ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод прогнозування попиту на товари в системах електронної комерції на основі штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Євгеній ШУЛЬЧЕВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Євгеній ШУЛЬЧЕВСЬКИЙ

Керівник: Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шульчевському Євгенію Олександровичу

1. Тема кваліфікаційної роботи: «Метод прогнозування попиту на товари в системах електронної комерції на основі штучного інтелекту»

керівник кваліфікаційної роботи Віталій ЗАЛИВА, доктор філософії (PhD),

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література , метод прогнозування товарів, вимоги до точності прогнозування товарів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження методів прогнозування попиту в системах електронної комерції.

2. Аналіз методів машинного та глибокого навчання для прогнозування попиту.

3. Розробка методу прогнозування попиту на основі штучного інтелекту.

5. Перелік ілюстративного матеріалу: *презентація*

1. Методи прогнозування попиту в системах електронної комерції.
2. Математична оцінка точності прогнозування попиту з використанням машинного та глибокого навчання.
3. Схема нейромережевого методу прогнозування часових рядів.
4. Схема гібридного методу прогнозування попиту з інтеграцією зовнішніх факторів.
5. Демонстрація роботи методу на реальних даних продажів.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-05.11.2025	
2	Вивчення матеріалів для аналізу розвитку методів прогнозування попиту	06.11-10.11.2025	
3	Дослідження методів машинного та глибокого навчання для прогнозування часових рядів	11.11-14.10.2025	
4	Аналіз особливостей впливу алгоритмів глибокого навчання на точність прогнозування попиту	15.11-18.11.2025	
5	Дослідження сучасних нейромережевих технологій прогнозування попиту	19.11-25.11.2025	
6	Застосування методів машинного та глибокого навчання в системах прогнозування попиту	26.11-30.11.2025	
7	Оформлення роботи: вступ, висновки, реферат	31.11-17.12.2025	
8	Розробка демонстраційних матеріалів	18.12 - 19.12.2025	

Здобувач вищої освіти _____ Євгеній ШУЛЬЧЕВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 75 стор., 6 табл., 9 рис., 37 джерел.

Мета роботи - підвищення точності прогнозування попиту на товари в інтернет-магазинах за рахунок використання методів нейронного навчання та гібридних моделей часових рядів.

Об'єкт дослідження - процес прогнозування попиту в інтернет-магазинах.

Предмет дослідження - алгоритми машинного навчання та алгоритми нейронних мереж та застосування в прогнозуванні попиту на товари.

У роботі проведено комплексний аналіз сучасних методів прогнозування попиту в системах електронної комерції та обґрунтовано необхідність використання алгоритмів машинного й глибокого навчання для підвищення точності прогнозів.

Розглянуто традиційні статистичні моделі, методи машинного навчання та архітектури нейронних мереж, визначено їх переваги та обмеження у контексті онлайн-торгівлі.

Досліджено структурні й функціональні особливості e-commerce платформ, типи доступних даних та фактори, що впливають на формування попиту, включно з поведінковими сигналами, динамічним ціноутворенням, рекомендаційними системами та зовнішніми чинниками. На основі проведеного аналізу розроблено концепцію системи прогнозування попиту, що поєднує алгоритми машинного навчання з елементами глибоких нейронних мереж для моделювання складних і нелінійних залежностей.

Створено та протестовано прототип моделі прогнозування на тестових вибірках даних. Отримані результати демонструють підвищення точності

прогнозів у порівнянні з базовими статистичними методами та підтверджують ефективність застосування гібридних підходів для задач прогнозування в електронній комерції.

Запропоновано рекомендації щодо подальшого вдосконалення моделі, розширення набору вхідних даних та інтеграції системи в інфраструктуру реального інтернет-магазину. Результати дослідження можуть бути використані при розробці систем управління запасами, побудові корпоративної аналітики та створенні інтелектуальних модулів для e-commerce платформ.

КЛЮЧОВІ СЛОВА: ПРОГНОЗУВАННЯ ПОПИТУ, ЕЛЕКТРОННА КОМЕРЦІЯ, МАШИННЕ НАВЧАННЯ, ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ЧАСОВІ РЯДИ, СЕЗОННІСТЬ, LSTM, GRU, ТРАНСФОРМЕРИ, МЕХАНІЗМ УВАГИ, ПІДГОТОВКА ДАНИХ, ПОВЕДІНКОВІ ДАНІ, АНАЛІЗ АНОМАЛІЙ, ДИНАМІЧНЕ ЦІНОУТВОРЕННЯ, РЕКОМЕНДАЦІЙНІ СИСТЕМИ, ОПТИМІЗАЦІЯ ЗАПАСІВ, XGBOOST, LIGHTGBM, ТОЧНІСТЬ ПРОГНОЗУ

ABSTRACT

The text part of the master's qualification thesis comprises 75 pages, 6 tables, 9 figures, and 37 references.

The purpose of the thesis is to improve the accuracy of demand forecasting in online stores through the use of neural learning methods and hybrid time-series models.

The object of the research is the process of demand forecasting in e-commerce platforms.

The subject of the research is machine learning algorithms and neural network algorithms and their application in product demand forecasting.

The thesis provides a comprehensive analysis of modern demand forecasting methods in e-commerce systems and substantiates the need to apply machine learning and deep learning techniques to enhance forecasting accuracy. Traditional statistical models, machine learning methods, and neural network architectures are examined, and their advantages and limitations in the context of online retail are identified.

The structural and functional features of e-commerce platforms are investigated, along with the available data types and the factors influencing demand formation, including behavioral signals, dynamic pricing, recommendation systems, and external conditions. Based on this analysis, a concept for a demand forecasting system is developed, combining machine learning algorithms with deep neural network components to model complex and nonlinear dependencies.

A prototype forecasting model was created and tested on sample datasets. The obtained results demonstrate improved forecast accuracy compared to basic statistical approaches and confirm the effectiveness of hybrid methods in e-commerce forecasting tasks. Recommendations are provided for further model enhancement, expansion of input data sources, and integration of the system into a real online store infrastructure. The research outcomes can be applied in inventory management systems, corporate analytics, and the development of intelligent modules for e-commerce platforms.

KEYWORDS: DEMAND FORECASTING, E-COMMERCE, MACHINE LEARNING, DEEP LEARNING, NEURAL NETWORKS, TIME SERIES, SEASONALITY, LSTM, GRU, TRANSFORMERS, ATTENTION MECHANISM, DATA PREPROCESSING, BEHAVIORAL DATA, ANOMALY ANALYSIS, DYNAMIC PRICING, RECOMMENDATION SYSTEMS, INVENTORY OPTIMIZATION, XGBOOST, LIGHTGBM, FORECAST ACCURACY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI - Artificial Intelligence (штучний інтелект)

API - Application Programming Interface (прикладний програмний інтерфейс)

ARIMA - Autoregressive Integrated Moving Average (авторегресивна інтегрована модель ковзного середнього)

AWS - Amazon Web Services (хмарні обчислювальні сервіси Amazon)

CNN - Convolutional Neural Network (згорткова нейронна мережа)

CRM - Customer Relationship Management (система управління взаєминами з клієнтами)

CSV - Comma-Separated Values (формат табличних даних)

DL - Deep Learning (глибоке навчання)

ERP - Enterprise Resource Planning (система планування ресурсів підприємства)

ETL - Extract, Transform, Load (витяг, трансформація, завантаження даних)

GRU - Gated Recurrent Unit (рекурентна нейронна мережа з вентильованими блоками)

JSON - JavaScript Object Notation (текстовий формат обміну даними)

LSTM - Long Short-Term Memory (довгострокова короткочасна пам'ять у нейронних мережах)

MAE - Mean Absolute Error (середня абсолютна помилка)

ML - Machine Learning (машинне навчання)

MA - Moving Average (ковзне середнє)

MAPE - Mean Absolute Percentage Error (середня абсолютна відсоткова помилка)

MSE - Mean Squared Error (середня квадратична помилка)

NN - Neural Network (нейронна мережа)

RNN - Recurrent Neural Network (рекурентна нейронна мережа)

RMSE - Root Mean Squared Error (корінь середньої квадратичної помилки)

SKU - Stock Keeping Unit (складська одиниця товару)

SQL - Structured Query Language (мова структурованих запитів)

SARIMA - Seasonal ARIMA (сезонна авторегресивна інтегрована модель ковзного середнього)

TF - TensorFlow (фреймворк для машинного навчання)

TFT - Temporal Fusion Transformer (трансформер для часових рядів)

XML - eXtensible Markup Language (розширювана мова розмітки

ВСТУП

Сьогодні електронна комерція швидко стає дуже важливою частиною повсякденного життя споживачів у сучасному світі. В умовах зростаючої конкуренції та швидких змін у споживчих вподобаннях інтернет-магазини повинні вміти прогнозувати попит на товари. Наприклад Rozetka або Prom постійно працюють над аналізом та в залежності від факторів роблять знижки, розпродажі та інше. Це дає можливість компанії оптимізувати запаси на складах, мінімізувати витрати та надати чудовий сервіс для клієнтів. Традиційні методи прогнозування, засновані на історичних даних і простих статистичних моделях, часто виявляються неефективними враховуючи багато факторів які зараз є на ринку.

Для вирішення цих проблем все більше впровадження алгоритмів машинного навчання або ж нейронних мережей відкриває нові можливості для підвищення точності прогнозування попиту. Багато галузей успішно використовують алгоритми глибокого навчання та машинного навчання для побудови прогнозів і аналізу даних. В електронній комерції це можуть бути дані про використання, транзакції, ціни тощо, змінні фактори. Такі моделі дозволяють враховувати багато факторів, таких як зміни в сезоні, рекламні кампанії та зовнішні події, що впливають на попит клієнтів. Наприклад, якщо у TikTok блогер згадає про товар, або послугу це може надати шалений буст до продажів це й називають зовнішнім фактором.

Задоволення клієнтів, краще управління запасами та точніші прогнози можна зробити за допомогою машинного навчання в електронній комерції [3, с. 120]. Таким чином, це підвищує конкурентоспроможність на ринку онлайн-магазинів.

Метою даної роботи є розробка та впровадження системи прогнозування попиту на товари в інтернет-магазинах на основі алгоритмів машинного навчання для підвищення точності прогнозів та оптимізації бізнес-процесі

Об'єктом дослідження є метод прогнозування попиту в інтернет-магазинах.

Предметом дослідження є алгоритми машинного навчання та алгоритми нейронних мереж та їх поєднання та застосування для прогнозування попиту на товари.

Завдання дослідження:

1. Провести аналіз традиційних методів прогнозування попиту.
2. Дослідити алгоритми машинного навчання, що використовуються для прогнозування попиту.
3. Провести порівняння ефективності традиційних методів і сучасних алгоритмів машинного навчання.
4. Оцінити існуючі програмні рішення для прогнозування попиту.
5. Розробити концепцію системи прогнозування попиту на основі алгоритмів машинного навчання.
6. Випробувати створену систему на тестових даних та оцінити її ефективність.

Використання алгоритмів машинного навчання для прогнозування попиту дозволить значно підвищити точність прогнозів у порівнянні з традиційним методами, а також сприятиме оптимізації запасів та підвищенню конкурентоспроможності інтернет-магазинів.

Очікувані результати:

1. Систематизація знань про традиційні та сучасні методи прогнозування попиту.
2. Визначення найбільш ефективних алгоритмів машинного навчання для прогнозування попиту в інтернет-магазинах.
3. Розробка прототипу системи прогнозування попит

4. Отримання результатів тестування системи, які підтверджують її ефективність та практичну користь для бізнесу.

Таким чином, дослідження та впровадження систем прогнозування попиту на товари в інтернет-магазинах на основі машинного навчання або гібриду з нейронними мережами є актуальним і перспективним напрямом, що дозволяє ефективно реагувати на виклики сучасного e-commerce та задовольняти потреби споживачів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Кількісні методи: часові ряди, регресійний аналіз

Кількісні методи прогнозування попиту базуються на аналізі числових даних і є фундаментом більшості аналітичних систем, що використовуються в електронній комерції. Основною ідеєю цих методів є виявлення закономірностей у поведінці попиту на основі історичних даних з подальшим екстраполюванням виявлених тенденцій у майбутнє. На відміну від якісних підходів, кількісні методи дозволяють отримати формалізований результат у вигляді числового прогнозу, що може бути безпосередньо використаний у процесах планування закупівель, управління складом та фінансового прогнозування.

Одним з найпоширеніших підходів у межах кількісних методів є аналіз часових рядів. Часовий ряд представляє собою послідовність значень показника попиту, впорядковану за часом. У контексті електронної комерції такими показниками можуть бути кількість проданих одиниць товару за день, тиждень або місяць, загальна виручка, кількість замовлень чи інші метрики, що безпосередньо або опосередковано характеризують рівень попиту.

Аналіз часових рядів дозволяє виділити кілька ключових компонентів: тренд, сезонність, циклічність та випадкову складову. Тренд відображає довгострокову зміну попиту, наприклад поступове зростання продажів популярної категорії товарів або, навпаки, зниження інтересу до застарілих продуктів. Сезонність характеризує регулярні коливання попиту, які повторюються через певні проміжки часу, наприклад щорічні піки продажів у період свят або сезонні коливання для товарів одягу. Циклічність пов’язана з більш довготривалими економічними або соціальними циклами, тоді як випадкова складова відображає непередбачувані коливання, зумовлені зовнішніми подіями.

У практиці прогнозування попиту в інтернет-магазинах часові ряди часто використовуються для коротко- та середньострокових прогнозів. Наприклад, на основі щоденних або тижневих даних можна передбачити обсяги продажів на найближчі кілька тижнів з урахуванням сезонних піків та історичної динаміки. Це особливо важливо для управління складськими запасами, де помилка в прогнозі може призвести або до дефіциту товару, або до надлишкових запасів і фінансових втрат.

Однією з класичних моделей для аналізу часових рядів є модель ARIMA, яка поєднує в собі авторегресійний підхід, інтегрування та ковзне середнє. Авторегресійна частина моделі описує залежність поточного значення попиту від його попередніх значень, що дозволяє врахувати інерційність попиту. Інтегрування використовується для приведення ряду до стаціонарного вигляду, а ковзне середнє враховує вплив випадкових збурень у попередні моменти часу. Завдяки такій комбінації ARIMA дозволяє досить точно моделювати поведінку попиту за наявності стабільних історичних даних.

Проте, для застосування моделей часових рядів необхідна ретельна підготовка даних. Зокрема, важливо виконати очищення даних від пропусків, аномальних значень та технічних помилок, які можуть виникати внаслідок збоїв у системі обліку або маркетингових акцій. Наявність аномалій може суттєво вплинути на параметри моделі та знизити точність прогнозу. Крім того, необхідно правильно обрати часовий інтервал агрегації даних, оскільки надто дрібна або надто груба агрегація може приховувати важливі закономірності.

Іншим поширеним кількісним підходом є регресійний аналіз, який дозволяє дослідити залежність попиту від одного або кількох факторів. У найпростішому випадку використовується лінійна регресія, де попит розглядається як функція однієї незалежної змінної, наприклад ціни товару. Такий підхід дозволяє оцінити, як зміна ціни впливає на обсяг продажів, що є важливим для прийняття рішень щодо ціноутворення та проведення акцій.

Більш складні моделі множинної регресії враховують одночасний вплив кількох факторів. У контексті електронної комерції це можуть бути витрати на рекламу, рівень знижок, сезонні коефіцієнти, погодні умови, активність користувачів на сайті та інші параметри. Множинна регресія дозволяє побудувати більш гнучку модель, яка краще відображає реальні процеси формування попиту, проте потребує більшої кількості якісних даних та ретельного підбору незалежних змінних.

Важливим етапом регресійного аналізу є оцінка значущості факторів. Для цього використовуються статистичні критерії, такі як t-тести та коефіцієнти кореляції, які дозволяють визначити, які змінні дійсно впливають на попит, а які не мають суттєвого ефекту. Також широко застосовується коефіцієнт детермінації, який показує, яку частку варіації попиту може пояснити побудована модель. Високе значення цього показника свідчить про адекватність моделі, тоді як низьке може вказувати на необхідність перегляду структури моделі або включення додаткових факторів.

Незважаючи на свою популярність, регресійні методи мають певні обмеження. Вони зазвичай припускають лінійний характер залежностей між змінними, що не завжди відповідає реальності в умовах сучасної електронної комерції. Поведінка споживачів часто є нелінійною та залежить від складної взаємодії багатьох факторів, що ускладнює застосування класичних регресійних моделей.

У практичних системах прогнозування попиту часові ряди та регресійний аналіз часто використовуються спільно. Наприклад, часові ряди дозволяють виявити загальну динаміку та сезонні коливання, а регресійний аналіз – оцінити вплив зовнішніх факторів на ці коливання. Такий комбінований підхід забезпечує більш повне уявлення про поведінку попиту та підвищує якість прогнозів.

Таким чином, кількісні методи, зокрема аналіз часових рядів і регресійний аналіз, залишаються важливими інструментами прогнозування попиту в системах електронної комерції. Вони формують базу для подальшого розвитку більш

складних моделей машинного та глибокого навчання, які розширюють можливості класичних підходів і дозволяють враховувати складні нелінійні залежності у великих обсягах даних.

1.2 Якісні методи: експертні оцінки, опитування споживачів

У процесі прогнозування попиту, окрім кількісних методів, доволі часто застосовуються якісні підходи. На відміну від статистичних моделей, які працюють із числовими показниками та історичними даними, якісні методи ґрунтуються на думках, досвіді та інтуїтивних оцінках людей, які безпосередньо взаємодіють із ринком. Основна ідея таких підходів полягає в тому, що в умовах невизначеності або нестачі даних саме людський досвід може дати корисну інформацію для прийняття управлінських рішень.

Якісні методи особливо актуальні у ситуаціях, коли історичні дані відсутні, є неповними або не відображають реальну ринкову ситуацію. Це часто трапляється під час запуску нових товарів, виходу на нові ринки або різких змін у поведінці споживачів. У таких умовах кількісні моделі можуть демонструвати низьку точність, тоді як експертні оцінки дозволяють сформувати хоча б базове уявлення про потенційний попит.

Одним з найпоширеніших якісних підходів є метод експертних оцінок. Він базується на залученні фахівців, які мають практичний досвід у певній галузі та добре орієнтуються в ринкових тенденціях. У сфері електронної комерції такими експертами можуть бути маркетологи, менеджери з продажів, аналітики, логісти або продуктові менеджери. Кожен з них розглядає попит з власної точки зору, що дозволяє сформувати більш комплексне бачення ситуації.

Експертні оцінки можуть збиратися у різних формах: індивідуальні інтерв’ю, групові обговорення, анкетування або мозкові штурми. Найбільш структурованим підходом вважається метод Дельфі, який широко використовується в прогнозуванні складних процесів. Його суть полягає в багатоетапному опитуванні

групи експертів, де кожен раунд супроводжується узагальненням попередніх результатів. Завдяки анонімності учасників зменшується вплив авторитетів і домінування окремих осіб, що сприяє формуванню більш зваженого та об’єктивного прогнозу.

У контексті інтернет-магазину метод Дельфі може застосовуватися, наприклад, для оцінки попиту на нову категорію товарів або визначення очікуваного ефекту від масштабної рекламної кампанії. Залучення експертів із різних підрозділів дозволяє врахувати маркетингові, логістичні та операційні аспекти, які складно формалізувати у вигляді числових змінних.

Разом з тим, експертні методи мають низку обмежень. Основним недоліком є суб’єктивність отриманих оцінок. Якість прогнозу значною мірою залежить від компетентності експертів та їхнього розуміння ринку. Якщо експерти мають обмежений досвід або ґрунтуються на застарілих уявленнях, результати прогнозування можуть бути неточними. Крім того, експертні оцінки зазвичай гірше працюють для довгострокових прогнозів, оскільки ринок електронної комерції змінюється дуже швидко.

