Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: *"Інтелектуальна система аналізу та прогнозування ринку криптовалют"*

Виконав: студент 6 курсу групи КН-62м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва спеціальності)

Романко С. А.
(прізвище та ініціали)

Керівник *Процик Ю. С.*
(прізвище та ініціали)

Рецензент Сороптук О.І.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І. Б.
"10" грудня 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Романку Станіславу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *"Інтелектуальна система аналізу та прогнозування ринку криптовалют"*

керівник роботи *Процик Юрій Степанович, к.ф-м.н.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"29" квітня 2025 року* № *С-288*
2. Термін подання студентом роботи *10 грудня 2025 р.*
3. Вихідні дані до роботи *Аналіз шляхів вирішення задачі, організаційна структура системи*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Перелік скорочень та умовних позначень. Вступ.
Розділ 1. Стан проблемної області.
Розділ 2. Інформаційне забезпечення.
Розділ 3. Математичне забезпечення.
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту.
Висновки. Список використаних джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)

6. Дата видачі завдання *1 травня 2025 року*

КАЛЕНДАРНИЙ ПЛАН

№ з п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз стану проблемної області	*01.05.2025 р.* *02.06.2025 р.*	виконано
2.	Реалізація засобів збору та попередньої обробки даних	*03.06.2025 р.* *16.07.2025 р.*	виконано
3.	Вибір і реалізація архітектур нейронних мереж, налаштування гіперпараметрів та порівняння якості моделей	*17.07.2025 р.* *11.09.2025 р.*	виконано
4.	Автоматизація оновлення прогнозів, збереження фактичних та прогнозованих даних	*12.09.2025 р.* *03.10.2025 р.*	виконано
5.	Створення веборієнтованої системи	*06.10.2025 р.* *19.11.2025 р.*	виконано
6.	Оформлення пояснювальної записки та здача на рецензування	*20.11.2025 р.* *10.12.2025 р.*	виконано

Студент _____ *Романко С. А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Процик Ю. С.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 86 сторінок пояснювальної записки, 14 рисунків, 3 таблиці, 1 додаток та 19 джерел.

У роботі розроблено інтелектуальну систему аналізу та прогнозування ринку криптовалют на основі нейронних мереж LSTM, GRU, CNN-LSTM, тощо. Використано погодинні дані отримані через API CoinGecko.

Створено веборієнтований застосунок для автоматизованого збирання, обробки та збереження даних у DuckDB, а також формування щогодинних прогнозів. Інтерфейс, реалізований у Streamlit з використанням Plotly, забезпечує інтерактивну візуалізацію часових рядів і прогнозних моделей.

Розроблена система може слугувати аналітичним інструментом для трейдерів і дослідників фінансових ринків.

Ключові слова: Python, машинне навчання, нейронні мережі, LSTM, GRU, часові ряди, прогнозування, криптовалюта.

ABSTRACT

The master's thesis includes 86 pages of documentation, 14 figures, 3 tables, 1 appendix, and 19 references.

This work presents an intelligent system for cryptocurrency market analysis and forecasting based on LSTM, GRU, CNN-LSTM, etc. Hourly data from the CoinGecko API was used.

A web-based application was developed to automate data collection, preprocessing, storage in DuckDB, and hourly forecast generation. The interface, built with Streamlit and Plotly, provides interactive visualization of time series and model predictions.

The developed system can be used as an analytical tool for traders and financial market researchers.

Keywords: Python, machine learning, neural networks, LSTM, GRU, time series, forecasting, cryptocurrency.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити програмне та алгоритмічне забезпечення інтелектуальної системи аналізу та прогнозування ринку криптовалют, а саме:

- реалізувати засоби автоматизованого збору погодинних даних про криптовалюту та їх попередню обробку;
- обрати та реалізувати кілька архітектур нейронних мереж (LSTM, GRU, CNN-LSTM, тощо), виконати налаштування їхніх гіперпараметрів та оцінити якість роботи моделей;
- перевірити роботу найкращої моделі на тестовій вибірці;
- реалізувати механізм автоматизованого оновлення прогнозів, а також збереження фактичних та прогнозованих даних для подальшого аналізу;
- розробити веборієнтоване програмне забезпечення для представлення даних, результатів прогнозування та інтерактивної візуалізації;
- провести інтерпретацію отриманих результатів та сформувати висновки щодо ефективності розробленої системи.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, інтерфейс прикладного програмування

BTC – Bitcoin, перша та найбільша криптовалюта

CNN – Convolutional Neural Network, згорткова нейронна мережа

DB – Database, база даних

DuckDB – колонково-орієнтована аналітична СУБД, що використовується для локального зберігання даних

GRU – Gated Recurrent Unit, рекурентна нейронна мережа зі спрощеною структурою

JSON – JavaScript Object Notation, формат передачі даних

LSTM – Long Short-Term Memory, рекурентна мережа з довгою короткочасною пам’яттю

MAE – Mean Absolute Error, середня абсолютна похибка

ML – Machine Learning, машинне навчання

MVC – Model-View-Controller, архітектурний шаблон

OHLCV – Open, High, Low, Close, Volume – набір ціни та об’єму торгів

Parquet – колонково-орієнтований формат збереження даних

ReLU – Rectified Linear Unit, функція активації

RMSE – Root Mean Squared Error, середньоквадратична похибка

RNN – Recurrent Neural Network, рекурентна нейронна мережа

SQL – Structured Query Language, мова запитів до баз даних

Streamlit – фреймворк для створення веб інтерфейсів Python-додатків

t+1 – прогноз наступного часового кроку

URL – Uniform Resource Locator, адреса ресурсу в Інтернеті

ВСТУП

Швидкий розвиток фінансових технологій та зростання популярності криптовалют призвели до появи нових підходів до аналізу та прогнозування динаміки цифрових активів.

Актуальність. Ринок криптовалют характеризується високою волатильністю, нелінійністю та залежністю від широкого спектра внутрішніх і зовнішніх факторів, що ускладнює застосування традиційних аналітичних методів та актуалізує потребу у створенні інтелектуальних систем прогнозування. Використання моделей глибинного навчання, здатних враховувати складні часові взаємозв'язки, розглядається як ефективний інструмент для аналізу фінансових часових рядів у таких умовах.

Об'єктом дослідження є процес прогнозування часових рядів фінансових даних криптовалютного ринку.

Предметом дослідження є методи, моделі та алгоритми глибинного машинного навчання для короткострокового прогнозування вартості криптовалюти Bitcoin, а також засоби їх програмної реалізації у вигляді веборієнтованої аналітичної системи.

Метою магістерської роботи є розроблення інтелектуальної системи для аналізу та прогнозування вартості криптовалюти Bitcoin з використанням глибинних моделей машинного навчання, автоматизованого збору та обробки даних, а також інтерактивної візуалізації результатів у режимі реального часу.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати особливості криптовалютного ринку та сучасні підходи до прогнозування фінансових часових рядів;
- дослідити архітектури моделей глибинного навчання (LSTM, GRU, CNN-LSTM, Attention) та визначити їх придатність для прогнозування волатильних рядів;
- зібрати та підготувати погодинні ринкові дані з використанням відкритих API;

- реалізувати модулі обробки даних та формування ознак;
- побудувати, навчити та порівняти декілька моделей машинного навчання за метриками точності;
- розробити вебінтерфейс, що забезпечує доступ до історичних даних, прогнозів та аналітичних графіків;
- інтегрувати усі компоненти в єдину систему з автоматичним оновленням даних і прогнозів t+1 у режимі реального часу.

Наукова новизна роботи полягає у поєднанні методів глибинного навчання із серверною автоматизованою платформою короткострокового прогнозування криптовалют, що дозволяє підвищити оперативність доступу до прогнозів та забезпечує безперервність роботи моделі. Додатковою складовою новизни є практичне порівняння сучасних нейронних архітектур у контексті волатильних фінансових рядів та визначення оптимальних рішень для реальних умов ринку.

Практичне значення розробленої системи полягає у можливості її використання трейдерами, аналітиками, дослідниками та освітніми закладами як інструмента для моніторингу ринку, формування прогнозів та створення торгових стратегій. Архітектура платформи забезпечує масштабованість рішення й дозволяє розширювати його функціональність за рахунок підтримки нових криптовалют, моделей машинного навчання та аналітичних механізмів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Актуальність прогнозування вартості криптовалют

Ринок криптовалют за останнє десятиліття перетворився на одну з найбільш динамічних та волатильних фінансових екосистем у світі. Bitcoin, як перша та найвпливовіша криптовалюта, формує загальні ринкові тренди, а зміни його ціни впливають на поведінку інших цифрових активів і настрій інституційних інвесторів [1]. Постійні коливання вартості, спричинені глобальними новинами, макроекономічними змінами, політикою регуляторів та ринковими настроями, роблять прогнозування ціни криптовалют складним, але вкрай важливим завданням.

На відміну від традиційних фінансових ринків, криптовалютний ринок працює безперервно – 24 години на добу, 7 днів на тиждень – що вимагає застосування автоматизованих інтелектуальних систем для оперативного аналізу та формування прогнозів у реальному часі [2]. Більшість криптовалют не мають фундаментальних показників, характерних для акцій чи товарів, тому основну інформацію про них містять саме часові ряди цін та обсягів торгів.

Висока волатильність і нелінійність криптовалютних даних ускладнюють використання традиційних методів прогнозування, таких як ARIMA чи експоненціальне згладжування. У той же час глибинні нейронні мережі, зокрема моделі типу LSTM та GRU, демонструють високу ефективність у моделюванні довготривалих залежностей і складних патернів у часових рядах [3]. Завдяки здатності враховувати часовий контекст і нелінійні взаємозв’язки, такі моделі дозволяють отримувати точніші короткострокові прогнози.

Актуальність прогнозування криптовалют зростає й у зв’язку з розвитком алгоритмічної торгівлі. Автоматизовані торгові системи потребують стабільних і точних моделей прогнозування, від якості яких залежить рівень ризику, ефективність управління активами та прийняття рішень у високошвидкісному ринковому середовищі [4]. Надійні прогностичні системи дозволяють трейдерам своєчасно реагувати на зміни ринку та оптимізувати торгові стратегії.

Таким чином, прогнозування вартості криптовалют на основі сучасних методів машинного навчання є актуальним науковим і прикладним завданням, яке сприяє розвитку фінансових технологій, підвищенню точності аналітичних систем та зменшенню ризиків у торгових операціях.

1.1.1 Особливості криптовалютного ринку

Криптовалютний ринок суттєво відрізняється від традиційних фінансових ринків за структурою, динамікою та поведінкою учасників. Однією з ключових особливостей є його децентралізований характер: криптовалюти не контролюються жодною державою чи фінансовою установою, а функціонують у межах розподілених мереж, що забезпечують прозорість та стійкість до цензури [5]. Відсутність централізованого регулятора створює середовище зі значним рівнем свободи для інвесторів, але водночас збільшує ризики.

Ще однією важливою характеристикою є цілодобовий режим роботи ринку. Криптовалютні біржі функціонують без перерви – 24/7, що забезпечує високу ліквідність, проте сприяє підвищеній волатильності, яка виникає внаслідок миттєвої реакції учасників на новини, технічні фактори чи зміни ринкових настроїв [6]. Такий режим ускладнює ручний аналіз і робить автоматизовані системи прогнозування практично необхідними.

Велике значення має структура торгів: ринок криптовалют характеризується переважною участю роздрібних інвесторів, швидкою зміною трендів та високою чутливістю до інформаційного фону. Дослідження показують, що соціальні мережі, новини, аналітичні огляди та навіть публікації впливових осіб можуть спричиняти різкі цінові коливання, що практично не характерно для традиційних ринків у такому масштабі [7].

Крім того, ринок криптовалют має низьку бар’єрність входу. Для купівлі чи продажу цифрових активів достатньо мобільного застосунку та доступу до Інтернету, що стимулює масовість та активну участь користувачів. Така доступність

формує високий рівень спекулятивності, що також збільшує складність прогнозування цінових рухів.

Важливою особливістю є й відсутність класичних фундаментальних показників. На відміну від акцій, криптовалюти не мають фінансової звітності, дивідендної політики чи чітких макроекономічних прив’язок. Тому основним інформаційним носієм для аналітики є саме часові ряди цін та об'єми торгів, які відображають поведінку ринку та дозволяють будувати моделі короткострокового прогнозування [8].

Таким чином, криптовалютний ринок характеризується поєднанням високої волатильності, децентралізації, доступності та інформаційної чутливості. Це робить його складним, але надзвичайно важливим об’єктом для дослідження за допомогою сучасних математичних методів та інтелектуальних інформаційних систем.

1.1.2 Волатильність та ризики цифрових активів

Волатильність є ключовою характеристикою криптовалютного ринку, яка значною мірою визначає його ризики та складність прогнозування. Порівняно з традиційними фінансовими інструментами, криптовалюти демонструють значно більшу амплітуду цінових коливань навіть протягом коротких часових інтервалів. Така поведінка пов’язана з низкою факторів: недостатньою ліквідністю окремих активів, підвищеною спекулятивною активністю, відсутністю фундаментальних показників та швидкою реакцією учасників ринку на інформаційні події.

Висока волатильність створює як можливості для потенційного прибутку, так і суттєві ризики для інвесторів. Нерідко протягом доби відбуваються різкі цінові стрибки, зумовлені публікаціями у медіа, активністю великих гравців, технічними збоями бірж або змінами в регуляторному полі. Такі коливання ускладнюють застосування класичних методів аналізу та потребують використання глибинних моделей, здатних фіксувати приховані патерни та нелінійні залежності у часових рядах.

Особливим видом ризику є інституційна невизначеність, зокрема нестабільність правового статусу криптовалют у різних країнах. Регуляторні новини можуть спричиняти суттєві цінові коливання. Дослідження показують, що ринок криптовалют є надзвичайно чутливим до інформації, яка формує очікування інвесторів, що підсилює ефект раптових змін [9].

Також важливим фактором є ризик маніпуляцій ринком. Через відсутність централізованого регулювання та домінування роздрібних інвесторів криптовалюти часто стають об’єктом штучного розкручування цін, “pump and dump”-схем та інших непрозорих практик. Це створює додаткові труднощі для побудови точних моделей прогнозування.

Таким чином, волатильність та ризиковість криптовалютного ринку визначають необхідність застосування сучасних аналітичних підходів, орієнтованих на роботу з нестабільними та нелінійними процесами.

