вхідних даних на відповідність заданим моделям. Система гарантує, що вхідні параметри для ML-моделі (наприклад, тип судна, дистанція рейсу, погодні умови) відповідають очікуваним типам даних, діапазонам значень та обов'язковості полів ще до того, як вони будуть передані в алгоритм прогнозування. Це виключає помилки виконання (runtime errors), пов'язані з некоректними даними, та спрощує код обробників, звільняючи його від рутинних перевірок;

– автоматична генерація документації: завдяки використанню стандартів OpenAPI (раніше відомого як Swagger) та JSON Schema, FastAPI автоматично

генерує інтерактивну документацію API (/docs). Це значно спрощує процес розробки, тестування ендпоінтів та інтеграції ML-сервісу з основним .NET бекендом, забезпечуючи чіткий контракт взаємодії між мікросервісами;

– зручність впровадження (Dependency Injection): фреймворк має вбудовану легковажну систему впровадження залежностей, що дозволяє ефективно керувати життєвим циклом компонентів, таких як завантажені у пам'ять "важкі" ML-моделі та скейлери даних. Це дозволяє завантажувати модель лише один раз при старті контейнера (подія startup), а не при кожному запиті, що мінімізує затримки (latency) при отриманні прогнозу.

4.1.3 Клієнтська частина (Frontend)

Клієнтський рівень системи "OptiFuel" реалізовано у вигляді сучасного односторінкового застосунку (Single Page Application – SPA) на базі бібліотеки React (версія 18+). Вибір архітектури SPA обумовлений необхідністю забезпечення високої інтерактивності та швидкодії інтерфейсу, наближеної до десктопних програм. На відміну від традиційних багатосторінкових сайтів (MPA), де кожна дія користувача призводить до перезавантаження сторінки та повторного рендерингу на сервері, SPA завантажує код інтерфейсу один раз, а надалі обмінюється з сервером лише легковажними JSON-даними. Це суттєво зменшує навантаження на канал зв'язку, що є критично важливим для роботи в умовах супутникового інтернету на суднах.

Технологічний стек фронтенду базується на наступних компонентах:

React та компонентна модель

Використання React дозволило застосувати декларативний підхід до опису інтерфейсів. Інтерфейс системи розбито на незалежні, ізольовані компоненти (наприклад, PredictionForm, HistoryTable, ExplanationChart), які можна повторно використовувати в різних частинах програми.

– реактивність та Virtual DOM: ключовою перевагою є механізм віртуального DOM, який мінімізує кількість звернень до реального DOM браузера.

Це дозволяє створювати високодинамічні інтерфейси, де зміни даних (наприклад, оновлення статусу "світлофора" при введенні фактичного споживання або перебудова графіка в реальному часі) відбуваються миттєво, без помітних затримок для користувача;

– управління станом (Hooks): використання хуків (useState, useEffect, useContext) дозволило ефективно керувати локальним станом компонентів та глобальним станом застосунку (наприклад, статусом авторизації користувача) без надмірного ускладнення коду.

Mantine UI (Бібліотека компонентів)

Для забезпечення професійного вигляду, ергономічності та адаптивності інтерфейсу було використано бібліотеку Mantine UI. Це потужний набір React-компонентів, який надає готові рішення для побудови складних інтерфейсів.

– адаптивність (Responsive Design): система автоматично адаптується до розміру екрана пристрою завдяки вбудованій сітковій системі (Grid System). Це гарантує коректне відображення аналітичних дашбордів як на великих моніторах у диспетчерських центрах, так і на планшетах капітанів;

– прискорення розробки: використання готових, протестованих компонентів (таких як AppShell для макета, Table з підтримкою скролінгу, Modal для діалогових вікон, Notifications для сповіщень) дозволило сфокусувати зусилля на реалізації специфічної бізнес-логіки моніторингу палива, а не на рутинній верстці CSS-стилів та забезпеченні кросбраузерності.

Recharts (Візуалізація даних)

Оскільки "OptiFuel" є аналітичною системою, критично важливим аспектом є якісна візуалізація великих масивів даних. Для цього було інтегровано бібліотеку Recharts, яка базується на D3.js, але адаптована для React.

– декларативний підхід: бібліотека дозволяє описувати графіки як набір компонентів, що спрощує їх інтеграцію в JSX-код;

– SVG-рендеринг: всі графіки (гістограми розподілу відхилень, лінійні графіки трендів, діаграми ефективності) рендеряться у форматі SVG, що забезпечує

ідеальну чіткість зображення на екранах будь-якої роздільної здатності та дозволяє реалізувати інтерактивність (підсвічування при наведенні, клікабельні елементи для фільтрації таблиць – "drill-down").

Взаємодія з API (Axios)

Для організації HTTP-запитів до серверної частини використано бібліотеку Axios. Налаштовано механізм перехоплювачів (interceptors), який автоматично додає JWT-токен авторизації до заголовків кожного запиту та централізовано обробляє помилки (наприклад, автоматичне перенаправлення на сторінку входу у випадку закінчення терміну дії сесії), що підвищує безпеку та стабільність роботи клієнтської частини.

4.1.4 Система управління базами даних

Для реалізації рівня збереження даних (Persistence Layer), що відповідає за надійне зберігання історії рейсів, профілів користувачів, конфігурацій системи та результатів аналітичних розрахунків, було обрано PostgreSQL (версія 15+). Це потужна об'єктно-реляційна система управління базами даних (ORDBMS) з відкритим вихідним кодом, яка є промисловим стандартом для побудови надійних корпоративних систем.

Вибір PostgreSQL як основного сховища даних обґрунтований комплексом архітектурних та експлуатаційних переваг:

– гарантія цілісності даних (ACID): система повністю відповідає принципам ACID (Atomicity, Consistency, Isolation, Durability). Це є критично важливим для предметної області моніторингу палива, оскільки операції збереження даних про рейс (які включають фінансові та кількісні показники) повинні виконуватися як атомарні транзакції. У разі збою живлення або помилки сервера механізм журналювання транзакцій (Write-Ahead Logging – WAL) гарантує, що база даних не залишиться у неузгодженому стані, і жоден біт інформації про витрати не буде втрачено;

– ефективна робота з конкурентним доступом (MVCC): PostgreSQL використовує механізм багатоверсійного керування конкурентним доступом (Multi-Version Concurrency Control). Це дозволяє системі ефективно обслуговувати одночасні запити від багатьох користувачів (наприклад, коли один менеджер формує важкий аналітичний звіт за рік, а інший в цей час вносить нові дані про рейс). Завдяки MVCC операції читання не блокують операції запису, що забезпечує високу пропускну здатність системи "OptiFuel" навіть під навантаженням;

– потужний аналітичний інструментарій: СУБД має розширені можливості для виконання складних SQL-запитів, які необхідні для модуля аналітики. Підтримка віконних функцій, CTE (Common Table Expressions) та оптимізованих агрегатних функцій дозволяє виконувати складні розрахунки (наприклад, обчислення ковзного середнього споживання або ранжування суден за ефективністю) безпосередньо на рівні бази даних, мінімізуючи обсяг даних, що передаються на сервер застосунку;

– інтеграція з екосистемою .NET: PostgreSQL має високопродуктивний драйвер Npgsql, який забезпечує нативну підтримку у середовищі .NET. Це дозволяє ORM-фреймворку Entity Framework Core ефективно транслювати LINQ-запити з коду C# в оптимізований SQL-код PostgreSQL, використовуючи специфічні можливості бази даних (такі як типи даних JSONB або масиви);

– придатність до контейнеризації: PostgreSQL ідеально підходить для розгортання в середовищі Docker. Офіційні образи (на базі Alpine Linux) є легковажними та безпечними, що спрощує налаштування середовища розробки та продуктивного розгортання через docker-compose. Підтримка монтування томів (Docker Volumes) забезпечує збереження даних навіть при перестворенні контейнерів.

4.2 Реалізація бази даних та прикладних програмних інтерфейсів (API)

Ефективність функціонування інформаційної системи безпосередньо залежить від правильно спроектованої моделі даних та надійності програмних

інтерфейсів, що забезпечують обмін інформацією між компонентами. У цьому підрозділі описано фізичну реалізацію бази даних та архітектуру RESTful API серверної частини.

4.2.1 Проектування та реалізація схеми бази даних

В основу підсистеми зберігання даних покладено реляційну модель. Для взаємодії з СУБД PostgreSQL було використано технологію Entity Framework Core з підходом Code-First. Цей підхід дозволяє визначати структуру бази даних за допомогою класів мови C#, після чого система автоматично генерує SQL-скрипти (міграції) для створення відповідних таблиць та зв'язків (рис4.2. . Це забезпечує синхронізацію коду та схеми БД, а також спрощує процес розгортання системи.

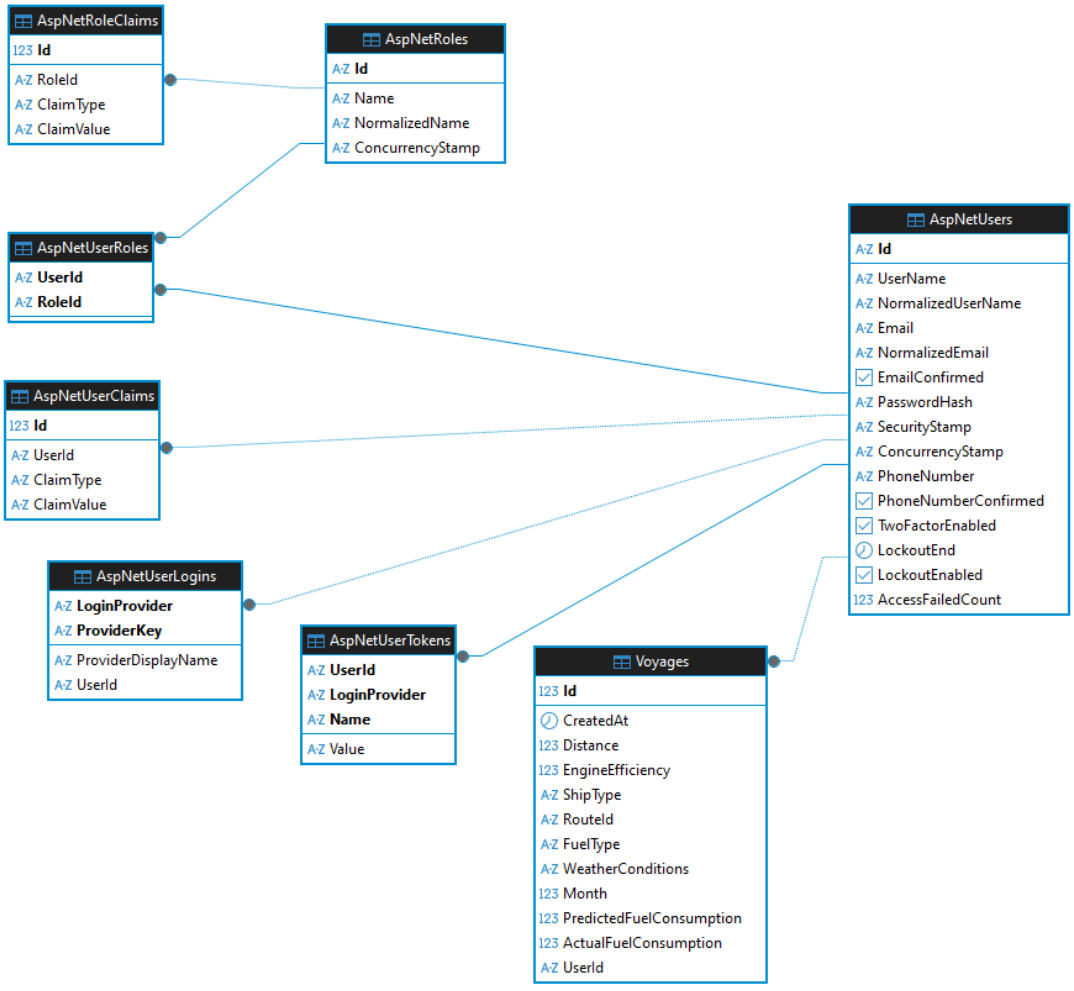

Рисунок 4.2 – ER-діаграма бази даних

Центральним елементом моделі даних є сутність PredictionHistory (Історія прогнозів), яка акумулює всю інформацію про життєвий цикл рейсу. Структура цієї сутності розроблена з урахуванням вимог до аналітичної звітності та містить наведені нижче атрибути.