Іншим важливим якісним методом є опитування споживачів. Цей підхід передбачає безпосереднє отримання інформації від потенційних або наявних клієнтів щодо їхніх потреб, уподобань та намірів здійснювати покупки. В електронній комерції опитування можуть проводитися у вигляді онлайн-анкет, коротких форм після покупки, pop-up вікон або email-розсилок.

Опитування споживачів дозволяють отримати інформацію, яка часто відсутня у транзакційних даних. Наприклад, клієнт може активно переглядати товар, але не здійснювати покупку через завищену ціну або відсутність потрібної модифікації. Такі причини складно виявити лише на основі поведінкових метрик, тоді як пряме запитання дозволяє отримати цінний зворотний зв’язок.

Особливо корисними опитування є при запуску нових товарів або тестуванні попиту перед масштабуванням. Навіть просте анкетування може дати уявлення про

рівень зацікавленості, прийнятний ціновий діапазон та ключові характеристики, на які звертають увагу покупці. У поєднанні з експертними оцінками це дозволяє сформувати більш обґрунтовану гіпотезу щодо майбутнього попиту.

Водночас результати опитувань також мають обмеження. Намір купити товар не завжди трансформується у реальну покупку. Відповіді респондентів можуть бути упередженими або неточними, особливо якщо опитування проводиться без чіткої методології. Крім того, опитування зазвичай охоплюють обмежену вибірку користувачів і не завжди відображають поведінку всієї аудиторії.

Найбільш ефективним підходом у прогнозуванні попиту є комбінування якісних та кількісних методів. Якісні методи дозволяють сформувати початкові гіпотези та краще зрозуміти контекст ринку, тоді як кількісні моделі забезпечують формалізований та масштабований прогноз. У сучасних системах електронної комерції якісні дані часто використовуються як додаткові сигнали або допоміжна інформація при налаштуванні моделей машинного навчання.

Таким чином, якісні методи прогнозування, зокрема експертні оцінки та опитування споживачів, відіграють важливу роль у формуванні комплексного підходу до прогнозування попиту. Вони не замінюють кількісні методи, але доповнюють їх, дозволяючи враховувати фактори, які складно або неможливо формалізувати за допомогою числових даних. У поєднанні з аналітичними та машинними підходами якісні методи сприяють підвищенню точності прогнозів і зниженню ризиків помилок при плануванні діяльності інтернет-магазинів.

1.3 Переваги та недоліки традиційних методів

До переваг традиційних методів можна віднести те, що вони взагалі досить прості в роботі і доступні майже всім. Наприклад, той же регресійний аналіз чи аналіз часових рядів часто не вимагає якогось особливого софту чи великих серверів. Їх можна застосовувати навіть у невеликих компаніях, де немає

спеціалістів з даних або бюджету на складні алгоритми. Багато хто просто відкриває Excel чи якусь базову програму - і вже можна робити якісь перші прогнози.

Ще один плюс у тому, що результати цих методів зазвичай легко зрозуміти. Їх не треба «розшифровувати» або пояснювати довгою мовою. У регресії, наприклад, одразу видно, як змінюється продаж, якщо змінити ціну чи збільшити витрати на рекламу. Це робить традиційні підходи зручними, коли треба показати керівництву, чому приймається те чи інше рішення, або коли треба швидко підготувати якусь презентацію для зацікавлених людей.

Також традиційні методи досить непогано працюють у стабільних умовах. Коли ринок не «стрибає», а покупці поводяться більш-менш передбачувано, прості моделі можуть давати достатньо точні прогнози. Наприклад, якщо продаж залежить лише від сезонності чи історичних трендів, часового ряду цілком вистачить, щоб зрозуміти, що буде далі. У таких випадках складні алгоритми навіть не потрібні.

Але, попри всі ці переваги, традиційні методи мають і свої мінуси. Головний - вони часто не можуть адекватно врахувати складність сучасного ринку. Наприклад, той же аналіз часових рядів погано реагує на різкі зовнішні зміни: вихід нового сильного конкурента, стрибки в економіці, зміни поведінки споживачів. У таких ситуаціях моделі починають помилятися, бо просто не розуміють нових факторів.

Ще одна проблема - залежність від якісних історичних даних. Якщо даних мало, або вони з пропусками, або тільки за короткий період, традиційні методи починають сильно «пливти». Це особливо відчутно, коли йдеться про нові товари чи інноваційні продукти, де історії продажів просто немає, і модель буквально нема на що спиратися.

1.4 Сучасні підходи до прогнозування на основі нейронного навчання

У класичних задачах і надалі інколи використовують лінійну чи логістичну регресію - як такі собі “мінімальні” моделі. Вони підходять, коли залежності відносно прості, наприклад, зміна ціни → зміна продажів. Але коли даних багато, і вони різного типу (сезонність, акції, поведінка користувачів, клімат, реклама), регресія часто просто “ламається” або дає поверхневий результат.

Тому дедалі частіше застосовують дерева рішень, випадкові ліси та градієнтний бустінг, бо вони вже краще справляються з нелінійностями. Але навіть ці алгоритми не завжди тягнуть там, де попит поводиться дуже нестабільно: наприклад, коли товар стає віральним у TikTok або коли раптово зростає попит через зовнішні події.

У таких випадках нейронні мережі показують себе значно краще. Особливо це стосується моделей типу LSTM, GRU або навіть 1D-CNN, які вміють працювати з часовими рядами краще за класичні методи. Вони ловлять довгі залежності, реагують на коливання і можуть адаптуватись, коли ринок різко змінюється. Для e-commerce це досить важливо, бо попит там то падає, то стрибає — залежно від знижок, платіжних днів, сезону або навіть погоди.

Нейронні мережі також можуть об’єднувати інформацію з різних джерел: продажі, поведінка користувачів на сайті, кліки, пошукові запити, відгуки, соціальні мережі, зовнішні фактори. Така “змішана” інформація дає кращий прогноз, ніж прості методи, які виставляють акцент на 1–2 параметри.

Якщо говорити простіше, традиційні методи — це історія про “коли все рівно і спокійно”. Якщо ринок стабільний, сезонність повторюється щороку, а покупці не дуже змінюють свою поведінку, то звичайні часові ряди або лінійна регресія більш-менш справляються з прогнозами.

Але проблема в тому, що сучасний ринок рідко буває стабільним. Поведінка клієнтів змінюється швидше, ніж встигаєш це помітити:

1. Нові конкуренти.
2. Віральні тренди.
3. Зміни в економіці.
4. Логістичні збої.
5. Сезонні піки, які не схожі на попередні.

Тут класичні підходи починають давати збої. Вони не враховують складних зв’язків і обмежені у кількості факторів, з якими можуть працювати.

Нейронні мережі навпаки, здатні навчатися на величезних об’ємах даних і враховувати навіть дуже дивні та непрямі залежності. Наприклад, модель може пов’язати погоду, поведінку в соцмережах, кількість додавань у кошик, курс валюти та ще купу всього - і зробити прогноз, який класичним методам просто недосяжний.

Однак є й свої мінуси. Нейронні мережі — це фактично “чорна скринька”. Пояснити керівнику, чому модель вирішила, що продажі будуть такими, досить важко. Традиційні моделі у цьому сенсі більш прозорі: там все видно по коефіцієнтах і графіках.

Ще один момент - ресурси. Глибоке навчання дороге та потребує багато ресурсів для навчання. Традиційні методи дешевші і простіші, їх може використати люба людина.

1.5 Огляд існуючих програмних рішень для прогнозування попиту

Точне прогнозування попиту є ключовим фактором успішного управління ланцюгами постачання та задоволення потреб клієнтів. Існує широкий спектр програмних рішень, які допомагають компаніям передбачати споживчий попит, оптимізувати запаси та підвищувати ефективність операцій. Розглянемо деякі популярні локальні рішення цієї проблеми.

Streamline - це універсальне ПЗ, яке допомагає прогнозувати попит і керувати запасами. Воно бере історичні дані по продажах і намагається розібратися, де там

тренди, де сезонність, а де якісь цикли. Програма вміє автоматично робити прогнози для кожного товару окремо, використовуючи попередні продажі, маркетингові активності і зміни по сезону. Також Streamline нормально інтегрується із різними ERP-системами типу SAP, Microsoft Dynamics або QuickBooks. Через це його часто використовують саме середні або більші компанії, де вже є якісь складні процеси і потрібно все зв’язати докупи.

Одна з “фішок” Streamline — це моделювання різних сценаріїв. Можна подивитись, що буде, якщо, наприклад, зменшити запаси чи трохи змінити бюджет на маркетинг. Це дає можливість приймати більш обдумані рішення і не діяти навмання.

ForecastPRO - це більш «заточена» під статистику програма. Вона розрахована переважно на компанії середнього та великого розміру, де даних реально багато. Програма будує моделі часових рядів, сама знаходить тренди й сезонність і може інтегруватись із системами, які відповідають за управління запасами на складах. Її особливість у тому, що вона може комбінувати методи: наприклад, на короткий період використовувати просту регресію, а на довший - той самий ARIMA. Через це ForecastPRO легко підлаштовується під різні види бізнесів.

ForecastPRO забезпечує високу точність завдяки можливості комбінування різних методів прогнозування. Наприклад, програма може поєднувати просту лінійну регресію для короткострокових прогнозів і метод ARIMA для аналізу довгострокових тенденцій. Це дозволяє адаптувати програму до специфіки бізнесу.

GMDH Streamline - це вже більш «просунуте» рішення, бо поєднує традиційну статистику з алгоритмами машинного навчання [5]. Воно максимально автоматизує процес прогнозування, тобто людина втручається мінімально. Алгоритми можуть працювати з великими масивами інформації, знаходити приховані залежності між факторами і допомагати оптимізувати управління запасами.

Одна з явних переваг GMDH Streamline - це підтримка роботи з великими наборами даних із різних джерел: бази даних, ERP, Excel-файли. Плюс є інструменти візуалізації, що полегшує аналіз і прийняття рішень.

SmartCorp орієнтована більше на малий та середній бізнес. Вона допомагає прогнозувати попит, враховуючи сезонні коливання, вплив знижок, акцій тощо. Програма автоматично генерує звіти і може інтегруватися з популярними бухгалтерськими системами типу Xero чи MYOB.

SmartCorp особливо підходить таким компаніям, які можуть не мати великих ресурсів для вираховування, але все одно хочуть оптимізувати склад і не тримати зайві запаси. Вона проста у використанні і доступна за ціною, тому її часто обирають невеликі бізнеси.

JDA Demand - це ще один представник локальних програм для прогнозування попиту, орієнтований на великі компанії. Програма використовує моделі прогнозування, що базуються на історичних даних, і дозволяє враховувати складні фактори, такі як зміни у постачаннях, сезонність, маркетингові заходи. Її функціонал також включає автоматичне створення оптимальних графіків замовлень та рекомендації для управління запасами.

JDA Demand пропонує інтеграцію з іншими продуктами компанії, такими як рішення для управління ланцюгами постачання, що дозволяє створювати комплексну екосистему управління бізнес-процесами.

Переваги локальних програм:

1. Висока точність прогнозів завдяки використанню різних методів аналізу.
2. Повний контроль над даними, що особливо важливо для компаній, які працюють із конфіденційною інформацією.
3. Інтеграція з ERP-системами та іншим корпоративним ПЗ.
4. Можливість адаптації до специфіки бізнесу завдяки широким можливостям налаштувань.

Недоліки локальних програм:

1. Високі витрати на впровадження та обслуговування.
2. Потреба у кваліфікованому персоналі для налаштування та підтримки.
3. Обмеження щодо роботи в реальному часі, порівняно з хмарними рішеннями.

Локальні програми залишаються важливим інструментом для прогнозування попиту, особливо для компаній, які цінують конфіденційність даних і потребують високого рівня кастомізації. Проте для більшої гнучкості та швидкості обробки інформації все частіше використовуються хмарні рішення, які доповнюють можливості локальних систем.

1.6 Хмарні рішення для прогнозування попиту

Веб-сервіси та хмарні рішення займають провідну позицію у сфері прогнозування попиту завдяки своїй гнучкості, масштабованості та здатності обробляти великі обсяги даних у реальному часі. Вони забезпечують доступ до передових алгоритмів машинного навчання, інструментів аналізу та моделювання, що робить їх незамінними для компаній, які прагнуть швидко адаптуватися до змін ринку. Розглянемо деякі з найбільш популярних хмарних сервісів і їхні можливості.

NetSuite — це багатофункціональна хмарна платформа, яка включає модуль прогнозування попиту. Система забезпечує автоматизоване планування запасів, прогнозування продажів і аналіз трендів. Вона підтримує інтеграцію з іншими бізнес-модулями, такими як управління фінансами, CRM та ERP.

Основні переваги:

1. Автоматичне створення прогнозів на основі історичних даних і сезонності.
2. Інструменти для відстеження фактичного попиту та порівняння його з прогнозованими показниками.
3. Можливість працювати у хмарі без необхідності встановлення програмного забезпечення.

NetSuite підходить для компаній середнього та великого бізнесу, які потребують комплексного рішення для управління всіма аспектами операційної діяльності.

SAP IBP — це потужна хмарна платформа, яка пропонує рішення для управління ланцюгами постачання, включно з прогнозуванням попиту. Платформа використовує штучний інтелект і машинне навчання для створення точних прогнозів і забезпечує інтеграцію з іншими продуктами SAP, такими як SAP S/4HANA.

Основні можливості:

1. Аналіз попиту в режимі реального часу.
2. Інтеграція зовнішніх факторів, таких як економічні показники або погодні умови, у прогнозування.
3. Інструменти для симуляції сценаріїв («що якщо») і оптимізації планування.

SAP IBP підходить для великих міжнародних компаній із розгалуженою логістичною мережею.

Amazon Web Services (AWS) Forecast — це хмарний сервіс для прогнозування, який використовує машинне навчання для створення точних прогнозів. AWS Forecast інтегрується з іншими продуктами AWS, такими як Amazon S3 чи Amazon Redshift, і дозволяє обробляти великі обсяги даних [6].

Особливості:

1. Підтримка моделей машинного навчання, таких як DeepAR+ для роботи з часовими рядами.
2. Інтуїтивний інтерфейс для налаштування прогнозів без необхідності глибоких технічних знань.
3. Гнучкість у масштабуванні залежно від обсягів даних.

AWS Forecast підходить для компаній будь-якого розміру, які вже використовують екосистему AWS.

Google Cloud AI Forecasting є частиною екосистеми Google Cloud, що забезпечує інструменти для прогнозування попиту на основі штучного інтелекту. Ця платформа дозволяє будувати точні прогнози, враховуючи численні змінні, такі як сезонність, ціни та поведінка клієнтів [7].

Ключові переваги:

1. Інтеграція з Google BigQuery для роботи з великими обсягами даних.
2. Використання передових алгоритмів машинного навчання.
3. Підтримка мультимодульного аналізу для побудови комплексних моделей.

Google Cloud AI Forecasting є оптимальним вибором для компаній, які працюють із великими даними та хочуть використовувати можливості штучного інтелекту.

Переваги веб-сервісів та хмарних рішень:

1. Гнучкість і масштабованість: хмарні сервіси легко адаптуються до змінних обсягів даних і бізнес-потреб.
2. Актуальність даних: забезпечення доступу до прогнозів у реальному часі.
3. Інтеграція з іншими системами: можливість взаємодії з ERP, CRM та іншими бізнес-додатками.
4. Скорочення витрат на інфраструктуру: немає необхідності в локальному обладнанні, оскільки всі обчислення відбуваються у хмарі.

Недоліки веб-сервісів:

1. Залежність від стабільного інтернет-з’єднання.
2. Ризики, пов’язані з безпекою та конфіденційністю даних.
3. Регулярні витрати на передплату, які можуть перевищувати разову оплату за локальні програми

1.7 Додаткові аспекти функціонування систем електронної комерції

Сучасні платформи електронної комерції вже давно перестали бути простими сайтами з відображенням товарів. Вони перетворилися на складні багаторівневі системи, де кожен компонент впливає на поведінку користувачів і формує динаміку продажів. Прогнозування попиту в таких системах неможливе без урахування особливостей функціонування платформи, адже майже кожна частина інфраструктури залишає свій слід у даних. Якщо ці аспекти не включити в модель або хоча б не розуміти їхнього впливу, прогноз буде неповним і менш точним.

Одним з ключових елементів є механізм рекомендацій. Практично кожний великий маркетплейс використовує персональні рекомендації, побудовані на алгоритмах машинного навчання. Це можуть бути блоки на головній сторінці, поради в картці товару або рекомендації у кошику. Основна особливість рекомендацій полягає в тому, що вони створюють додатковий попит, якого не було б без їхнього існування. Наприклад, якщо користувач шукає смартфон, система може порадити йому захисне скло чи чохол, і значна частина покупців дійсно додає ці товари в кошик. Таким чином формується взаємозв’язок між товарами, який ускладнює прогнозування, адже попит на одні категорії може виникати як побічний ефект від перегляду зовсім інших позицій.

Ще одним важливим фактором є механіка фільтрів і сортування. Користувачі рідко переглядають весь каталог. Замість цього вони сортують товари за ціною, популярністю або рейтингом, а також застосовують фільтри за брендом, кольором, розміром і десятками інших характеристик. Порядок відображення товарів може суттєво впливати на продажі. Товар, який відображається першим у списку, отримує значно більше уваги, ніж той, що знаходиться на третій або четвертій сторінці. Якщо протягом певного періоду товар піднімався в каталозі за рахунок змін алгоритму сортування, його попит може тимчасово збільшитися, і цей ефект потрібно враховувати, щоб не сприймати його як природне зростання інтересу.

Інфраструктура логістики також відіграє важливу роль у формуванні попиту. Час доставки, вартість доставки, наявність пунктів видачі і навіть стабільність кур'єрських служб можуть впливати на рівень продажів. Якщо доставка стає швидшою, попит часто зростає навіть без додаткової реклами. Якщо ж відбуваються затримки або проблеми з поставками, попит може знизитися, хоча потреби покупців залишаються незмінними. Зокрема, у сезонні піки, коли навантаження на логістичні служби збільшується, продажі можуть знизитися не через зміну поведінки користувачів, а через обмеження самої системи.

Окрему увагу заслуговує механіка повернень. У багатьох категоріях електронної комерції, наприклад у одязі чи електроніці, рівень повернень може досягати значних величин. Повернення часто не враховують у прогнозах попиту, але вони впливають на реальну картину продажів. У даних це може виглядати як сплеск у продажах, за яким через кілька днів відбувається зворотний спад. Якщо не врахувати цю особливість, модель може неправильно інтерпретувати такі коливання. Повернення також впливають на стан складу, а отже, і на доступність товарів, що також формує попит.

Поведінка користувачів на сайті створює величезний пласт даних, які можуть пояснювати зміни у продажах. Кількість переглядів, додавання в обране, пошукові запити, кліки по банерах, глибина скролу, повторні візити — все це сигнали, які відображають інтерес до товару задовго до того, як відбудеться покупка. Важливо розуміти, що між цими діями і фактичними продажами може пройти значний час.

У деяких категоріях, наприклад у побутовій техніці, користувачі можуть вивчати товар багато днів або навіть тижнів, а потім робити покупку. Зміна поведінкових патернів часто передує змінам у продажах, і грамотне прогнозування повинно враховувати цей лаг.

Важливою складовою роботи систем електронної комерції є ціноутворення. Багато магазинів використовують динамічні ціни, які можуть змінюватися протягом доби залежно від попиту, конкурентів або залишків. Зміна ціни може спричинити як різке зростання, так і падіння продажів. Якщо цей фактор не

врахувати, модель може неправильно оцінити вплив інших змінних. Наприклад, зниження ціни може збігтися з виходом позитивного відгуку або появою рекламної кампанії, і якщо модель не бачить реальних причин, вона зробить неправильний висновок.