1.1.3 Складність побудови прогнозів у реальному часі

Побудова прогнозів вартості криптовалют у режимі реального часу є складним завданням через специфіку ринку та особливості поведінки його учасників. Однією з головних проблем є швидкоплинність ринкових процесів: новини, події та зміни у торговельній активності впливають на ціну практично миттєво, створюючи умови, за яких прогнози швидко втрачають актуальність. Це вимагає використання моделей, здатних оперативно адаптуватися до оновлених даних та формувати короткострокові передбачення з високою частотою оновлення.

Важливою складністю є також висока чутливість криптовалют до шуму в даних. Часові ряди цін містять значну частку випадкових флуктуацій, які ускладнюють виділення стабільних закономірностей. Класичні статистичні методи часто не справляються з таким рівнем нестабільності, тому все більшого поширення набувають методи глибинного навчання, зокрема моделі рекурентних нейронних мереж, здатні враховувати контекст попередніх значень та виконувати багатовимірний аналіз структури ряду [10].

Ще одним аспектом, що ускладнює прогнозування, є відсутність фіксованих тимчасових патернів. На відміну від традиційних фінансових інструментів, криптовалюти не підпорядковуються класичним циклам торгів, які характерні для фондових ринків. Різниця в активності учасників залежно від часових поясів, а також глобальна доступність бірж створюють нерівномірне навантаження на ринок, що впливає на формування трендів.

Не менш вагомою проблемою є обмеження у швидкості обробки даних. Для створення ефективної системи прогнозування в реальному часі необхідно забезпечити швидке завантаження даних, їх попередню обробку, нормалізацію та передачу на вхід моделі. У веборієнтованих аналітичних системах це потребує оптимізації архітектури програмного забезпечення та застосування легковагових інструментів для забезпечення стабільної роботи у фоновому режимі.

Таким чином, складність побудови прогнозів у реальному часі визначається високою частотою оновлення даних, нестабільністю ринкових процесів, шумністю часових рядів та технологічними вимогами до швидкості обробки інформації. Це робить застосування сучасних моделей глибинного навчання та оптимізованих програмних рішень необхідною умовою створення ефективних прогнозних систем.

1.2 Аналіз сучасних підходів до прогнозування часових рядів

Прогнозування часових рядів є одним із ключових завдань у сфері фінансової аналітики, економіки та машинного навчання. Існує широкий спектр методів, що застосовуються для моделювання динаміки даних, кожен з яких має власні переваги й обмеження. Вибір конкретного підходу залежить від властивостей ряду, його стабільності, рівня шуму, сезонності та наявності довгострокових залежностей.

Одним із найпоширеніших класичних методів прогнозування є моделі ARIMA, які добре працюють на стаціонарних часових рядах з вираженою автокореляцією. Проте їх ефективність знижується у випадках високої волатильності та нелінійних залежностей, що характерні для криптовалютного ринку [11]. Інші статистичні підходи, зокрема експоненціальне згладжування (модель Холта–Вінтерса),

демонструють хороші результати за умов регулярної сезонності, однак малопридатні для ринків з хаотичною динамікою.

Методи машинного навчання, такі як випадкові ліси та градієнтний бустинг, забезпечують більшу гнучкість завдяки здатності моделювати нелінійні взаємозв'язки. Проте їх ефективність у прогнозуванні часових рядів обмежена тим, що вони не враховують часову структуру даних без спеціальної побудови ознак. Для коректної роботи таких моделей необхідно додатково формувати лагові ознаки, ковзаючі середні та інші статистичні характеристики, що підвищує складність підготовки даних.

У останні роки значного поширення набули методи глибинного навчання, зокрема рекурентні нейронні мережі (RNN), моделі LSTM та GRU. Їхньою основною перевагою є здатність враховувати часовий контекст і моделювати довготривалі залежності без ручного створення спеціалізованих ознак [12]. Такі моделі показують високу ефективність у задачах короткострокового прогнозування фінансових даних, у тому числі на ринках із високою волатильністю.

Окреме місце посідають комбіновані моделі, зокрема CNN-LSTM, що поєднують здатність згорткових нейронних мереж до виділення локальних патернів із перевагами рекурентних механізмів. Завдяки цьому такі моделі демонструють стійкість до шуму й здатність враховувати як місцеві, так і довгострокові залежності в даних.

Таким чином, сучасний підхід до прогнозування часових рядів передбачає поєднання статистичних, машинних та глибинних методів, що дозволяє комплексно аналізувати дані та обирати оптимальні моделі залежно від характеру конкретного ринку. У контексті криптовалют найбільш перспективними є моделі глибинного навчання, які забезпечують високу адаптивність та точність прогнозів у динамічному середовищі.

1.2.1 Класичні статистичні методи

Класичні статистичні методи прогнозування часових рядів тривалий час були основним інструментом фінансової аналітики. Їхню популярність зумовлено простотою застосування, зрозумілою математичною інтерпретацією та здатністю моделювати дані з вираженими закономірностями. Найбільш відомими серед цих методів є моделі AR, MA, ARMA, ARIMA та експоненціальне згладжування.

Моделі ARIMA широко застосовуються для прогнозування стаціонарних часових рядів. Вони враховують автокореляцію даних, а також дозволяють представляти ряд як комбінацію авторегресії, ковзного середнього та інтегрування. Проте ефективність ARIMA суттєво знижується за умов високої волатильності та різких структурних зламів, що характерно для криптовалютних рядів [13].

Метод експоненціального згладжування, зокрема модель Холта–Вінтерса, добре працює для рядів із вираженою сезонністю та плавними трендами. Він дозволяє виділяти основні компоненти ряду – рівень, тренд і сезонність. Однак для криптовалют, де сезонність змінюється нерегулярно, а поведінка ціни є хаотичною, точність таких методів обмежена.

Достатньо часто класичні статистичні моделі показують хороші результати на стабільних економічних даних, проте демонструють низьку ефективність на ринках із високою частотою змін. Вони не враховують нелінійні залежності, різкі стрибки, асиметричні реакції ринку та інформаційний вплив новин. У випадку криптовалютного ринку це створює обмеження для використання таких моделей без додаткових модифікацій.

Водночас статистичні методи мають важливу перевагу – інтерпретованість. Вони дозволяють чітко пояснити структуру моделі, значення параметрів та їхній вплив на результат. Це робить їх корисними як інструменти для попереднього аналізу та порівняння з більш складними моделями машинного навчання.

Таким чином, класичні статистичні методи являють собою основу для аналізу часових рядів, проте їхня ефективність на ринку криптовалют є обмеженою через нелінійність, високу волатильність та відсутність сталих закономірностей у даних.

1.2.2 Методи машинного навчання

Традиційні методи машинного навчання займають проміжне місце між класичними статистичними моделями та сучасними глибинними нейронними мережами. Вони здатні враховувати нелінійні залежності в даних, працювати зі складними структурами та демонструвати високу точність за умови правильної побудови ознак. На відміну від глибинних моделей, більшість алгоритмів машинного навчання не оперує часовою структурою даних безпосередньо, тому їх ефективність залежить від якості попереднього формування ознак.

Одним із найпоширеніших підходів є використання дерев рішень та їх ансамблів. Моделі випадкового лісу (Random Forest) дозволяють враховувати складні нелінійні взаємозв'язки та стійкі до шуму, проте їхній прогноз є усередненим, що іноді зменшує чутливість до різких змін у даних. Градієнтний бустинг, зокрема XGBoost та LightGBM, показує високу точність завдяки послідовному навчанию композиції слабких моделей, що робить його ефективним для складних задач регресії [14].

Проте головним недоліком класичних ML-моделей у задачах прогнозування часових рядів є необхідність ручного створення ознак. Для ефективної роботи таких моделей потрібно формувати лагові змінні, ковзаючі середні, індикатори зміни об'єму та інші статистичні характеристики. Це збільшує трудомісткість підготовки даних та ускладнює адаптацію моделі до швидких змін ринку, таких як ті, що спостерігаються у криптовалютах.

Ще однією особливістю методів машинного навчання є їхня обмежена здатність до врахування довготривалих залежностей. Навіть за наявності великої кількості штучно створених ознак модель може не враховувати глобальні тренди або приховані зв'язки, які простежуються лише на великих проміжках часу.

Попри це, методи машинного навчання залишаються важливим інструментом у фінансовому прогнозуванні. Вони можуть бути використані як базові моделі для порівняння, а також як частина комбінованих систем, що поєднують ML-підходи з глибинними нейронними мережами.

Таким чином, методи машинного навчання забезпечують достатню гнучкість і здатність моделювати нелінійні процеси, проте їх ефективність у прогнозуванні криптовалют обмежена потребою у складній побудові ознак та відсутністю механізмів роботи з часовими залежностями.

1.2.3 Глибинні нейронні мережі в задачах прогнозування

Глибинні нейронні мережі (Deep Learning) стали одним із найефективніших інструментів для прогнозування часових рядів, особливо у сферах з високою нестабільністю та складними нелінійними залежностями, таких як криптовалютні ринки. Їхня здатність автоматично виділяти ознаки, адаптуватися до нових даних та працювати з довготривалими трендами забезпечує значні переваги над класичними статистичними та традиційними ML-підходами.

Одним із найважливіших класів моделей для аналізу часових рядів є рекурентні нейронні мережі (RNN). Вони здатні враховувати послідовність даних, зберігати інформацію про попередні стани та передавати її на наступні кроки, що робить їх придатними для прогнозування фінансових рядов. Проте класичні RNN мають суттєві обмеження, зокрема проблему зникнення градієнтів, що ускладнює навчання на довгих послідовностях.

Для подолання цих недоліків було розроблено моделі LSTM та GRU, які використовують спеціальні елементи керування інформаційними потоками. Моделі LSTM, завдяки комбінації комірок пам’яті та механізмів забування, дозволяють точно моделювати довготривалі залежності та ефективно працюють на даних з високою волатильністю. GRU є спрощеною версією LSTM, але водночас забезпечують подібну точність за меншої кількості параметрів, що робить їх придатними для задач, де важлива швидкість обчислень.

Глибинні моделі також включають архітектури із згортковими шарами, такі як CNN-LSTM. Згорткові мережі здатні виявляти локальні патерни у часових рядах, зокрема короткострокові структури, які важко вловити традиційними методами.

Поєднання CNN і LSTM забезпечує подвійний ефект: виділення локальних ознак та моделювання їх динаміки у часі.

Окрему категорію становлять моделі з механізмом уваги (Attention), які дозволяють мережі самостійно визначати, які частини послідовності є найбільш значущими для прогнозу. Завдяки цьому такі системи досягають високої точності навіть у випадках, коли традиційні методи не здатні обробляти складні або нерегулярні патерни [15]. У фінансовому прогнозуванні attention-моделі отримали широке застосування, оскільки вони компенсують недоліки RNN, пов'язані з обмеженням довжини пам'яті.

Таким чином, глибинні нейронні мережі є найперспективнішим напрямом у прогнозуванні криптовалютних часових рядів завдяки здатності обробляти нелінійні залежності, шумні дані та складні ринкові закономірності. Вони забезпечують високу точність прогнозів і добре адаптуються до мінливих умов ринку, що робить їх оптимальним вибором для побудови сучасних інтелектуальних систем аналізу фінансових даних.

Висновки до розділу

У результаті проведеного огляду встановлено, що криптовалютний ринок має низку специфічних особливостей, які суттєво ускладнюють процес прогнозування його динаміки. Децентралізований характер ринку, цілодобовий режим роботи, висока залежність від інформаційного фону та значна доля спекулятивної активності створюють середовище з підвищеною волатильністю. Це робить традиційні підходи до фінансового аналізу малоефективними та вимагає застосування сучасних методів прогнозування.

Розглянуті класичні статистичні моделі, зокрема ARIMA та методи експоненційного згладжування, демонструють високу ефективність на стаціонарних даних, проте мають суттєві обмеження у випадку ринків з нерегулярною та хаотичною динамікою, що властиво криптовалютам. Традиційні методи машинного навчання, хоча й здатні моделювати нелінійні залежності, потребують складного

формування ознак та не враховують часову структуру даних без додаткових модифікацій.

Найбільш перспективними для прогнозування часових рядів на криптовалютному ринку є глибинні нейронні мережі, зокрема LSTM, GRU, CNN-LSTM та моделі з механізмом уваги. Вони забезпечують високу адаптивність, здатність обробляти нелінійні процеси та формувати точні короткострокові прогнози навіть у системах з високою волатильністю.

Таким чином, аналіз сучасних підходів підтвердив необхідність застосування глибинного навчання для побудови інтелектуальної системи прогнозування криптовалют, що й визначає подальший напрям дослідження в наступних розділах.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Джерела даних та інструменти збору інформації

Для побудови системи прогнозування вартості криптовалюти необхідним є доступ до якісних історичних та актуальних ринкових даних. Основним типом даних, що використовується у фінансовому аналізі, є часові ряди, які містять інформацію про відкриту (Open), максимальну (High), мінімальну (Low), закриту (Close) ціну активу та обсяг торгів (Volume). Такі дані позначаються загальновідомим скороченням OHLCV і є стандартом фінансової індустрії.

У межах даної роботи основним джерелом даних виступає інтерфейс програмного доступу CoinGecko API. Це один із найпопулярніших відкритих сервісів, що надає актуальні та історичні дані про криптовалюти, включно з погодинними часовими рядами, ринковою капіталізацією, об’ємами торгів та іншими характеристиками. CoinGecko API забезпечує стабільний доступ, високу частоту оновлення та зручний формат відповіді у вигляді JSON, що дозволяє ефективно інтегрувати його у програмні системи [16].

Важливою перевагою цього джерела є безкоштовна модель доступу та відсутність жорстких обмежень на кількість запитів, що робить його оптимальним вибором для освітніх, дослідницьких та експериментальних цілей. Крім того, сервіс має високу репутацію в індустрії завдяки точності даних та стабільності роботи.

Для отримання даних використовується спеціалізований модуль збору інформації, який звертається до API, отримує часові ряди за вибраний період та зберігає їх у структурованому форматі. Дані попередньо приводяться до єдиної часової шкали, що є важливою умовою коректного навчання моделей машинного навчання. Після збору інформації здійснюється її перетворення у внутрішній формат системи та подальше збереження у локальне сховище.

Окрім зовнішнього джерела даних, важливу роль відіграє внутрішня інфраструктура їх зберігання. Для цього використовується DuckDB – аналітична колонкова база даних, оптимізована для локальних розрахунків. Вона забезпечує

швидку обробку великих обсягів інформації, ефективну роботу з Parquet-файлами та інтеграцію з Python, що дозволяє безперешкодно здійснювати подальші етапи обробки та аналізу даних.