1. Ідентифікатори та метадані:

– Id (Primary Key, integer): унікальний ідентифікатор запису, що використовується для індексації;

– CreatedAt (DateTime): часова мітка створення прогнозу у форматі UTC, необхідна для побудови часових рядів та аналізу трендів;

– UserId (Foreign Key, string): зовнішній ключ, що забезпечує зв'язок "один-до-багатьох" з таблицею користувачів. Це гарантує розмежування доступу до даних.

2. Вхідні експлуатаційні параметри (Input Features): збереження цих даних є критично важливим для реалізації функції моніторингу "План-Факт".

– RouteId (string): ідентифікатор маршруту;

– ShipType (string): тип судна (танкер, буксир тощо);

– Distance (double): планова дистанція рейсу в морських милях;

– EngineEfficiency (double): ККД двигуна у відсотках;

– WeatherConditions (string): категорія погодних умов;

– FuelType (string): тип використовуваного палива.

3. Результати роботи системи:

– PredictedFuelConsumption (double): нормативне значення витрат палива, розраховане нейромережевою моделлю;

– ActualFuelConsumption (double, nullable): фактичне значення витрат. Це поле може приймати значення NULL на етапі створення рейсу і заповнюється користувачем лише після завершення перевезення. Саме наявність цього поля дозволяє розраховувати відхилення (Deviation) та будувати графіки ефективності.

Для керування користувачами та забезпечення безпеки використано стандартні таблиці бібліотеки ASP.NET Core Identity, які автоматично інтегруються в схему бази даних (таблиці AspNetUsers, AspNetUserRoles тощо).

4.2.2 Архітектура та реалізація .NET Web API

Серверна частина реалізована як RESTful API, що дотримується принципів stateless-архітектури. Для забезпечення гнучкості та тестопридатності коду широко використано патерн Dependency Injection (Впровадження залежностей). Усі сервіси (робота з БД, клієнт ML-сервісу) реєструються в контейнері DI при старті застосунку.

Для чіткого розмежування внутрішньої моделі даних (Entity) та зовнішнього контракту API використано патерн DTO (Data Transfer Objects):

– PredictionRequest: використовується для отримання даних від клієнта. Містить атрибути валідації (Data Annotations), що гарантує коректність вхідних даних (наприклад, дистанція не може бути від'ємною);

– PredictionResponse: повертає клієнту лише необхідні дані, приховуючи службову інформацію;

– PredictionHistoryPreviewDto: оптимізований об'єкт для відображення списків у таблицях, що зменшує обсяг трафіку.

Основну бізнес-логіку інкапсульовано у контролерах:

а) VoyageController: відповідає за операційну діяльність:

1) Метод CreateVoyage реалізує складний сценарій: валідація вхідних даних → відправка запиту на ML-сервіс → отримання прогнозу → збереження повного запису в БД → повернення результату клієнту.;

2) Метод UpdateActualConsumption дозволяє вносити фактичні дані, замикаючи цикл моніторингу.

б) HistoryController: забезпечує доступ до історичних даних:

1) Реалізовано серверну пагінацію та сортування за допомогою бібліотеки System.Linq.Dynamic.Core. Це дозволяє ефективно працювати з

великими масивами даних, завантажуючи їх частинами (сторінками), а не повністю;

2) Реалізовано фільтрацію за категоріями відхилень ("Critical", "Normal"), що дозволяє будувати інтерактивні звіти.

в) AnalyticsController: спеціалізований контролер для агрегації даних. Виконує аналітичні SQL-запити через LINQ (групування GroupBy, усереднення Average) на стороні сервера баз даних, повертаючи на фронтенд лише готову статистику для графіків.

4.2.3 Реалізація API сервісу машинного навчання (ML Service)

Мікросервіс на базі FastAPI (Python) спроектовано для високопродуктивної обробки запитів на прогнозування.

Ключовою особливістю реалізації є використання менеджера життєвого циклу (lifespan), який забезпечує завантаження "важких" артефактів (моделі best_model.joblib, скалера scaler.joblib та експлейнера SHAP) в оперативну пам'ять лише один раз – при старті контейнера. Це дозволяє скоротити час обробки кожного запиту до мілісекунд, оскільки не витрачається час на читання файлів з диска.

Реалізовано два спеціалізовані ендпоінти:

– POST /predict: приймає "сирі" дані, виконує їх препроцесинг (функція preprocess_input) – кодування категорій та циклічних змінних, нормалізацію – і передає підготовлений вектор у модель XGBoost. Результатом є числове значення витрат палива;

– POST /explain: реалізує функціонал пояснюваного штучного інтелекту (XAI). Використовує об'єкт shap.TreeExplainer для розрахунку значень Шеплі (SHAP values). Відповідь містить список ознак та їх числовий внесок у відхилення прогнозу від базового значення, що дозволяє візуалізувати логіку прийняття рішень моделлю.

Взаємодія між .NET API та Python ML Service реалізована через внутрішню мережу Docker за допомогою HTTP-клієнта, що забезпечує ізоляцію компонентів та високу швидкість обміну даними.

4.3 Інтерфейс користувача та сценарії використання

Клієнтська частина системи "OptiFuel" розроблена відповідно до сучасних стандартів веб-інженерії як односторінковий застосунок (SPA). Це забезпечує плавність роботи інтерфейсу, миттєвий відгук на дії користувача та відсутність перезавантажень сторінки при переході між розділами. Дизайн системи реалізовано з використанням бібліотеки компонентів Mantine UI, що гарантує адаптивність (Responsive Design) – коректне відображення як на настільних моніторах диспетчерів, так і на планшетах капітанів суден.

Нижче описано основні сценарії взаємодії користувача з системою.

4.3.1 Сценарій публічного доступу та швидкого прогнозування

Точкою входу до системи є головна сторінка (Landing Page), яка виконує інформаційну функцію, ознайомлюючи користувача з можливостями платформи. Для незареєстрованих користувачів (гостей) доступний базовий функціонал – "Публічний калькулятор" (Fuel Forecaster).

Інтерфейс форми прогнозування

Реалізовано інтуїтивно зрозумілу форму введення параметрів рейсу (рис. 4.3).

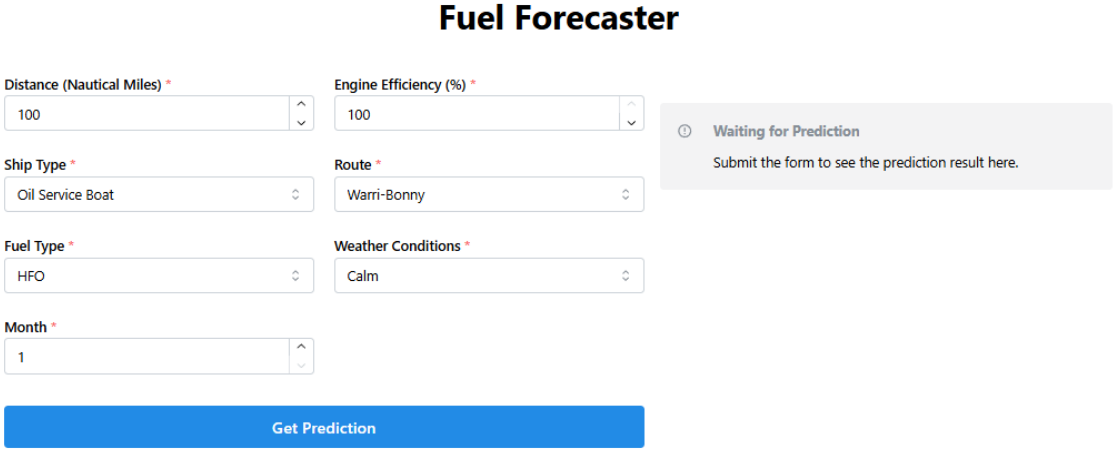

Рисунок 4.3 – Головна сторінка та форма прогнозування

Для мінімізації помилок оператора (human error) використано наступні підхідні рішення:

– випадаючі списки (Select inputs): для категоріальних полів (Ship Type, Route, Fuel Type, Weather) замість текстового вводу використано фіксовані списки значень. Це гарантує, що дані будуть передані на сервер у форматі, який відповідає словникам моделі машинного навчання;

– валідація на стороні клієнта: числові поля (Distance, Engine Efficiency) оснащені механізмами перевірки діапазонів. Наприклад, система не дозволить ввести від'ємну дистанцію або ефективність двигуна більше 100%;

– індикація завантаження: під час обробки запиту сервером форма блокується та відображається анімація завантаження (LoadingOverlay), що забезпечує зворотний зв'язок з користувачем.

Після розрахунку результат відображається у вигляді візуального компонента (Ring Progress), де центральним елементом є прогнозований обсяг палива у літрах. Цей сценарій дозволяє швидко оцінити бюджет майбутнього рейсу без необхідності реєстрації.

4.3.2 Сценарій операційного моніторингу та аналітики

Для авторизованих користувачів, які виконують роль менеджерів флоту або логістів-аналітиків, система надає доступ до закритого модуля – "Панелі історії та аналітики" (History & Analytics Dashboard). Ця сторінка спроектована як центральний робочий простір (Workspace) для здійснення ретроспективного аналізу перевезень та оперативного контролю ефективності флоту.

Функціонал дашборду реалізує концепцію "від загального до конкретного" (Overview first, zoom and filter, then details-on-demand), що дозволяє користувачеві швидко оцінити загальний стан справ і за необхідності заглибитися в деталі конкретних інцидентів.