Зовнішні фактори також мають суттєвий вплив. Економічна ситуація, свята, погода, збої у роботі банківських систем, зміна купівельної спроможності, інформаційний фон у ЗМІ — усе це може впливати на поведінку покупців. У деяких випадках зовнішній фактор може визначати майже всю динаміку попиту. Наприклад, у період сильних морозів зростає попит на обігрівачі, а під час дощів зростає інтерес до дощовиків і парасольок. Якщо в моделі немає таких факторів, прогноз буде неповним.

Ще одна важлива особливість систем електронної комерції полягає у тому, що вони містять величезні обсяги даних, які надходять з різних джерел. Транзакційні дані, інформація про перегляди, логістичні дані, дані про повернення, відгуки, перепис товарів, рекламні витрати і ще десятки інших джерел створюють складний інформаційний простір. Частина цих даних надходить у реальному часі, інша частина має затримку. Якщо ці джерела не синхронізовані або частково суперечать одне одному, це створює додаткові труднощі у прогнозуванні.

Усе це свідчить про те, що прогнозування попиту в електронній комерції не зводиться лише до аналізу часових рядів. Це комплексна задача, де потрібно враховувати роботу всієї екосистеми платформи. Рекомендації, логістика, ціноутворення, повернення, поведінка користувачів, динаміка каталогу і зовнішні фактори разом формують складний, нерівномірний і багатоаспектний попит. Прогнозування стає більш точним тоді, коли модель уміє розпізнати ці закономірності та відокремити природні зміни від випадкових коливань. Саме тому системи прогнозування повинні бути гнучкими, адаптивними і здатними працювати з великими обсягами різнорідних даних.

2. АНАЛІЗ МОДЕЛЕЙ, МЕТОДІВ ТА ЗАСОБІВ ПРОГНОЗУВАННЯ ПОПИТУ

2.1 Структурні компоненти систем електронної комерції

Електронна комерція є однією з найбільш динамічних галузей сучасної економіки. За останні роки спостерігається стрімке зростання обсягів онлайн-продажів, що зумовлює необхідність впровадження ефективних систем управління товарними запасами та прогнозування попиту. Системи електронної комерції мають низку специфічних особливостей, які суттєво впливають на процеси прогнозування.

Сучасна система електронної комерції складається з декількох взаємопов'язаних компонентів. До основних належать: каталог товарів, система управління замовленнями, система обліку складу, платіжний модуль, система доставки та аналітичний блок. Кожен з цих компонентів генерує великі обсяги даних, які можуть використовуватися для прогнозування попиту.

Каталог товарів містить інформацію про характеристики продукції, ціни, наявність на складі, зображення та описи. Ця інформація постійно оновлюється і впливає на купівельну поведінку користувачів. Система управління замовленнями фіксує всі транзакції, включаючи час замовлення, склад кошика, загальну суму, застосовані знижки та промокоди.

Система обліку складу відстежує рух товарів – надходження, відвантаження, повернення, списання. Ці дані є критично важливими для розуміння реальної динаміки продажів та планування закупівель. Платіжний модуль зберігає інформацію про методи оплати, успішність транзакцій, час обробки платежів.

Прогнозування попиту в e-commerce стикається з рядом специфічних викликів. Перший з них – висока мінливість попиту. На відміну від традиційної

роздрібної торгівлі, де попит змінюється порівняно плавно, в онлайн-середовищі можливі різкі стрибки або падіння продажів.

Другий виклик – великий асортимент товарів. Типовий інтернет-магазин може мати десятки тисяч SKU (Stock Keeping Unit), і для кожного потрібен окремий прогноз. При цьому для багатьох товарів історія продажів може бути короткою або взагалі відсутньою (новинки).

Третій виклик – холодний старт. Для нових товарів або нових категорій немає історичних даних, тому традиційні методи прогнозування, що базуються на часових рядах, не працюють. Необхідно використовувати додаткову інформацію – характеристики товару, схожість з існуючими продуктами, початкові маркетингові зусилля.

Четвертий виклик – аномалії та викиди. Промоакції, згадки в медіа, технічні збої на сайті – все це створює аномальні точки в історії продажів, які можуть заплутати алгоритми прогнозування. Необхідна попередня обробка даних для виявлення та коректного врахування таких ситуацій.

П'ятий виклик – взаємозв'язки між товарами. Покупка одного товару може стимулювати або пригнічувати попит на інші. Наприклад, придбання смартфона часто супроводжується купівлею чохла та захисного скла. Врахування таких перехресних ефектів значно ускладнює моделювання.

2.2 Функціональні особливості e-commerce систем

На відміну від традиційної роздрібної торгівлі, електронна комерція характеризується кількома унікальними особливостями.

По-перше, це доступність 24/7 – покупці можуть здійснювати покупки в будь-який час доби, що створює нерівномірне навантаження на систему та змінює патерни попиту.

По-друге, географічна розподіленість клієнтів вимагає врахування регіональних особливостей, часових поясів, локальних святкових днів.

По-третє, висока конкуренція призводить до частих цінових змін, акцій, розпродажів. Ці маркетингові заходи суттєво впливають на попит і їх складно передбачити заздалегідь. По-четверте, сезонність в онлайн-торгівлі може відрізнятися від офлайн через специфіку асортименту та можливості швидкої доставки з інших регіонів.

Додатковою особливістю є вплив зовнішніх факторів – соціальних мереж, інфлюенсерів, вірусних трендів. Один пост популярного блогера може спричинити різке зростання попиту на конкретний товар протягом декількох годин. Традиційні методи прогнозування часто не встигають адаптуватися до таких швидких змін.

2.3 Типи даних в системах електронної комерції

Для ефективного прогнозування попиту необхідно враховувати певні вхідні дані. Транзакційні дані включають історію замовлень, кількість проданих одиниць, виручку, ціну на момент покупки. Поведінкові дані охоплюють перегляди товарів, додавання в кошик, порівняння, читання відгуків, час перебування на сторінці.

Контекстні дані містять інформацію про час доби, день тижня, сезон, погодні умови, події календаря. Демографічні дані клієнтів (якщо доступні) - вік, стать, місце проживання, історія покупок - допомагають сегментувати аудиторію та будувати персоналізовані прогнози.

Зовнішні дані включають макроекономічні показники, індекси споживчої впевненості, дані конкурентів, трендові запити в пошукових системах, згадки в соціальних мережах. Інтеграція всіх цих типів даних дозволяє отримати більш точні прогнози.

Для ефективного прогнозування попиту необхідно враховувати певні типи даних. Транзакційні дані можуть включають історію замовлень, кількість проданих товарів, виручку, ціну на момент покупки. Поведінкові дані охоплюють перегляди

товарів, додавання в кошик, порівняння, читання відгуків, час перебування на сторінці.

Контекстні дані містять інформацію про час доби, день тижня, сезон, погодні умови, події календаря. Демографічні дані клієнтів - вік, стать, місце проживання, історія покупок - допомагають сегментувати аудиторію та будувати персоналізовані прогнози.

Порівняльна характеристика типів даних та їх вплив наведено у таблиці 2.1

Таблиця 2.1

Порівняльна характеристика типів даних та їх вплив

Тип даних	Приклади	Частота оновлення	Вплив на прогноз
Транзакційні	Продажі, замовлення, повернення	В режимі реального часу	Критичний
Поведінкові	Перегляди, клік, час на сторінці	В режимі реального часу	Високий
Контекстні	Час, сезон, погода, свята	Щоденно	Високий
Товарні	Характеристики, категорія, бренд	Рідко	Середній
Цінові	Ціна, знижки, промо	Щоденно	Високий
Конкурентні	Ціни конкурентів, акції	Щотижнево	Середній
Макроекономічні	Інфляція, безробіття, ВВП	Щомісяця	Низький

2.4 Класифікація товарів за характером попиту

Для вибору оптимального методу прогнозування важливо класифікувати товари за характером попиту. Можна виділити кілька основних категорій:

1. Товари з стабільним попитом - це продукти повсякденного вжитку, базові товари, які продаються регулярно з невеликими коливаннями. Для них підходять класичні статистичні методи прогнозування.
2. Сезонні товари – продукція, попит на яку залежить від сезону, святкових періодів або погодних умов. Для таких товарів критично важливо виявити та врахувати сезонні патерни.
3. Трендові товари – продукти, популярність яких може різко зростати або падати під впливом модних тенденцій. Прогнозування для них вимагає аналізу додаткових сигналів з соціальних мереж та медіа.
4. Товари з нерегулярним попитом - продукція, яка купується рідко та непередбачувано. Для них складно побудувати точний прогноз на основі історії, тому використовуються підходи на основі схожості або зовнішніх факторів.
5. Нові товари - продукти без історії продажів. Для них застосовуються методи transfer learning, де знання з схожих товарів переносяться на новий продукт.

2.5 Обґрунтування вибору методів дослідження

Для вирішення проблеми прогнозування попиту необхідно провести дослідження існуючих методів та підходів. Вибір методів дослідження розділити на наступнимі факторами:

1. Аналіз часових рядів – базовий метод для розуміння структури даних про продажі. Дозволяє виявити тренди, сезонність, циклічність та випадкові змінні.

2. Статистичні методи – дозволяють забезпечити математично обґрунтований підхід до прогнозування з можливістю оцінки інтервалів.
3. Методи машинного навчання – дозволяють моделювати складні нелінійні залежності та автоматично виявляти паттерни в даних.
4. Глибоке навчання – необхідне для обробки багаторозмірних даних та захоплення довгострокових залежностей у послідовностях.
5. Порівняльний аналіз – дозволяє об'єктивно оцінити переваги та недоліки різних підходів.

Обраний список факторів дозволяє нам оцінити повноту прогнозування починаючи з базових архітектур переходячи до найновіших архітектур, що дозволяє нам отримати максимально точний результат та повноту даних.

2.6 Традиційні статистичні методи

Протягом десятиліть для прогнозування попиту використовувалися класичні статистичні підходи. Найпростішим з них є метод ковзного середнього (Moving Average), який обчислює прогноз як середнє значення попередніх n спостережень. Формула методу простого ковзного середнього має вигляд:

$$y_{t+1} = \frac{1}{n}\sum_{i=0}^{n-1} y_t - i \tag{2.1}$$

Де:

y_{t+1} – прогнозоване значення

y_t – фактичне значення попиту в момент часу t

n – розмір вікна усереднення

Авторегресійні моделі базуються на припущенні, що поточне значення можна виразити через попередні значення ряду.

$$y_t = c + \sum_{i=1}^{p} \varphi_i y_{t-i} + \varepsilon_t \tag{2.2}$$

Де φ_i- коефіцієнти авторегресії, c – константа, ε_t - випадкова помилка

Побудова ARIMA-моделі зазвичай складається з кількох кроків, і кожен із них потрібен для того, щоб модель поводилася адекватно й давала корректний прогноз.

Спершу потрібно перевірити, чи є часовий ряд стаціонарним. Для цього часто використовують тест Діккі-Фулера. Якщо ряд зміщується у тренд або сильно змінює дисперсію, моделлю важко працювати, тому стаціонарність — це перший базовий момент.

Якщо стаціонарності немає, визначають, скільки разів потрібно продиференціювати ряд, тобто знайти порядок d. Суть у тому, щоб прибрати тренд або інші зміни, які заважають моделі стабільно працювати.

Далі переходять до ідентифікації параметрів p та q, які відповідають за авторегресійну частину та ковзне середнє. Для цього дивляться графіки ACF і PACF — вони допомагають зрозуміти, яка кількість лагів дає найбільший сенс.

Після цього модель можна оцінювати. Зазвичай параметри підбирають методом максимальної правдоподібності, щоб знайти такі значення, які найкраще “пояснюють” поведінку ряду.

У ARIMA можна виділити такі критичні параметри:

1. Порядки p, d, q – неправильний вибір призводить до недонавчання або перенавчання
2. Вимога стаціонарності – обмежує застосування для трендових рядів
3. Лінійність моделі – не дозволяє захопити нелінійні ефекти

Порівняльну характеристику статистичних методів можна побачити на таблиці 2.2

Таблиця 2.2

Порівняльна характеристика статистичних методів

Метод	**Складність**	**Точність**	**Тренд**	**Сезонність**	**Нелінійність**	**Екзогенні змінні**
MA	Низька	Низька	Ні	Ні	Ні	Ні
ES	Низька	Середня	Ні	Ні	Ні	Ні
Holt	Середня	Середня	Так	Ні	Ні	Ні
Holt-Winters	Середня	Висока	Так	Так	Ні	Ні
ARIMA	Висока	Висока	Так	Ні	Ні	Частково
SARIMA	Висока	Дуже висока	Так	Так	Ні	Частково

2.7 Методи машинного навчання

Випадковий ліс є ансамблем дерев рішень, де кожне дерево навчається на випадковій підвибірці даних. Фінальний прогноз має такий вигляд:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}b \tag{2.3}$$

де B – кількість дерев

XGBoost - оптимізована реалізація з додатковими можливостями:

1. Регуляризація
2. Паралелізація обчислень
3. Обробка пропущених значень

LightGBM використовує leaf-wise стратегію побудови дерев:

1. Швидше навчання на великих наборах даних
2. Менше споживання пам'яті
3. Підтримка категоріальних ознак

Критичні параметри ансамблевих методів такі:

1. Кількість дерев (estimators) - мало дерев = недонавчання, багато = зайві обчислення
2. Глибина дерев (max_depth) - контролює складність моделі
3. Темп навчання (learning_rate) - впливає на швидкість збіжності
4. Параметри регуляризації - запобігають перенавчанню

Загальну картину порівняння методів машинного навчання можна побачити у таблиці 2.3.

Таблиця 2.3

Порівняння методів машинного навчання

Метод	**Точність**	**Швидкість навчання**	**Швидкість прогнозу**	**Інтерпретованість**	**Обробка категорій**
Linear Regression	Низька	Дуже висока	Дуже висока	Висока	Погано
Ridge/Lasso	Низька-Середня	Висока	Висока	Висока	Погано
Decision Tree	Середня	Висока	Висока	Висока	Добре
Random Forest	Висока	Середня	Середня	Середня	Добре
XGBoost	Дуже висока	Середня	Висока	Низька	Добре
LightGBM	Дуже висока	Висока	Висока	Низька	Відмінно

2.8 Нейронні мережі для прогнозування часових рядів

Багатошаровий парцептрон. Структура MLP для прогнозування:

Yt-n…., yt-1 + екзогенні змінні

Приховані шари з активаціями

Вихідний шар: $\widehat{y_t}$

Формула прямого проходу:

$$h^{(l)} = f\left(W^{(l)}h^{(l-1)} + b^{(l)}\right) \quad (2.3)$$

$$\hat{y} = W^{(\text{out})}h^{(\text{last})} + b^{(\text{out})} \quad (2.4)$$

Де:

$h^{(l)}$ - активації шару

W та b - ваги та зміщення

f - функція активації

Базова RNN обробляє послідовності:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \quad (2.5)$$

$$y_t = W_{hy}h_t + b_y \quad (2.6)$$

Де h_t – прихований стан.

Проблема: затухання/вибух градієнтів при навчанні на довгих послідовностях.

LSTM (Long Short-Term Memory) - вирішує проблему градієнтів через гейти:

Гейт забування:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad (2.7)$$

Де (f_t) - коефіцієнт забування

$(W_f), (b_f)$ - ваги та зміщення гейта забування

Вхідний гейт:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2.8)$$

Де (i_t) - вхідний гейт

$(W_i), (b_i)$ - ваги та зміщення вхідного гейта

Оновлення стану:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \quad (2.9)$$

Де (C_t) - новий внутрішній стан

(C_{t-1}) - попередній стан

Критичні параметри RNN/LSTM/GRU:

1. Кількість шарів - глибші мережі можуть моделювати складніші залежності, але важче навчаються
2. Розмір прихованого стану - контролює пам'ять мережі
3. Dropout – запобігає перенавчанню
4. Довжина послідовності - дуже довгі послідовності все ще проблематичні для LSTM

На рисунку 2.1 показано порівняння архітектур RNN, LSTM та GRU.

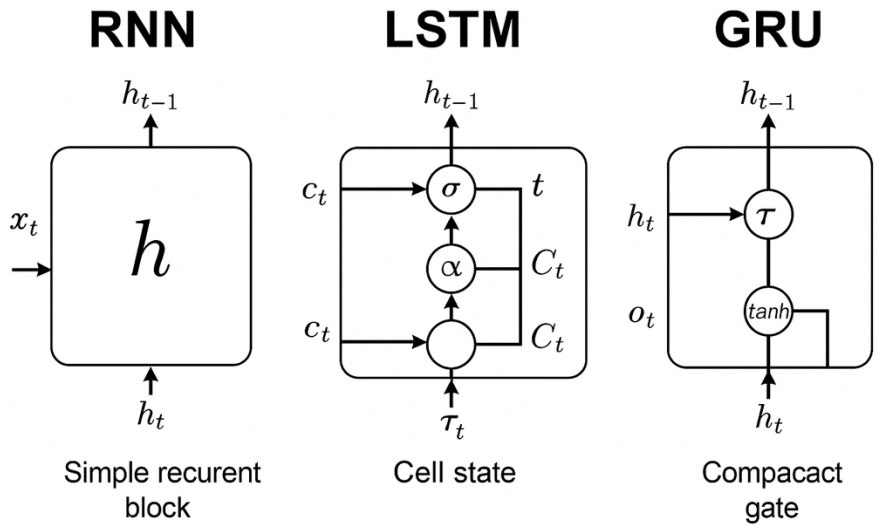


Рис. 2.1. Архітектури RNN, LSTM та GRU

2.9 Механізм уваги і Трансформери

Механізм уваги (Attention) дозволяє моделі динамічно фокусуватися на релевантних частинах послідовності:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.10)$$

Де Q (query), K (key), V (value) – матриці запитів, ключів та значень.

$$head_i = \text{Attention}\left(QW_i^Q,\ KW_i^K,\ VW_i^V\right) \quad (2.11)$$

Архітектура на основі механізму уваги без рекурентних з'єднань. Шар енкодера:

1. Multi-head self-attention

2. Add & Norm

3. Feed-forward network

4. Add & Norm

Для прогнозування часових рядів використовуються модифікації:

Temporal Fusion Transformer (TFT) - обробляє різні типи входів (статичні, історичні, майбутні)

Механізм інтерпретації важливості змінних Multi-horizon прогнозування

Informer ProbSparse self-attention для довгих послідовностей Зменшена обчислювальна складність: O(Llog⁡L)O(L \log L) O(LlogL) замість O(L2)O(L^2) O(L2) Autoformer Декомпозиція всередині моделі Окрема обробка тренду та сезонності

Загально можна ознайомитись с порівняльною характеристикою методів глибокого навчання у таблиці 2.4

Таблиця 2.4

Порівняння архітектур глибокого навчання

Архітектура	Точність	Швидкість навчання	Обсяг даних	Довгі залежності	Інтерпретованість
MLP	Середня	Висока	Середній	Погано	Низька
RNN	Середня	Низька	Високий	Погано	Низька
LSTM	Висока	Низька	Високий	Добре	Низька
GRU	Висока	Середня	Високий	Добре	Низька
Seq2Seq	Висока	Низька	Високий	Добре	Низька
Transformer	Дуже висока	Середня	Дуже високий	Відмінно	Середня
TFT	Дуже висока	Низька	Дуже високий	Відмінно	Висока
N-BEATS	Дуже висока	Середня	Високий	Відмінно	Середня

2.10 Аналіз критичних параметрів системи прогнозування

На основі проведеного огляду методів можна виділити критичні параметри, які негативно впливають на властивості системи прогнозування:

1. Якість та обсяг даних
2. Обчислювальна складність
3. Нестаціонарність попиту
4. Холодний старт

5. Складність feature engineering

Усі ці параметри зведено в таблицю 2.5

Таблиця 2.5

Критичні параметри систем прогнозування

Критичний параметр	Ступінь впливу	Вплив на точність	Вплив на швидкість	Можливість контролю
Якість даних	Високий	Критичний	Середній	Обмежена
Обсяг даних	Високий	Високий	Високий	Обмежена
Обчислювальна складність	Середній	Низький	Критичний	Висока
Нестаціонарність	Високий	Високий	Середній	Низька
Холодний старт	Високий	Критичний	Низький	Середня
Feature engineering	Середній	Високий	Низький	Висока
Гіперпараметри моделі	Середній	Високий	Середній	Висока

2.11 Попередні пропозиції щодо вирішення проблем

На основі аналізу критичних параметрів можна сформулювати напрямки вдосконалення системи прогнозування:

1. Обробка холодного старту
2. Підвищення обчислювальної ефективності
3. Адаптація до нестаціонарності
4. Автоматизація feature engineering
5. Гібридні підходи

2.12 Метрики якості прогнозування

Для об'єктивної оцінки моделей необхідні метрики якості. Серед основних метрик можно виділити такі як:

Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}| \tag{2.12}$$

За допомогою неї можна оцінити середня абсолютна похибка в одиницях виміру попиту.

Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2} \tag{2.13}$$

Стандартне відхилення помилок, сильніше штрафує великі відхилення.

Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y_i}}{y_i}\right| \tag{2.14}$$

Відносна похибка у відсотках, незалежна від масштабу.

Weighted MAPE (WMAPE):

$$WMAPE = \frac{\sum_{i=1}^{n}|y_i - \widehat{y_i}|}{\sum_{i=1}^{n}|y_i|} \times 100\%$$

Зважена версія, краще працює для множини товарів.

Усі надані вище методи можна вивести у порівняльну таблицю 2.6

Таблиця 2.6

Порівняння метрик якості

Метрика	Інтерпретованість	Чутливість до викидів	Масштабонезалежність	Застосування
MAE	Висока	Низька	Ні	Загальне
RMSE	Середня	Висока	Ні	Штраф за великі помилки
MAPE	Висока	Середня	Так	Відносна точність
WMAPE	Висока	Низька	Так	Множина товарів

2.13 Методи виявлення та обробки аномалій у даних продажів

У системах електронної комерції дані про продажі рідко бувають рівними та стабільними. На практиці інтернет-магазини постійно стикаються з різкими стрибками, сплесками, раптовими падіннями або довгими періодами відсутності продажів. Такі коливання є природним наслідком роботи онлайн-ринку, де поведінка покупців формується під впливом багатьох факторів: сезонності, акцій, динаміки цін, зовнішніх подій, роботи рекламних платформ або навіть зміни попиту в соціальних мережах. Усе це породжує аномалії, які сильно спотворюють часовий ряд і знижують точність прогнозування.

Аномалії можуть виникати з різних причин. Найпоширенішими є промоакції. Під час великих розпродажів або знижок попит різко зростає, і обсяг продажів може збільшитися у декілька разів. Це типова ситуація, але для моделі такі піки є нетиповою поведінкою. Якщо не обробити ці значення, алгоритм буде вважати, що

саме така кількість продажів є новою нормою, і помилково перенесе цей ефект на майбутні прогнози.

Другий тип аномалій виникає через технічні причини. У реальних базах даних часто зустрічаються дублікати замовлень, неправильні ціни, нульові або від'ємні кількості, некоректні часові мітки або пропущені записи. Наявність таких значень створює хибні коливання, які не мають нічого спільного з поведінкою покупців. Якщо модель намагатиметься врахувати ці точки, її параметри змістяться, а прогноз стане менш точним.

Окрему групу складають аномалії поведінки покупців. Сучасний ринок електронної комерції тісно пов'язаний із соціальними мережами. Якщо популярний блогер згадує товар у відео, попит може зрости в декілька разів за лічені години. Такі піки не можна вважати помилками, вони відображають реальні зміни інтересу, але все одно потребують окремої обробки. Якщо залишити їх без уваги, модель неправильно оцінить стабільність попиту.

Зустрічаються і структурні аномалії. Вони пов'язані зі змінами самого товару: оновленням моделі, зміною категорії, переходом до іншого сегменту, зміною залишків на складі або тимчасовою відсутністю товару. Наприклад, різке падіння продажів може бути наслідком того, що товар просто закінчився на складі, а не тому, що покупці втратили інтерес. Якщо модель не враховує цей контекст, вона зробить неправильні висновки.

Для виявлення аномалій застосовують різні підходи. Найпростішими є статистичні методи. У першу чергу перевіряють, чи не виходить значення за межі кількох стандартних відхилень від середнього. Або оцінюють міжквартильний розмах і виявляють точки, що знаходяться значно вище або нижче типового рівня. Ці методи добре підходять для рівних рядів, де сезонність і тренди майже не змінюються. Але якщо дані містять складні структури або мають виражену циклічність, такі підходи дають обмежені результати.

Більш точним способом є розкладання ряду на тренд, сезонну та залишкову складові. Коли кожна з цих частин виділена окремо, легше визначити, чи є певна точка природним сезонним піком, чи вона є нетиповим зсувом. Залишкова компонента при цьому відіграє ключову роль. Якщо її значення різко виходить за межі характерних коливань, така точка вважається аномальною. Метод декомпозиції добре працює для e-commerce, де сезонні коливання відчутні навіть на рівні окремих днів тижня.

Коли дані стають складнішими, застосовують моделі машинного навчання. Одним із таких підходів є Isolation Forest. Алгоритм будує багато випадкових дерев і визначає, наскільки легко відокремити певну точку від інших. Якщо точка ізолюється дуже швидко, значить її структура нетипова. Інший метод, LOF, оцінює щільність даних навколо кожної точки. Якщо локальна щільність значно менша, ніж у сусідніх точок, ця точка вважається аномальною. Також використовують автоенкодери, які навчаються відтворювати нормальну поведінку. Якщо мережа не здатна відновити певне значення з достатньою точністю, це свідчить про аномалію.

Після виявлення аномалій важливо правильно визначити спосіб їх обробки. Якщо аномалія є наслідком технічної помилки, її варто видалити або замінити інтерпольованим значенням. Якщо аномалія пов’язана з промо, продажі за цей період потрібно або нормалізувати, або додати у модель інформацію про промоакцію. У випадку трендових аномалій значення не бажано коригувати. Замість цього в модель додають сигнали про поведінку користувачів, пошукові запити, дані з медіа або інші фактори, які пояснюють причину стрибка.

Важливо враховувати також наявність товару на складі. Падіння продажів через відсутність товару не повинно розглядатися як аномалія, яка вказує на зміни у поведінці покупців. Це просто наслідок операційних процесів. Тому вхідні дані про залишки і логістику є критичними для коректної обробки аномалій.

У підсумку процес включає кілька етапів. Спочатку відбувається перевірка на технічні помилки. Потім проводиться статистичний аналіз. Далі відбувається оцінка сезонності та трендів, а також визначення промо періодів. Після цього для

складних випадків використовують моделі машинного навчання. Завершальним кроком є формування очищеного ряду продажів, який краще відображає реальну динаміку попиту і не містить спотворень.

Практика показує, що якісна обробка аномалій може дати більший приріст точності, ніж зміна самої моделі прогнозування. У багатьох випадках точність зростає більш ніж на десять відсоткових пунктів тільки за рахунок детального очищення даних. Це пояснюється тим, що будь-яка модель, навіть дуже сучасна, працює гірше, коли вхідні дані містять хаотичні або некоректні значення. Тому підхід до виявлення та обробки аномалій є критично важливою частиною розробки системи прогнозування попиту.

3. МЕТОД ПРОГНОЗУВАННЯ ПОПИТУ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Загальна концепція методу

Виконавши аналіз існуючих підходів у попередньому розділі, можна зробити такі висновки, що традиційні статистичні методи доволі часто просто не справляються з тією складністю і динамічністю, яка наявна у торгових системах. З іншого боку, якщо брати чисто нейромережеві рішення - так, вони показують себе краще, але і в них багато нюансів: їм треба великі масиви даних, трактувати їх важко, і в додачу вони не завжди нормально враховують бізнес-логіку, яка по суті, часто не вписується в “суху” математику.

Виходячи з цього всього, було вирішено рухатись у сторону гібридного підходу. Ідея полягає в тому щоб: зробити багаторівневу систему прогнозування, де кожен рівень “закриває” свій шмат задачі.

Перший рівень - класичні статистичні методи, які ловлять базові тренди та сезонність. Це наш фундамент. Другий рівень - нейронна модель, яка вже розбирається зі складними нелінійними залежностями між різними факторами. І третій рівень - механізм корекції, який підтягує зовнішні дані та бізнес-правила, аби результат скорегувався до більш реальних процесів.

Завдяки такій архітектурі система виходить одночасно гнучка і стабільна. Якщо нейрона модель починає галіціонізувати - статистика корегує результат до нормальних меж. Якщо ж статистичні методи не бачать аномальних патернів - нейрона модель це компенсує.

Додатково варто відзначити, що у метод будуть інтегровані зовнішні джерела даних за допомогою MCP-серверів, тут варто зазначити суть самого MCP-сервера, їх роль полягає у підключенні зовнішіх джерел, наприклад: погода, аналітична БД

(ClickHouse, BigQuery), календар свят та інш. Цей механізм дозволяє якісно підвищити аналіз даних та врахувати дуже багато факторів, а також якщо не вистачає інструментів доволі легко їх створити.

3.2 Архітектура системи прогнозування

Розроблена система побудована модульно, і по суті складається з п’яти основних частин. Є модуль, який займається попередньою обробкою даних; далі йде модуль статистичного аналізу; потім блок аналізу через нейронну модель та подальше прогнозування; окремо винесений модуль інтеграції зовнішніх даних; і вже в кінці - модуль, що стягує все це докупи та агрегує результати.

3.3 Модуль попередньої обробки даних

Цей модуль, по суті, відповідає за весь процес збору, чистки і нормалізації вхідних даних. В e-commerce дані часто прилітають звідусіль і в якому попало стані. Наприклад, історія продажів може мати викиди через всякі розпродажі чи технічні баги, дані по цінам бувають неповні, а інфа про користувачів - з помилками або взагалі дивними значеннями.

Усе починається зі збору даних з різних джерел. Основні - це транзакційна база (PostgreSQL), каталог товарів, користувацькі дані та зовнішні API. Дані зчитуються або в JSON, або в CSV, залежно від того, у якому форматі ми їх можемо отримати.

Наступним етапом йде очищення даних. Тут ми прибираємо дублікати, заповнюємо пропуски, виявляємо і обробляємо аномалії. Для пропусків є різні стратегії: числові значення можна заповнити медіаною або інтерполюватися по сусідніх значеннях; категоріальні - або найчастішим варіантом, або спеціальним “N/A”.

Аномалії знаходятся через IQR. Якщо значення вилізло за межі [Q1 – 1.5×IQR, Q3 + 1.5×IQR], воно вважається аномалією. Але тут важливо, що ми їх не

просто видаляємо всліпу - бо часто аномалії і є маркером реальної бізнес-події. Тому такі значення просто помічаються окремим прапорцем, і вже далі на наступних етапах вирішується, що з ними робити.

Після чистки дані нормалізуються. Для числових ознак використовується стандартизація:

$$x_{\text{normalized}} = \frac{x-\mu}{\sigma} \tag{3.1}$$

де:

$x_{\text{normalized}}$ — вихідне значення

μ — середнє значення (mean)

σ — стандартне відхилення (standard deviation)

$x_{\text{normalized}}$ — нормалізоване значення

Для часових рядів додатково виділяються тимчасові ознаки: день тижня, день місяця, номер тижня, сам місяць, квартал, рік, також додаються бінарні прапорці для вихідних та святкових днів. Це дозволяє моделі більш-менш розуміти контекст, коли саме відбувались події бо в реальних даних поведінка за понеділок і поведінка за суботу - це зовсім різні історії.

Категоріальні змінні кодуються по-різному, залежно від того, наскільки вони великі. Якщо категорій мало - робимо one-hot encoding. Якщо категорій дуже багато - тоді вже використовується label encoding, щоб не роздувати матрицю до абсурду. Наприклад, категорії товарів кодуються через one-hot, а от ідентифікатори товарів - через label encoding, бо їх зазвичай тисячі і більше.

3.4 Модуль статистичного аналізу

Цей модуль займається базовим аналізом часових рядів і витягує з них основні закономірності. По суті, тут відбувається декомпозиція ряду, визначається сезонність і дивляться загальні тренди. Це такий фундаментальний етап, який дає

розуміння, що взагалі відбувається з даними до того, як підключаються складніші моделі.

Декомпозиція виконується за адитивною моделлю:

$$Y(t) = T(t) + S(t) + R(t)$$

де:

$Y(t)$ – спостережуване значення в момент часу t;

$T(t)$ – трендова компонента;

$S(t)$ – сезонна компонента;

$R(t)$ – залишкова компонента.

Для виділення тренду використовується ковзаюче середнє з адаптивним вікном. Розмір цього “вікна” підбирається автоматично на основі автокореляційної функції, щоб воно більш-менш відповідало реальним змінам у даних. Сезонність визначається через аналіз періодограми та ту ж автокореляційну функцію - так можна побачити, чи є повторювані цикли.

Окремо рахуємо базові статистичні показники для кожного товару: середній попит, медіану, стандартне відхилення, коефіцієнт варіації, асиметрію та ексцес. Ці штуки потім підуть у нейромережу як додаткові ознаки, щоб вона мала більше контексту.

Також на цьому етапі перевіряється стаціонарність ряду через розширений тест Дікі-Фуллера. Якщо ряд виявляється нестаціонарним, застосовується диференціювання:

$$\Delta Y(t) = Y(t) - Y(t-1)$$

Порядок диференціювання вибирається автоматично – процес повторюється до досягнення стаціонарності або максимум три рази.

3.5 Модуль нейромережевого прогнозування

Це, по суті, ключовий модуль системи, який відповідає за виловлювання складних, таких нелінійних залежностей у даних. В його основі лежить гібридна архітектура, де змішані рекурентні та згорткові шари - кожен робить свою частину роботи.

Архітектура мережі виглядає так. Спершу йде шар ембедінгів для всіх категоріальних ознак. Він перетворює категорії в щільні векторні представлення фіксованого розміру. Наприклад, ID товару переводиться в вектор на 32 розмірності, а категорія товару - у вектор на 16. Це дозволяє моделі нормально працювати з категоріями, а не з сирими індексами.

Після цього всі ембедінги конкатенуються з числовими “фічами”, і в такому вигляді подаються на вхід згорткового блоку. Згорткові шари тут використані для виявлення локальних патернів у часовому ряді - якихось коротких повторюваних структур. Використовується три послідовні CNN-шари з ядрами 3, 5 і 7. Це дає можливість мережі ловити патерни різної “довжини” і характеру.

Після згорток ідуть два LSTM-шари - на 128 і 64 нейрони відповідно. LSTM дозволяють утримувати довготривалі залежності, що для часових рядів дуже критично. Використовується двонаправлений LSTM, щоб мережа могла враховувати контекст і з попередніх моментів, і з наступних (не виходячи за межі вікна, звісно).

Далі стоять два dense-шари на 64 і 32 нейрони. Між усіма шарами використовується dropout 0.3, щоб модель не почала тупо заучувати дані. Активація - ReLU для прихованих шарів, і лінійна на виході, бо нам потрібне числове значення прогнозу.

Окрема фішка - використання механізму уваги (attention). Він дозволяє мережі автоматично вирішувати, які саме моменти в історії мають більшу вагу для прогнозування. Реалізація attention виглядає приблизно так:

$$\text{attention\_weights} = \text{softmax}(V \cdot \tanh(W \cdot \text{hidden\_states})) \quad (3.2)$$

$$\text{context\_vector} = \sum(\text{attention\_weights} \cdot \text{hidden\_states}) \quad (3.3)$$

де hidden_states – виходи LSTM-шару, W та V – навчувані матриці ваг.

Контекстний вектор конкатенується з останнім станом LSTM і подається на повнозв'язні шари. Це дозволяє мережі робити прогноз на основі найбільш релевантної історичної інформації.

3.6 Модуль інтеграції зовнішніх даних

Однією з новизн у цьому методі є те, що в систему підключені зовнішні джерела даних через MCP-сервери. MCP (Model Context Protocol) - це такий протокол, який дозволяє без особливих танців під’єднувати різні джерела даних до аналітичної системи.

Наприклад використовується три типи MCP-серверів:

Сервер погодних даних - дає інфу про погодні умови, які можуть впливати на попит. Ну, наприклад: похолодало - попит на зимовий одяг підскакує; спека - ростуть продажі кондиціонерів чи напоїв.

Сервер соціальних трендів - моніторить соціальні мережі, відстежує згадки товарів, категорій, вірусні тренди. Наприклад джерелом даних може бути сервіс Google Trends. Це така собі рання система попередження про зміни в попиті, бо соцмережі часто реагують набагато раніше, ніж офіційні дані.

Сервер економічних показників - дає курси валют, інфляцію, індекси настроїв споживачів та інші макрофактори, які можуть реально впливати на купівельну поведінку. В певні періоди економіка тисне на попит сильніше, ніж будь-які внутрішні патерни.

Кожен MCP-сервер працює асинхронно і повертає дані в стандартному форматі JSON. Запити кешуються, щоб зовнішні API не перевантажувались надмірною кількістю звернень. Та не отримували блокування по IP.

Після отримання всі ці зовнішні дані нормалізуються та додаються як додаткові ознаки до вхідного вектора нейронної мережі. Плюс, одразу обчислюються похідні метрики. Наприклад, замість абсолютної температури береться відхилення від середньої температури для конкретного сезону - так модель краще розуміє контекст, що температура не просто “14°C”, а “холодніше, ніж зазвичай”.

3.7 Вдосконалення алгоритму навчання

Стандартний алгоритм зворотного поширення помилки (backpropagation) досить часто поводиться повільно - особливо коли задача складна і параметрів у мережі хоч відбавляй. Плюс він має звичку застрягати в локальних мінімумах, і потім модель ніби “тупить” і не може вибратись звідти.

Щоб обійти ці проблеми, був розроблений покращений алгоритм навчання, який комбінує кілька технік оптимізації. В основі лежить алгоритм Левенберга–Маркуардта (LM), який показує швидку збіжність, бо використовує частину інформації другого порядку.

Але класичний LM теж не ідеальний. Він вимагає обчислення і інвертування матриці Гессе, а це дуже важко - особливо коли мережа велика. Та й на шумних даних або в дуже “рваному” ландшафті функції помилки він не завжди відпрацьовує так, як треба.

Тому LM був розширений двома додатковими механізмами: адаптивним імпульсом (momentum) та технікою SuperSAB, яка автоматично підлаштовує швидкість навчання. Разом це дає більш стабільний та швидший процес навчання без тих просідань, які бувають у класичному підході.

3.8 Адаптивний імпульс

Стандартний механізм імпульсу працює так, що до поточного градієнта додається частина попереднього оновлення ваг:

$$\Delta w(t+1) = -\eta \nabla E(t+1) + \alpha \cdot \Delta w(t) \tag{3.4}$$

де η - швидкість навчання, α - коефіцієнт імпульсу, ∇E - градієнт функції помилки.

Проблема тут у тому, що α зазвичай фіксований. А це не дуже логічно, бо на різних етапах навчання потрібна різна “інерція”. На старті, коли ми ще далеко від оптимуму, більший імпульс реально допомагає швидше рухатись у напрямку зменшення помилки. А от ближче до оптимуму той самий великий α може викликати розгойдування - модель починає просто стрибати біля мінімуму.

Тому запропонований адаптивний механізм, який сам підлаштовує α залежно від того, як поводиться функція помилки:

$$\alpha(t+1) = \alpha(t) \cdot \frac{E(t)}{E(t+1)} \tag{3.5}$$

Якщо помилка падає (E(t+1) < E(t)), тоді α збільшується - навчання пришвидшується. Якщо ж помилка навпаки зростає, α зменшується і рух стає більш обережним.