Таким чином, використання CoinGecko API як основного джерела даних, у поєднанні з DuckDB як інструментом локального зберігання, забезпечує надійний і гнучкий фундамент для побудови інтелектуальної системи прогнозування криптовалют.

2.1.1 CoinGecko API як основне джерело погодинних даних

CoinGecko API є одним із найбільш доступних та надійних інструментів для отримання погодинних та щоденних даних про криптовалюти. Завдяки своїй відкритості та безкоштовній моделі використання, сервіс став стандартом у дослідницьких проєктах, аналітичних системах і навчальних роботах. Він надає широкий набір ринкових показників, включаючи часові ряди OHLCV, історичні ціни, ринкову капіталізацію, об'єми торгів та інші метрики, потрібні для фінансового аналізу.

Однією з ключових переваг CoinGecko API є висока частота оновлення даних. Погодинні ринкові дані дозволяють моделювати короткострокові тренди та зменшувати часову затримку між реальною ринковою динамікою та входом моделей машинного навчання. Завдяки цьому система отримує актуальну інформацію, що є критично важливим у задачах прогнозування цін криптовалют, де навіть незначні затримки можуть впливати на точність передбачень.

Сервіс надає дані у форматі JSON, що значно спрощує їхню інтеграцію у програмні системи. Формат відповіді містить однозначно структуровані масиви, де кожен елемент включає часову мітку та значення ціни або об’єму. Така структура дозволяє без додаткових труднощів перетворювати дані у формати, потрібні для подальшої обробки, включно з побудовою датафреймів та формуванням ознак для моделей.

З технічної точки зору, CoinGecko API вирізняється стабільною роботою та порівняно високими лімітами на кількість запитів. Це робить сервіс зручним для реалізації автоматизованих систем збору та оновлення даних, де здійснюється періодичне звернення до API в режимі реального часу. В умовах швидкозмінного ринку криптовалют така можливість дозволяє забезпечити безперервне оновлення інформації та підтримувати моделі у актуальному стані.

У контексті цієї роботи CoinGecko API використовується як основне джерело даних для отримання погодинних цін Bitcoin (BTC-USD). Це забезпечує достатню деталізацію часових рядів, необхідну для побудови високоточних моделей прогнозування та аналізу ринкових закономірностей.

2.1.2 Особливості структури OHLCV-даних

OHLCV-дані є стандартним форматом представлення фінансових часових рядів, який використовується біржами, трейдерами, аналітичними платформами та системами автоматичного прогнозування. Абревіатура OHLCV означає Open, High, Low, Close та Volume – ключові параметри, що описують поведінку активу протягом певного інтервалу часу.

Показник Open відображає ціну відкриття періоду, тобто вартість активу на початку відповідного часового інтервалу. Значення High і Low представляють найбільшу та найменшу зафіксовану ціну в межах цього інтервалу, що дозволяє оцінити діапазон коливань та рівень ринкової активності. Показник Close є одним із найважливіших елементів, оскільки він відображає ціну закриття періоду та зазвичай використовується під час навчання моделей прогнозування.

Параметр Volume містить інформацію про кількість активів, що були продані або куплені за відповідний проміжок часу. Цей показник дозволяє оцінити інтенсивність торгів і часто використовується як допоміжний індикатор у фінансовому аналізі.

Нижче наведено приклад структури OHLCV-даних у погодинному розрізі:

Таблиця 2.1 – Приклад OHLCV-даних для Bitcoin (BTC-USD)

Дата та час	Open	High	Low	Close	Volume
2025-01-01 10:00	43250,12	43410,55	43180,20	43390,80	152,34
2025-01-01 11:00	43390,80	43520,10	43290,45	43450,30	164,92
2025-01-01 12:00	43450,30	43600,80	43370,20	43590,10	171,04

Для покращення наочності на рисунку 2.1 подано графічне представлення OHLCV у вигляді класичної японської свічки.

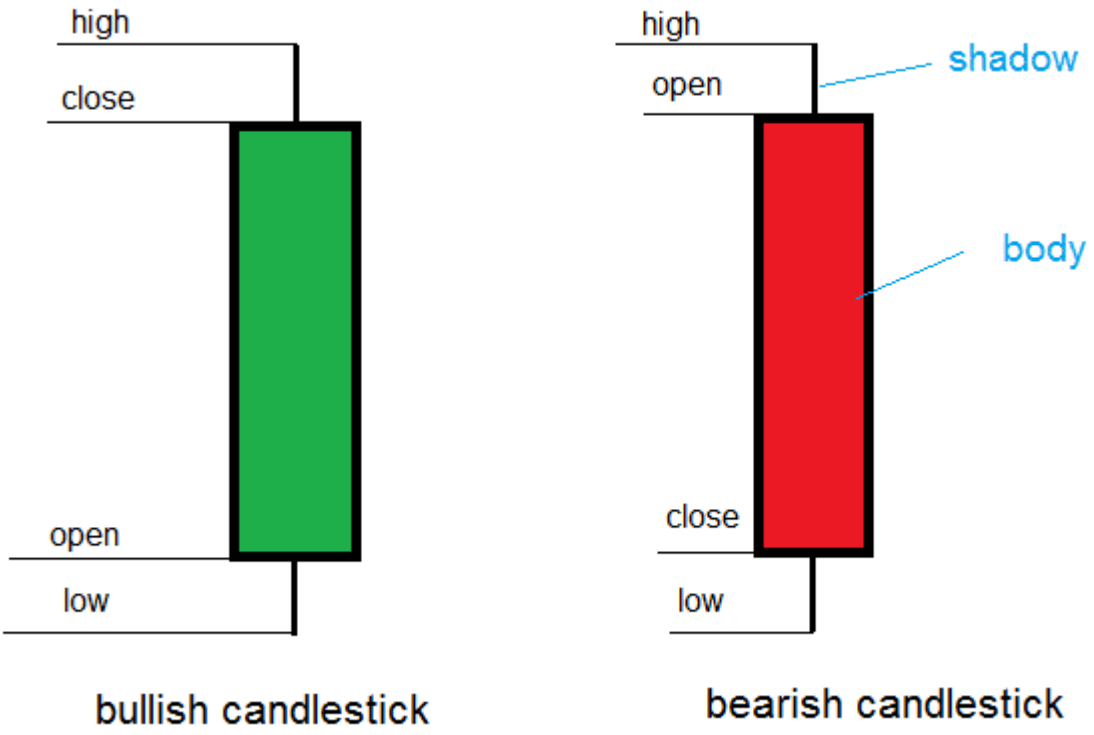


Рисунок 2.1 – графічне представлення OHLCV у вигляді свічки

Рисунок 2.2 – Графічне представлення графіку цін у свічковому вигляді

Свічковий графік (див. рис. 2.2) демонструє ті самі значення, що містяться в таблиці:

- Верх тіла свічки відповідає High,
- Низ тіла – Low,
- Open та Close – кольорові межі тіла свічки (залежно від напрямку руху ціни).

Структурованість OHLCV-даних робить їх універсальним форматом для фінансових досліджень. Вони дозволяють відстежувати як локальні зміни ціни, так і глобальні тренди, що є критично важливим для побудови моделей машинного навчання. Погодинні дані забезпечують оптимальний баланс між деталізацією та рівнем шуму, дозволяючи моделі реагувати на ринкові зміни оперативно і водночас не перевантажувати її надмірною кількістю даних.

Таким чином, OHLCV-дані забезпечують повноцінну картину ринкової динаміки за певний інтервал, створюючи основу для побудови точних, стійких і надійних прогнозних моделей.

2.2 Реалізація збору та зберігання даних

Процес збору та зберігання даних є одним із ключових етапів побудови інтелектуальної системи прогнозування криптовалют. Від ефективності цього процесу залежить якість вхідних даних, стабільність роботи моделей машинного навчання та можливість оперативного оновлення прогнозів у реальному часі.

У межах даної роботи реалізовано спеціалізований модуль автоматизованого збору даних, що здійснює взаємодію з CoinGecko API та формує погодинні OHLCV-дані для криптовалюти Bitcoin (BTC-USD). Модуль включає механізм обробки HTTP-запитів, перевірку коректності відповіді сервера та конвертацію отриманих значень у структурований формат. Важливою складовою цього етапу є приведення часових міток до єдиного стандарту UTC, що дозволяє уникнути розбіжностей при подальшому об'єднанні та порівнянні даних.

Після збору інформації дані проходять базову обробку, зокрема сортування, нормалізацію часових проміжків та заповнення можливих пропусків. Для зменшення ймовірності появи аномалій здійснюється перевірка відхилень у межах одного інтервалу та відкидання некоректних значень.

Збереження даних реалізовано з використанням DuckDB – високопродуктивної колонкової аналітичної бази даних, оптимізованої для роботи з локальними системами. DuckDB забезпечує швидку вибірку, ефективну обробку великої кількості рядків та можливість інтеграції з Parquet-файлами, що є перевагою в системах, де дані оновлюються з високою частотою.

Архітектура зберігання побудована таким чином, що історичні дані записуються у форматі Parquet, який добре стискає числову інформацію та дозволяє виконувати аналітичні операції без необхідності повного завантаження файлу в оперативну пам’ять. Паралельно DuckDB використовується як шар для SQL-запитів, що дозволяє швидко фільтрувати, агрегувати та трансформувати дані без додаткових перетворень.

Рішення доповнюється окремим модулем автоматизованого оновлення, що запускається з фіксованим інтервалом (раз на годину) та виконує синхронізацію з CoinGecko API. Завдяки цьому система завжди працює з актуальними ринковими даними, що особливо важливо в задачах короткострокового прогнозування.

Таким чином, модуль збору та зберігання даних забезпечує комплексний підхід до отримання та обробки інформації, формуючи надійну основу для

подальшої роботи моделей машинного навчання та інтелектуальної системи в цілому.

2.2.1 Модуль історичного збору даних

Модуль історичного збору даних є основним компонентом системи, який забезпечує отримання повного набору ринкової інформації за попередні періоди. Це необхідно для навчання моделей машинного навчання, їх подальшої валідації та формування бази для аналізу ринкових тенденцій. Побудова історичного набору даних дозволяє моделі працювати не лише з останніми значеннями, а й виявляти довготривалі закономірності, які нерідко визначають загальний напрямок ринку.

Реалізація модуля включає механізм надсилання серії запитів до CoinGecko API з параметрами, що відповідають часовим межам історичного інтервалу. Отримані дані містять погодинні OHLCV-значення, які формують основу для аналізу та побудови трендів. Модуль працює таким чином, що щоразу виконується перевірка коректності та повноти відповіді, після чого дані нормалізуються та перетворюються у внутрішній формат системи.

Важливою складовою модуля є обробка можливих пропусків або спотворень у даних. Через технічні особливості роботи криптовалютних бірж та API можливі ситуації, коли окремі часові інтервали містять некоректні або неповні значення. Для усунення таких проблем модуль виконує перевірку наявності всіх часових відміток та здійснює заповнення пропусків або відкидання некоректних записів у разі критичних відхилень.

Історичні дані завантажуються у форматі JSON, після чого перетворюються у табличну структуру. Після цього модуль зберігає результати у файл Parquet, який використовується як основний формат для збереження великих обсягів структурованих числових даних. Використання Parquet дозволяє оптимізувати процес читання та запису, а також забезпечує можливість подальшої обробки даних методами колонкової аналітики.

Модуль також враховує технічні обмеження API, зокрема затримки між запитами та максимальну кількість дозволених звернень за короткий проміжок часу. Це дозволяє уникнути блокування доступу та забезпечує стабільну роботу системи протягом усього циклу збору історичних даних. Додатково передбачено фіксацію останньої успішної часової позначки, що дозволяє відновити процес збору у разі технічних збоїв або переривання роботи програми.

Таким чином, модуль історичного збору даних забезпечує надійну, послідовну та автоматизовану побудову повного набору погодинних OHLCV-даних для криптовалюти Bitcoin. Це створює якісну основу для навчання моделей та забезпечує високу достовірність аналізу ринкових процесів

2.2.2 Використання DuckDB та формату Parquet

Ефективне зберігання та обробка даних є критичним елементом будь-якої системи, що працює з фінансовими часовими рядами та машинним навчанням. Обсяг історичних даних з часом збільшується, тому вибір оптимального формату та інструментів бази даних визначає продуктивність системи, швидкість завантаження даних та можливість виконання аналітичних операцій у режимі реального часу.

У розробленій системі використовується поєднання паркет-файлів (Parquet) та аналітичної бази даних DuckDB. Такий підхід забезпечує високу продуктивність, ефективне використання пам’яті та зручність інтеграції з інструментами машинного навчання.

Формат Parquet є колонково-орієнтованим форматом зберігання даних, який оптимізовано для робочих навантажень, пов’язаних з великими масивами числових значень. Завдяки колонковій структурі Parquet забезпечує високу ступінь стиснення та мінімізує обсяг зайнятого місця, що є важливою перевагою під час тривалого накопичення ринкових даних. Крім того, Parquet дозволяє виконувати вибіркове читання окремих колонок без необхідності завантаження всього файлу, що значно прискорює обробку даних на етапах попереднього аналізу та формування ознак.

DuckDB виступає інструментом локальної аналітичної бази даних, спеціально оптимізованої для роботи з даними, що зберігаються у колонкових форматах. На відміну від класичних реляційних СУБД, DuckDB орієнтована на OLAP-навантаження, що дозволяє виконувати складні аналітичні запити та агрегації з високою швидкістю навіть на звичайному локальному комп'ютері. Її архітектура побудована таким чином, що виконання SQL-запитів здійснюється у пам'яті, що суттєво підвищує продуктивність.

Взаємодія між Parquet і DuckDB забезпечує додаткові переваги. DuckDB може безпосередньо працювати з файлами Parquet без імпортування даних у внутрішню базу, що дозволяє виконувати SQL-запити безпосередньо поверх зовнішніх наборів даних. Це значно пришвидшує робочий процес та зменшує потребу у дублюванні інформації. Завдяки цьому аналітичні операції, такі як фільтрація, обчислення статистичних показників або агрегації по часових інтервалах, виконуються максимально ефективно.

У межах системи DuckDB використовується як основний інструмент для читання, фільтрації та підготовки даних перед передачею їх у модель машинного навчання. Зокрема, база даних забезпечує швидке формування вибірок, створення часових інтервалів, обчислення волатильності та інших характеристик, необхідних для побудови прогнозів.

Таким чином, поєднання DuckDB та Parquet забезпечує оптимальний баланс між швидкістю, ефективністю зберігання та зручністю інтеграції, що робить їх відповідним вибором для аналітичної системи прогнозування криптовалют, побудованої у межах даної роботи.