Інтерактивне табличне представлення даних

Центральним елементом інтерфейсу є динамічна таблиця ("Data Grid"), що консолідує історичні дані про всі розраховані рейси (рис. 4.4). Враховуючи потенційно великий обсяг накопичених даних (Big Data), у системі реалізовано механізм серверної пагінації (Server-Side Pagination). На відміну від клієнтської пагінації, цей підхід передбачає завантаження даних з бази даних PostgreSQL невеликими порціями (сторінками), що значно знижує навантаження на мережу та оперативну пам'ять браузера, забезпечуючи високу швидкість відгуку інтерфейсу навіть при наявності мільйонів записів.

Таблиця підтримує динамічне сортування за будь-яким атрибутом (дата, маршрут, обсяг споживання), що дозволяє, наприклад, миттєво ранжувати рейси від найбільшого перевитрати до найменшого для пріоритизації розслідувань.

Date	Ship	Route	Dist.	Eff.	Fuel	Weather	Predicted (L)	Actual (L)
28.11.2025	Oil Service Boat	Escravos-Lagos	376 nm	95.43%	Diesel	Moderate	**15 583,54**	14 349
28.11.2025	Surfer Boat	Escravos-Lagos	376 nm	95.43%	Diesel	Moderate	**14 949,55**	17 931
28.11.2025	Surfer Boat	Warri-Bonny	245 nm	95.43%	Diesel	Moderate	**10 380,19**	11 361
28.11.2025	Fishing Trawler	Warri-Bonny	245 nm	95.43%	Diesel	Moderate	**11 205,71**	12 561
28.11.2025	Fishing Trawler	Escravos-Lagos	137 nm	95.43%	Diesel	Moderate	**3 896,3**	3 651
28.11.2025	Fishing Trawler	Escravos-Lagos	137 nm	95.43%	HFO	Moderate	**3 801,05**	3 651
28.11.2025	Fishing Trawler	Lagos-Apapa	452 nm	95.43%	Diesel	Calm	**20 967,23**	19 354
24.11.2025	Oil Service Boat	Warri-Bonny	100 nm	100%	HFO	Calm	**2 993,75**	3 100
24.11.2025	Tanker Ship	Lagos-Apapa	1254 nm	87%	Diesel	Stormy	**17 844,85**	19 824
24.11.2025	Surfer Boat	Warri-Bonny	1251 nm	100%	Diesel	Calm	**19 433,63**	19 230

‹ 1 2 3 ›

Рисунок 4.4 – Табличне представлення даних

Візуальна аналітика та агреговані метрики (KPIs)

Для зменшення когнітивного навантаження на оператора, над табличною частиною розміщено блок візуальної аналітики (рис. 4.5), реалізований на базі бібліотеки Recharts. Він включає:

– панель ключових показників ефективності (KPI Cards): відображає миттєвий зріз стану флоту: загальну кількість виконаних рейсів, сумарний обсяг спожитого палива та, що є критично важливим, глобальний середній відсоток

відхилення (Global Average Deviation). Цей інтегральний показник слугує індикатором "здоров'я" логістичної системи: якщо він перевищує порогове значення (наприклад, 3%), це сигналізує про системні проблеми в управлінні або технічному стані флоту;

– гістограма розподілу відхилень (Deviation Distribution): цей графік дозволяє оцінити стохастичну природу процесу споживання палива. Він візуалізує частотний розподіл рейсів за категоріями ефективності: "Економія", "Норма", "Попередження" та "Критично". Наявність "важкого хвоста" в зоні критичних відхилень вказує на наявність систематичних порушень;

– аналіз ефективності в розрізі типів суден: стовпчикова діаграма дозволяє провести порівняльний аналіз (benchmarking) різних класів суден. Це дає можливість виявити, чи є перевитрати проблемою конкретного типу техніки (наприклад, застарілих танкерів), чи вони розподілені рівномірно по всьому флоту.

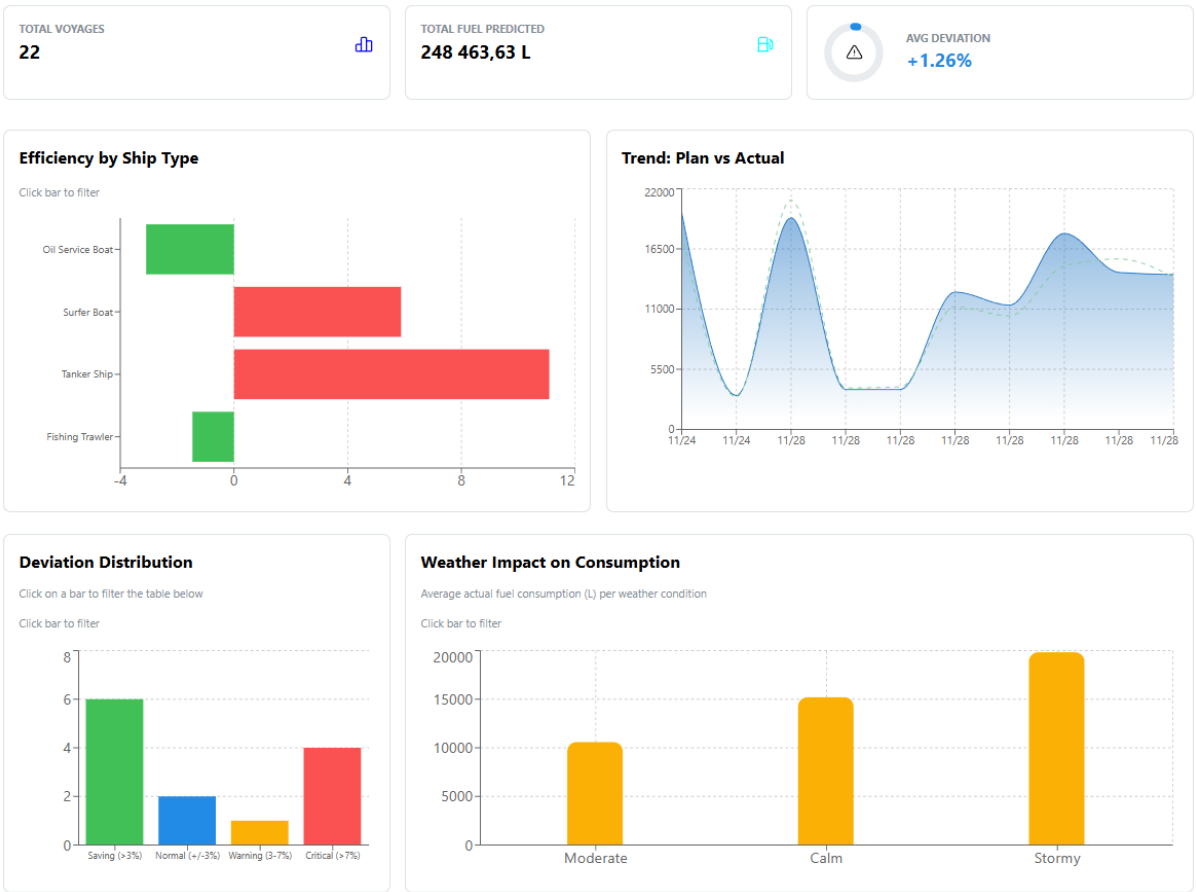

Рисунок 4.5 – Візуальна аналітика та агреговані метрики

Інтерактивна фільтрація (Drill-down Interaction)

Важливою особливістю реалізації є глибока інтеграція графічних елементів з табличними даними. Реалізовано патерн "Drill-down": графіки є не просто статичними зображеннями, а інтерактивними фільтрами. Клік на певний сегмент графіка (наприклад, на червоний стовпчик "Critical" на гістограмі або на категорію "Tanker Ship") автоматично формує складний пошуковий запит до API. В результаті таблиця даних миттєво оновлюється, відображаючи лише ті записи, що відповідають обраному критерію.

Такий підхід дозволяє менеджеру за лічені секунди локалізувати проблему, переходячи від стратегічного огляду всього флоту до списку конкретних проблемних рейсів, що потребують адміністративного втручання.

4.3.3 Сценарій детального аналізу рейсу та пояснення ШІ (XAI)

При виборі конкретного рейсу з таблиці користувач потрапляє на сторінку "Деталі рейсу" (Voyage Details) (рис. 4.6). Цей інтерфейс вирішує задачу порівняльного аналізу "План-Факт".

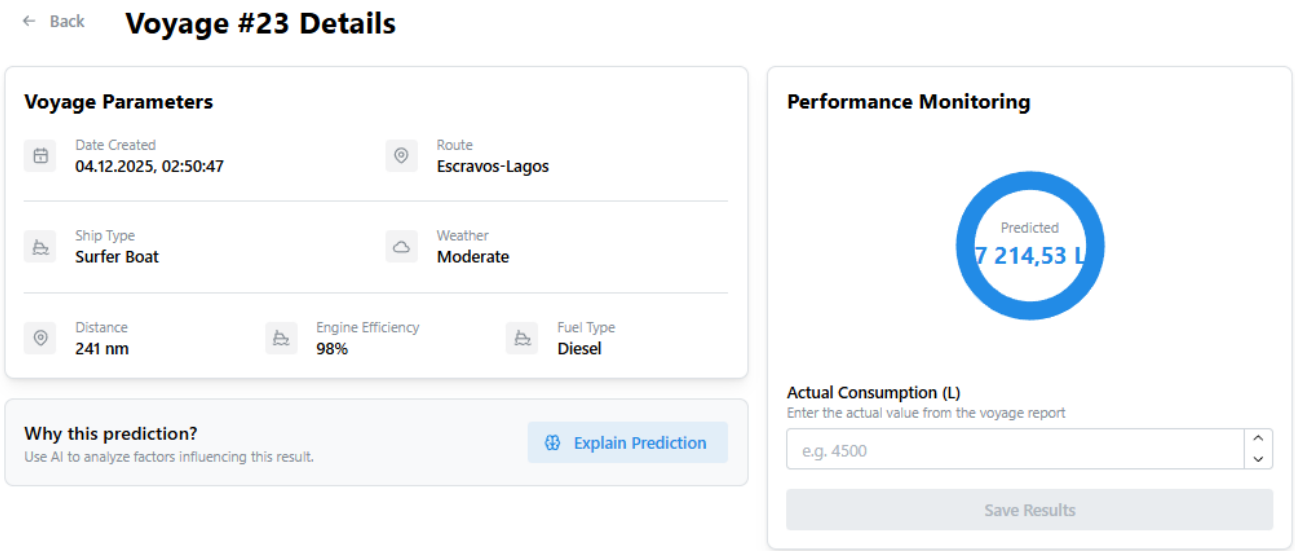

Рисунок 4.6 – Деталі рейсу

Функціонал моніторингую

Сторінка розділена на дві логічні зони. Зліва відображаються незмінні параметри рейсу (паспорт рейсу). Справа – панель моніторингу. Користувач вводить фактичне значення споживання палива (отримане зі звіту капітана), і система миттєво розраховує відхилення (рис. 4.7).