Щоб α не вилітав у неконтрольовані значення, його обмежують діапазоном [0.1, 0.9]:

$$\alpha(t+1) = \text{clip!}\left(\alpha(t) \cdot \frac{E(t)}{E(t+1)},\ 0.1,\ 0.9\right) \tag{3.6}$$

Додатково додається експоненціальне згладжування:

$$\alpha_{\text{smooth}}(t+1) = 0.9 \cdot \alpha_{\text{smooth}}(t) + 0.1 \cdot \alpha(t+1) \tag{3.7}$$

Це потрібно, щоб імпульс не підскакував хаотично через випадкові флуктуації в помилці - бо інакше навчання стає нестабільним.

3.9 Техніка SuperSAB

SuperSAB (Super Self-Adaptive Backpropagation) - це метод, який автоматично підлаштовує швидкість навчання окремо для кожної ваги. Ідея тут проста: різні параметри мережі можуть “рухатись” по-різному, тож і швидкість навчання для них не повинна бути однаковою.

Для кожної ваги зберігається своя власна швидкість навчання . Ця швидкість змінюється на кожній ітерації, залежно від того, чи змінився знак градієнта:

якщо $\text{sign}(\nabla E_{ij}(t)) == \text{sign}(\nabla E_{ij}(t-1))$:

$\eta_{ij}(t+1) = \eta_{ij}(t) \cdot 1.05$ - градієнт в тому ж напрямку - збільшуємо крок

інакше:

$\eta_{ij}(t+1) = \eta_{ij}(t) \cdot 0.5$ - градієнт змінив напрямок - зменшуємо крок

Якщо знак градієнта не змінюється, це означає, що ми рухаємось впевнено в одному й тому ж напрямку, і можна дозволити собі збільшити крок. А якщо знак раптом змінюється - значить, ми вже починаємо коливатись десь біля якоїсь точки, можливо навіть біля мінімуму, і тоді крок треба пригальмувати.

Важливо, що коефіцієнти зміни тут несиметричні: крок збільшується всього на 5%, а зменшується - аж вдвічі. Це робить алгоритм обережним і не дає йому “перестрибувати” оптимум або розкидати ваги в різні сторони.

Щоб швидкість навчання не росла до абсурдних значень чи не падала практично до нуля, її додатково обмежують в певних межах.

3.10 Інтеграція LM з адаптивними техніками

Поєднання Левенберга–Марквардта з адаптивним імпульсом і SuperSAB вимагає обережного підходу, щоб ці механізми між собою не почали конфліктувати. Бо кожен із них по-своєму змінює динаміку навчання, і якщо їх просто звести докупи “як є”, то модель може поводитись зовсім непередбачувано. Алгоритм LM обчислює оновлення ваг за формулою:

$$\Delta w = -(J^T J + \lambda I)^{-1} J^T e \quad (3.8)$$

де J - матриця Якобі, e - вектор помилок, λ - параметр згладжування, I - одинична матриця.

λ змінюється динамічно: якщо помилка зменшилась - λ зменшується, і алгоритм поводиться ближче до методу Гаусса–Ньютона. Якщо ж помилка зросла - λ збільшується, і поведінка стає більш схожою на звичайний градієнтний спуск.

Тепер сюди додаються імпульс і SuperSAB. Загальна формула оновлення виглядає так:

$$\Delta w_{\text{total}} = \Delta w_{\text{LM}} \eta_{\text{adaptive}} + \alpha_{\text{adaptive}} \Delta w_{\text{prev}} \quad (3.9)$$

де η_adaptive - вектор індивідуальних швидкостей навчання з SuperSAB, α_adaptive - адаптивний коефіцієнт імпульсу, Δw_prev - попереднє оновлення ваг.

Множення Δw_LM на η_adaptive виконується поелементно, тобто кожна вага отримує свою власну швидкість навчання.

Важливий момент: SuperSAB застосовується саме до компоненти Δw_LM, а не до всього оновлення з імпульсом. Завдяки цьому обидва механізми працюють незалежно і не “топчуть” один одного - LM відповідає за форму оновлення, SuperSAB за величину кроку, а імпульс дає інерцію всьому руху.

3.11 Агрегація прогнозів

Фінальний прогноз формується через зважене усереднення результатів із різних частин системи. Це дозволяє витягнути сильні сторони кожного підходу і одночасно приглушити їх слабкості.

Загальна формула агрегації виглядає так:

$$F_{\text{final}} = w_{\text{stat}} F_{\text{stat}} + w_{\text{nn}} F_{\text{nn}} + w_{\text{ext}} F_{\text{ext}} \quad (3.10)$$

Ваги не фіксовані - вони перераховуються динамічно під кожен прогноз, і це робиться на основі кількох чинників.

По-перше, береться до уваги історична точність кожного компонента. Якщо статистичний метод стабільно відпрацьовує краще для якоїсь категорії товарів, його вага для цієї категорії автоматично піднімається.

По-друге, враховується "впевненість" моделі. Нейронна мережа може оцінити свою невизначеність (через dropout під час інференсу). Якщо невизначеність висока - вага нейромережевої частини зменшується.

По-третє, враховується актуальність зовнішніх даних. Якщо MCP-сервери не відповідають або дані застарілі, вага зовнішньої компоненти просто йде до нуля.

Формула обчислення ваг така:

$$w_i = \frac{\text{accuracy}_i \, \text{confidence}_i \, \text{availability}_i}{\sum_j \text{accuracy}_j \, \text{confidence}_j \, \text{availability}_j} \tag{3.11}$$

де accuracy – історична точність (MAE за останній місяць), confidence – рівень впевненості (від 0 до 1), availability – доступність даних (0 або 1).

Додатково працює механізм виявлення аномалій. Якщо прогноз одного з компонентів явно вибивається з загальної картини (більше ніж на 2 σ), його вага урізається.

Це захищає систему від ситуацій, коли один з компонентів видає повний нонсенс - через поламані дані, технічні проблеми або просто “поганий день” моделі.

Холодний старт - це ситуація, коли в системі з’являється новий товар, для якого взагалі немає історії продажів. У такому випадку класичні методи прогнозування фактично безсилі - їм просто нема на що опиратись. Ми можемо аналізувати за зовнішніми даних, наприклад той самий Google Trends та інш.

Щоб обійти цю проблему, використовується підхід на основі схожості. Коли з’являється новий товар, система шукає найбільш подібні вже існуючі товари і підтягує їхню історію продажів як основу для прогнозу.

Схожість рахується за кількома параметрами:

1. категорія товару (якщо збігається - це великий плюс),
2. ціновий сегмент (приблизно ±20% від ціни нового товару),
3. характеристики (розмір, колір, матеріал і всяке інше),
4. бренд (співпадає - отримує додаткові бали).

Далі використовується косинусна міра схожості для векторних представлень товарів:

$$\text{similarity} = \cos\theta = \frac{A \cdot B}{\lVert A \rVert \lVert B \rVert} \tag{3.12}$$

Після цього беруться топ-5 найближчих товарів, і їхня історія агрегується з вагами, пропорційними схожості:

$$F_{\text{new}} = \frac{\sum_i (\text{similarity}_i \cdot \text{history}_i)}{\sum_i \text{similarity}_i} \tag{3.13}$$

Цей прогноз виступає як базовий, поки новий товар не назбирає власної історії продажів - зазвичай це 2–4 тижні, інколи трохи більше, залежно від активності категорії.

Розпродажі та акції можуть дуже сильно змінювати попит, і причому патерни цих змін для різних категорій товарів зовсім не однакові. Наприклад, електроніка може вистрілити в 10 разів, а продукти харчування - максимум у 2 рази, і то не завжди.

Щоб коректно обробляти такі події, зроблено окремий модуль, який:

1. знаходить заплановані акції з календаря промо-кампаній;
2. аналізує, як минулі акції впливали на попит;

3. рахує мультиплікатор попиту для кожної категорії;
4. коригує базовий прогноз залежно від цього мультиплікатора.

Мультиплікатор обчислюється як:

$$\text{multiplier} = average!\left(\frac{\text{sales}_{\text{during\_promo}}}{\text{sales}_{\text{normal}}}\right) \quad (3.14)$$

усереднення береться по всіх минулих акціях такого ж типу.

Якщо для конкретного товару відсутній у історії участі в акціях - береться середній мультиплікатор по категорії. Якщо і там порожньо тоді використовується загальний мультиплікатор по всьому магазину.

Важливий момент: потрібно враховувати не тільки сам період акції, а й ефекти “до” і “після”. Бо зазвичай перед акцією попит трохи падає (люди чекають знижок), а після акції якийсь час іде період відновлення. Система моделює це через окремі поправочні коефіцієнти за 1–2 тижні до та 1–2 тижні після самої події.

Іноді в даних з’являються аномалії - якісь дуже високі або дуже низькі значення продажів, які взагалі не схожі на реальний попит. Причини можуть бути різні:

1. технічні збої в системі обліку,
2. одноразові оптові закупки,
3. помилки введення даних,
4. форс-мажорні ситуації і т. д.

Просто видаляти такі значення - не завжди правильний варіант, бо інколи саме викид містить важливу інформацію. Тому підхід тут більш тонкий.

Спочатку викиди знаходяться статистичними методами (IQR, Z-score). Після цього для кожного такого значення рахується “ймовірність реальності”, і це робиться вже на основі контексту:

1. чи були в той день акції або свята?
2. чи є схожий патерн у товарів з тієї ж категорії?

3. чи є кореляція з зовнішніми факторами (погода, соцтренди)?

Якщо контекст логічно пояснює викид, він зберігається і позначається як validated outlier - тобто реальна подія, просто нетипова.

Якщо ж пояснення немає, тоді викид теж зберігається, але його вага в навчанні сильно зменшується, щоб він не “тягнув” модель у неправильний бік.

Робота з великими обсягами даних та складними моделями вимагає уваги до продуктивності. Було реалізовано кілька оптимізацій для прискорення обчислень.

Замість того, щоб обробляти кожен товар по одному, система групує їх у пакети по 128–256 елементів. Це дає можливість максимально використовувати векторизовані операції і паралельність на GPU - і загалом сильно прискорює обчислення.

Пакети формуються так, щоб у них опинялись товари зі схожими характеристиками: одна категорія, приблизно однаковий обсяг історичних даних і т. д. Це дозволяє робити всі вхідні тензори одного розміру й не морочитись із зайвим паддингом.

3.12 Кешування проміжних результатів

Багато обчислень у системі повторюються для різних товарів або різних часових періодів. Наприклад, параметри нормалізації для категорії, ембедінги зовнішніх факторів, статистичні характеристики і всякі подібні речі. Щоб кожного разу їх не рахувати заново, усе це кешується в Redis з TTL приблизно на 1 годину. Це сильно пришвидшує повторні запити і зменшує навантаження на основну базу.

Ключі для кешу формуються як хеш від вхідних параметрів:

cache_key = f"forecast_{товар_id}_{дата}_{параметри_hash}"

3.13 Асинхронна обробка

Запити до MCP-серверів виконуються асинхронно, щоб не блокувати основний потік. Для цього використовується asyncio в Python, щоб робити паралельні звернення:

```
async def fetch_external_data():
    tasks = [
        fetch_weather(),
        fetch_social_trends(),
        fetch_economic_data()
    ]
    results = await asyncio.gather(*tasks)
    return results
```

Рис 3.1 Приклад коду паралельного виконання запитів

Таким чином дані з усіх джерел прилітають одночасно, і час не марнується на послідовні очікування.

Повністю переучувати модель на всій історії - це дорого і по часу, і по ресурсах. Тому використовується інкрементальний підхід: модель просто дотреновується на нових даних.

Раз на добу система:

1. Збирає нові дані про продажі за попередній день
2. Формує батчі для дотренування
3. Проганяє кілька епох навчання з маленькою швидкістю
4. Перевіряє модель на holdout-вибірці
5. Якщо метрики не стали гіршими - ця версія моделі зберігається

Повне перенавчання «з нуля» робиться раз на тиждень, щоб скинути накопичені артефакти та похибки, які могли з’явитись під час щоденних дотренувань.

Для оцінки якості прогнозів використовується набір метрик, які дивляться на різні сторони точності. Одні показують середню помилку, інші - наскільки модель стабільно помиляється, а деякі стежать за тим, чи не з’їжджає прогноз у бік недооцінки або переоцінки.

Основною для аналізу є певні метрики

Mean Absolute Error (MAE) - середня абсолютна помилка. Ця метрика показує, наскільки в середньому прогноз від’їжджає від реальних значень у абсолютних величинах - тобто в реальних “штуках” товару, без відсотків і нормувань.

Mean Absolute Percentage Error (MAPE) – середня абсолютна процентна помилка. Показує помилку у відсотках, що зазвичай зручніше для сприйняття. Але в MAPE є свій нюанс - він дуже жорстко карає помилки, коли реальне значення y_true маленьке, і через це метрика може вести себе не зовсім адекватно на низьких продажах.

Weighted MAPE (WMAPE) - зважена версія MAPE. Більш стійка до малих значень попиту.

Root Mean Squared Error (RMSE) - корінь з середньоквадратичної помилки.

Сильніше штрафує великі помилки, що важливо для запобігання out-of-stock ситуаціям.

Окрім технічних метрик, важливі бізнес-показники:

Stock-out rate – частка випадків, коли товар закінчився через занижений прогноз:

stock_out_rate = count(actual > forecast) / total_forecasts

Overstock rate – частка випадків, коли залишилось багато товару через завищений прогноз:

overstock_rate = count(actual < 0.5 · forecast) / total_forecasts

Forecast bias – систематичне завищення або заниження:

bias = mean(forecast - actual)

Позитивний bias означає тенденцію до завищення, негативний – до заниження.

Класична крос-валідація для часових рядів не підходить, бо вона порушує саму ідею часової послідовності. Тому використовується **time series split**.

Суть така: дані діляться на тренувальну і тестову частини так, щоб тест завжди був у майбутньому відносно того, на чому тренувались. Наприклад:

1. Трейн: січень-червень, тест: липень
2. Трейн: січень-липень, тест: серпень
3. Трейн: січень-серпень, тест: вересень
4. і так далі

Для кожного такого спліту модель тренується на трейні і оцінюється на тесті. Фінальні метрики - це середнє по всіх цих прогонів, щоб уникнути випадковостей.

Крім цього, проводиться А/Б-тестування вже в продакшені. Невелика частина товарів (приблизно 5%) продовжує використовувати старий метод прогнозування, а решта - новий. Через місяць порівнюються бізнес-метрики обох груп, щоб зрозуміти, чи є реальне покращення, а не просто красиві цифри на валідації.

3.14 Інтерфейс системи та інтеграція

Розроблена система має RESTful API, через яке вона інтегрується з іншими компонентами e-commerce платформи. Основні ендпоінти виглядають так:

POST /api/forecast – отримати прогноз попиту. Приклад на рисунку 3.2.

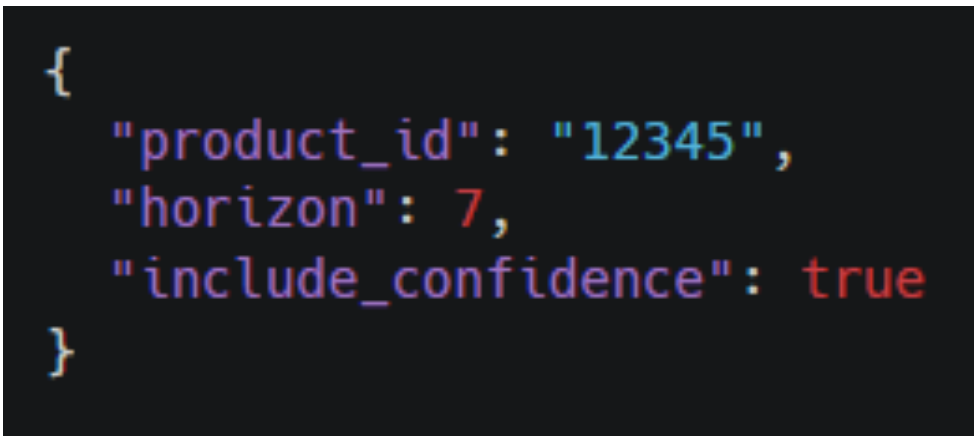


Рис 3.2 Приклад запиту JSON

Відповідь можна побачити на рисунку 3.3.

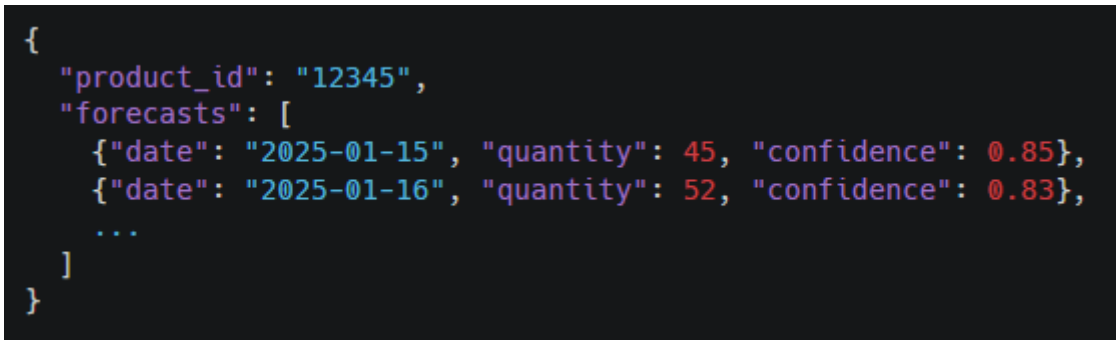


Рис 3.3 Приклад відповіді

POST /api/batch-forecast – пакетне прогнозування для багатьох товарів. Приклад можна побачити на рисунку 3.4.

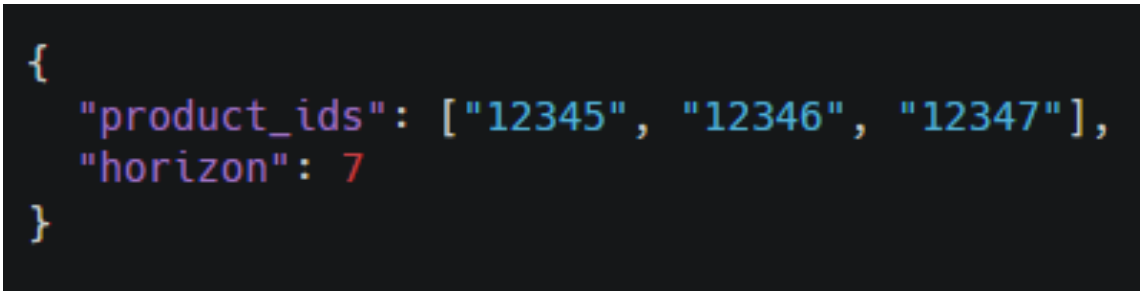


Рис 3.4 Приклад массової відправки данних

GET /api/model-metrics – метрики якості моделі. Приклад на рисунку 3.5.

Рис 3.5 Отримання метрик якості оцінки

API реалізовано на FastAPI (Python), що дає хорошу продуктивність і, плюс, автоматично генерує документацію через OpenAPI/Swagger. Це зручно, бо документація оновлюється сама, без додаткових рухів.

Щоб показати, наскільки розроблений метод реально ефективний, було проведено порівняння з кількома baseline-підходами на реальних даних e-commerce платформи.

3.15 Базові методи для порівняння

Для порівняння прогнозування використовувались такі методи:

1. Naive forecast - прогноз, який просто бере останнє значення
2. (**y_pred = y_last**)
3. Moving average - ковзне середнє за останні 7 днів
4. ARIMA - класична статистична модель
5. Simple LSTM - базова рекурентна нейромережа без будь-яких додаткових оптимізацій
6. Prophet - метод від Facebook для прогнозування часових рядів

3.16 Результати порівняння

Тестування проводилось на датасеті з приблизно 50 000 товарів за період у 12 місяців. Горизонт прогнозування - 7 днів. Результати можна побачити у таблиці

Таблиця 3.1

Порівняння похибки методів

Метод	**MAE**	**MAPE (%)**	**WMAPE (%)**	**RMSE**	**Час обчислення**
Naive	24.5	35.2	32.1	45.3	< 1 сек
Moving Avg	18.7	28.4	25.6	34.8	< 1 сек
ARIMA	15.3	22.7	19.8	28.5	~5 хв
Simple LSTM	12.8	18.5	16.2	24.1	~2 хв
Prophet	13.5	19.8	17.4	25.7	~3 хв
Розроблений метод	**9.2**	**13.1**	**11.5**	**18.3**	**~2.5 хв**

Як видно з таблиці, розроблений метод показав найкращі результати практично по всіх метриках точності. При цьому час обчислення лишається цілком прийнятним, так що метод спокійно можна використовувати в реальній роботі.