2.3 Попередня обробка та підготовка часових рядів

Попередня обробка даних є невід'ємним етапом побудови будь-якої системи прогнозування, оскільки саме на цьому етапі формуються вхідні дані для моделей машинного навчання. Якість підготовлених часових рядів значною мірою визначає точність моделей, їхню стабільність та здатність адаптуватися до змін ринку.

Першим кроком підготовки є сортування даних за часовою шкалою та приведення часових міток до єдиного формату. Для забезпечення сумісності даних усі часові позначки конвертуються у часову зону UTC, що дозволяє уникнути зміщень під час об’єднання історичних та нових отриманих даних.

На наступному етапі виконується перевірка наявності пропущених значень. У ринкових даних можуть виникати пропуски через технічні збої бірж, нестабільність мережі або неповні відповіді API. У таких випадках застосовуються методи заповнення пропусків, зокрема інтерполяція, копіювання попереднього значення або відкидання інтервалів за наявності критичних відхилень.

Після перевірки даних здійснюється обчислення допоміжних статистичних показників, які можуть бути корисними для подальшого аналізу. Наприклад, коефіцієнт волатильності може визначатися як відношення різниці між максимальним і мінімальним значеннями до ціни закриття. Такі показники можуть використовуватися як додаткові ознаки для моделей машинного навчання чи для візуального аналізу.

Окремим етапом є нормалізація даних. Оскільки моделі глибинного навчання чутливі до масштабів вхідних значень, дані приводяться до діапазону [0, 1] або [-1, 1] за допомогою методів мінімакс-нормалізації. Особливо важливо виконувати масштабування на навчальній вибірці та використовувати ті самі параметри для тестових та реальних даних, щоб уникнути витоку інформації.

Для формування послідовностей застосовується ковзне вікно. У межах даної роботи формується набір входів, де кожна послідовність містить WINDOW попередніх значень (наприклад, 48 годин), а вихід – це прогнозоване значення t+1. Такий підхід дозволяє моделі аналізувати останню динаміку ринку та використовувати її для формування короткострокового прогнозу.

Особливу увагу приділено структурі набору даних для моделі. Навчальна частина включає перші 80-90 % ряду, тоді як тестова частина містить останню його частину з невеликим перекриттям, що дозволяє забезпечити коректне формування перших послідовностей у тестовій вибірці.

Таким чином, попередня обробка часових рядів забезпечує якісну підготовку даних, формує коректну структуру послідовностей та створює основу для побудови стабільних моделей прогнозування криптовалют.

2.3.1 Очищення даних

Очищення даних є одним з ключових етапів підготовки часових рядів, оскільки навіть невеликі спотворення або пропуски можуть суттєво вплинути на стабільність та точність моделей машинного навчання. Ринкові дані, зокрема погодинні OHLCV-значення, можуть містити неточності, що виникають через відмови у роботі API, некоректні показники на біржах або розриви у часових інтервалах. Тому очищення є обов'язковою умовою формування якісного набору даних.

Першим кроком очищення є перевірка послідовності часових міток. Оскільки дані збираються з погодинною частотою, кожні дві сусідні точки мають відрізнятися рівно на одну годину. Якщо модуль виявляє відсутність окремих часових відрізків, система здійснює їх коректне заповнення або позначає такі інтервали як недостовірні. У випадках, коли пропуск невеликий і дані незначно впливають на загальну послідовність, застосовується інтерполяція або копіювання останнього коректного значення.

Наступним етапом є виявлення аномальних значень. Аномалії можуть проявлятися у вигляді різких стрибків ціни, нетипових обсягів або неконсистентних значень High і Low (наприклад, коли High менше за Open чи Close). Подібні значення можуть призводити до помилкових висновків під час навчання моделі. Тому алгоритм очищення автоматично перевіряє логічну відповідність параметрів OHLCV та відкидає записи, що виходять за межі допустимих відхилень.

Важливим аспектом очищення є обробка значень з нульовим об'ємом торгів. У деякі періоди на окремих біржах можуть не здійснюватися операції або API може повернути нульові значення для Volume. Якщо такі значення є системними, вони

впливають на статистичні характеристики ринку. Модуль очищення перевіряє ці випадки та визначає, чи є це справжнім станом ринку, чи помилкою вибірки.

Крім того, у процесі очищення виконується приведення типів даних до єдиного формату. Числові значення конвертуються у стандартні типи із фіксованою точністю, що дозволяє уникнути помилок при подальшому масштабуванні та обробці даних. Також перевіряється наявність дублікатів, які можуть з’являтися при повторних запитах до API або у разі нестабільності мережі.

Завдяки виконанню очищення даних забезпечується формування цілісного, логічно узгодженого та репрезентативного набору погодинних значень. Це дозволяє зменшити шум у даних, запобігти виникненню артефактів під час навчання та підвищити точність передбачень моделей, які працюють на основі цих часових рядів.

2.3.2 Масштабування та формування ознак

Масштабування та формування ознак є одним з найважливіших етапів у підготовці даних для моделей глибинного навчання. Моделі типу LSTM, GRU та їх комбінації чутливі до діапазонів вхідних значень, тому правильне масштабування дозволяє прискорити навчання, підвищити стабільність градієнтів та забезпечити коректну роботу мереж.

Першим кроком є застосування мінімакс-нормалізації, яка перетворює вихідні значення до діапазону [0, 1] або [-1, 1]. У межах цієї роботи використовується класичний MinMaxScaler, який розраховується виключно на навчальній вибірці. Це дозволяє уникнути витоку інформації та гарантує, що тестові дані та дані реального часу обробляються за тими самим параметрами масштабування. Залежність між цінами може бути нелінійною, тому приведення до єдиної шкали підвищує точність і стабільність прогнозування для більшості глибинних моделей.

Після масштабування здійснюється формування ознак (feature engineering). Попри те, що глибинні нейронні мережі здатні самостійно виділяти важливі патерни в часовому ряді, формування додаткових ознак може покращити якість прогнозу. До

таких ознак можуть належати ковзаючі середні, показники волатильності, різниця між ціною закриття та цінами максимуму або мінімуму, а також індикатори імпульсу. Ці ознаки дозволяють моделі швидше знаходити ключові взаємозв'язки у даних.

Однією з ключових операцій є формування послідовностей за допомогою ковзного вікна. Моделі типу LSTM та GRU прогнозують значення на основі попередніх станів, тому побудова послідовностей із довжиною WINDOW (наприклад, 48 годин) дозволяє мережі враховувати динаміку цін та зміни трендів за попередній період. Вихідною змінною є значення t+1 – прогноз ціни закриття наступної години.

Важливо зазначити, що під час формування послідовностей необхідно зберігати хронологічний порядок даних та уникати випадкового перемішування. Послідовності формуються таким чином, щоб кожне вікно містило часові значення, які впливали один на одного у реальному ринку. Це забезпечує відповідність моделі реальній поведінці даних та покращує здатність мережі передбачати майбутні значення.

Завершальним етапом є поділ даних на навчальну та тестову вибірки. Тренувальна частина охоплює основний обсяг історичних даних, тоді як тестова – останні значення з частковим перекриттям, що дозволяє моделі формувати коректні стартові послідовності. Такий підхід забезпечує адекватну оцінку здатності моделі прогнозувати значення на нових, ще не бачених даних.

Таким чином, масштабування та формування ознак створюють оптимальні умови для роботи моделей глибинного навчання, забезпечують правильну побудову послідовностей та дозволяють нейронним мережам максимально ефективно використовувати інформацію, що міститься у часових рядах.

2.4 Формулювання задачі прогнозування та параметри дослідження

Задача прогнозування вартості криптовалюти передбачає визначення майбутнього значення ціни на основі історичних даних. У межах цієї роботи здійснюється прогноз ціни закриття Bitcoin (BTC-USD) на один часовий крок уперед

– так званий прогноз t+1. Такий тип прогнозування є ключовим у короткострокових торговельних стратегіях та дозволяє моделі реагувати на зміну ринкової динаміки з високою частотою.

Задача формулюється як задача регресії, де вихідною змінною є значення ціни закриття наступного інтервалу. Вхід моделі складається з послідовності довжини WINDOW, яка містить історичні значення ціни та, за необхідності, додаткові ознаки. Метою моделі є мінімізація різниці між передбаченим значенням та фактичним результатом, що оцінюється за допомогою стандартних метрик, таких як MAE (Mean Absolute Error) та RMSE (Root Mean Squared Error).

Основними параметрами дослідження є:

WINDOW – довжина вхідної послідовності. У межах системи обрано значення 48 годин, оскільки воно дозволяє моделі врахувати короткострокову динаміку, не перенавантажуючи її зайвою інформацією.

HORIZON – горизонт прогнозування. У цій роботі він дорівнює одному часовому кроку (t+1), що відповідає практичним потребам короткострокового прогнозування.

TRAIN_RATIO – співвідношення між навчальною і тестовою вибірками. Для забезпечення надійності моделі використовується співвідношення 80-90 % для навчання та 10-20 % для тестування, що дозволяє виділити достатню кількість даних для оцінки якості прогнозу.

SCALING – метод нормалізації даних. Використовується мінімакс-нормалізація, яка забезпечує однаковий масштаб для всіх параметрів та стабільність навчання.

MODEL_TYPE – тип архітектури моделі. У межах дослідження порівнюються LSTM, GRU, CNN-LSTM та LSTM із механізмом уваги. Це дозволяє визначити, який із підходів найкраще працює на погодинних криптовалютних рядах.

BATCH_SIZE та EPOCHS – параметри навчання, що регулюють кількість даних у одному пакеті та кількість проходів моделі через навчальну вибірку. Вибір цих параметрів впливає на збіжність моделі та стабільність навчання.

DATA_FREQUENCY – частота оновлення даних. У цій системі використовується погодинний інтервал, що дозволяє поєднати точність прогнозування зі швидкістю оновлення ринку.

Таким чином, задача прогнозування формулюється як регресійна задача з одночасним урахуванням довготривалих та короткострокових залежностей. Вибір параметрів дослідження відображає необхідність побудови точних, стабільних та адаптивних моделей, здатних працювати з високоволатильними даними криптовалютного ринку.

Висновки до розділу

У даному розділі було розглянуто інформаційне забезпечення інтелектуальної системи прогнозування вартості криптовалют, зокрема структуру вхідних даних, джерела їх отримання та особливості обробки. Встановлено, що погодинні OHLCV-дані є оптимальним форматом для моделювання короткострокової динаміки Bitcoin, оскільки вони відображають як локальні коливання, так і загальні ринкові тенденції.

Проаналізовано можливості CoinGecko API як відкритого та надійного джерела ринкової інформації. Його використання дозволяє забезпечити регулярне оновлення даних та автоматизувати процес збору інформації без додаткових витрат. Внутрішня інфраструктура системи, побудована на базі DuckDB та формату Parquet, забезпечує ефективне зберігання великих обсягів даних, швидке виконання запитів та гнучкість у подальшій аналітичній обробці.

Було детально розглянуто процес попередньої обробки даних, який включає очищення часових рядів від пропусків та аномалій, перевірку коректності часових інтервалів, нормалізацію вхідних значень та формування послідовностей за допомогою ковзного вікна. Такий підхід дозволяє створити якісний і структурований набір даних, придатний для навчання моделей глибинного навчання.

Сформульовано задачу прогнозування ціни Bitcoin як задачу регресії на один часовий крок уперед (t+1), визначено ключові параметри дослідження, включно з

довжиною вхідного вікна, часткою навчальної вибірки та методами масштабування. Обґрунтовано вибір цих параметрів з огляду на високу волатильність криптовалютного ринку та потребу у точних короткострокових прогнозах.

Таким чином, у цьому розділі сформовано надійне інформаційне підґрунтя для побудови моделі прогнозування. Результати забезпечують основу для подальшого математичного моделювання, вибору архітектури нейронних мереж та проведення експериментальних досліджень, що розглядаються у наступних розділах.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Основні поняття та моделі часових рядів

Часовий ряд – це впорядкована послідовність значень, які фіксуються у певні часові моменти. У контексті фінансових ринків часові ряди містять інформацію про динаміку цін, обсяг торгів та інші показники, що дозволяє аналізувати поведінку активів у часі. Основною особливістю часових рядів є залежність поточного значення від попередніх, що відрізняє їх від незалежних вибірок у класичних статистичних задачах.

Ключовими властивостями часових рядів є тренд, сезонність, циклічність і шумові компоненти. Тренд відображає довгострокову тенденцію зміни значень, сезонність – періодичні коливання, притаманні певним часовим інтервалам, тоді як випадковий шум відповідає за нерегулярні коливання, зумовлені стохастичною природою ринку.

У задачах прогнозування важливу роль відіграє стаціонарність – властивість ряду, при якій його статистичні характеристики залишаються сталими в часі. Багато класичних моделей вимагають стаціонарності, тоді як сучасні глибинні нейронні мережі здатні працювати й з нестабільними, сильно волатильними рядами.

Оцінювання якості прогнозів здійснюється за допомогою метрик MAE (Mean Absolute Error) і RMSE (Root Mean Squared Error). Обидві метрики використовуються у цій роботі для оцінки точності моделей:

Формула середньої абсолютної похибки:

$$MAE = (1/n) * \Sigma |y_i - \hat{y}_i| \quad (3.1)$$

Формула середньоквадратичної похибки:

$$RMSE = sqrt((1/n) * \Sigma (y_i - \hat{y}_i)^2) \quad (3.2)$$

y_i – фактичне значення,

$\hat{y}_i$ – прогноз моделі,

n – кількість спостережень.

У фінансовому прогнозуванні RMSE більш чутливий до великих помилок, тоді як MAE стабільніший при роботі зі значною волатильністю. Саме тому оцінювання моделі за двома метриками дозволяє отримати збалансовану картину її ефективності.

Таким чином, робота з часовими рядами потребує розуміння їхніх особливостей, властивостей та специфіки обробки, що формує основу для побудови ефективних моделей прогнозування, які розглядаються у наступних підрозділах.

3.1.1 Часові ряди та сезонність

Часовий ряд у фінансовому аналізі являє собою послідовність вимірювань, упорядкованих у часі. У випадку криптовалют це, як правило, ціни закриття, обсяги торгів або інші ринкові показники, які змінюються з кожною годиною або хвилиною. На відміну від звичайних статистичних даних, часові ряди містять інформацію про залежність поточних значень від попередніх, що робить їх об’єктом спеціалізованих методів аналізу.

Типова структура часового ряду складається з кількох компонентів:

тренду, сезонності, циклічності та випадкового шуму. Декомпозиція дозволяє розділити ряд на окремі складові з метою їх кращого розуміння та подальшої побудови прогнозної моделі.