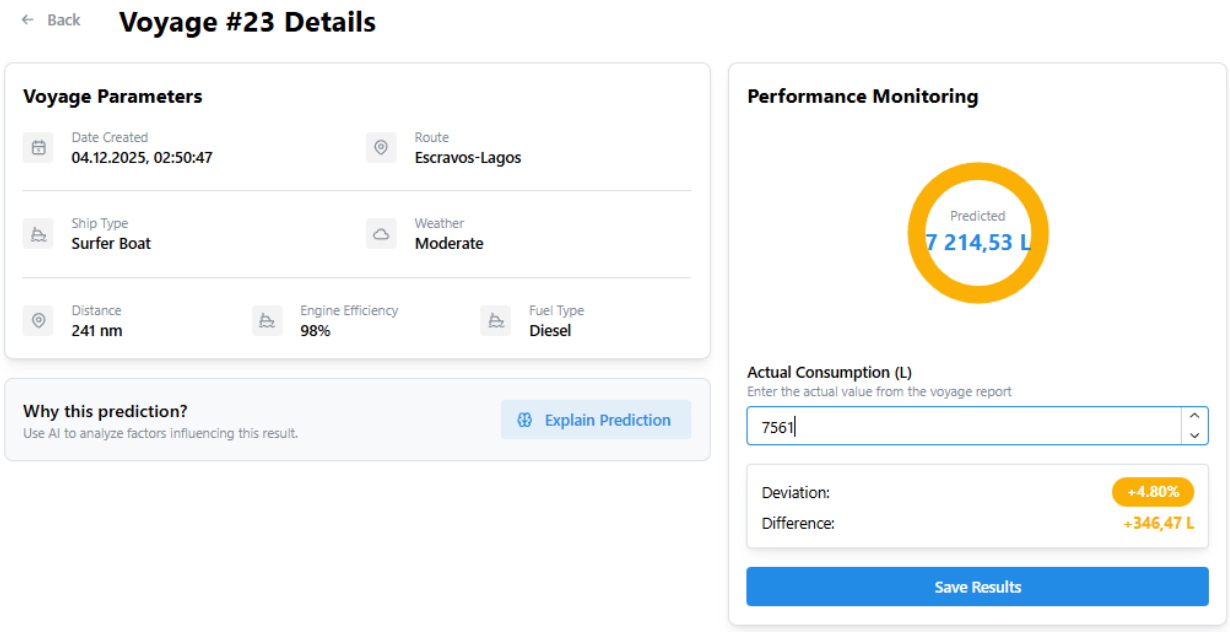

Рисунок 4.7 – Приклад введення фактичного значення споживання палива

Застосовано колірне кодування статусу (Badging):

– зелений колір: відхилення в межах норми ($\leq 3\%$);
– жовтий колір: попередження ($3 - 7\%$);
– червоний колір: аномалія ($> 3\%$).

Модуль пояснення (Explainable AI)

Унікальною особливістю системи є інтеграція пояснюваного штучного інтелекту. Якщо менеджер бачить дивний прогноз або аномалію, він може скористатися функцією "Explain Prediction" (рис. 4.7 – 4.8).

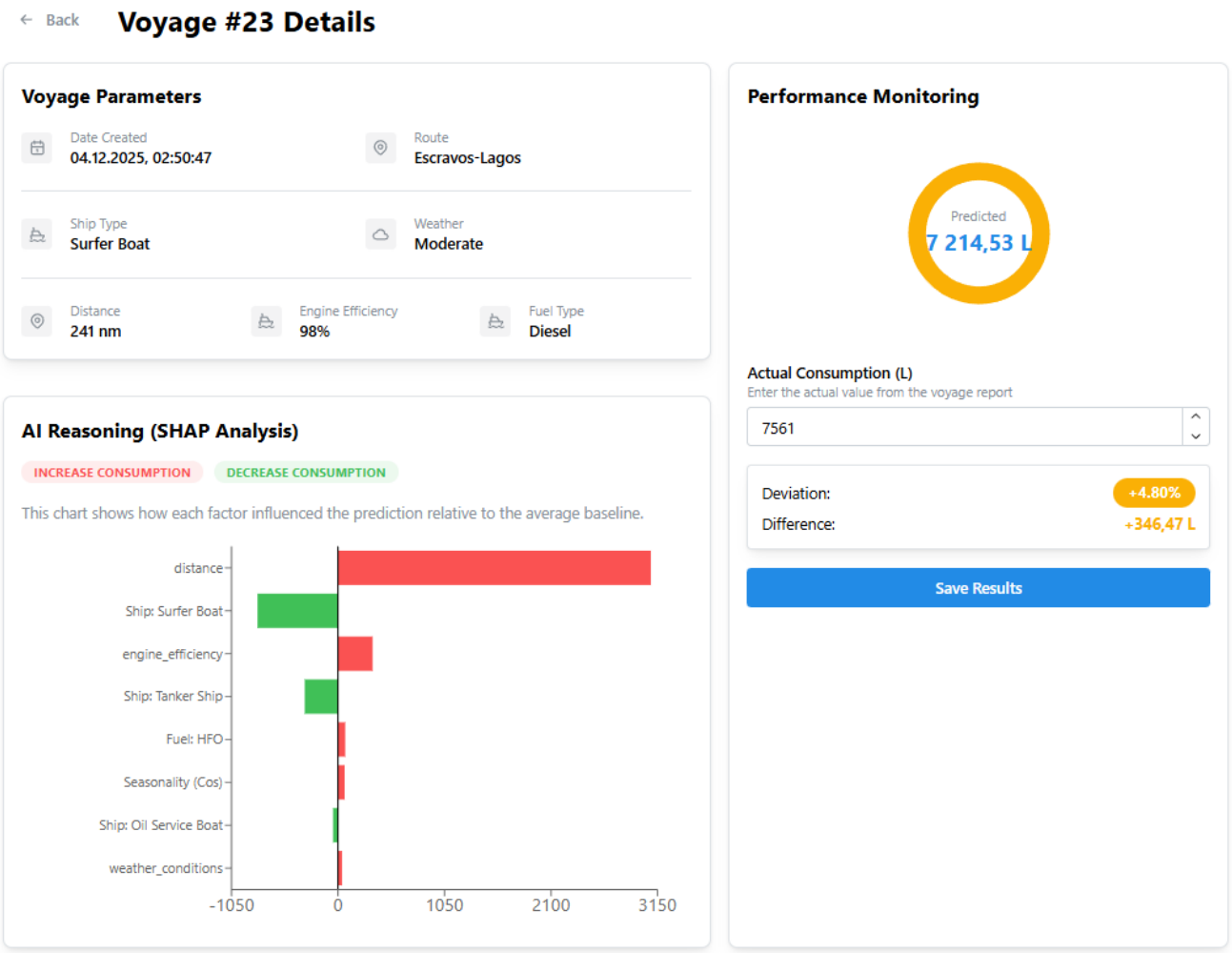

Рисунок 4.8 – Приклад використання функції "Explain Prediction"

Система генерує запит до модуля SHAP і відображає графік впливу факторів (Explanation Chart). Цей графік візуалізує, як кожен вхідний параметр (погода, тип судна, дистанція) вплинув на фінальну цифру. Наприклад, графік може показати, що незважаючи на коротку дистанцію, прогноз високий через фактор "Штормова погода" (червона смуга впливу).

Така візуалізація перетворює модель "чорної скриньки" на прозорий інструмент, підвищуючи довіру користувача до автоматизованих розрахунків та допомагаючи відрізнити об'єктивні перевитрати (через погоду) від суб'єктивних (крадіжка або недбалість).

4.4 Оцінка ефективності впровадження системи

Впровадження інтелектуальної системи "OptiFuel" у виробничі процеси транспортно-логістичних компаній дозволяє досягти комплексного ефекту, який виражається не лише у прямих фінансових показниках, а й у якісній зміні підходів до управління флотом. Ефективність системи оцінюється за трьома основними напрямками: економічним, операційним та стратегічним.

4.4.1 Економічна ефективність та зменшення експлуатаційних витрат (OPEX)

Витрати на бункерування (паливо) традиційно складають левову частку операційних витрат судна – від 40% до 60% залежно від типу судна. Тому навіть незначне підвищення ефективності у відсотковому відношенні конвертується у значні фінансові заощадження.

Система "OptiFuel" забезпечує економічний ефект за рахунок наступних механізмів:

– виявлення та усунення аномальних перевитрат: завдяки розробленому алгоритму детекції аномалій (порівняння факту з динамічною нормою), система дозволяє оперативно виявляти рейси з критичним відхиленням (>7%). Це дає можливість миттєво реагувати на випадки несанкціонованого зливу палива (крадіжки) або нецільового використання ресурсів. За оцінками галузевих експертів, мінімізація крадіжок може заощадити до 3-5% загального паливного бюджету;

– предиктивне технічне обслуговування (Predictive Maintenance): систематичні відхилення у "жовтій зоні" (3-7%) для конкретного судна при нормальних погодних умовах є раннім індикатором технічних проблем (наприклад, забруднення корпусу, знос гвинта або проблеми з паливною апаратурою). Система дозволяє ідентифікувати такі судна та направити їх на технічне обслуговування до того, як несправність призведе до серйозної аварії або значних втрат палива;

– оптимізація бюджетування: використання високоточної моделі Gradient Boosting ($R^2 > 95\%$) замість лінійних нормативів дозволяє точніше планувати

закупівлі палива. Це зменшує ризики заморожування обігових коштів у надлишкових запасах палива або, навпаки, необхідності екстреної дозаправки за завищеними цінами.

4.4.2 Операційна ефективність та автоматизація

Впровадження системи дозволяє трансформувати роботу диспетчерських служб та відділів моніторингу:

– зменшення впливу людського фактору: автоматизація розрахунку нормативів виключає суб'єктивні помилки диспетчерів при ручному плануванні. Система працює 24/7, забезпечуючи стабільну точність розрахунків незалежно від навантаження персоналу;

– перехід до управління за відхиленнями (Management by Exception): замість ручного перегляду звітів по кожному з сотень рейсів, менеджери фокусують увагу лише на тих записах, які система підсвітила червоним або жовтим кольором. Це значно скорочує час на аналіз даних та прийняття рішень, підвищуючи продуктивність праці аналітичного відділу;

– прискорення документообігу: інтеграція системи через API дозволяє автоматично формувати маршрутні листи та звіти, що зменшує бюрократичне навантаження на екіпаж та офісний персонал.

4.4.3. Стратегічна та екологічна ефективність

Окрім прямих фінансових вигод, система створює довгострокову стратегічну цінність:

– прозорість та обґрунтованість рішень (Data-Driven Decisions): інтеграція модуля XAI (Explainable AI) на базі бібліотеки SHAP вирішує проблему довіри до штучного інтелекту. Коли система вказує на перевитрату, вона надає математичне обґрунтування (наприклад, "вплив погодних умов враховано, перевитрата викликана іншими факторами"). Це дозволяє вести аргументований діалог з капітанами та підрядниками, уникаючи безпідставних звинувачень;

– зменшення вуглецевого сліду (Decarbonization): оптимізація споживання палива прямо корелює зі зменшенням викидів CO_2 та інших парникових газів. Використання системи "OptiFuel" допомагає судноплавним компаніям відповідати суворим міжнародним екологічним стандартам (наприклад, вимогам IMO 2020/2030) та покращує екологічний імідж компанії.