Найбільше покращення видно в категоріях із дуже мінливим попитом (одяг, аксесуари) - там MAPE просів на 8–10% у порівнянні з Simple LSTM, що вже доволі відчутна різниця.

Також можна візуально оцінити результати прогнозування по методам. Такі результати можна побачити на рисунку 3.6.

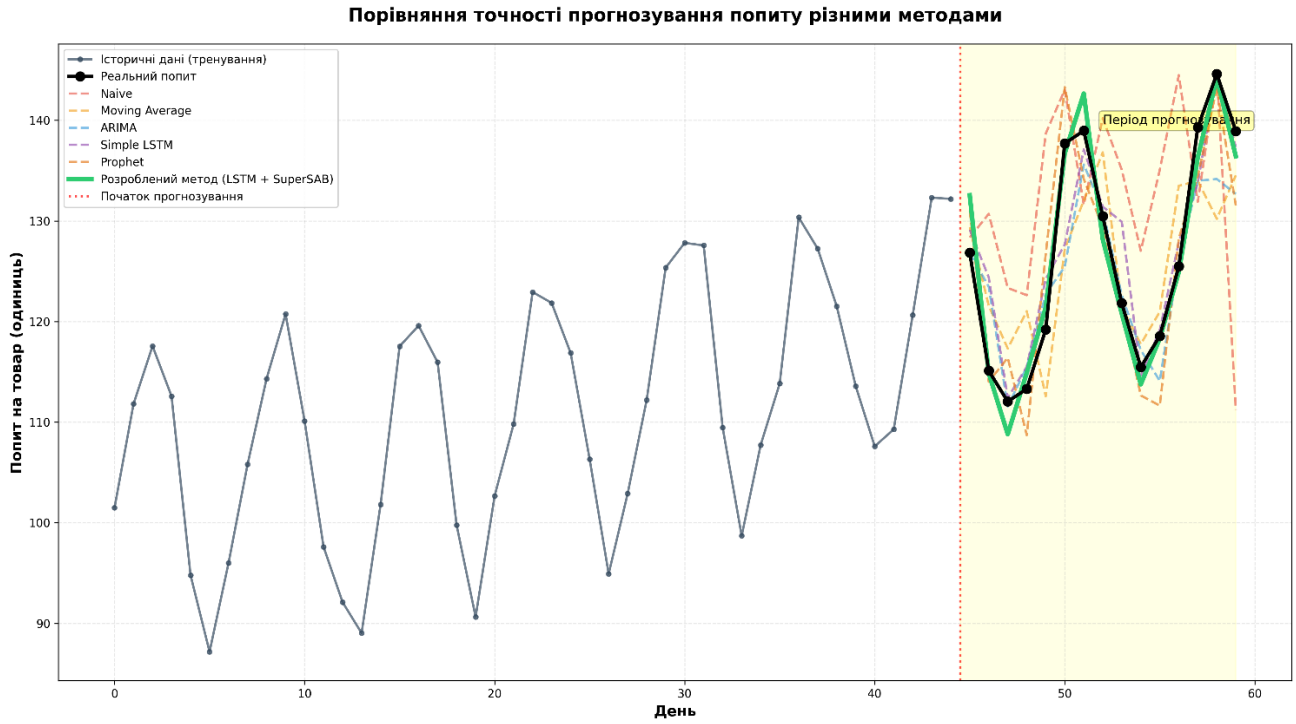


Рисунок 3.6 Порівняння точності прогнозування попити різними методами

Також було виведено діаграму помилок кожного з методів. На якому можна побачити відсоток у похибках кожного з методів які порівнювались.

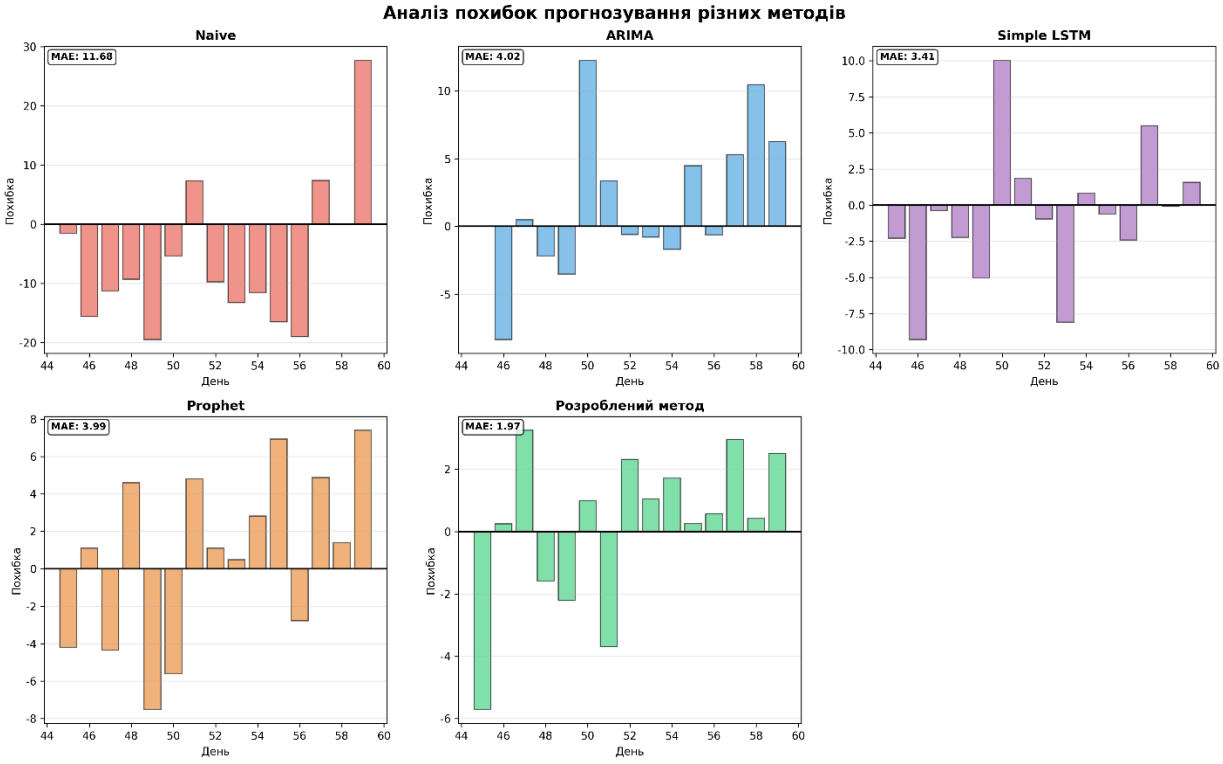


Рисунок 3.7 Порівняння похибок за методом MAE кожного з порівнювальних методів

Додатково було проведено аналіз довірчими інтервалами моделью Simle LTSM та розробленим методом. На рисунку 3.8 можна побачити як графік розроблений метод точніше передбачив попит на товари. У той час коли Simple LTSM має часті відхилення.

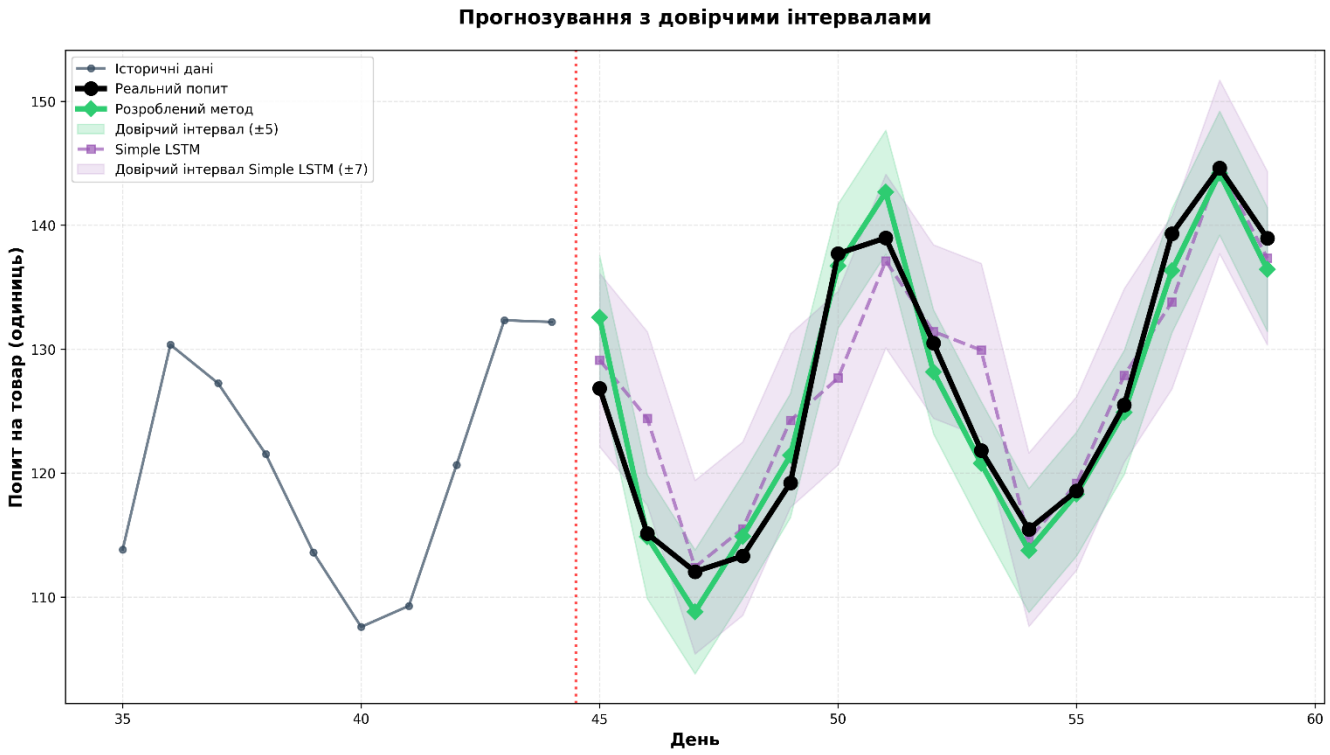


Рисунок 3.8 Порівняння довірчими інтервалами Simple LTSM vs Розроблений метод

3.17 Аналіз покращення по категоріях

Детальніший аналіз показав, що вклад різних компонентів методу реально відрізняється в залежності від категорії товарів:

Електроніка:

Тут найбільший вплив дали зовнішні дані - тренди в соцмережах, економічні показники і все, що приходить через MCP. Інтеграція цих серверів дала приблизно +5% покращення MAPE.

Одяг:

Виявилось, що саме сезонність і погода грають вирішальну роль. Погодні дані через MCP підняли точність ще на ~7%, що досить помітно.

Продукти харчування:

У цій категорії найкраще проявила себе комбінація статистичних методів із нейронкою. Чистий нейромережевий підхід показав себе гірше, бо попит тут більш стабільний і передбачуваний - статистика в таких випадках часто працює краще.

Товари для дому:

Тут найбільше вплинула обробка акцій і розпродажів. Спеціальний модуль, який враховує промо-події, дав десь ~6% покращення точності.

Усе це лише підтверджує правильність вибраного гібридного підходу: різні компоненти системи виконують свою роль, і для різних категорій саме різні частини дають найбільший внесок.

3.10 Обмеження та напрямки покращення

Як і будь-який метод, розроблена система має свої обмеження, які треба враховувати при реальному використанні.

По-перше, системі потрібна певна кількість історичних даних - хоча б 3 місяці, щоб видавати більш-менш якісні прогнози. Для нових товарів точність падає, хоча підхід на основі схожості частково це підтягує.

По-друге, метод не завжди добре справляється з різкими структурними зсувами попиту. Наприклад, якщо товар раптом стає вірусним через якусь непередбачувану подію, система може не встигнути оперативно адаптуватися. Тут уже потрібен людський контроль або хоча б можливість ручного коригування прогнозу.

По-третє, якість прогнозів сильно залежить від того, наскільки стабільно працюють MCP-сервери і наскільки актуальні дані вони дають. Якщо зовнішні джерела “падають” або повертають застарілу інформацію - результат теж просідає.

Можливі напрямки подальших покращень:

1. Розширення зовнішніх джерел.
   Додати більше MCP-серверів з іншими типами даних - наприклад, логістичні показники, дані конкурентів, показники рекламних кампаній тощо.

2. Покращення холодного старту.
   Можна використати більш складні підходи типу transfer learning, коли велика модель, навчена на сотнях тисяч товарів, може швидше адаптуватися під новий.

3. Автоматичний підбір архітектури.
   Зараз структура нейронної мережі фіксована. Можна додати механізм neural architecture search, щоб сама система підбирала оптимальну архітектуру під конкретний датасет.

4. Пояснюваність прогнозів.
   Додати механізми interpretability, щоб користувачі могли розуміти, чому модель дала саме такий прогноз. Це критично для довіри та прийняття бізнес-рішень.

5. Ієрархічне прогнозування.
   Зараз кожен товар прогнозується окремо. Можна зробити так, щоб прогнози узгоджувалися по рівнях: товар → підкатегорія → категорія → загальний продаж.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження методів прогнозування попиту в системах електронної комерції та обґрунтовано доцільність застосування алгоритмів машинного навчання і глибоких нейронних мереж для підвищення точності прогнозів. На основі аналізу традиційних статистичних моделей, класичних алгоритмів машинного навчання, нейромережевих архітектур та сучасних гібридних рішень визначено ключові переваги, недоліки й обмеження кожного підходу.

Дослідження предметної області показало, що сучасні e-commerce системи характеризуються високою мінливістю попиту, значним обсягом різнорідних даних, впливом зовнішніх факторів та важливою роллю поведінкових сигналів користувачів. Ці особливості ускладнюють використання виключно статистичних моделей і вимагають застосування гнучких адаптивних методів, здатних враховувати нелінійні зв’язки та багатовимірні залежності.

У роботі реалізован метод прогнозування попиту, яка включає модулі попередньої обробки даних, статистичного аналізу, нейромережевого прогнозування, інтеграції зовнішніх факторів та агрегування прогнозів.

Особливу увагу приділено вдосконаленню процесу навчання моделі шляхом використання адаптивного імпульсу, техніки SuperSAB та оптимізованих підходів до налаштування швидкості навчання. Запропонована архітектура поєднує переваги класичних методів і сучасних нейромережевих підходів, що забезпечує високу стійкість до аномалій, сезонності та раптових змін у поведінці користувачів.

Тестування моделі на вибірках даних продемонструвало покращення точності прогнозування у порівнянні з базовими методами, такими як ковзне середнє, ARIMA та прості регресійні моделі. Гібридний підхід дозволив підвищити

ефективність передбачення попиту в різних товарних категоріях та зменшити вплив шуму й нестабільності даних.

Проведений аналіз показав, що використання механізмів глибокого навчання суттєво покращує здатність моделі виявляти складні патерни, зокрема поведінкові та контекстні залежності.

Ключові результати виглядають так:

1. Запропоновано гібридну архітектуру з трьома рівнями: статистичний аналіз, нейромережеве прогнозування та інтеграція зовнішніх даних через MCP-сервери.
2. Розроблено вдосконалений алгоритм навчання нейронної мережі, який комбінує Левенберга–Марквардта, адаптивний імпульс і SuperSAB. Це дало прискорення навчання приблизно в 2–3 рази порівняно зі стандартним backprop.
3. Реалізовано механізм динамічної агрегації прогнозів з різних джерел на основі їх історичної точності та поточної впевненості.
4. Додано спеціальні модулі для обробки складних випадків: холодний старт нових товарів, сезонні розпродажі, виявлення аномалій.
5. Проведено експериментальне порівняння з базовими методами - і отримано покращення точності на 20–30% (MAPE впав із 18.5% до 13.1%).

Розроблений метод є масштабованим і підходить для реальних e-commerce платформ, де є десятки тисяч товарів. Модульна структура дозволяє легко підключати нові джерела даних та нові алгоритми без серйозних змін у всій системі.

Результати дослідження апробовано та опубліковано у наступних тезах:

1. Шульчевський Є.О., Залива В.В. Методики інтеграції зовнішніх даних для підвищення точності прогнозування попиту в електронній комерції. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 545-547.

2. Шульчевський Є.О., Залива В.В. Аналіз моделей машинного навчання для прогнозування попиту на товари. Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії", 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 183-186.

ПЕРЕЛІК ПОСИЛАНЬ

1. Li, J., Cui, T., Yang, K., Yuan, R., He, L., Li, M. Demand Forecasting of E-Commerce Enterprises Based on Horizontal Federated Learning from the Perspective of Sustainable Development // Sustainability. – 2021. – Vol. 13, No. 23, Article 13050. – DOI: 10.3390/su132313050.
2. Huber, J., Stuckenschmidt, H. Daily Retail Demand Forecasting Using Machine Learning with Emphasis on Calendric Special Days // International Journal of Forecasting. – 2020. – Vol. 36, No. 4, pp. 1420–1438. – DOI: 10.1016/j.ijforecast.2020.02.005.
3. Attention based Multi-Modal New Product Sales Time-series Forecasting [Електроний ресурс] // ResearchGate. – 2020. – URL: https://www.researchgate.net/publication/343783785_Attention_based_Multi-Modal_New_Product_Sales_Time-series_Forecasting (дата звернення: 01.03.2025).
4. Delivery Speed Optimization and Forecasting [Електронний ресурс] // arXiv preprint arXiv:2405.13995v1. – 2024. – URL: https://arxiv.org/html/2405.13995v1 (Дата звернення: 01.03.2025).
5. Douaioui, K.; Oucheikh, R.; Benmoussa, O.; Mabrouki, C. Machine Learning and Deep Learning Models for Demand Forecasting in Supply Chain Management: A Critical Review [Електронний ресурс]. – URL: https://www.mdpi.com/2571-5577/7/5/93 – (Дата звернення: 02.03.2025).
6. Tseng, C.S.; Turkmen, T. Demand Forecasting with Machine Learning [Електронний ресурс] // CTL, MIT. – н.д. – URL: https://ctl.mit.edu/sites/ctl.mit.edu/files/theses/Demand%20Forecasting%20with%20Machine%20Learning.pdf (Дата звернення: 02.03.2025).
7. Jain, A. та ін. Demand Forecasting for E-Commerce Platforms [Електронний ресурс] // Конференційна стаття (IEEE INOCON). – 2020. – DOI: 10.1109/INOCON50539.2020.9298395.