Загальна адитивна модель декомпозиції часового ряду має вигляд:

$$Y(t) = T(t) + S(t) + C(t) + \varepsilon(t) \quad (3.3)$$

$Y(t)$ – значення ряду у момент часу t

$T(t)$ – трендова складова

$S(t)$ – сезонність

$C(t)$ – циклічна компонента

$\varepsilon(t)$ – випадковий шум

У фінансових даних тренд може відображати довгострокове зростання або спад ринку. Циклічність проявляється у великих періодичних коливаннях, зазвичай пов’язаних з економічними циклами або ринковими фазами. Випадковий шум

відображає короткострокові непередбачувані зміни, пов’язані з новинним фоном, ринковими подіями, поведінкою великих інвесторів та іншими факторами.

Сезонність у класичному розумінні рідко проявляється у криптовалютах через відсутність прив’язки до робочого часу бірж, але певні закономірності можуть виникати, наприклад, підвищення активності в окремі години доби або на початку торгових тижнів. Такі шаблони дозволяють моделі краще розуміти структуру даних і більш точно прогнозувати майбутні значення.

Для криптовалютних часових рядів характерною є висока волатильність та відсутність стабільної сезонності, що ускладнює використання класичних методів декомпозиції. Саме тому глибинні нейронні мережі, здатні автоматично визначати залежності й тренди без явного декомпозування ряду, є більш ефективними у таких задачах.

Таким чином, розуміння структури часових рядів та наявність їхніх компонентів формує основу для подальшої побудови прогнозних моделей та дозволяє оцінити, наскільки складним є процес аналізу криптовалютних даних.

3.1.2 Типи моделей прогнозування часових рядів

Моделі прогнозування часових рядів поділяються на декілька основних груп, кожна з яких має свої особливості, переваги та сфери застосування. Вибір конкретного підходу залежить від властивостей даних, рівня їх волатильності та типу залежностей, які необхідно змоделювати.

Першу групу становлять класичні статистичні моделі, зокрема AR (авторегресійні), MA (моделі ковзного середнього) та ARIMA, які широко використовуються у традиційному фінансовому аналізі. У загальному вигляді авторегресійна модель AR(p) описується формулою:

$$Y(t) = c + \varphi_1 Y(t-1) + \varphi_2 Y(t-2) + \ldots + \varphi_p Y(t-p) + \varepsilon(t) \qquad (3.4)$$

c – константа,

$\varphi_1 \ldots \varphi_p$ – коефіцієнти моделі,

ε(t) – випадкова похибка.

Такі моделі добре працюють у випадках, коли часовий ряд є стаціонарним та має виражену автокореляцію. Однак вони менш ефективні у середовищах з різкими стрибками, що характерно для криптовалют.

Другу групу складають методи машинного навчання, такі як випадкові ліси, градієнтний бустинг і методи підтримувальних векторів. Вони дозволяють моделювати нелінійні залежності та працювати з великою кількістю ознак, проте не враховують часовий порядок без попереднього формування спеціальних послідовностей і лагових ознак.

Найбільш актуальними для прогнозування фінансових рядів є глибинні нейронні мережі, які становлять третю групу моделей. До них належать рекурентні мережі (RNN), LSTM, GRU, CNN-LSTM, а також архітектури з механізмами уваги (Attention). Їх головною перевагою є здатність враховувати довготривалі та короткострокові залежності, не вимагаючи ручного створення ознак. Завдяки цьому такі моделі демонструють високу точність на даних із нестабільною динамікою.

Окрему категорію становлять комбіновані моделі, що поєднують можливості різних архітектур, наприклад CNN для виділення локальних патернів та LSTM для аналізу часової послідовності. Такі моделі є особливо ефективними при роботі з шумними фінансовими даними.

Таким чином, моделі прогнозування часових рядів становлять широкий спектр підходів – від простих статистичних методів до складних нейронних мереж. Завданням дослідника є вибір моделі, яка найкраще відображає властивості ряду та забезпечує необхідну точність у контексті конкретної задачі.

3.2 Нейронні мережі для прогнозування

Нейронні мережі є одним із найбільш ефективних інструментів для моделювання часових рядів, особливо коли йдеться про дані з високою волатильністю та складними нелінійними залежностями. На відміну від класичних статистичних моделей та традиційних методів машинного навчання, нейронні

мережі здатні автоматично виділяти приховані закономірності у послідовностях та враховувати вплив попередніх станів на майбутні значення.

Основна перевага нейронних мереж у задачах прогнозування полягає в їхній здатності працювати із залежностями різної тривалості. Криптовалютні ринки характеризуються нерегулярними коливаннями, різкими змінами трендів та високою чутливістю до глобальних і локальних новин. Такі особливості вимагають застосування моделей, здатних обробляти як короткострокові, так і довготривалі патерни, що традиційним методам часто недоступно.

У межах цієї роботи розглядаються чотири основні архітектури нейронних мереж:

- LSTM (Long Short-Term Memory) – модель, здатна зберігати інформацію на довгі проміжки часу завдяки спеціальним елементам пам’яті;
- GRU (Gated Recurrent Unit) – спрощена рекурентна модель, яка зберігає більшість переваг LSTM, але є більш компактною та швидкою при навчанні;
- CNN-LSTM – комбінована модель, що поєднує здатність згорткових шарів виділяти локальні патерни з рекурентною обробкою послідовностей;
- моделі з механізмом уваги (Attention) – сучасний підхід, що дозволяє моделі самостійно визначати, які частини послідовності є найбільш важливими для прогнозу.

Усі ці архітектури працюють із часовими послідовностями, однак кожна має різні механізми обробки інформації та по-різному реагує на шум та структурні зміни ринку. Включення декількох моделей у дослідження дозволяє отримати об’єктивне порівняння їхньої продуктивності та обрати оптимальний підхід для прогнозування ціни Bitcoin на один часовий крок уперед.

Таким чином, нейронні мережі становлять основу математичного апарату, використаного у даній роботі, та відкривають можливість побудови високоточних прогнозних систем для фінансових часових рядів.

3.2.1 Архітектура LSTM

LSTM (Long Short-Term Memory) – це різновид рекурентних нейронних мереж, спеціально розроблений для ефективного моделювання довготривалих залежностей у часових рядах. На відміну від класичних RNN, які страждають від проблеми зникнення або вибуху градієнтів, LSTM використовує механізм керування інформаційними потоками через систему «гейтів» (воріт). Це дає можливість мережі зберігати або забувати інформацію залежно від її важливості для прогнозування майбутніх значень.

Основним елементом LSTM є комірка пам'яті, яка зберігає стан на різних часових кроках. Для керування даними використовуються три типи воріт:

• forget gate – визначає, яку частину попереднього стану потрібно забути

• input gate – контролює, яку нову інформацію слід записати

• output gate – визначає, яку частину стану передати на вихід

Математичний опис роботи LSTM має вигляд:

Forget gate:

$$f_t = \sigma(Wf \cdot [h_{t-1}, x_t] + bf) \tag{3.5}$$

Оновлений стан пам'яті:

$$C_t = f_t * C_{t-1} + i_t * \overline{C}_t \tag{3.6}$$

Вихідний стан:

$$\overline{C}_t = tanh(Wc \cdot [h_{t-1}, x_t] + bc) \tag{3.7}$$

x_t – вхідні дані у момент часу t

h_t – прихований стан

C_t – стан пам'яті

σ – сигмоїдна функція активації

У задачах прогнозування волатильних фінансових часових рядів LSTM зазвичай демонструє стабільну поведінку завдяки здатності зберігати інформацію на довгих інтервалах та пом'якшувати вплив короткострокових коливань. Механізм воріт дозволяє моделі гнучко контролювати потік інформації, що робить LSTM

стійкою до шуму та різких змін тренду. Проте ефективність такої моделі значною мірою залежить від правильного вибору гіперпараметрів і достатнього обсягу даних, оскільки надмірно складні конфігурації можуть призводити до перенавчання на високоволатильних ринках.

3.2.2 Архітектура GRU

GRU (Gated Recurrent Unit) – це спрощений різновид рекурентних нейронних мереж, який був розроблений як альтернатива LSTM з меншою кількістю параметрів та швидшим навчанням. Попри спрощену структуру, GRU демонструє порівнянну точність з LSTM у багатьох задачах прогнозування часових рядів, зокрема на фінансових даних.

На відміну від LSTM, GRU не має окремого стану пам'яті C_t. Вся інформація зберігається у прихованому стані h_t, що робить модель компактнішою та менш обчислювально затратною. Управління інформаційними потоками здійснюється двома воротами:

- update gate (z_t) – визначає, яку частину попереднього стану зберегти
- reset gate (r_t) – визначає, яку частину інформації з минулого не брати до уваги

Робота GRU описується наступними формулами:

Update gate:

$$z_t = \sigma(Wz \cdot [h_{t-1}, x_t] + b\_z) \tag{3.8}$$

Reset gate:

$$r_t = \sigma(Wr \cdot [h_{t-1}, x_t] + b\_r) \tag{3.9}$$

Кандидат на новий стан:

$$\bar{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t] + b) \tag{3.10}$$

Оновлений прихований стан:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \bar{h}_t \tag{3.11}$$

x_t – вхід у момент часу t

h_t – прихований стан

σ – сигмоїдна функція активації

$tanh$ – гіперболічний тангенс

GRU, маючи спрощену структуру порівняно з LSTM, добре підходить для роботи з волатильними рядами завдяки своїй компактності та меншій кількості параметрів. Вона швидше навчається, менш схильна до перенавчання і може ефективно моделювати середньо- та довгострокові залежності. GRU часто показує стабільну роботу на даних із вираженим шумом, оскільки механізм оновлення стану дозволяє моделі швидше адаптуватися до змін динаміки ринку. Водночас спрощення архітектури іноді може обмежувати її здатність до моделювання дуже складних або структурно різноманітних патернів.

3.2.3 Комбінована модель CNN-LSTM

Комбінована модель CNN-LSTM поєднує можливості згорткових нейронних мереж (CNN) і рекурентних мереж типу LSTM, що дозволяє одночасно виділяти локальні патерни в часових рядах та моделювати довгострокові залежності.

Основна ідея полягає у використанні згорткових шарів для первинної обробки послідовності, після чого результати передаються у рекурентний блок LSTM. Завдяки цьому модель спочатку виявляє локальні структури – різкі зміни, мікро-тренди, короткі імпульси – а потім аналізує їхню динаміку в часі.

Архітектура CNN-LSTM зазвичай включає такі елементи:

- 1D-згортковий шар (Conv1D) для виділення локальних шаблонів у часовій послідовності
- Шар підвибірки (Pooling) для зменшення розмірності та усунення шуму
- LSTM-шар, який опрацьовує отриману послідовність ознак та моделює довготривалі залежності

Операція одновимірної згортки над часовим рядом x□ у CNN-шарі може бути записана як:

$$y_t = \sum_{k=0}^{K-1} w_k \cdot x_{(t+k)} + b \qquad (3.12)$$

y_t – вихід згорткового фільтра у момент часу t

$x_{(t+k)}$ – елемент вхідної послідовності

w_k – ваги фільтра згортки

b – зміщення

k – розмір ядра згортки

Саме наявність згорткового шару, який навчається виділяти локальні патерни у часі, є ключовою відмінністю CNN-LSTM від чистих рекурентних моделей.

У контексті прогнозування волатильних часових рядів модель CNN-LSTM має як сильні, так і слабкі сторони. Завдяки згортковим шарам вона краще виявляє короткострокові локальні патерни, такі як різкі коливання ціни або імпульсні зміни тренду. Проте поєднання CNN і LSTM підвищує складність моделі, що може призводити до більшої чутливості до шуму, необхідності точнішого налаштування гіперпараметрів та підвищених вимог до якості даних. Таким чином, CNN-LSTM здатна давати добрі результати за умов правильно підібраної архітектури, але її стабільність та узагальнювальна здатність значно залежать від властивостей вхідного ряду.

3.2.4 Механізм уваги (Attention)

Механізм уваги (Attention) є сучасним підходом у глибинному навчанні, який дозволяє моделі автоматично визначати, які частини послідовності мають найбільший вплив на прогноз. На відміну від класичних рекурентних мереж, де інформація передається послідовно, Attention дає змогу моделі безпосередньо «звертатися» до будь-якого попереднього стану, оцінюючи його важливість.

На кожному кроці прогнозування для поточного прихованого стану h_t обчислюються ваги важливості попередніх станів h_i. Коефіцієнт уваги для і-го елемента послідовності може бути записаний як:

$$\alpha_{(t,\,i)} = exp(h_t \cdot h_i) \,/\, \sum_j exp(h_t \cdot h_j) \tag{3.13}$$

$\alpha_{(t,\,i)}$ – вага важливості і-го кроку для прогнозу в момент t

h_t – поточний прихований стан

h_i – прихований стан у момент часу i

$h_t \cdot h_i$ – скалярний добуток (міра подібності)

Після цього контекстний вектор для моменту t обчислюється як зважена сума всіх попередніх станів:

$$context = \sum_j \alpha_{(t,\,i)} \cdot h_i \tag{3.14}$$

Саме така схема з обчисленням ваг $\alpha_{(t,\,i)}$ і побудовою $context$ є ключовою відмінністю моделей з Attention від класичних RNN, LSTM та GRU, де кожен крок опрацьовується послідовно без явного механізму вибору найважливіших частин історії.

Для волатильних фінансових часових рядів механізм уваги має потенційну перевагу: він може автоматично виділяти окремі важливі моменти в історії, не покладаючись на послідовну обробку даних. Це робить Attention придатним для випадків, коли ключові патерни розташовані на значній відстані один від одного. Водночас моделі з увагою зазвичай є складнішими, вимагають більше обчислювальних ресурсів та потребують великих обсягів якісних даних для стабільного навчання. На сильно шумних або коротких рядах такі моделі можуть демонструвати нестійку поведінку, оскільки алгоритм уваги не завжди здатен виділити справді релевантні часові фрагменти.

3.3 Оптимізація та вибір гіперпараметрів

Процес навчання моделей глибинного навчання істотно залежить від правильного вибору гіперпараметрів, які впливають на швидкість збіжності, здатність моделі узагальнювати інформацію та якість підсумкового прогнозу. На відміну від параметрів, що визначаються під час навчання моделі, гіперпараметри задаються дослідником вручну і потребують підбору за допомогою експериментального аналізу.