Узагальнена оцінка

Розроблена система не вимагає значних капітальних інвестицій у апаратне забезпечення (використовуються існуючі дані телеметрії) та легко масштабується завдяки хмарній архітектурі та контейнеризації Docker. Це забезпечує високий показник повернення інвестицій (ROI) та робить систему доступною навіть для невеликих логістичних операторів.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи було виконано повний цикл програмної реалізації та інженерного проектування інтелектуальної системи моніторингу палива "OptiFuel".

Основні результати розділу полягають у наступному:

– реалізовано надійну архітектуру: спроектовано та впроваджено мікросервісну архітектуру системи, що складається з трьох незалежних компонентів: клієнтського застосунку, сервера бізнес-логіки та спеціалізованого ML-сервісу. Такий підхід забезпечив гнучкість розробки, можливість незалежного масштабування модулів та технологічну ізоляцію (поєднання .NET та Python екосистем). Використання контейнеризації Docker дозволило уніфікувати середовище розгортання та спростити процес впровадження системи (CI/CD);

– застосовано передовий технологічний стек: в якості ядра системи використано новітню платформу .NET 9, що забезпечило високу продуктивність обробки запитів та надійність бізнес-логіки завдяки суворій типізації C# 13. Для реалізації модуля штучного інтелекту використано стандарт де-факто – Python та

швидкий асинхронний фреймворк FastAPI. Надійність збереження даних гарантується використанням СУБД PostgreSQL, що відповідає вимогам ACID;

– створено функціональний веб-інтерфейс: розроблено адаптивний SPA-застосунок на базі React та Mantine UI. Інтерфейс підтримує різні сценарії використання: від швидкого прогнозування для гостей до глибокого ретроспективного аналізу для авторизованих менеджерів. Реалізовано інтерактивні дашборди з візуалізацією ключових метрик (KPIs) та механізмом фільтрації даних ("drill-down"), що значно пришвидшує процес прийняття управлінських рішень;

– інтегровано модуль пояснюваного ШІ (XAI): унікальною особливістю реалізації є впровадження бібліотеки SHAP, що дозволило вирішити проблему "чорної скриньки". Система не лише детектує аномалії за допомогою алгоритму "світлофора", але й візуалізує вплив кожного фактора (погоди, типу судна, дистанції) на кінцевий прогноз. Це забезпечує прозорість роботи системи та підвищує довіру користувачів до результатів автоматизованого аналізу;

– підтверджено практичну ефективність: оцінка ефективності впровадження показала, що система "OptiFuel" здатна забезпечити значний економічний ефект шляхом мінімізації крадіжок палива, оптимізації планування бюджету та переходу до предиктивного обслуговування суден. Автоматизація розрахунків та звітності дозволяє знизити операційне навантаження на персонал та мінімізувати вплив людського фактору.

Таким чином, розроблений програмний комплекс є завершеним, високотехнологічним продуктом, готовим до промислової експлуатації, що повністю вирішує поставлені задачі дослідження.

ВИСНОВКИ

У даній кваліфікаційній роботі вирішено актуальну науково-прикладну задачу підвищення ефективності експлуатації водного транспорту шляхом розробки інтелектуальної системи моніторингу та прогнозування витрат паливно-мастильних матеріалів.

Основні наукові та практичні результати роботи полягають у наступному:

– проведено комплексний аналіз предметної області. Встановлено, що традиційні методи контролю витрат палива, які базуються на статичних нормах, є неефективними в умовах динамічної зміни зовнішніх факторів (погода, течії, завантаження). Це призводить до фінансових втрат та неможливості оперативно виявляти зловживання. Обґрунтовано необхідність переходу до динамічного моделювання норм за допомогою методів машинного навчання;

– виконано експериментальне дослідження методів машинного навчання. На реальних даних телеметрії проведено навчання та порівняння трьох класів регресійних моделей: лінійної регресії, випадкового лісу та градієнтного бустингу. Також, доведено, що залежність споживання палива від експлуатаційних факторів має нелінійний характер. Найкращі результати продемонструвала модель Gradient Boosting (XGBoost), яка досягла коефіцієнта детермінації $R^2 \approx 95.5\%$ та мінімальної середньоквадратичної помилки на тестовій вибірці;

– розроблено методику інтелектуального моніторингу. Створено алгоритм виявлення аномалій, який базується на порівнянні фактичного споживання з динамічною нормою, розрахованою моделлю. Впроваджено трирівневу систему оцінки відхилень («Світлофор»), що дозволяє автоматично класифікувати рейси на ефективні (відхилення $\leq 3\%$) підозрілі ($3 - 7\%$) та аномальні (>7%);

– запропоновано та реалізовано підхід до інтерпретації прогнозів (XAI). Для вирішення проблеми недовіри до «чорної скриньки» ШІ, у систему інтегровано модуль на базі методу SHAP. Це дозволило розкладати прогноз на складові та пояснювати користувачу вплив кожного фактора (наприклад, погодних умов чи

технічного стану) на кінцевий результат, що забезпечує прозорість прийняття рішень;

– створено програмний комплекс "OptiFuel". Розроблено повнофункціональну систему з використанням сучасної мікросервісної архітектури. Технологічний стек включає: .NET 9 (Backend API), Python FastAPI (ML Service), React.js (Frontend) та PostgreSQL (Database). Використання контейнеризації Docker забезпечило ізольованість компонентів та легкість розгортання. Реалізовано зручний веб-інтерфейс з інтерактивними аналітичними дашбордами, механізмами фільтрації даних («drill-down») та рольовою моделлю доступу;

– підтверджено практичну цінність розробки. Впровадження системи дозволяє перейти від реактивного контролю до проактивного управління ефективністю флоту. Очікуваний економічний ефект досягається за рахунок мінімізації крадіжок палива, оптимізації планування бюджету та своєчасного виявлення технічних несправностей суден.

Таким чином, мета роботи досягнута, а поставлені завдання виконані в повному обсязі. Розроблена система є завершеним програмним продуктом, готовим до впровадження у виробничі процеси транспортних компаній.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Global Fleet Management Market Size, Share & Trends Analysis Report 2024-2030. Grand View Research, 2024. URL: https://www.grandviewresearch.com/industry-analysis/fleet-management-market (дата звернення: 17.10.2025).

2. Wialon GPS Tracking & Fleet Management Platform. Gurtam, 2024. URL: https://wialon.com/ (дата звернення: 19.10.2025).

3. UMT Fleet Management Solutions. Ukrtrans Telematics, 2024. URL: https://umt.ua/ (дата звернення: 19.10.2025).

4. Teltonika Telematics Solutions. Teltonika Networks, 2024. URL: https://teltonika-gps.com/ (дата звернення: 19.10.2025).

5. Zhou M., Jin H., Wang W. A review of vehicle fuel consumption models to evaluate eco-driving and eco-routing. Transportation Research Part D: Transport and Environment, 2016. Vol. 49. P. 203–218. DOI: 10.1016/j.trd.2016.09.008.

6. Hamidreza Abediasl, Amir Ansari, Vahid Hosseini, Charles Robert Koch, Mahdi Shahbakhti. Real-time vehicular fuel consumption estimation using machine learning and on-board diagnostics data. Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 2024. DOI: 10.1177/09544070231185609.

7. Rykała M., Grzelak M., Rykała Ł., Voicu D., Stoica R.-M. Modeling Vehicle Fuel Consumption Using a Low-Cost OBD-II Interface. Energies, 2023. Vol. 16(21). P. 7266. DOI: 10.3390/en16217266.

8. Wickramanayake S., Bandara H.M.N.D. Fuel consumption prediction of fleet vehicles using Machine Learning: A comparative study. Proceedings of the Moratuwa Engineering Research Conference (MERCon), 2016. P. 90-95. DOI: 10.1109/MERCon.2016.7480121.

9. Perrotta F., Parry T., Neves L.C. Application of machine learning for fuel consumption modelling of trucks. 2017 IEEE International Conference on Big Data, 2017. P. 3810-3815. DOI: 10.1109/BigData.2017.8258382.

10. James G., Witten D., Hastie T., Tibshirani R. An Introduction to Statistical Learning: with Applications in R. 2nd Edition. Springer, 2021. 607 p. DOI: 10.1007/978-1-0716-1418-1.

11. Breiman L. Random Forests. Machine Learning, 2001. Vol. 45. P. 5–32. DOI: 10.1023/A:1010933404324.

12. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. P. 785–794. DOI: 10.1145/2939672.2939785.

13. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems, 2017. Vol. 30. P. 3146-3154.

14. Abukhalil T., Almahirah M., Abdelkareem M.A., Alami A.H., Olabi A.G. Machine learning regression algorithms for modeling fuel consumption in light-duty vehicles. Energy Reports, 2022. Vol. 8. P. 6626-6637. DOI: 10.1016/j.egyr.2022.05.021.

15. Moradi E., Miranda-Moreno L. Vehicular fuel consumption estimation using real-world measures through cascaded machine learning modeling. Transportation Research Part D: Transport and Environment, 2020. Vol. 88. Article 102576. DOI: 10.1016/j.trd.2020.102576.

16. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation, 1997. Vol. 9(8). P. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

17. Cho K., Van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv preprint, 2014. arXiv:1406.1078.

18. Barbado A., Corcho O. Interpretable machine learning models for predicting and explaining vehicle fuel consumption anomalies. Engineering Applications of Artificial Intelligence, 2022. Vol. 115. Article 104979. DOI: 10.1016/j.engappai.2022.104979.

19. Barbado A., Corcho O. Anomaly detection in average fuel consumption with XAI techniques for dynamic generation of explanations. AAAI Conference on Artificial Intelligence, 2020.

20. Liu F.T., Ting K.M., Zhou Z.-H. Isolation Forest. 2008 Eighth IEEE International Conference on Data Mining, 2008. P. 413–422. DOI: 10.1109/ICDM.2008.17.

21. Schölkopf B., Platt J.C., Shawe-Taylor J., Smola A.J., Williamson R.C. Estimating the Support of a High-Dimensional Distribution. Neural Computation, 2001. Vol. 13(7). P. 1443-1471. DOI: 10.1162/089976601750264965.

22. Wahab A., Quek C., Tan C.K., Takeda K. Driving Profile Modeling and Recognition Based on Soft Computing Approach. IEEE Transactions on Neural Networks, 2009. Vol. 20(4). P. 563-582. DOI: 10.1109/TNN.2008.2007906.

23. MacQueen J. Some methods for classification and analysis of multivariate observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967. Vol. 1. P. 281-297.

24. Miotti M., Needell Z.A., Ramakrishnan S., et al. Quantifying the impact of driving style changes on light-duty vehicle fuel consumption. Transportation Research Part D: Transport and Environment, 2021. Vol. 98. Article 102918. DOI: 10.1016/j.trd.2021.102918.

25. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 p.Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf. (дата звернення: 01.11.2025).

26. Python 3.12 Documentation. (2024). Python Software Foundation. URL: https://docs.python.org/3/. (дата звернення: 01.11.2025).

27. Scikit-learn: Machine Learning in Python. (2024). Scikit-learn developers. URL: https://scikit-learn.org/stable/. (дата звернення: 01.11.2025).