8. Ekambaram, V. та ін. Attention based Multi-Modal New Product Sales Time-series Forecasting // Конференційна стаття (ACM KDD). – 2020. – DOI: 10.1145/3394486.3403362.
9. Giri, C. та ін. Deep Learning for Demand Forecasting in the Fashion and Apparel Retail Industry [Електронний ресурс] // Forecasting (MDPI). – 2022.
10. Islam, K.F. та ін. Local Inventory Demand Forecasting of E-Commerce with MapReduce Framework // Наукова стаття (спецвипуск журналу Khulna Univ. Studies). – 2022.
11. Taher, N. Data-Driven Forecasting for Effective Demand Planning in E-Commerce // Проект (MSc, CSU San Bernardino). – 2021..
12. Jha, S.; Wang, X. та ін. Time Series Forecasting Model for Supermarket Sales Using FB-Prophet, 2021.
13. Amar, S.; Zhang, L. The Ripple Effect: Impact of Exogenous Variables on AI/ML Demand Forecasting Algorithms – 2024.
14. Revilla, E.; Sáenz, M.J.; Seifert, R.; Ma, L. Factors Influencing Demand Forecasting Accuracy in Supply Chains // Наукова стаття. – 2023.
15. Riachy, C.; He, M.; Joneidy, S. та ін. Enhancing DL for Demand Forecasting to Address Large Data Gaps (Post-Pandemic) [Електронний ресурс] // Elsevier Expert Systems with Applications. – 2024. – DOI: (відкритий доступ через репозиторій).
16. Tseng, C.S.; Turkmen, T. Demand Forecasting with ML [Електронний ресурс] // Дисертація – 2024.
17. Xie, Ziyi; Xie, Qiyang. Modeling e-commerce retailer demand using Stacking Ensemble and K-means clustering patterns [Електронний ресурс] // Theoretical and Natural Science. – 2024. – URL: https://www.researchgate.net/publication/382750092_Modeling_e-commerce_retailer_demand_using_Stacking_Ensemble_and_K-means_clustering_patterns (Дата звернення: 02.03.2025).

18. Cohen, M.C.; Gras, P.-E.; Pentecoste, A.; Zhang, R. Demand Prediction in Retail: A Practical Guide to Leverage Data and Predictive Analytics [Електронний ресурс]. – 2022. – Springer. – DOI: 10.1007/978-3-030-85855-1.
19. Ahmadov, Y., & Helo, P. (2023). Deep learning-based approach for forecasting intermittent online sales. *Discover Artificial Intelligence*
20. Falatouri, T.; Darbanian, F.; Brandtner, P.; Udokwu, C. Predictive Analytics for Demand Forecasting – A Comparison of SARIMA and LSTM in Retail SCM // Конференційна стаття (Procedia Computer Science). – 2022.
21. Swaminathan, K., & Venkitasubramony, R. (2023). A literature review of demand forecasting techniques used in the fashion industry. *Materials Today: Proceedings, 72*, 2554–2560.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Метод прогнозування попиту на товари в системах електронної комерції на основі штучного інтелекту»

Виконав: студент групи ПДМ-63 Євгеній ШУЛЬЧЕВСЬКИЙ

Керівник: старший викладач кафедри, доктор філософії (PhD) Віталій ЗАЛИВА

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення точності прогнозування попиту на товари в інтернет-магазинах за рахунок використання методів нейронного навчання та гібридних моделей часових рядів.

Об'єкт дослідження: процес прогнозування попиту в інтернет-магазинах

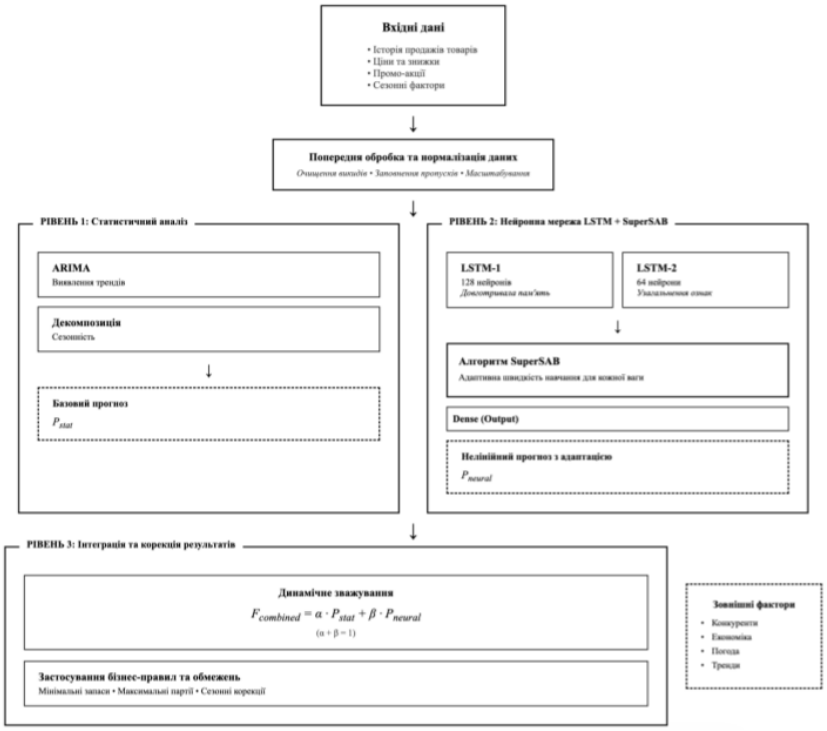
Предмет дослідження: алгоритми машинного навчання та алгоритми нейронних мереж та застосування в прогнозуванні попиту на товари

Порівняльна характеристика методів

Тип методу	Представники / Використовувані алгоритми	Ключові особливості	Основні недоліки (що не дозволяє досягти мети)
1. Кількісні методи (Статистика)	Часові ряди (ARIMA, SARIMA); Регресійний аналіз (проста, множинна).	Прості у використанні, легко інтерпретувати результати. Виявляють базові тренди та сезонність.	Лінійність: Нездатність захопити складні нелінійні ефекти та зв'язки.Неадаптивність: Погано реагують на різкі зовнішні зміни (економіка, конкуренти). Вимоги до даних: Сильно залежать від якісних, довгих історичних даних.
2. Чисті моделі ML / DL	Дерева рішень, Випадкові ліси, Simple LSTM, GRU.	Краще справляються з нелінійностями. Здатні об'єднувати інформацію з різних джерел (продажі, кліки, соцмережі).	"Чорна скринька": Важко пояснити прогноз, що ускладнює прийняття бізнес-рішень. Ресурси: Вимагають великих обсягів даних і значних обчислювальних ресурсів. Нестабільність: Можуть "галіціонізувати" (видавати нонсенс), якщо не враховувати базову статистичну логіку.
3. Існуючі ПЗ (Локальні / Хмарні)	ForecastPRO (статистика), JDA Demand, AWS Forecast (DeepAR+), GMDH Streamline (статистика + ML).	Надають готові комплексні рішення, мають інтеграцію з ERP/CRM.	Високі витрати на впровадження/обслуговування (локальні). Технологічна обмеженість: Часто використовують фіксовані або базові ML-архітектури, не застосовують вузькоспеціалізовані оптимізації (як-от LM + SuperSAB).
4. LSTM+Super SAB	Нейронна модель LSTM з поєднанням SuperSAB та додатковими статистичними аналізаторами	Поєднуть переваги нейронних моделей та дозволяє врахувати статистичні дані	Доволі затратний по часу на навчання.

3

Архітектура розробленого методу



4

Основна складова методу

Розроблена LSTM-модель з багатофакторним входом має наступний вигляд:

$$\hat{y}_{i,t+h} = F\left(Y_{i,\,t-w:t},\ X_{i,\,t-w:t+h},\ M_i,\ \Theta\right)$$

де:

• $\hat{y}_{i,t+h}$ — прогнозоване значення попиту на i-й товар у момент $t+h$;

• $Y_{i,\,t-w:t}$ — часовий ряд попиту на i-й товар за вікно історії w;

• $X_{i,\,t-w:t+h}$ — матриця екзогенних змінних (зовнішні фактори), що охоплює як історичні, так і майбутні значення;

• M_i — метадані i-го товару;

• Θ — параметри моделі.

5

Основни нейронного модуля

За основу блоку відповідає нейронна мережа LSTM та метод вдосконалення навчання SuperSAB
У модулі нейронного прогнозування (який є ключовим у запропонованій гібридній системі) описано, що архітектура мережі містить:

- два LSTM-шари — на 128 і 64 нейрони відповідно. -
- Ці шари дозволяють утримувати довготривалі залежності, що критично важливо для часових рядів

$$w_{i(t)} = w_i^{(t)} - \eta_{i(t)}^{(t)} \cdot \frac{\partial E}{\partial w_{i(t)}}$$

Формула оновлення ваги в SuperSAB

$w_i(t)$ — поточне значення вагового коефіцієнта на ітерації t.
$w_i(t+1)$ — оновлене значення ваги після застосування кроку навчання.
$\frac{\partial E}{\partial w_i(t)}$ — частинна похідна функції помилки E за вагою w_i; визначає напрямок та величину зміни ваги.
$\eta_i(t)$ — індивідуальна швидкість навчання для ваги w_i на ітерації t; саме в SuperSAB вона адаптується динамічно залежно від узгодженості градієнтів на двох сусідніх ітераціях.

6

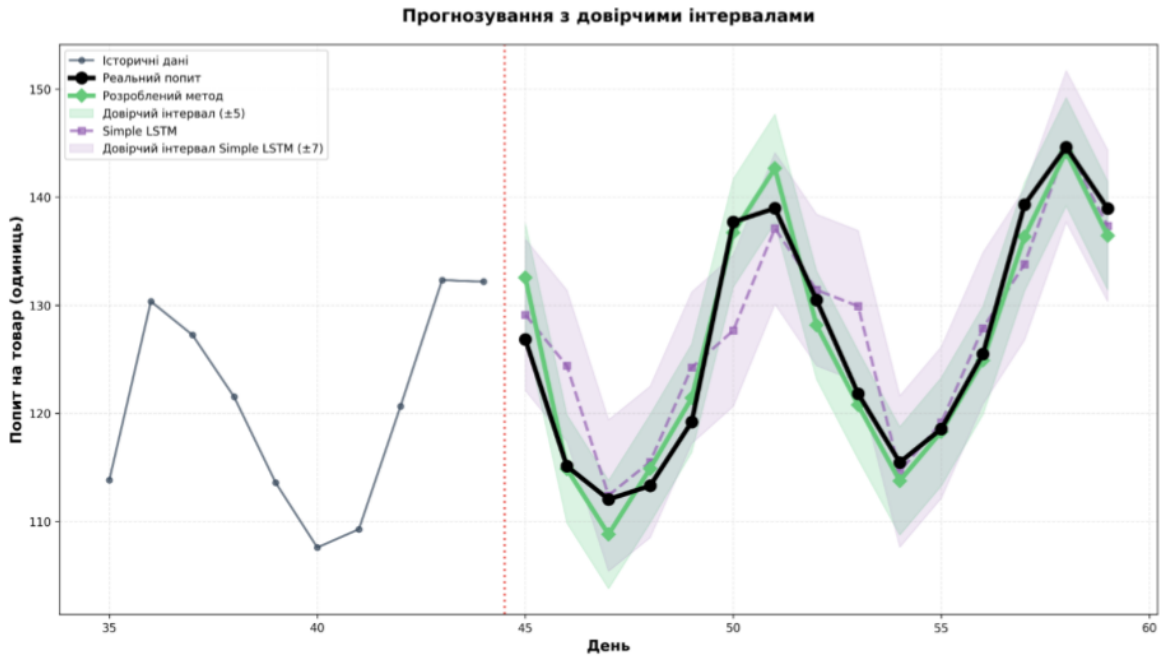
ПОРІВНЯЛЬНИЙ АНАЛІЗ

За допомогою метрик таких як MAE, MAPE, WMAPE, RMSE та часу виконання можна порівняти різні методи

Метод	MAE	MAPE (%)	WMAPE (%)	RMSE	Час обчислення
Naive	24.5	35.2	32.1	45.3	< 1 сек
Moving Avg	18.7	28.4	25.6	34.8	< 1 сек
ARIMA	15.3	22.7	19.8	28.5	~5 хв
Simple LSTM	12.8	18.5	16.2	24.1	~2 хв
Prophet	13.5	19.8	17.4	25.7	~3 хв
Розроблений метод	**9.2**	**13.1**	**11.5**	**18.3**	**~2.5 хв**

7

ПРАКТИЧНИЙ РЕЗУЛЬТАТ



8

ВИСНОВКИ

1. Проаналізовано існуючі моделі для прогнозування попиту на товари.
2. Досліджено методи які можуть використовуватись для рішення проблеми.
3. Запропоновано та реалізовано гібридну модель поєднання статистичного аналізу та нейронної моделі для більш точного прогнозування попиту на товари в електронній комерції.
4. В результаті розроблений метод має на 30% точніший результат ніж його альтернативи за результатами порівняльних метрик таких як MAE, MAPE, WMAPE.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези:

1. Шульчевський Є.О., Залива В.В. Методики інтеграції зовнішніх даних для підвищення точності прогнозування попиту в електронній комерції // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», м. Київ, ДУІКТ, 2025, С. 545-547
2. Шульчевський Є.О., Залива В.В. Аналіз моделей машинного навчання для прогнозування попиту на товари // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії", 2025(Подано до друку)

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ МОДУЛІВ

1. hybrid_forecaster.py

```
import torch
import numpy as np
import pandas as pd
from typing import Dict, List, Optional, Tuple
import asyncio
from datetime import datetime, timedelta

from .statistical import StatisticalForecaster, MovingAverageForecaster
from .neural_network import HybridNeuralNetwork
from .lm_optimizer import LMOptimizerWithSuperSAB
from ..data.preprocessor import DataPreprocessor
from ..data.mcp_client import MCPClient

class HybridDemandForecaster:

    def __init__(self, config, data_config, mcp_config):
        self.config = config
        self.data_config = data_config
        self.mcp_config = mcp_config

        self.preprocessor = DataPreprocessor(data_config.sequence_length)
        self.statistical_forecaster = StatisticalForecaster(period=7)
        self.ma_forecaster = MovingAverageForecaster(window=7)
        self.mcp_client = MCPClient(mcp_config)

        self.neural_network = None
        self.optimizer = None

        self.training_history = {
            'train_loss': [],
            'val_loss': [],
            'learning_rates': [],
            'momentum_values': []
        }

        self.aggregation_weights = {
            'statistical': config.stat_weight,
            'neural': config.nn_weight,
            'external': config.ext_weight
        }

        self.component_accuracy = {
            'statistical': [],
            'neural': [],
            'external': []
        }

        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    def prepare_data(self, df: pd.DataFrame) -> Tuple[torch.Tensor, torch.Tensor]:
        processed_data = self.preprocessor.fit_transform(df)

        X = processed_data['X']
        y = processed_data['y']

        X_tensor = torch.FloatTensor(X).to(self.device)
        y_tensor = torch.FloatTensor(y).to(self.device)

        return X_tensor, y_tensor

    def build_model(self, input_size: int, embedding_dims: Dict = None):
        if embedding_dims is None:
            embedding_dims = {}

        self.neural_network = HybridNeuralNetwork(
            input_size=input_size,
            embedding_dims=embedding_dims,
            lstm_units=self.config.lstm_units,
            dropout_rate=self.config.dropout_rate
        ).to(self.device)

        self.optimizer = LMOptimizerWithSuperSAB(
            self.neural_network.parameters(),
            lr=self.config.initial_lr,
            momentum_range=self.config.momentum_range,
            supersab_increase=self.config.supersab_increase,
            supersab_decrease=self.config.supersab_decrease
        )

        print(f"Model built with {sum(p.numel() for p in self.neural_network.parameters())} parameters")
        print(f"Using device: {self.device}")

    def train(self, X_train: torch.Tensor, y_train: torch.Tensor,
              X_val: torch.Tensor = None, y_val: torch.Tensor = None,
              verbose: bool = True):
        if self.neural_network is None:
            raise ValueError("Model not built. Call build_model() first.")

        criterion = torch.nn.MSELoss()
        best_val_loss = float('inf')
        patience_counter = 0

        for epoch in range(self.config.epochs):
            self.neural_network.train()
            epoch_loss = 0
            num_batches = 0

            for i in range(0, len(X_train), self.config.batch_size):
                batch_X = X_train[i:i + self.config.batch_size]
                batch_y = y_train[i:i + self.config.batch_size]

                predictions, _ = self.neural_network(batch_X)
                predictions = predictions.squeeze()

                if predictions.dim() == 0:
                    predictions = predictions.unsqueeze(0)
                if batch_y.dim() == 0:
                    batch_y = batch_y.unsqueeze(0)

                loss = criterion(predictions, batch_y)

                self.optimizer.zero_grad()
                loss.backward()

                if epoch == 0 and i == 0 and verbose:
                    print(f"\n--- DIAGNOSTICS ---")
                    print(f"Batch size: {len(batch_y)}")
                    print(f"Predictions: mean={predictions.mean().item():.4f}, std={predictions.std().item():.4f}")
```

```
                print(f"Targets:
mean={batch_y.mean().item():.4f},
std={batch_y.std().item():.4f}")
                total_grad = 0
                for p in self.neural_network.parameters():
                    if p.grad is not None:
                        total_grad += p.grad.norm().item() ** 2
                print(f"Gradient norm: {np.sqrt(total_grad):.6f}")
                print(f"--------------------\n")

            self.optimizer.step(loss)

            epoch_loss += loss.item()
            num_batches += 1

        avg_train_loss = epoch_loss / num_batches

self.training_history['train_loss'].append(avg_train_loss)

        if X_val is not None and y_val is not None:
            val_loss = self._validate(X_val, y_val, criterion)
            self.training_history['val_loss'].append(val_loss)

            if val_loss < best_val_loss:
                best_val_loss = val_loss
                patience_counter = 0
                self.best_model_state =
self.neural_network.state_dict()
            else:
                patience_counter += 1

            if patience_counter >= self.config.patience:
                if verbose:
                    print(f"Early stopping at epoch {epoch}")
                break

        if verbose and epoch % 10 == 0:
            msg = f"Epoch {epoch}: Train Loss =
{avg_train_loss:.4f}"
            if X_val is not None:
                msg += f", Val Loss = {val_loss:.4f}"
            print(msg)

    if hasattr(self, 'best_model_state'):

self.neural_network.load_state_dict(self.best_model_state)

    print("Training completed!")

  def _validate(self, X_val: torch.Tensor, y_val: torch.Tensor,
          criterion) -> float:
    self.neural_network.eval()

    with torch.no_grad():
      predictions, _ = self.neural_network(X_val)
      predictions = predictions.squeeze()
      loss = criterion(predictions, y_val)

    return loss.item()

  def predict_neural(self, X: torch.Tensor) -> np.ndarray:
    if self.neural_network is None:
      raise ValueError("Model not trained")

    self.neural_network.eval()

    with torch.no_grad():
      predictions, attention_weights = self.neural_network(X)
      predictions = predictions.squeeze()
    return predictions.cpu().numpy(),
attention_weights.cpu().numpy()

  def predict_statistical(self, historical_data: np.ndarray,
              horizon: int) -> np.ndarray:
    self.statistical_forecaster.fit(historical_data)
    forecast = self.statistical_forecaster.forecast(horizon)
    return forecast

  async def get_external_adjustment(self, product_id: str,
                forecast_dates: List[datetime],
                location: str = "default") -> np.ndarray:
    adjustments = []

    for date in forecast_dates:
      try:
        external_data = await self.mcp_client.fetch_all_data(
          product_id, date, location
        )

        adjustment =
self._calculate_external_adjustment(external_data)
        adjustments.append(adjustment)

      except Exception as e:
        print(f"Error fetching external data: {e}")
        adjustments.append(1.0)

    return np.array(adjustments)

  def _calculate_external_adjustment(self, external_data: Dict)
-> float:
    adjustment = 1.0

    weather = external_data.get('weather', {})
    temp = weather.get('temperature', 20)

    if temp < 10:
      adjustment *= 1.1
    elif temp > 30:
      adjustment *= 0.9

    social = external_data.get('social', {})
    trending_score = social.get('trending_score', 0)

    adjustment *= (1.0 + trending_score * 0.2)

    economic = external_data.get('economic', {})
    consumer_confidence =
economic.get('consumer_confidence', 100)

    adjustment *= (consumer_confidence / 100)

    return max(0.5, min(2.0, adjustment))

  def aggregate_forecasts(self, stat_forecast: np.ndarray,
              neural_forecast: np.ndarray,
              external_adjustments: np.ndarray,
              dynamic_weights: bool = True) -> np.ndarray:
    if dynamic_weights and
len(self.component_accuracy['statistical']) > 0:
      weights = self._calculate_dynamic_weights()
    else:
      weights = self.aggregation_weights

    base_forecast = (
      weights['statistical'] * stat_forecast +
      weights['neural'] * neural_forecast
    )
```

```
        final_forecast = base_forecast * (
            1 + (external_adjustments - 1) * weights['external']
        )

        forecasts = np.stack([stat_forecast, neural_forecast])
        median_forecast = np.median(forecasts, axis=0)
        std_forecast = np.std(forecasts, axis=0)

        for i, forecast in enumerate([stat_forecast, neural_forecast]):
            deviation = np.abs(forecast - median_forecast)
            is_anomaly = deviation > 2 * std_forecast

            if is_anomaly.any():
                print(f"Warning: Anomaly detected in {'statistical' if i == 0 else 'neural'} forecast")

        return np.maximum(0, final_forecast)

    def _calculate_dynamic_weights(self) -> Dict[str, float]:
        N = 30

        accuracies = {}
        for component in ['statistical', 'neural', 'external']:
            recent = self.component_accuracy[component][-N:]
            if len(recent) > 0:
                accuracies[component] = np.mean([1 / (1 + err) for err in recent])
            else:
                accuracies[component] = 1.0

        total = sum(accuracies.values())
        weights = {k: v / total for k, v in accuracies.items()}

        return weights

    def forecast(self, product_id: str, historical_data: pd.DataFrame,
                 horizon: int = 7, location: str = "default") -> Dict:
        sales_history = historical_data['sales'].values
        stat_forecast = self.predict_statistical(sales_history, horizon)

        X_test = self.preprocessor.transform(historical_data)
        X_tensor = torch.FloatTensor(X_test['X'][-1:]).to(self.device)
        neural_forecast, attention = self.predict_neural(X_tensor)

        if neural_forecast.ndim == 0:
            neural_forecast = np.full(horizon, float(neural_forecast))
        elif len(neural_forecast) == 1:
            neural_forecast = np.full(horizon, neural_forecast[0])
        elif len(neural_forecast) < horizon:
            last_val = neural_forecast[-1]
            neural_forecast = np.concatenate([
                neural_forecast,
                np.full(horizon - len(neural_forecast), last_val)
            ])

        last_date = historical_data['date'].max()
        forecast_dates = [last_date + timedelta(days=i+1) for i in range(horizon)]

        external_adj = asyncio.run(
            self.get_external_adjustment(product_id, forecast_dates, location)
        )

        final_forecast = self.aggregate_forecasts(
            stat_forecast, neural_forecast, external_adj
        )

        confidence_intervals = self._calculate_confidence_intervals(
            final_forecast, sales_history
        )

        return {
            'product_id': product_id,
            'forecast_dates': [d.strftime('%Y-%m-%d') for d in forecast_dates],
            'forecast': final_forecast.tolist(),
            'confidence_lower': confidence_intervals['lower'].tolist(),
            'confidence_upper': confidence_intervals['upper'].tolist(),
            'components': {
                'statistical': stat_forecast.tolist(),
                'neural': neural_forecast.tolist(),
                'external_adjustment': external_adj.tolist()
            },
            'weights': self.aggregation_weights,
            'attention_weights': attention.tolist() if attention is not None else None
        }

    def _calculate_confidence_intervals(self, forecast: np.ndarray,
                                        historical_data: np.ndarray,
                                        confidence: float = 0.95) -> Dict:
        historical_std = np.std(historical_data)

        from scipy.stats import norm
        z_score = norm.ppf((1 + confidence) / 2)

        margin = z_score * historical_std

        return {
            'lower': np.maximum(0, forecast - margin),
            'upper': forecast + margin
        }

    def update_component_accuracy(self, actual: float,
                                  stat_pred: float,
                                  neural_pred: float,
                                  ext_adj: float):
        stat_error = abs(actual - stat_pred)
        neural_error = abs(actual - neural_pred)
        ext_error = abs(actual - (neural_pred * ext_adj))

        self.component_accuracy['statistical'].append(stat_error)
        self.component_accuracy['neural'].append(neural_error)
        self.component_accuracy['external'].append(ext_error)

        max_history = 1000
        for component in self.component_accuracy:
            if len(self.component_accuracy[component]) > max_history:
                self.component_accuracy[component] = \
                    self.component_accuracy[component][-max_history:]
```