До ключових гіперпараметрів моделей LSTM, GRU та інших мереж для часових рядів належать:

- кількість нейронів у прихованих шарах – визначає здатність моделі захоплювати складні залежності;
- кількість шарів – збільшує виразну потужність моделі, але може підвищувати ризик перенавчання;
- довжина вхідної послідовності (window size) – визначає, яку кількість попередніх значень модель використовує для прогнозування;
- розмір мінібатча – впливає на стабільність оновлення ваг і швидкість тренування;
- коефіцієнт навчання (learning rate) – контролює величину кроку під час оптимізації;
- кількість епох – визначає, скільки разів модель проходить через тренувальний набір.

Задача вибору гіперпараметрів пов’язана з необхідністю досягти балансу між здатністю моделі точно апроксимувати дані та її стійкістю до перенавчання. Типи гіперпараметрів, такі як кількість шарів або розмір прихованого стану, значною мірою визначають складність моделі, тоді як значення learning rate або batch size впливають на динаміку оптимізації.

Для покращення якості навчання використовується метод ранньої зупинки (early stopping), коли тренування припиняється у разі відсутності покращення на валідаційній вибірці після певної кількості епох. Це дозволяє запобігти перенавчанню та зберегти параметри моделі, що забезпечують найкращий результат.

Ще одним важливим аспектом є вибір оптимізатора. Найчастіше застосовуються алгоритми Adam, RMSProp та SGD, які визначають спосіб оновлення ваг моделі. Вони відрізняються швидкістю збіжності та чутливістю до форми функції втрат, що робить їх вибір важливою частиною процесу оптимізації.

Залежно від характеристик часового ряду, зокрема його волатильності, рівня шуму та тривалості доступної історії, вибір гіперпараметрів може істотно змінювати поведінку моделі. Тому процес оптимізації зазвичай передбачає серію експериментів, у межах яких дослідник порівнює різні конфігурації моделі, аналізує

метрики помилки та обирає параметри, які забезпечують найкращий баланс між точністю та стабільністю прогнозування.

3.4 Порівняльний аналіз моделей

Порівняльний аналіз моделей є важливою частиною процесу дослідження, оскільки дозволяє оцінити сильні та слабкі сторони різних архітектур глибинного навчання у задачі прогнозування волатильних часових рядів. У межах даного розділу наведено узагальнення результатів роботи чотирьох моделей: LSTM, GRU, CNN-LSTM та моделі з механізмом уваги (Attention). Для їх оцінювання використовувалися метрики середньої абсолютної помилки (MAE) та середньоквадратичної помилки (RMSE), а також візуальний аналіз прогнозів.

Оскільки фінансові часові ряди характеризуються високою волатильністю, сильним шумом та часто різкими змінами напрямку, моделі демонструють різну чутливість до таких властивостей. Тому порівняння проводиться не лише за значеннями метрик, але й за поведінкою прогнозів у характерних умовах: під час різких стрибків, відкатів, короткострокових циклів та змін тренду.

Нижче наведено узагальнюючу таблицю, у яку будуть внесені результати обчислень після експериментального етапу:

Таблиця 3.1 – Порівняння моделей за метриками MAE та RMSE

Модель	MAE	RMSE
LSTM	554,934	718,110
GRU	434,272	572,263
CNN-LSTM	604,987	779,384
Attention	1017,675	1227,291

Для більш глибокого аналізу доцільно доповнити таблицю візуальними графіками прогнозів кожної моделі. На таких графіках добре видно здатність моделі

повторювати тенденції ряду, її реакцію на різкі зміни та якість наближення до реальних значень.

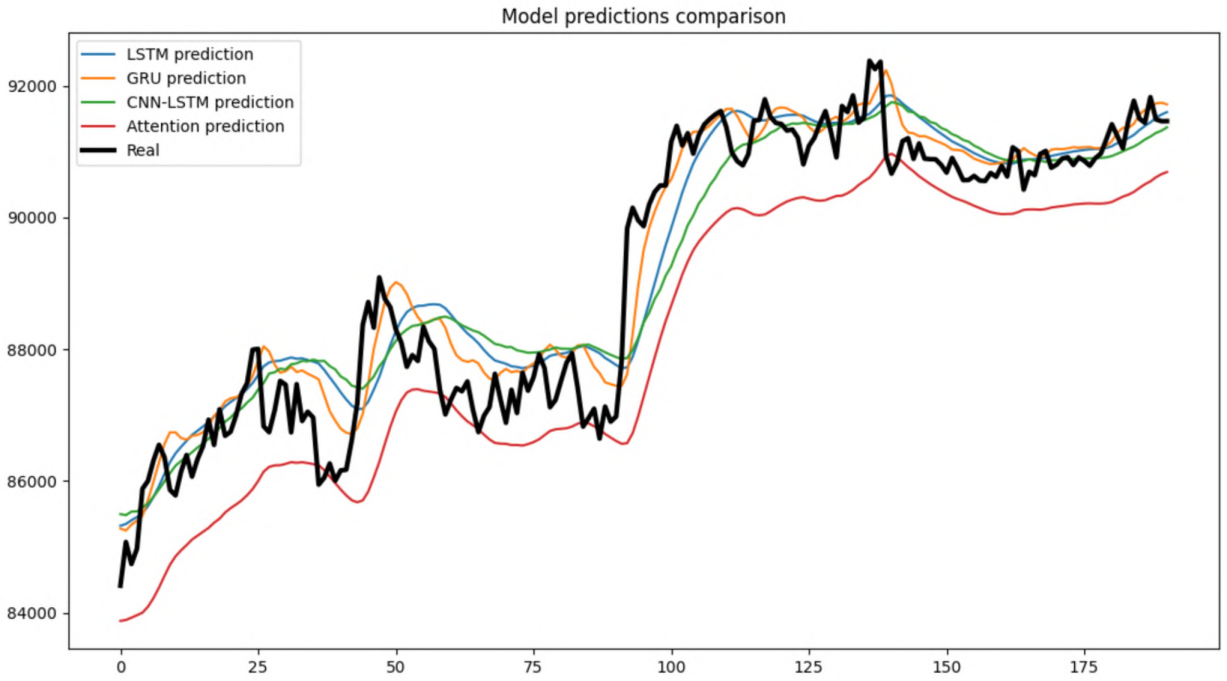


Рисунок 3.1 – Порівняння передбачень моделей LSTM, GRU, CNN-LSTM та Attention на погодинних даних BTC-USD

Аналізуючи графік на рисунку 3.1 можна зробити наступні висновки:

- LSTM і GRU показують найкращу адаптацію до волатильності
- CNN-LSTM дає непоганий результат, але може «перегладжувати» і помилятися
- Attention вловлює тренд, але не працює добре на коротких рядах

Узагальнюючи поведінку розглянутих моделей, можна зазначити, що кожна з них має власні особливості у роботі з волатильними фінансовими часовими рядами.

Модель LSTM демонструє збалансовану точність та стійкість завдяки здатності накопичувати інформацію про попередні стани та контролювати її вплив через механізм воріт. Це дозволяє їй ефективніше реагувати на різкі коливання та уникати надмірного згладжування прогнозів.

Модель GRU, яка має спрощену архітектуру, зазвичай показує близькі результати до LSTM, але при цьому навчається швидше та менше схильна до перенавчання. Завдяки компактній структурі GRU добре адаптується до шумних

даних і здатна забезпечувати стабільний прогноз навіть за обмеженого обсягу історичних спостережень.

Комбінована модель CNN-LSTM здатна виявляти локальні короткострокові патерни, проте ця ж властивість робить її чутливою до ринкового шуму. У фазах високої волатильності згортковий шар іноді підсилює випадкові цінові коливання, що може призводити до нестійких або викривлених прогнозів, особливо за короткого тренувального періоду.

Моделі з механізмом Attention мають потенціал ефективно виділяти важливі часові моменти, проте для цього вони потребують значно більшої кількості навчальних даних, ніж доступно у коротких вибірках. За умов високого рівня шуму та обмеженої довжини ряду Attention-моделі можуть формувати надмірно згладжені або неточні прогнози, оскільки їм складно визначити стійкі релевантні залежності.

Загалом результати порівняння свідчать, що розглянуті архітектури по-різному реагують на характер фінансових часових рядів. Тому вибір моделі має ґрунтуватися не лише на значеннях метрик точності, але й на здатності алгоритму забезпечувати стабільні, інтерпретовані та узгоджені з реальною динамікою ринку прогнози.

Висновки до розділу

У цьому розділі було розглянуто математичні засади, які лежать в основі моделювання та прогнозування фінансових часових рядів. Описано ключові властивості часових послідовностей, зокрема тренд, сезонність та шум, що є визначальними факторами при виборі методів моделювання. Зазначено, що волатильність криптовалютних ринків суттєво ускладнює процес прогнозування, оскільки моделі повинні адаптуватися до різких змін динаміки.

У межах розділу було проаналізовано чотири архітектури глибинного навчання: LSTM, GRU, CNN-LSTM та модель з механізмом уваги (Attention). Для кожної з них розглянуто принцип роботи, основні компоненти та особливості обчислень. Показано, що кожна архітектура має свої переваги та обмеження у задачах прогнозування волатильних фінансових рядів: рекурентні мережі добре

моделюють послідовності, CNN-компоненти виділяють локальні патерни, а Attention дозволяє зосереджуватися на найбільш важливих часових моментах.

Також було описано загальні принципи оптимізації та вибору гіперпараметрів, включно з вибором розміру вікна спостережень, кількості нейронів, оптимізатора та механізмів запобігання перенавчанню. Наголошено, що якість моделей значною мірою залежить від належного налаштування цих параметрів та властивостей вхідних даних.

Порівняльний аналіз моделей дозволив визначити загальні тенденції їх поведінки на волатильних часових рядах, а також виділити ключові фактори, що впливають на точність та стабільність прогнозів. Отримані результати та спостереження формують основу для вибору моделей, які будуть інтегровані у програмну систему, що описується в наступному розділі.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Огляд інструментарію та технологій

Програмна система для аналізу та прогнозування криптовалютного ринку була реалізована з використанням сучасного інструментарію, що поєднує засоби глибинного навчання, автоматизованого збору даних, локальної аналітики та вебінтерфейсу. У цьому підрозділі наведено загальний огляд технологій, що були використані під час розробки, а також їх роль у побудові програмного комплексу.

Основною мовою розробки є Python, який завдяки своїй простоті та розвиненій екосистемі бібліотек став стандартом у сфері машинного навчання та фінансової аналітики. Для роботи з табличними структурами та числовими даними використовувалися бібліотеки Pandas та NumPy, що забезпечують широкий спектр можливостей для обробки часових рядів.

Для побудови моделей глибинного навчання застосовано фреймворк PyTorch, який дозволяє гнучко конструювати нейронні мережі, реалізовувати власні архітектури та виконувати обчислення на графічному процесорі [17]. На основі PyTorch реалізовано моделі LSTM та GRU, що використовуються системою для прогнозування ціни Bitcoin.

Отримання ринкових даних здійснюється через CoinGecko API, який надає доступ до історичної та поточної інформації про ціни криптовалют. API дозволяє завантажувати OHLCV-структури (Open, High, Low, Close, Volume), що є стандартом у фінансових часових рядах [16].

Збереження та аналітична обробка даних виконуються у локальній базі DuckDB, яка поєднує швидкість колонкового формату та зручність інтеграції з Python. Використання DuckDB дає можливість працювати з великими обсягами часових рядів локально, без необхідності розгортання складних серверних рішень [18].

Для побудови вебінтерфейсу використано фреймворк Streamlit, який дозволяє створювати інтерактивні аналітичні панелі без необхідності реалізації розділеної архітектури «клієнт-сервер». Завдяки Streamlit система забезпечує миттєве оновлення графіків, інтерактивність та зручну взаємодію з користувачем. Візуалізація даних у вебінтерфейсі реалізована за допомогою бібліотеки Plotly, яка підтримує масштабування, наведення, інтерактивне дослідження графіків і чудово підходить для фінансових даних.

Використання зазначених інструментів дозволило створити комплексну систему, що автоматично збирає дані, обробляє їх, формує прогнози та забезпечує користувачеві зручний доступ до результатів аналізу. Обраний стек технологій забезпечує як ефективність обчислень, так і простоту підтримки та можливість подальшого масштабування проєкту.

4.2 Архітектура розробленої системи

Архітектура розробленої системи побудована за модульним принципом та включає чотири основні підсистеми: модуль збору даних, модуль обробки та збереження інформації, модуль прогнозування та модуль вебінтерфейсу. Така структуризація забезпечує можливість незалежного розвитку компонентів, зручність тестування, а також гнучкість при модифікації або розширенні функціональності.

У контексті задачі прогнозування фінансових часових рядів система має працювати у режимі, максимально наближеному до реального часу, тому архітектура орієнтована на автоматизацію оновлення даних і формування прогнозу за фіксованою часовою частотою. Кожен з модулів виконує чітко визначену роль, а взаємодія між ними реалізована у вигляді послідовного пайплайна, який гарантує узгодженість даних на всіх етапах роботи.

Загальна логіка системи складається з таких кроків:

1. Збір ринкових даних з API CoinGecko, включно з історичними та актуальними OHLCV-значеннями.

2. Збереження отриманих даних у базі DuckDB, що забезпечує швидкий доступ, фільтрацію та подальшу аналітику.
3. Формування технічних ознак (features), необхідних моделі для прогнозування.
4. Генерація прогнозу t+1 за допомогою моделей LSTM або GRU.
5. Запис прогнозів у базу даних та створення історії попередніх передбачень.
6. Відображення результатів у вебінтерфейсі Streamlit, який надає користувачу інтерактивні графіки та аналітичну інформацію.

Система побудована таким чином, щоб кожен компонент міг працювати окремо:

- модулі ingest та jobs можуть виконуватися за розкладом, навіть без запуску вебінтерфейсу;
- модулі models дозволяють тренувати моделі незалежно від інтерфейсу або бази даних;
- модулі ui формують інтерфейс на основі актуальних даних, без необхідності повторної обробки чи тренування моделей.

Це дозволяє масштабувати функціональність, підключати нові моделі, змінювати джерела даних або адаптувати інтерфейс без повного рефакторингу проєкту. Модульна архітектура також полегшує деплоймент: система може бути розгорнута як на локальному комп’ютері, так і на сервері з мінімальними модифікаціями.