28. XGBoost Documentation. (2024). XGBoost developers. URL: https://xgboost.readthedocs.io/en/stable/. (дата звернення: 01.11.2025).

29. Pandas documentation. (2024). The pandas development team. URL: https://pandas.pydata.org/docs/. (дата звернення: 04.11.2025).

30. NumPy Documentation. (2024). NumPy Developers. URL: https://numpy.org/doc/stable/. (дата звернення: 04.11.2025).

31. Pydantic Documentation. (2024). Samuel Colvin. URL: https://docs.pydantic.dev/latest/. (дата звернення: 04.11.2025).

32. FastAPI Documentation. (2024). Sebastián Ramírez. URL: https://fastapi.tiangolo.com/. (дата звернення: 04.11.2025).

33. FastAPI: Main Tutorial. (2024). Tiago, P. URL: https://fastapi.tiangolo.com/tutorial/. (дата звернення: 05.11.2025).

34. Introduction to .NET. (2024). *Microsoft Docs*. URL: https://learn.microsoft.com/en-us/dotnet/core/introduction. (дата звернення: 06.11.2025).

35. Tutorial: Create a web API with ASP.NET Core. (2024). *Microsoft Docs*. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api. (дата звернення: 06.11.2025).

36. Fowler, M. (2014). Microservices: a definition of this new architectural term. *martinfowler.com*. URL: https://martinfowler.com/articles/microservices.html. (дата звернення: 06.11.2025).

37. PostgreSQL Documentation. (2024). The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/. (дата звернення: 06.11.2025).

38. ASP.NET Core Documentation. (2024). Microsoft. URL: https://learn.microsoft.com/en-us/aspnet/core/. (дата звернення: 06.11.2025).

39. Entity Framework Core Documentation. (2024). Microsoft. URL: https://learn.microsoft.com/en-us/ef/core/. (дата звернення: 06.11.2025).

40. Npgsql: PostgreSQL provider for .NET. (2024). Npgsql. URL: https://www.npgsql.org/efcore/. (дата звернення: 06.11.2025).

41. Docker Documentation. (2024). Docker Inc. URL: https://docs.docker.com/. (дата звернення: 16.11.2025).

42. Docker Compose Overview. (2024). Docker Inc. URL: https://docs.docker.com/compose/. (дата звернення: 16.11.2025).

43. React: The library for web and native user interfaces. (2024). Meta Platforms. URL: https://react.dev/. (дата звернення: 14.11.2025).

44. Vite: Next Generation Frontend Tooling. (2024). ViteJS. URL: https://vitejs.dev/. (дата звернення: 14.11.2025).

45. Mantine: A fully featured React component library. (2024). Mantine. URL: https://mantine.dev/. (дата звернення: 14.11.2025).

46. React Router Documentation. (2024). Remix Software Inc. URL: https://reactrouter.com/en/main. (дата звернення: 16.11.2025).

47. Recharts: A composable charting library built on React components. (2024). Recharts Group. URL: https://recharts.org/en-US/. (дата звернення: 20.11.2025).

48. SHAP (SHapley Additive exPlanations) Documentation. (2024). Scott Lundberg. URL: https://shap.readthedocs.io/en/latest/. (дата звернення: 22.11.2025).

49. Axios: Promise based HTTP client for the browser and node.js. (2024). Axios. URL: https://axios-http.com/docs/intro. (дата звернення: 22.11.2025).

50. Tabler Icons Documentation. (2024). Paweł Kuna. URL: https://tabler.io/icons. (дата звернення: 20.11.2025).

ДОДАТОК А Вміст файлу main.py

```
import logging
import joblib
from pathlib import Path
import numpy as np
import pandas as pd
from fastapi import FastAPI, HTTPException
from contextlib import asynccontextmanager
import shap

from .models import PredictionRequest, PredictionResponse

logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s -
%(message)s")

ml_artifacts = {}
explainer = None

@asynccontextmanager
async def lifespan(app: FastAPI):
    global explainer
    logging.info("Application startup: Loading ML artifacts...")

    artifacts_path = Path("/app/artifacts")

    try:
        ml_artifacts["model"] = joblib.load(artifacts_path / "best_model.joblib")
        ml_artifacts["scaler"] = joblib.load(artifacts_path / "scaler.joblib")
        ml_artifacts["feature_order"] = joblib.load(artifacts_path / "feature_order.joblib")
        logging.info("ML artifacts loaded successfully.")

        try:
            explainer = shap.TreeExplainer(ml_artifacts["model"])
            logging.info("SHAP Explainer initialized successfully.")
        except Exception as e:
            logging.warning(f"Failed to initialize SHAP explainer: {e}")
    except FileNotFoundError as e:
        logging.error(f"Artifact loading error: {e}. Run the training pipeline first.")

    yield

    logging.info("Application shutdown: Clearing ML artifacts...")
    ml_artifacts.clear()
```

```
# --- FastAPI App Initialization ---
app = FastAPI(
    title="OptiFuel: Ship Fuel Consumption API",
    description="An intelligent system to predict fuel consumption for maritime
vessels.",
    version="1.0.0",
    lifespan=lifespan
)

def preprocess_input(data: pd.DataFrame, feature_order: list) -> pd.DataFrame:
    """
    Перетворює вхідні дані з запиту у формат, придатний для моделі.
    """
    processed_data = data.copy()

    weather_mapping = {'Calm': 0, 'Moderate': 1, 'Stormy': 2}
    processed_data['weather_conditions'] =
processed_data['weather_conditions'].map(weather_mapping)

    processed_data['month_sin'] = np.sin(2 * np.pi * processed_data['month'] / 12)
    processed_data['month_cos'] = np.cos(2 * np.pi * processed_data['month'] / 12)
    processed_data.drop('month', axis=1, inplace=True)

    categorical_cols = ['ship_type', 'route_id', 'fuel_type']
    processed_data = pd.get_dummies(processed_data, columns=categorical_cols)

    processed_data = processed_data.reindex(columns=feature_order, fill_value=0)

    return processed_data


@app.get("/", tags=["General"])
def read_root():
    return {"message": "Welcome to the OptiFuel API!"}


@app.post("/predict", response_model=PredictionResponse, tags=["Prediction"])
def predict(request: PredictionRequest):
    if not all(k in ml_artifacts for k in ["model", "scaler", "feature_order"]):
        raise HTTPException(
            status_code=503,
            detail="Service Unavailable: ML artifacts not loaded. Check application logs."
        )
```

```
    try:
        input_df = pd.DataFrame([request.dict()])
        processed_df = preprocess_input(input_df, ml_artifacts["feature_order"])

        scaled_features = ml_artifacts["scaler"].transform(processed_df)

        prediction = ml_artifacts["model"].predict(scaled_features)

        return PredictionResponse(predicted_fuel_consumption=round(prediction[0], 2))

    except Exception as e:
        logging.error(f"Error during prediction: {e}", exc_info=True)
        raise HTTPException(status_code=400, detail=f"Failed to process request:
{str(e)}")


@app.post("/explain")
def explain(request: PredictionRequest):
    if not explainer or not ml_artifacts["scaler"]:
        raise HTTPException(status_code=503, detail="Explainer or Scaler not loaded")

    try:
        input_df = pd.DataFrame([request.dict()])
        processed_df = preprocess_input(input_df, ml_artifacts["feature_order"])
        scaled_features = ml_artifacts["scaler"].transform(processed_df)

        shap_values = explainer.shap_values(scaled_features)

        values = shap_values[0] if isinstance(shap_values, list) else shap_values

        if len(values.shape) > 1:
            values = values[0]

        explanation = {}
        for i, col_name in enumerate(ml_artifacts["feature_order"]):
            explanation[col_name] = float(values[i])

        return explanation

    except Exception as e:
        logging.error(f"Explanation error: {e}")

        raise HTTPException(status_code=400, detail=f"Explanation failed: {str(e)}")
```

ДОДАТОК Б Вміст файлу preprocessor.py

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from pathlib import Path
import logging
import joblib

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

class FuelDataProcessor:
    """
    Клас для повного циклу передпроцесингу даних про ефективність палива на
суднах.
    """
    def __init__(self, raw_data_path: Path, processed_data_dir: Path):
        self.raw_data_path = raw_data_path
        self.processed_data_dir = processed_data_dir
        self.raw_df = None
        self.processed_df = None
        self.X_train, self.X_test, self.y_train, self.y_test = [None] * 4
        self.scaler = StandardScaler()
        self.feature_order = None

    def execute(self, target_column: str, test_size: float = 0.2, random_state: int = 42):
        """Запускає повний конвеєр обробки даних."""
        self._load_data()
        self._preprocess()
        self._split_and_scale_data(target_column, test_size, random_state)
        self._save_artifacts()

    def _load_data(self):
        """Завантажує дані з CSV-файлу."""
        logging.info(f"Завантаження даних з {self.raw_data_path}...")
        try:
```

```
            self.raw_df = pd.read_csv(self.raw_data_path)
            logging.info("Дані успішно завантажено.")
        except FileNotFoundError:
            logging.error(f"Файл не знайдено за шляхом: {self.raw_data_path}")
            raise

    def _preprocess(self):
        """Виконує обробку даних: кодування, інжиніринг ознак, обробку
мультиколінеарності."""
        logging.info("Початок передпроцесингу даних...")
        df = self.raw_df.copy()

        df = self._encode_categorical(df)
        df = self._engineer_features(df)
        df = self._handle_multicollinearity(df)

        self.processed_df = df.drop(columns=['ship_id'])
        logging.info("Передпроцесинг завершено.")

    def _encode_categorical(self, df: pd.DataFrame) -> pd.DataFrame:
        """Кодує категоріальні ознаки."""
        logging.info("Кодування категоріальних ознак...")
        weather_mapping = {'Calm': 0, 'Moderate': 1, 'Stormy': 2}
        df['weather_conditions'] = df['weather_conditions'].map(weather_mapping)
        df = pd.get_dummies(df, columns=['ship_type', 'route_id', 'fuel_type'],
drop_first=True)
        return df

    def _engineer_features(self, df: pd.DataFrame) -> pd.DataFrame:
        """Створює нові, більш інформативні ознаки."""
        logging.info("Інжиніринг ознак...")
        month_map = {
            'January': 1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June': 6,
            'July': 7, 'August': 8, 'September': 9, 'October': 10, 'November': 11, 'December':
12
        }
        df['month_num'] = df['month'].map(month_map)
        df['month_sin'] = np.sin(2 * np.pi * df['month_num'] / 12)
```

```
        df['month_cos'] = np.cos(2 * np.pi * df['month_num'] / 12)
        return df.drop(columns=['month', 'month_num'])

    def _handle_multicollinearity(self, df: pd.DataFrame) -> pd.DataFrame:
        """Видаляє висококорельовані ознаки."""
        logging.info("Обробка мультиколінеарності (видалення CO2_emissions)...")
        return df.drop(columns=['CO2_emissions'])

    def _split_and_scale_data(self, target_column: str, test_size: float, random_state: int):
        """Розділяє дані та масштабує ознаки."""
        logging.info("Розділення даних на тренувальну та тестову вибірки...")
        X = self.processed_df.drop(columns=[target_column])
        y = self.processed_df[target_column]

        self.feature_order = X.columns.tolist()