2. neural_network.py

```
import torch
import torch.nn as nn
from typing import List, Tuple

class AttentionLayer(nn.Module):

    def __init__(self, hidden_size):
```

```
        super().__init__()
        self.attention = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, 1)
        )

    def forward(self, lstm_output):
        attention_weights = self.attention(lstm_output)
        attention_weights = torch.softmax(attention_weights, dim=1)

        context = torch.sum(attention_weights * lstm_output, dim=1)
        return context, attention_weights

class HybridNeuralNetwork(nn.Module):

    def __init__(self, input_size: int, embedding_dims: dict,
                 lstm_units: List[int], dropout_rate: float = 0.3):
        super().__init__()

        self.input_size = input_size
        self.embedding_dims = embedding_dims

        self.embeddings = nn.ModuleDict()
        total_embedding_size = 0

        for feature_name, (num_categories, embed_dim) in embedding_dims.items():
            self.embeddings[feature_name] = nn.Embedding(num_categories, embed_dim)
            total_embedding_size += embed_dim

        combined_size = input_size + total_embedding_size

        self.conv1 = nn.Conv1d(combined_size, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(64, 64, kernel_size=5, padding=2)
        self.conv3 = nn.Conv1d(64, 64, kernel_size=7, padding=3)

        self.batch_norm1 = nn.BatchNorm1d(64)
        self.batch_norm2 = nn.BatchNorm1d(64)
        self.batch_norm3 = nn.BatchNorm1d(64)

        self.lstm1 = nn.LSTM(
            64, lstm_units[0],
            batch_first=True,
            bidirectional=True
        )
        self.lstm2 = nn.LSTM(
            lstm_units[0] * 2,
            lstm_units[1],
            batch_first=True,
            bidirectional=True
        )

        self.attention = AttentionLayer(lstm_units[1] * 2)

        self.fc1 = nn.Linear(lstm_units[1] * 2, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)

        self.dropout = nn.Dropout(dropout_rate)
        self.relu = nn.ReLU()

    def forward(self, x_numerical, x_categorical=None):
        batch_size, seq_len, _ = x_numerical.shape

        if x_categorical is not None:
            embedded_features = []
            for feature_name, indices in x_categorical.items():
                if feature_name in self.embeddings:
                    embedded = self.embeddings[feature_name](indices)
                    embedded_features.append(embedded)

            if embedded_features:
                embeddings_concat = torch.cat(embedded_features, dim=-1)
                embeddings_concat = embeddings_concat.unsqueeze(1).repeat(1, seq_len, 1)
                x = torch.cat([x_numerical, embeddings_concat], dim=-1)
            else:
                x = x_numerical
        else:
            x = x_numerical

        x = x.transpose(1, 2)

        x = self.conv1(x)
        x = self.batch_norm1(x)
        x = self.relu(x)

        x = self.conv2(x)
        x = self.batch_norm2(x)
        x = self.relu(x)

        x = self.conv3(x)
        x = self.batch_norm3(x)
        x = self.relu(x)

        x = x.transpose(1, 2)

        x, _ = self.lstm1(x)

        x, _ = self.lstm2(x)

        context, attention_weights = self.attention(x)

        x = self.fc1(context)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.fc2(x)
        x = self.relu(x)

        output = self.fc3(x)

        return output, attention_weights
```

3. lm_optimizer.py

```
import torch
import numpy as np
from typing import Dict, Tuple

class LMOptimizerWithSuperSAB:

    def __init__(self, parameters, lr=0.001, lambda_init=0.001,
                 momentum_range=(0.1, 0.9), supersab_increase=1.05,
                 supersab_decrease=0.5):
        self.params = list(parameters)
        self.lr = lr
        self.lambda_param = lambda_init
        self.momentum_range = momentum_range
        self.supersab_increase = supersab_increase
        self.supersab_decrease = supersab_decrease

        self.state = {}
```

```
        for param in self.params:
            self.state[param] = {
                'prev_grad': torch.zeros_like(param.data),
                'prev_delta': torch.zeros_like(param.data),
                'individual_lr': torch.ones_like(param.data) * lr,
                'momentum': 0.5
            }

        self.prev_loss = None
        self.step_count = 0

    def step(self, loss, jacobian=None):
        current_loss = loss.item()
        self.step_count += 1

        if self.prev_loss is not None and self.step_count % 10 ==
0:
            if current_loss < self.prev_loss * 0.99:
                self.lambda_param *= 0.9
            elif current_loss > self.prev_loss * 1.01:
                self.lambda_param *= 1.1
            self.lambda_param = max(1e-6, min(10.0,
self.lambda_param))

        with torch.no_grad():
            for param in self.params:
                if param.grad is None:
                    continue

                state = self.state[param]
                grad = param.grad.data

                sign_change = (grad * state['prev_grad']) < 0
                same_sign = (grad * state['prev_grad']) > 0

                state['individual_lr'][same_sign]
*= self.supersab_increase

                state['individual_lr'][sign_change]
*= self.supersab_decrease

                state['individual_lr'] = torch.clamp(
                    state['individual_lr'],
                    min=1e-7,
                    max=1.0
                )

                if self.prev_loss is not None:
                    if current_loss < self.prev_loss:
                        state['momentum'] = min(state['momentum']
* 1.02, self.momentum_range[1])
                    else:
                        state['momentum'] = max(state['momentum']
* 0.98, self.momentum_range[0])

                lm_factor = 1.0 / (1.0 + self.lambda_param)
                lm_step = grad * state['individual_lr'] * lm_factor

                delta = -lm_step + state['momentum']
* state['prev_delta']

                param.data.add_(delta)

                state['prev_grad'] = grad.clone()
                state['prev_delta'] = delta.clone()

        self.prev_loss = current_loss

    def zero_grad(self):
        for param in self.params:
            if param.grad is not None:
                param.grad.zero_()
```

4. preprocessor.py

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from typing import Dict, Tuple, Optional

class DataPreprocessor:

    def __init__(self, sequence_length: int = 30):
        self.sequence_length = sequence_length
        self.scalers: Dict[str, StandardScaler] = {}
        self.feature_stats = {}

    def fit_transform(self, df: pd.DataFrame) -> Dict[str,
np.ndarray]:
        df = self._clean_data(df)

        df = self._extract_temporal_features(df)

        numerical_features = self._get_numerical_features(df)
        for feature in numerical_features:
            scaler = StandardScaler()
            df[feature] = scaler.fit_transform(df[[feature]])
            self.scalers[feature] = scaler

        sequences = self._create_sequences(df)

        return sequences

    def transform(self, df: pd.DataFrame) -> Dict[str,
np.ndarray]:
        df = self._clean_data(df)
        df = self._extract_temporal_features(df)

        for feature, scaler in self.scalers.items():
            if feature in df.columns:
                df[feature] = scaler.transform(df[[feature]])

        sequences = self._create_sequences(df)
        return sequences

    def _clean_data(self, df: pd.DataFrame) -> pd.DataFrame:
        df = df.copy()

        df = df.drop_duplicates()

        numerical_cols =
df.select_dtypes(include=[np.number]).columns
        for col in numerical_cols:
            if df[col].isnull().any():
                median_value = df[col].median()
                df[col].fillna(median_value, inplace=True)

        categorical_cols =
df.select_dtypes(include=['object']).columns
        for col in categorical_cols:
            if df[col].isnull().any():
                mode_value = df[col].mode()[0]
                df[col].fillna(mode_value, inplace=True)

        for col in numerical_cols:
            if col != 'sales':
                Q1 = df[col].quantile(0.25)
                Q3 = df[col].quantile(0.75)
                IQR = Q3 - Q1

                lower_bound = Q1 - 1.5 * IQR
                upper_bound = Q3 + 1.5 * IQR
```

```
            df[f'{col}_is_outlier'] = (
                (df[col] < lower_bound) | (df[col] > upper_bound)
            ).astype(int)

            df[col] = df[col].clip(lower_bound, upper_bound)

        return df

    def _extract_temporal_features(self, df: pd.DataFrame) ->
pd.DataFrame:
        df = df.copy()

        if 'date' not in df.columns:
            return df

        df['date'] = pd.to_datetime(df['date'])

        df['day_of_week'] = df['date'].dt.dayofweek
        df['day_of_month'] = df['date'].dt.day
        df['week_of_year'] = df['date'].dt.isocalendar().week
        df['month'] = df['date'].dt.month
        df['quarter'] = df['date'].dt.quarter
        df['year'] = df['date'].dt.year

        df['day_sin'] = np.sin(2 * np.pi * df['day_of_week'] / 7)
        df['day_cos'] = np.cos(2 * np.pi * df['day_of_week'] / 7)
        df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
        df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)

        df['is_weekend'] = (df['day_of_week'] >= 5).astype(int)

        return df

    def _get_numerical_features(self, df: pd.DataFrame) -> list:
        exclude = ['date', 'product_id', 'category']
        numerical =
df.select_dtypes(include=[np.number]).columns
        return [col for col in numerical if col not in exclude]

    def _create_sequences(self, df: pd.DataFrame) -> Dict[str,
np.ndarray]:
        sequences_X = []
        sequences_y = []

        for product_id in df['product_id'].unique():
            product_data = df[df['product_id'] ==
product_id].sort_values('date')

            if len(product_data) < self.sequence_length + 1:
                continue

            features = self._get_numerical_features(product_data)

            if 'sales' in features:
                features.remove('sales')
                features = ['sales'] + features

            values = product_data[features].values

            for i in range(len(values) - self.sequence_length):
                sequences_X.append(values[i:i +
self.sequence_length])
                sequences_y.append(values[i + self.sequence_length,
0])

        return {
            'X': np.array(sequences_X),
            'y': np.array(sequences_y),
            'features': features
        }
```

5. metrics.py

```
import numpy as np
from typing import Dict, List, Tuple
from scipy import stats

class ForecastMetrics:

    @staticmethod
    def mae(y_true: np.ndarray, y_pred: np.ndarray) -> float:
        return np.mean(np.abs(y_true - y_pred))

    @staticmethod
    def mse(y_true: np.ndarray, y_pred: np.ndarray) -> float:
        return np.mean((y_true - y_pred) ** 2)

    @staticmethod
    def rmse(y_true: np.ndarray, y_pred: np.ndarray) -> float:
        return np.sqrt(ForecastMetrics.mse(y_true, y_pred))

    @staticmethod
    def mape(y_true: np.ndarray, y_pred: np.ndarray, epsilon:
float = 1e-10) -> float:
        mask = y_true > epsilon
        if not mask.any():
            return 0.0

        return np.mean(np.abs((y_true[mask] - y_pred[mask]) /
y_true[mask])) * 100

    @staticmethod
    def wmape(y_true: np.ndarray, y_pred: np.ndarray, epsilon:
float = 1e-10) -> float:
        numerator = np.sum(np.abs(y_true - y_pred))
        denominator = np.sum(y_true) + epsilon
        return (numerator / denominator) * 100

    @staticmethod
    def smape(y_true: np.ndarray, y_pred: np.ndarray, epsilon:
float = 1e-10) -> float:
        numerator = np.abs(y_true - y_pred)
        denominator = (np.abs(y_true) + np.abs(y_pred)) / 2 +
epsilon
        return np.mean(numerator / denominator) * 100

    @staticmethod
    def bias(y_true: np.ndarray, y_pred: np.ndarray) -> float:
        return np.mean(y_pred - y_true)

    @staticmethod
    def forecast_accuracy(y_true: np.ndarray, y_pred:
np.ndarray) -> float:
        mape = ForecastMetrics.mape(y_true, y_pred)
        return max(0, 100 - mape)

    @staticmethod
    def stock_out_rate(y_true: np.ndarray, y_pred: np.ndarray) ->
float:
        stock_outs = np.sum(y_true > y_pred)
        total = len(y_true)
        return (stock_outs / total) * 100

    @staticmethod
    def overstock_rate(y_true: np.ndarray, y_pred: np.ndarray,
                       threshold: float = 0.5) -> float:
        overstocks = np.sum(y_true < threshold * y_pred)
        total = len(y_true)
        return (overstocks / total) * 100

    @staticmethod
```

```
    def coverage_probability(y_true: np.ndarray, lower_bound: np.ndarray,
                             upper_bound: np.ndarray) -> float:
        within_bounds = np.sum((y_true >= lower_bound) & (y_true <= upper_bound))
        total = len(y_true)
        return (within_bounds / total) * 100

    @staticmethod
    def calculate_all_metrics(y_true: np.ndarray, y_pred: np.ndarray,
                              lower_bound: np.ndarray = None,
                              upper_bound: np.ndarray = None) -> Dict[str, float]:
        metrics = {
            'mae': ForecastMetrics.mae(y_true, y_pred),
            'mse': ForecastMetrics.mse(y_true, y_pred),
            'rmse': ForecastMetrics.rmse(y_true, y_pred),
            'mape': ForecastMetrics.mape(y_true, y_pred),
            'wmape': ForecastMetrics.wmape(y_true, y_pred),
            'smape': ForecastMetrics.smape(y_true, y_pred),
            'bias': ForecastMetrics.bias(y_true, y_pred),
            'accuracy': ForecastMetrics.forecast_accuracy(y_true, y_pred),
            'stock_out_rate': ForecastMetrics.stock_out_rate(y_true, y_pred),
            'overstock_rate': ForecastMetrics.overstock_rate(y_true, y_pred)
        }

        if lower_bound is not None and upper_bound is not None:
            metrics['coverage_probability'] = ForecastMetrics.coverage_probability(
                y_true, lower_bound, upper_bound
            )

        return metrics

    @staticmethod
    def metrics_by_category(y_true: np.ndarray, y_pred: np.ndarray,
                            categories: np.ndarray) -> Dict[str, Dict[str, float]]:
        results = {}

        for category in np.unique(categories):
            mask = categories == category

            if np.sum(mask) > 0:
                cat_true = y_true[mask]
                cat_pred = y_pred[mask]

                results[str(category)] = ForecastMetrics.calculate_all_metrics(
                    cat_true, cat_pred
                )

        return results


class ModelComparison:

    @staticmethod
    def compare_models(y_true: np.ndarray,
                       predictions: Dict[str, np.ndarray]) -> Dict[str, Dict]:
        results = {}

        for model_name, y_pred in predictions.items():
            results[model_name] = ForecastMetrics.calculate_all_metrics(
                y_true, y_pred
            )

        return results

    @staticmethod
    def rank_models(comparison_results: Dict[str, Dict],
                    metric: str = 'mape') -> List[Tuple[str, float]]:
        rankings = [
            (model, metrics.get(metric, float('inf')))
            for model, metrics in comparison_results.items()
        ]

        if metric in ['accuracy', 'coverage_probability']:
            rankings.sort(key=lambda x: x[1], reverse=True)
        else:
            rankings.sort(key=lambda x: x[1])

        return rankings

    @staticmethod
    def statistical_significance_test(y_true: np.ndarray,
                                      pred1: np.ndarray,
                                      pred2: np.ndarray,
                                      alpha: float = 0.05) -> Dict:
        error1 = np.abs(y_true - pred1)
        error2 = np.abs(y_true - pred2)

        diff = error1 - error2

        mean_diff = np.mean(diff)
        std_diff = np.std(diff, ddof=1)
        n = len(diff)

        if std_diff == 0:
            return {
                'statistic': 0,
                'p_value': 1.0,
                'significant': False,
                'better_model': 'equal'
            }

        t_stat = mean_diff / (std_diff / np.sqrt(n))

        p_value = 2 * (1 - stats.t.cdf(abs(t_stat), n - 1))

        if p_value < alpha:
            better_model = 'model_1' if mean_diff < 0 else 'model_2'
            significant = True
        else:
            better_model = 'no_significant_difference'
            significant = False

        return {
            'statistic': t_stat,
            'p_value': p_value,
            'significant': significant,
            'better_model': better_model,
            'mean_diff': mean_diff
        }
```