4.2.1 Структура проєкту та модулі

Програмна система реалізована у вигляді модульного Python-проєкту з чітким поділом логіки на незалежні компоненти. Така організація спрощує підтримку, дозволяє легко розширювати функціональність, а також забезпечує прозорість взаємодії між підсистемами. На рисунку 4.1 наведено фактичну структуру каталогу проєкту:

Рисунок 4.1 – Структура проєкту наведена з репозиторію на GitHub [19]

Опис основних каталогів

config/ – Містить файли конфігурацій та налаштувань застосунку.

data/ – Тут зберігаються параметри середовища, налаштування моделей, шляхів до бази даних, історичні параметри завантаження даних та інші глобальні константи. Використання централізованої конфігурації спрощує підтримку системи та зміну робочих параметрів без модифікації коду.

Також даний каталог використовується для роботи з локальною базою даних DuckDB, що містить OHLCV-дані, історію прогнозів та службові таблиці.

У ньому реалізовано:

- створення та ініціалізацію структури бази;
- запис нових ринкових даних;
- зчитування історичних даних для моделювання;
- оновлення таблиці прогнозів;
- оптимізацію запитів до таблиць із великими часовими рядами.

DuckDB забезпечує швидку аналітику, компактність зберігання та повністю локальну роботу системи без додаткового серверного ПЗ.

features/ – Містить модулі для генерації технічних ознак. До них належать:

- ковзаючі середні;
- різницеві ряди;
- показники волатильності;
- нормалізовані зміни ціни;
- інші статистичні та технічні індикатори.

Структура цього каталогу дозволяє легко додавати нові фічі або модифікувати існуючі без впливу на інші підсистеми.

ingest/ - Каталог, відповідальний за збір ринкової інформації з API CoinGecko. Містить клієнт для HTTP-запитів та інструменти для:

- завантаження історичних OHLCV-даних,
- отримання поточних даних у режимі онлайн,
- перевірки коректності та цілісності відповідей API,
- перетворення timestamp у читаємий формат,
- підготовки даних до запису в DuckDB.

Цей модуль є ключовим елементом системи, який забезпечує її зв’язок із реальним ринком.

jobs/ – Містить скрипти автоматизації, які виконуються за розкладом або вручну:

- fetch_history – завантаження або оновлення історичних ринкових даних;
- run_forecast – генерування прогнозу на наступну годину;
- службові утиліти для інтеграції з планувальниками та тренування наших моделей.

Завдяки цьому компоненту система може працювати у напівавтоматичному чи повністю автоматичному режимі.

models/ – Каталог, що містить реалізації моделей прогнозування. Структура моделей стандартизована:

- config.py – параметри моделі (розмір вікна, кількість нейронів, навчальні параметри);
- model.py – архітектура нейронної мережі (LSTM або GRU);
- train.py – процес навчання моделі на історичних даних;
- inference.py – генерація прогнозу t+1 на основі останніх даних;
- artifacts/ – збережені ваги, скейлери та службові артефакти.

Такий поділ забезпечує повну ізоляцію процесів тренування та прогнозування та дозволяє незалежно оновлювати або переобучати моделі.

ui/ – Відповідає за вебінтерфейс системи, побудований на Streamlit.

У ньому реалізовано:

- завантаження даних з бази;
- оновлення графіків у реальному часі;
- вибір моделі для прогнозування;
- відображення прогнозу та історії попередніх передбачень;
- інтерактивні Plotly-графіки.

Каталог ui/ фактично формує видиму частину системи, забезпечуючи користувача інструментом аналітики.

Службові файли:

- .env.example – шаблон файлу налаштувань із ключами доступу та конфігурацією;
- requirements.txt – перелік залежностей Python;
- README.md – інструкція з розгортання та запуску системи;
- .gitignore – виключає службові файли з репозиторію.
- run_hourly_update.bat - погодинне оновлення даних та прогнозування наступного кроку.
- run_daily_model_retrain.bat - щоденне перетренування моделей

Таким чином, структура проєкту відображає модульну архітектуру розробленої системи та забезпечує логічне розділення функціональності на незалежні компоненти.

4.2.2 Пайплайн обробки даних та прогнозування

Пайплайн обробки даних у розробленій системі побудований таким чином, щоб забезпечити безперервну актуалізацію ринкової інформації, підготовку ознак, генерацію прогнозу та збереження результатів у базі даних. Архітектура пайплайна відповідає вимогам роботи з волатильними фінансовими рядами та побудована за принципом послідовного виконання окремих етапів (див. рис. 4.2), кожен з яких реалізований у відповідному модулі проєкту.

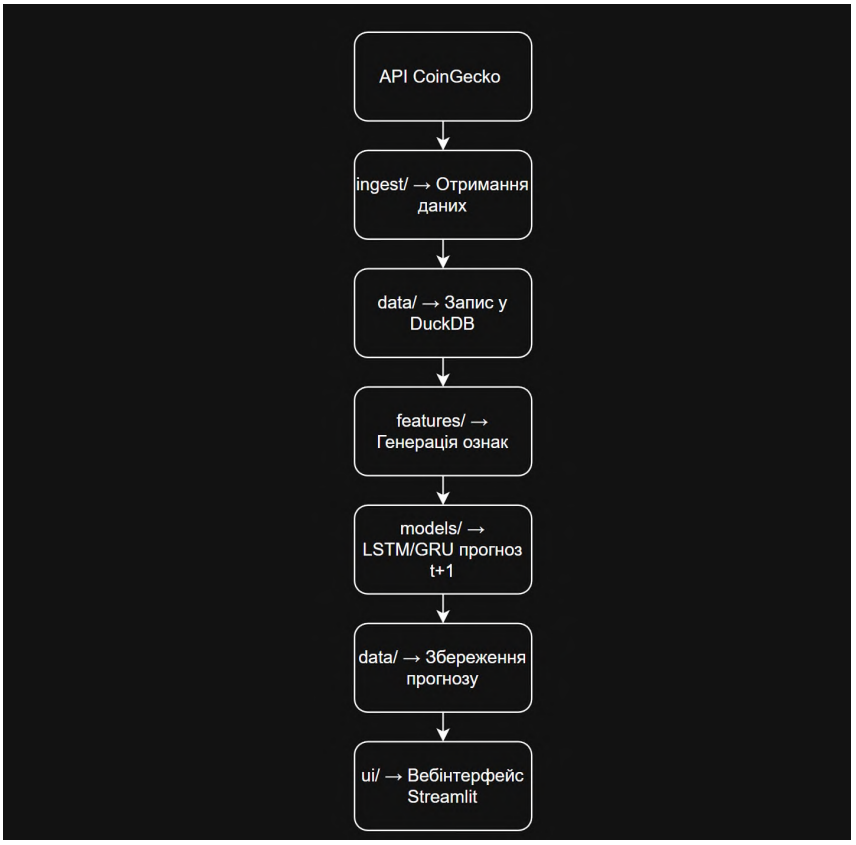


Рисунок 4.2 – Підсумкова схема пайплайна

Пайплайн включає такі ключові етапи:

Отримання даних з CoinGecko API

На першому етапі система завантажує нові ринкові дані через модуль ingest/:

- історичні OHLCV-значення (Open, High, Low, Close, Volume);
- поточні ринкові котирування для формування прогнозу t+1;
- коректне перетворення часових міток у формат UTC/локальний час.

Дані завантажуються через офіційний API, що забезпечує їх достовірність та узгодженість. Формат OHLCV відповідає вимогам фінансового аналізу та є придатним для подальшої обробки моделями.

Збереження та оновлення локальної бази DuckDB

Після отримання даних система виконує:

- перевірку відсутності дублювань;
- вставку нових записів у таблицю історичних цін;
- оновлення таблиці з прогнозами;
- створення резервних копій таблиць при потребі.

Використання DuckDB дозволяє зберігати великі обсяги історії та виконувати аналітичні SQL-запити локально з високою швидкістю. Це забезпечує можливість регулярної (погодинної) актуалізації без навантаження на систему.

Формування технічних ознак (features)

Перед подачею даних у модель виконується генерація ознак у каталозі features/.

Серед основних використовуваних фіч:

- ковзаючі середні та їх комбінації;
- відсоткові зміни ціни;
- показники локальної волатильності;
- нормалізовані дельти між сусідніми значеннями;
- формування фрейму фіч на основі заданого часового вікна (WINDOW).

Після генерації ознак дані нормалізуються за допомогою MinMaxScaler або іншого відповідного методу.

Підготовка вхідних тензорів для моделі

На цьому етапі формується послідовність розміром WINDOW × n_features, яка використовується моделями LSTM та GRU.

Підготовка включає:

- формування 3D-тензора (batch, seq_len, features);
- приведення даних до torch.FloatTensor;

- масштабування згідно зі збереженим скейлером;
- підбір порядку фіч для відповідності навченій моделі.

Генерація прогнозу моделями LSTM/GRU

У модулі models/.../inference.py виконується:

- завантаження моделі та нормалізатора зі збережених артефактів;
- інференс на останньому вікні історичних даних;
- отримання прогнозу t+1 у масштабованому вигляді;
- зворотне перетворення за допомогою scaler.inverse_transform;
- формування єдиного значення прогнозу.

Для забезпечення цілісності даних використовується функція ensure_forecast_continuity, яка перевіряє відповідність часових міток та запобігає пропускам або накладенням прогнозів.

Збереження прогнозу в DuckDB та створення історії передбачень

Після успішного інференсу прогноз:

- записується у таблицю forecast_hourly;
- зберігається з точною часовою міткою;
- використовується в інтерфейсі для побудови історичного ряду прогнозів;
- не перезаписується при оновленні моделі – всі старі прогнози зберігаються.

Це дозволяє відстежувати динаміку як ринку, так і поведінки моделей.

Відображення результатів у Streamlit

Останній етап – інтеграція з вебінтерфейсом у каталозі ui/:

- завантаження актуальної історії цін;
- підвантаження таблиці прогнозів;
- побудова інтерактивних Plotly-графіків;
- автоматичне оновлення сторінки через механізм st_autorefresh;
- візуалізація ручних або автоматичних запусків.

Інтерфейс працює в режимі реального часу, дозволяючи оцінювати:

- актуальний стан ринку,
- точність прогнозів,

- поведінку моделей у динаміці.

4.2.3 Організація бази даних DuckDB

Для зберігання ринкових даних, прогнозів та службової інформації у розробленій системі використовується локальна аналітична база даних DuckDB, що поєднує високу продуктивність, компактність зберігання та підтримку SQL-запитів. На відміну від класичних реляційних СУБД, DuckDB оптимізований під аналітичні навантаження (OLAP), що робить його ефективним інструментом для обробки великих часових рядів локально, без потреби у додатковому серверному середовищі [18].

Особливості DuckDB роблять його оптимальним вибором для фінансових моделей:

- колонковий формат зберігання даних значно прискорює роботу з широкими таблицями OHLCV;
- можливість виконання складних SQL-запитів без підняття окремого сервера;
- підтримка транзакцій і ACID-властивостей;
- нативна інтеграція з Python через бібліотеку duckdb;
- можливість зберігання таблиць у форматі Parquet.

У рамках системи DuckDB використовується для двох ключових завдань: зберігання історичних ринкових даних та накопичення прогнозів моделей.

Структура бази даних

База даних містить дві основні таблиці, необхідні для функціонування системи:

Таблиця ohlcv_hourly (історичні ринкові дані)

Ця таблиця містить основні ринкові показники криптовалюти Bitcoin у погодинному розрізі. Вона формується під час першого завантаження історичних даних, а далі – автоматично оновлюється кожну годину.

Таблиця 4.1 – Структура таблиці ohlcv_hourly

Поле	Тип даних	Опис
timestamp	TIMESTAMP	Часова мітка у UTC
open	DOUBLE	Ціна вікриття
high	DOUBLE	Макс. ціна за годину
low	DOUBLE	Мін. ціна за годину
close	DOUBLE	Ціна закриття
volume	DOUBLE	Обсяг торгів
source	VARCHAR	Джерело даних (CoinGecko)

Таблиця використовується:

- для побудови ознак (features),
- для тренування моделей,
- для формування останнього вікна історії під час інференсу.

Зберігання даних у колонковому форматі дозволяє швидко виконувати операції фільтрації, агрегації та вибірки за часовим діапазоном.

Таблиця forecast_hourly (таблиця прогнозів)

Ця таблиця використовується для накопичення всіх прогнозів, що генеруються моделями LSTM та GRU. Вона дозволяє будувати історію прогнозів та аналізувати точність моделей у динаміці.

Таблиця 4.2 – Структура таблиці forecast_hourly

Поле	Тип даних	Опис
coin_id	VARCHAR	Монета
vs_currency	VARCHAR	Валюта
ts_anchor	TIMESTAMP	Час, на основі якого формувався прогноз
ts_forecast	TIMESTAMP	Час, на який зроблено прогноз
model	VARCHAR	Назва моделі (LSTM або GRU)
y_pred	DOUBLE	Прогнозоване значення ціни
is_backfilled	BOOLEAN	Перевірка чи значення взято з прогнозу чи з реального значення
created_at	TIMESTAMP	Час формування прогнозу

На відміну від таблиці історичних даних, записи тут не перезаписуються, навіть якщо модель була перенавчена. Це дозволяє:

- порівнювати старі та нові моделі;
- проводити ретроспективну оцінку;
- формувати статистику поведінки алгоритмів.

Оновлення таблиці здійснюється через модуль jobs/run_forecast.

Робота з DuckDB у системі

Проєкт використовує уніфіковані функції доступу до бази, розташовані у каталозі data/, що включають:

- підключення до бази та ініціалізацію таблиць;

- додавання нових OHLCV-даних;
- отримання останнього вікна для інференсу;
- збереження передбачених значень;
- виконання вибірок для побудови графіків у Streamlit.

Завдяки цьому логіка роботи з даними сконцентрована в одному модулі, що зменшує дублювання коду та підвищує надійність.

Переваги обраного підходу

Архітектура бази даних робить систему:

- ефективною, оскільки DuckDB оптимізований під аналітичні операції;
- надійною, адже таблиці розташовані локально і не залежать від зовнішніх серверів;
- масштабованою, бо DuckDB підтримує роботу з великими табличними наборами;
- гнучкою, оскільки структура таблиць легко розширюється;
- зручною для інтеграції, особливо з Python і Streamlit.

Таке поєднання дозволяє системі працювати повністю автономно та забезпечувати високу продуктивність при мінімальних апаратних вимогах.

4.3 Реалізація вебінтерфейсу засобами Streamlit

Вебінтерфейс системи побудований із використанням фреймворку Streamlit, який дозволяє створювати інтерактивні аналітичні панелі на основі Python-коду без потреби розробки окремого клієнтського застосунку. Streamlit забезпечує миттєве оновлення елементів інтерфейсу, автоматичний ререндеринг компонентів та зручну інтеграцію з моделями глибинного навчання, що робить його оптимальним вибором для задач прогнозування фінансових часових рядів у реальному часі.