        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=test_size, random_state=random_state
        )

        logging.info("Масштабування числових ознак...")
        self.X_train = pd.DataFrame(self.scaler.fit_transform(self.X_train),
columns=self.X_train.columns)
        self.X_test = pd.DataFrame(self.scaler.transform(self.X_test),
columns=self.X_test.columns)

    def _save_artifacts(self):
        """Зберігає оброблені дані та артефакти (скейлер, порядок ознак)."""
        logging.info(f"Збереження артефактів у директорію
{self.processed_data_dir}...")
        self.processed_data_dir.mkdir(parents=True, exist_ok=True)

        # Збереження датасетів
        self.X_train.to_csv(self.processed_data_dir / 'X_train.csv', index=False)
        self.X_test.to_csv(self.processed_data_dir / 'X_test.csv', index=False)
        self.y_train.to_csv(self.processed_data_dir / 'y_train.csv', index=False,
header=True)
        self.y_test.to_csv(self.processed_data_dir / 'y_test.csv', index=False, header=True)
```

```
        joblib.dump(self.scaler, self.processed_data_dir / 'scaler.joblib')
        joblib.dump(self.feature_order, self.processed_data_dir / 'feature_order.joblib')

        logging.info("Всі артефакти передпроцесингу успішно збережено.")

if __name__ == '__main__':
    # --- Блок конфігурації для запуску цього файлу напряму ---
    BASE_ML_SERVICE_DIR = Path(__file__).parents[2]

    RAW_DATA_PATH = BASE_ML_SERVICE_DIR /
'data/raw/ship_fuel_efficiency.csv'
    PROCESSED_DATA_DIR = BASE_ML_SERVICE_DIR / 'data/processed'
    TARGET_COLUMN = 'fuel_consumption'

    # --- Запуск конвеєра обробки ---
    try:
        processor = FuelDataProcessor(
            raw_data_path=RAW_DATA_PATH,
            processed_data_dir=PROCESSED_DATA_DIR
        )
        processor.execute(target_column=TARGET_COLUMN)

        print(f"\nКонвеєр обробки даних успішно завершено.")
        print(f"Збережено у: {PROCESSED_DATA_DIR.resolve()}")
    except Exception as e:
        logging.error(f"Під час виконання сталася помилка: {e}", exc_info=True)
```

ДОДАТОК В Вміст файлу AnalyticsController.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.VisualBasic;
using OptiFuel.API.Data;

namespace OptiFuel.API.Controllers
{
   [ApiController]
   [Route("api/[controller]")]
   [Authorize]
   public class AnalyticsController : ControllerBase
   {
      private readonly AppDbContext _context;

      public AnalyticsController(AppDbContext context)
      {
         _context = context;
      }

      [HttpGet("summary")]
      public async Task<IActionResult> GetSummaty()
      {
         var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

         var query = _context.Voyages.Where(v => v.UserId == userId);

         var totalVoyages = await query.CountAsync();
         var totalPredicted = await query.SumAsync(v => v.PredictedFuelConsumption);
         var totalActual = await query.Where(h =>
h.ActualFuelConsumption.HasValue).SumAsync(v =>
v.ActualFuelConsumption!.Value);

         var completedVoyages = await query.Where(h =>
h.ActualFuelConsumption.HasValue).ToListAsync();
         double avgDeviation = 0;
         if (completedVoyages.Any())
         {
            avgDeviation = completedVoyages.Average(h =>
(h.ActualFuelConsumption!.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100);
```

```
        }

        var efficiencyByShip = completedVoyages
            .GroupBy(h => h.ShipType)
            .Select(g => new
            {
                ShipType = g.Key,
                AvgDeviation = g.Average(h => ((h.ActualFuelConsumption!.Value -
h.PredictedFuelConsumption) / h.PredictedFuelConsumption) * 100)
            })
            .OrderByDescending(x => x.AvgDeviation)
            .ToList();

        return Ok(new
        {
            TotalVoyages = totalVoyages,
            TotalPredictedVolume = totalPredicted,
            TotalActualVolume = totalActual,
            GlobalAverageDeviation = avgDeviation,
            ShipEfficiency = efficiencyByShip
        });
    }

    [HttpGet("charts")]
    public async Task<IActionResult> GetCharts()
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        var query = _context.Voyages
            .Where(h => h.UserId == userId && h.ActualFuelConsumption.HasValue);

        var completedVoyages = await query.ToListAsync();

        var shipStats = completedVoyages
            .GroupBy(h => h.ShipType)
            .Select(g => new
            {
                ShipType = g.Key,
                AvgDeviation = g.Average(h => ((h.ActualFuelConsumption!.Value -
h.PredictedFuelConsumption) / h.PredictedFuelConsumption) * 100)
            })
            .ToList();

        var trendStats = completedVoyages
```

```
            .OrderByDescending(h => h.CreatedAt)
            .Take(10)
            .OrderBy(h => h.CreatedAt)
            .Select(h => new
            {
                Date = h.CreatedAt.ToString("MM/dd"),
                Predicted = h.PredictedFuelConsumption,
                Actual = h.ActualFuelConsumption
            })
            .ToList();

        var deviations = completedVoyages.Select(h =>
            ((h.ActualFuelConsumption!.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption) * 100).ToList();

        var histogramStats = new List<object>
        {
            new { Range = "Saving (>3%)", Count = deviations.Count(d => d <= -3) },
            new { Range = "Normal (+/-3%)", Count = deviations.Count(d => d > -3 &&
d <= 3) },
            new { Range = "Warning (3-7%)", Count = deviations.Count(d => d > 3 &&
d <= 7) },
            new { Range = "Critical (>7%)", Count = deviations.Count(d => d > 7) }
        };

        var weatherStats = completedVoyages
            .GroupBy(h => h.WeatherConditions)
            .Select(g => new
            {
                Weather = g.Key,
                AvgConsumption = g.Average(h => h.ActualFuelConsumption!.Value)
            })
            .OrderBy(x => x.AvgConsumption)
            .ToList();

        return Ok(new
        {
            ShipStats = shipStats,
            TrendStats = trendStats,
            HistogramStats = histogramStats,
            WeatherStats = weatherStats});}}}
```

ДОДАТОК Г Вміст файлу HistoryController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using OptiFuel.API.Data;
using OptiFuel.API.Models;
using System.Security.Claims;
using System.Linq.Dynamic.Core;
using OptiFuel.API.DTOs;
using OptiFuel.API.DTOs.PredictionHistory;

namespace OptiFuel.API.Controllers;

[ApiController]
[Route("api/[controller]")]
[Authorize]
public class HistoryController : ControllerBase
{
   private readonly AppDbContext _dbContext;

   public HistoryController(AppDbContext dbContext)
   {
      _dbContext = dbContext;
   }

   [HttpGet]
   [ProducesResponseType(StatusCodes.Status200OK, Type =
typeof(IEnumerable<PredictionHistoryPreviewDto>))]
   public async Task<IActionResult> GetHistory([FromQuery]
ResourceQueryParameters queryParameters)
   {
      var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
      if (userId == null)
      {
         return Unauthorized();
      }
```

```
    var query = _dbContext.Voyages
                    .Where(h => h.UserId == userId)
                    .AsQueryable();

    if (!string.IsNullOrEmpty(queryParameters.DeviationCategory))
    {

       switch (queryParameters.DeviationCategory)
       {
          case "Saving": // <= -3%
             query = query.Where(h => h.ActualFuelConsumption.HasValue &&
                ((h.ActualFuelConsumption.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100) <= -3);
             break;
          case "Normal": // > -3% AND <= 3%
             query = query.Where(h => h.ActualFuelConsumption.HasValue &&
                ((h.ActualFuelConsumption.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100) > -3 &&
                ((h.ActualFuelConsumption.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100) <= 3);
             break;
          case "Warning": // > 3% AND <= 7%
             query = query.Where(h => h.ActualFuelConsumption.HasValue &&
                ((h.ActualFuelConsumption.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100) > 3 &&
                ((h.ActualFuelConsumption.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100) <= 7);
             break;
          case "Critical": // > 7%
             query = query.Where(h => h.ActualFuelConsumption.HasValue &&
                ((h.ActualFuelConsumption.Value - h.PredictedFuelConsumption) /
h.PredictedFuelConsumption * 100) > 7);
             break;
       }
    }

    if (!string.IsNullOrEmpty(queryParameters.ShipType))
    {
```

```
        query = query.Where(h => h.ShipType == queryParameters.ShipType);
    }

    if (!string.IsNullOrEmpty(queryParameters.WeatherCondition))
    {
        query = query.Where(h => h.WeatherConditions ==
queryParameters.WeatherCondition);
    }

    if (!string.IsNullOrWhiteSpace(queryParameters.SortBy))
    {
        var sortOrder = queryParameters.SortOrder?.ToLower() == "asc" ? "ascending" :
"descending";
        query = query.OrderBy($"{queryParameters.SortBy} {sortOrder}");
    }
    else
    {
        query = query.OrderByDescending(h => h.CreatedAt);
    }

    var totalItemCount = await query.CountAsync();

    var items = await query
        .Skip((queryParameters.PageNumber - 1) * queryParameters.PageSize)
        .Take(queryParameters.PageSize)
        .Select(h => new PredictionHistoryPreviewDto
        {
            Id = h.Id,
            CreatedAt = h.CreatedAt,
            RouteId = h.RouteId,
            ShipType = h.ShipType,
            PredictedFuelConsumption = h.PredictedFuelConsumption,
            Distance = h.Distance,
            EngineEfficiency = h.EngineEfficiency,
            FuelType = h.FuelType,
            WeatherConditions = h.WeatherConditions,
            ActualFuelConsumption = h.ActualFuelConsumption
        })
```

```
            .ToListAsync();

        var paginationMetadata = new PaginationMetadata
        {
            TotalItemCount = totalItemCount,
            PageSize = queryParameters.PageSize,
            CurrentPage = queryParameters.PageNumber,
            TotalPageCount = (int)Math.Ceiling(totalItemCount /
(double)queryParameters.PageSize)
        };

        var pagedResult = new Models.PagedResult<PredictionHistoryPreviewDto>
        {
            Items = items,
            Metadata = paginationMetadata
        };

        return Ok(pagedResult);
    }
}
```

ДОДАТОК Д Вміст файлу ForecastPage.jsx

```
import { useState } from "react";
import { Container, Grid, LoadingOverlay, Title, Alert } from "@mantine/core";
import { IconAlertCircle } from "@tabler/icons-react";
import { notifications } from "@mantine/notifications";

import PredictionForm from "../components/PredictionForm";
import PredictionResult from "../components/PredictionResult";

import { useAuth } from "../context/AuthContext";
import { getPrediction, createVoyage } from "../services/apiService";

function ForecastPage() {
  const [result, setResult] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  const { isAuthenticated } = useAuth();

  const handleFormSubmit = async (formData) => {
    setIsLoading(true);
    setError(null);
    setResult(null);

    let predictionResult;
    try {
      if (isAuthenticated) {
        predictionResult = await createVoyage(formData);
        setResult(predictionResult);
        notifications.show({
          title: "Prediction Successful",
          message: "Fuel consumption prediction received successfully.",
          color: "green",
        });
      } else {
        predictionResult = await getPrediction(formData);
```

```
        notifications.show({
          title: 'Success',
          message: 'Prediction received successfully!',
          color: 'green',
        });
      }

      const normalizedResult = {
        predicted_fuel_consumption: predictionResult.predictedFuelConsumption ||
predictionResult.predicted_fuel_consumption
      };

      setResult(normalizedResult);
    } catch (apiError) {
      const errorMessage = apiError.message || "An unexpected error occurred.";
      setError(errorMessage);
      notifications.show({
        title: "Prediction Failed",
        message: errorMessage,
        color: "red",
      });
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <Container size="lg" py="xl">
      <Title order={1} ta="center" mb="xl">
        Fuel Forecaster
      </Title>

      {error && (
        <Alert
          icon={<IconAlertCircle size="1rem" />}
          title="Error!"
          color="red"
          withCloseButton
```

```
        onClose={() => setError(null)}
        mb="md"
      >
        {error}
      </Alert>
    )}

    <Grid>
      <Grid.Col span={{ base: 12, md: 7 }}>
        <div style={{ position: "relative" }}>
          <LoadingOverlay
            visible={isLoading}
            zIndex={1000}
            radius="sm"
            blur={2}
          />
          <PredictionForm
            onPredict={handleFormSubmit}
            isLoading={isLoading}
          />
        </div>
      </Grid.Col>

      <Grid.Col span={{ base: 12, md: 5 }}>
        <PredictionResult result={result} />
      </Grid.Col>
    </Grid>
  </Container>
 );
}

export default ForecastPage;
```

ДОДАТОК Е Вміст файлу HistoryPage.jsx

```
import { useState, useEffect } from "react";
import {
  Table,
  Title,
  Container,
  Alert,
  Loader,
  Center,
  Group,
  Pagination,
  UnstyledButton,
  Text,
  Button,
  Badge,
  CloseButton
} from "@mantine/core";
import {
  IconAlertCircle,
  IconSelector,
  IconChevronUp,
  IconChevronDown,
} from "@tabler/icons-react";
import { useNavigate } from "react-router-dom";
import {
  getHistory,
  getAnalyticsSummary,
  getAnalyticsCharts,
} from "../services/apiService";
import classes from "./HistoryPage.module.css";

import AnalyticsStats from "../components/AnalyticsStats";
import AnalyticsCharts from "../components/AnalyticsCharts";

function Th({ children, reversed, sorted, onSort }) {
  const Icon = sorted
    ? reversed
```

```
      ? IconChevronUp
      : IconChevronDown
    : IconSelector;
  return (
    <Table.Th className={classes.th}>
      <UnstyledButton onClick={onSort} className={classes.control}>
        <Group justify="space-between">
          <Text fw={500} fz="sm">
            {children}
          </Text>
          <Center className={classes.icon}>
            <Icon size="0.9rem" stroke={1.5} />
          </Center>
        </Group>
      </UnstyledButton>
    </Table.Th>
  );
}

function HistoryPage() {
  const navigate = useNavigate();
  const [data, setData] = useState({ items: [], metadata: null });
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  const [analyticsData, setAnalyticsData] = useState(null);
  const [chartsData, setChartsData] = useState(null);

  const [queryParams, setQueryParams] = useState({
    page: 1,
    sortBy: "CreatedAt",
    sortOrder: "desc",
    deviationCategory: null,
    shipType: null,
    weatherCondition: null,
  });

  const applyFilter = (key, value) => {
```

```
  setQueryParams((prev) => ({
    ...prev,
    [key]: value,
    page: 1,
  }));
};

const handleDeviationClick = (label) =>
  applyFilter("deviationCategory", label.split(" ")[0]);
const handleShipClick = (data) => applyFilter("shipType", data.shipType);
const handleWeatherClick = (data) =>
  applyFilter("weatherCondition", data.weather);

const FilterBadge = ({ label, onRemove }) => (
  <Badge
    size="lg"
    variant="light"
    rightSection={<CloseButton size="xs" onClick={onRemove} />}
    pr={3}
  >
    {label}
  </Badge>
);

useEffect(() => {
  const fetchHistory = async () => {
    setLoading(true);
    setError(null);
    try {
      const response = await getHistory(queryParams);
      setData(response);
    } catch (err) {
      setError(err.message || "Failed to fetch prediction history.");
    } finally {
      setLoading(false);
    }
  };
  fetchHistory();
```

```
}, [queryParams]);

useEffect(() => {
  const fetchStats = async () => {
    try {
      const [stats, charts] = await Promise.all([
        getAnalyticsSummary(),
        getAnalyticsCharts(),
      ]);

      setAnalyticsData(stats);
      setChartsData(charts);
    } catch (err) {
      console.error("Failed to load analytics:", err);
    }
  };

  fetchStats();
}, []);

const handleSort = (field) => {
  const isAsc =
    queryParams.sortBy === field && queryParams.sortOrder === "asc";
  setQueryParams({
    ...queryParams,
    sortBy: field,
    sortOrder: isAsc ? "desc" : "asc",
    page: 1,
  });
};

const rows = data.items.map((row) => (
  <Table.Tr
    key={row.id}
    onClick={() => navigate(`/voyages/${row.id}`)}
    className={classes.tableRow}
  >
    <Table.Td>{new Date(row.createdAt).toLocaleDateString()}</Table.Td>
```

```
    <Table.Td>{row.shipType}</Table.Td>
    <Table.Td>{row.routeId}</Table.Td>

    <Table.Td>{row.distance} nm</Table.Td>
    <Table.Td>{row.engineEfficiency}%</Table.Td>
    <Table.Td>{row.fuelType}</Table.Td>
    <Table.Td>{row.weatherConditions}</Table.Td>

    <Table.Td fw={700}>
      {row.predictedFuelConsumption.toLocaleString()}
    </Table.Td>
    <Table.Td>
      {row.actualFuelConsumption
       ? row.actualFuelConsumption.toLocaleString()
       : "—"}
    </Table.Td>
  </Table.Tr>
));

if (loading) {
  return (
    <Center>
      <Loader />
    </Center>
  );
}

if (error) {
  return (
    <Container size="md">
      <Alert
        icon={<IconAlertCircle size="1rem" />}
        title="Error!"
        color="red"
      >
        {error}
      </Alert>
```

```
    </Container>
  );
}

return (
  <Container size="xl">
    <Title order={1} mb="xl">
      Forecast History
    </Title>

    {analyticsData && <AnalyticsStats data={analyticsData} />}

    {chartsData && (
      <AnalyticsCharts
        data={chartsData}
        onDeviationClick={handleDeviationClick}
        onShipClick={handleShipClick}
        onWeatherClick={handleWeatherClick}
      />
    )}

    <Group mb="md" style={{ minHeight: 30 }}>
      {queryParams.deviationCategory && (
        <FilterBadge
          label={`Deviation: ${queryParams.deviationCategory}`}
          onRemove={() => applyFilter("deviationCategory", null)}
        />
      )}
      {queryParams.shipType && (
        <FilterBadge
          label={`Ship: ${queryParams.shipType}`}
          onRemove={() => applyFilter("shipType", null)}
        />
      )}
      {queryParams.weatherCondition && (
        <FilterBadge
          label={`Weather: ${queryParams.weatherCondition}`}
          onRemove={() => applyFilter("weatherCondition", null)}
```

```
    />
  )}

  {(queryParams.deviationCategory ||
    queryParams.shipType ||
    queryParams.weatherCondition) && (
    <Button
      variant="subtle"
      color="gray"
      size="xs"
      onClick={() =>
        setQueryParams({
          page: 1,
          sortBy: "CreatedAt",
          sortOrder: "desc",
          deviationCategory: null,
          shipType: null,
          weatherCondition: null,
        })
      }
    >
      Clear All
    </Button>
  )}
</Group>

<div style={{ marginBottom: "20px" }}></div>

<Table.ScrollContainer minWidth={1000}>
  <Table striped highlightOnHover withTableBorder withColumnBorders>
    <Table.Thead>
      <Table.Tr>
        <Th
          sorted={queryParams.sortBy === "CreatedAt"}
          reversed={queryParams.sortOrder === "desc"}
          onSort={() => handleSort("CreatedAt")}
        >
          Date
```

```
</Th>

<Th
  sorted={queryParams.sortBy === "ShipType"}
  reversed={queryParams.sortOrder === "desc"}
  onSort={() => handleSort("ShipType")}
>
  Ship
</Th>
<Th
  sorted={queryParams.sortBy === "RouteId"}
  reversed={queryParams.sortOrder === "desc"}
  onSort={() => handleSort("RouteId")}
>
  Route
</Th>

<Th
  sorted={queryParams.sortBy === "Distance"}
  reversed={queryParams.sortOrder === "desc"}
  onSort={() => handleSort("Distance")}
>
  Dist.
</Th>
<Th
  sorted={queryParams.sortBy === "EngineEfficiency"}
  reversed={queryParams.sortOrder === "desc"}
  onSort={() => handleSort("EngineEfficiency")}
>
  Eff.
</Th>
<Th
  sorted={queryParams.sortBy === "FuelType"}
  reversed={queryParams.sortOrder === "desc"}
  onSort={() => handleSort("FuelType")}
>
  Fuel
</Th>
```

```
    <Th
      sorted={queryParams.sortBy === "WeatherConditions"}
      reversed={queryParams.sortOrder === "desc"}
      onSort={() => handleSort("WeatherConditions")}
    >
      Weather
    </Th>

    {/* STATS */}
    <Th
      sorted={queryParams.sortBy === "PredictedFuelConsumption"}
      reversed={queryParams.sortOrder === "desc"}
      onSort={() => handleSort("PredictedFuelConsumption")}
    >
      Predicted (L)
    </Th>
    <Th
      sorted={queryParams.sortBy === "ActualFuelConsumption"}
      reversed={queryParams.sortOrder === "desc"}
      onSort={() => handleSort("ActualFuelConsumption")}
    >
      Actual (L)
    </Th>

    {/* <Table.Th>Dev %</Table.Th> */}
  </Table.Tr>
</Table.Thead>
<Table.Tbody>
  {rows.length > 0 ? (
    rows
  ) : (
    <Table.Tr>
      <Table.Td colSpan={9}>
        <Center>No history found.</Center>
      </Table.Td>
    </Table.Tr>
  )}
</Table.Tbody>
```

```
      </Table>
    </Table.ScrollContainer>

    {data.metadata && data.metadata.totalPageCount > 1 && (
     <Group justify="center" mt="xl">
      <Pagination
       total={data.metadata.totalPageCount}
       value={queryParams.page}
       onChange={(newPage) =>
        setQueryParams({ ...queryParams, page: newPage })
       }
      />
     </Group>
    )}
   </Container>
  );
}

export default HistoryPage;
```