Основні переваги використання Streamlit:

- інтерфейс створюється виключно Python-кодом;
- відсутня потреба у фронтенд-фреймворках (HTML/CSS/JS);

- інтеграція з бібліотеками Plotly та Matplotlib;
- можливість автоматичного оновлення сторінки;
- підтримка розширених віджетів (selectbox, slider, expander, date input тощо);
- простота розгортання на локальному ПК або сервері.

У системі Streamlit використовується для відображення ринкових даних, прогнозів, технічних індикаторів та історії передбачень.

4.3.1 Інтерактивні графіки на Plotly

Для візуалізації ринкових даних та прогнозів використано бібліотеку Plotly, яка забезпечує:

- динамічні графіки з можливістю масштабування;
- навігацію по часовій шкалі;
- інтерактивні підказки (hover labels);
- підтримку OHLC і candlestick-графіків;
- багатошарові графіки (реальні дані + прогноз).

На основній сторінці інтерфейсу відображено:

- історичний графік ціни закриття (Close);
- інтерактивний графік прогнозованих значень;
- ринкові показники за останні години;
- порівняння фактичної та прогнозованої динаміки.

Використання Plotly дозволяє зручно досліджувати короткі цикли волатильності, перевіряти поведінку моделей та оцінювати якість роботи системи.

4.3.2 Механізм автооновлення прогнозів

Для забезпечення актуальності відображуваної інформації вебінтерфейс використовує два механізми:

Автооновлення вмісту сторінки

Реалізовано за допомогою функції st_autorefresh(), яка періодично поновлює

вміст сторінки Streamlit без перезавантаження сторінки у браузері. Це дозволяє інтерфейсу автоматично підхоплювати нові дані, що вже з'явилися у базі.

Автоматичне оновлення даних у базі

Здійснюється за допомогою Task Scheduler (див. рис. 4.3), який запускає job-скрипти проєкту:

- оновлення OHLCV-даних,
- формування прогнозу t+1,
- щоденне перетренування моделей,
- запис результатів у DuckDB.

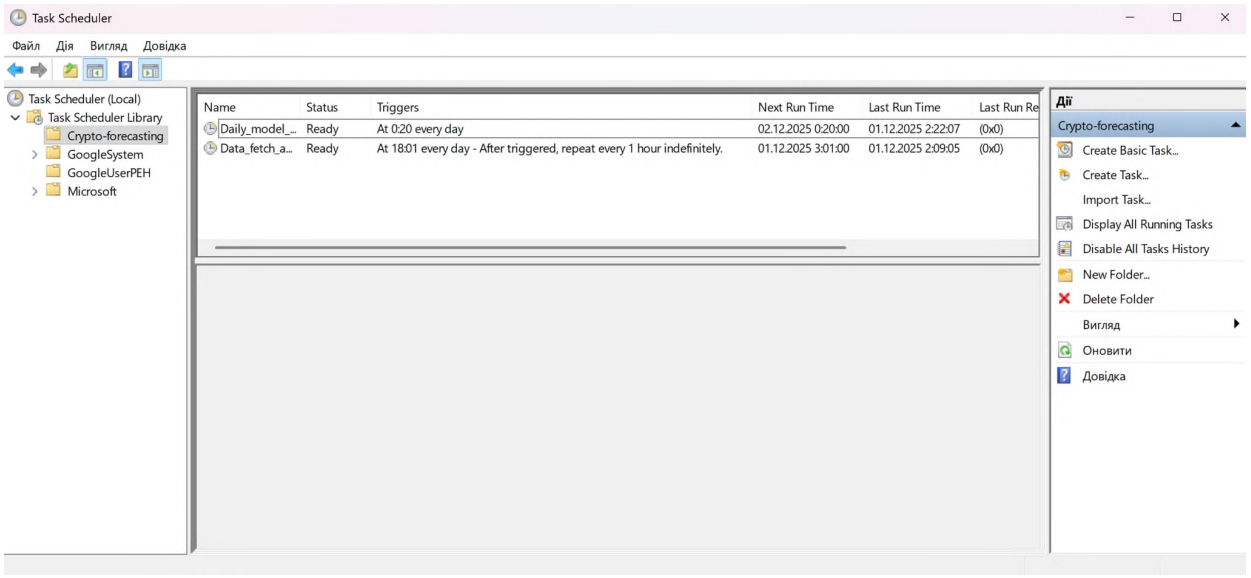
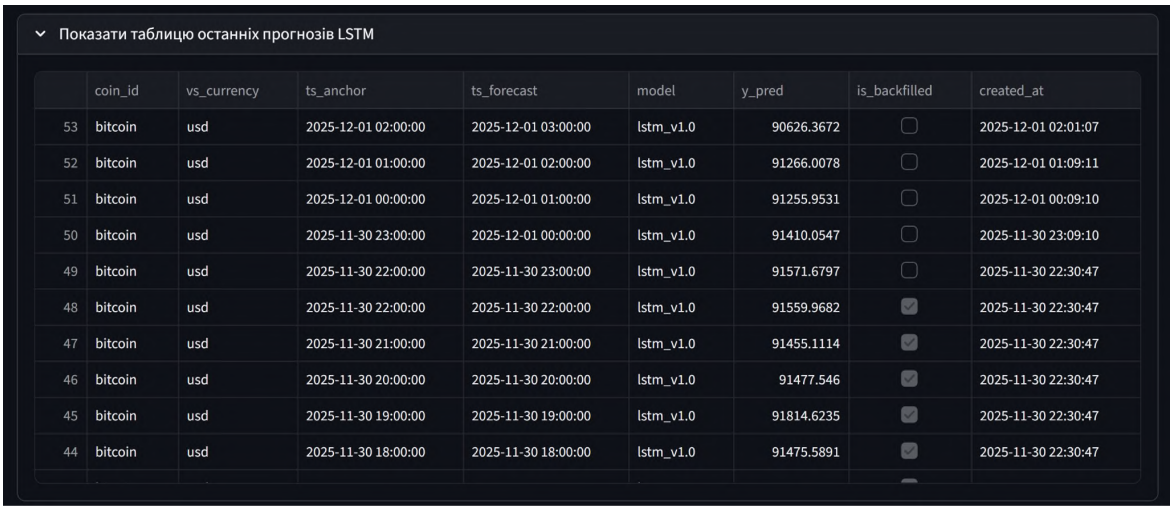


Рисунок 4.3 – Task Scheduler

Додатково для забезпечення безперебійної роботи системи було реалізовано функцію ensure_forecast_continuity(). Вона відповідає за наступні дії:

- заповнення пустих значень прогнозів реальними
- логування даних записів з особливою відміткою (is_backfilled)



	coin_id	vs_currency	ts_anchor	ts_forecast	model	y_pred	is_backfilled	created_at
53	bitcoin	usd	2025-12-01 02:00:00	2025-12-01 03:00:00	lstm_v1.0	90626.3672	☐	2025-12-01 02:01:07
52	bitcoin	usd	2025-12-01 01:00:00	2025-12-01 02:00:00	lstm_v1.0	91266.0078	☐	2025-12-01 01:09:11
51	bitcoin	usd	2025-12-01 00:00:00	2025-12-01 01:00:00	lstm_v1.0	91255.9531	☐	2025-12-01 00:09:10
50	bitcoin	usd	2025-11-30 23:00:00	2025-12-01 00:00:00	lstm_v1.0	91410.0547	☐	2025-11-30 23:09:10
49	bitcoin	usd	2025-11-30 22:00:00	2025-11-30 23:00:00	lstm_v1.0	91571.6797	☐	2025-11-30 22:30:47
48	bitcoin	usd	2025-11-30 22:00:00	2025-11-30 22:00:00	lstm_v1.0	91559.9682	☑	2025-11-30 22:30:47
47	bitcoin	usd	2025-11-30 21:00:00	2025-11-30 21:00:00	lstm_v1.0	91455.1114	☑	2025-11-30 22:30:47
46	bitcoin	usd	2025-11-30 20:00:00	2025-11-30 20:00:00	lstm_v1.0	91477.546	☑	2025-11-30 22:30:47
45	bitcoin	usd	2025-11-30 19:00:00	2025-11-30 19:00:00	lstm_v1.0	91814.6235	☑	2025-11-30 22:30:47
44	bitcoin	usd	2025-11-30 18:00:00	2025-11-30 18:00:00	lstm_v1.0	91475.5891	☑	2025-11-30 22:30:47

Рисунок 4.4 – Таблиця прогнозів моделі LSTM

На даному рисунку ми можемо бачити, що дані за період з 23:00 до 3:00 заповнені реальними прогнозами моделі, тоді як усі дані до цього моменту заповнені фактичною ціною.

Це дозволяє системі не виходити з ладу у випадку тимчасових збоїв з сервісами API або відсутності доступу до сервера на якому реалізована автоматизація, а також відслідкувати чіткі часові діапазони коли ці проблеми виникали.

Використання даних механізмів дозволяє забезпечити повну автономність системи без потреби втручання з боку користувача.

4.3.3 Елементи взаємодії з користувачем

Вебінтерфейс розробленої інтелектуальної системи аналізу та прогнозування ринку криптовалют реалізований на основі фреймворку Streamlit, що забезпечує інтерактивність, динамічне оновлення даних та зручну навігацію між розділами. Інтерфейс побудований таким чином, щоб користувач без спеціальних технічних знань міг швидко отримати доступ до історичних даних, переглянути сформовані ознаки, порівняти фактичні значення з прогнозами моделей та налаштувати параметри оновлення даних.

Головна навігація та структура інтерфейсу

Зліва розташоване бокове меню, яке забезпечує доступ до основних функціональних розділів (див. рис. 4.5):

- Data (Дані)
- Features (Ознаки)
- Forecast (Прогноз)
- Debugging (Налагодження моделей)
- Settings (Налаштування)

У верхній частині меню знаходиться селектор криптовалюти, що дозволяє користувачеві перемикатися між Bitcoin, Ethereum, Solana, BNB та іншими активами.

Рисунок 4.5 – Головне бокове меню та селектор криптовалюти

Розділ “Data”: перегляд історичних ринкових даних

Даний розділ відображає історичні погодинні дані по ціні активу (OHLCV), отримані через API CoinGecko та збережені у DuckDB. Графік будується

автоматично з урахуванням вибраного періоду (наприклад, 7 днів, 30 днів або повна історія).

Користувач отримує можливість:

- переглядати динаміку ціни у вигляді інтерактивного графіка;
- збільшувати, стискати, переміщувати часову шкалу;
- розкривати таблицю даних під графіком.

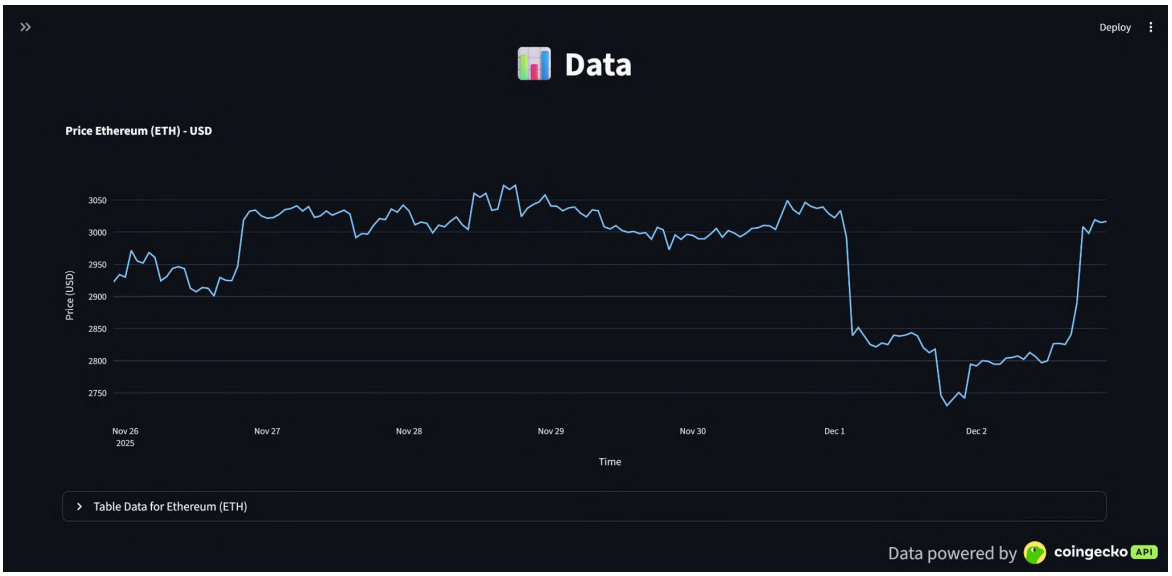


Рисунок 4.6 – Інтерактивний графік історії цін у розділі “Data”

Розділ “Features”: автоматично згенеровані ознаки

У цьому розділі відображаються ознаки, що використовуються моделями під час тренування та прогнозування:

- ковзні середні (SMA 5, 20, 50);
- доходність (returns);
- денна волатильність;
- об’єми торгів.

Інтерфейс демонструє кілька графіків, згрупованих у сітку 2×2, що дає змогу візуально оцінити характер і стабільність кожної ознаки.

Під графіками також доступна таблиця, де наведено повні значення всіх фіч.

Рисунок 4.7 – Графіки ознак (SMA, returns, volatility, volume) у розділі “Features”

Розділ “Forecast”: порівняння реальних і прогнозованих значень

Даний розділ призначений для перегляду результатів роботи двох моделей – LSTM та GRU.

На одному інтерактивному графіку відображаються три лінії:

- фактичні значення (Real);
- прогноз моделі LSTM;
- прогноз моделі GRU.

Система дозволяє:

- оцінити, наскільки точно модель передбачає локальні коливання;
- порівняти різні нейронні мережі між собою;
- переглянути лог-таблиці прогнозів для детального аналізу значень.

Рисунок 4.8 – Прогноз LSTM та GRU у порівнянні з реальними даними

Розділ “Debugging”: оцінка моделі на попередніх даних

Цей розділ створено для внутрішнього аналізу моделі та її поведінки. Він дозволяє:

- обрати модель (Baseline, LSTM або GRU);
- переглянути оцінки точності (MAE, RMSE);
- порівняти реальні значення, значення “майбутнього” (дані наступної доби) та прогноз, зроблений моделлю в умовах, максимально наближених до реального часу.

Цей режим використовується не кінцевим користувачем, а розробником для виявлення проблем зі скейлерами, зсувами у даних чи архітектурними недоліками моделі.

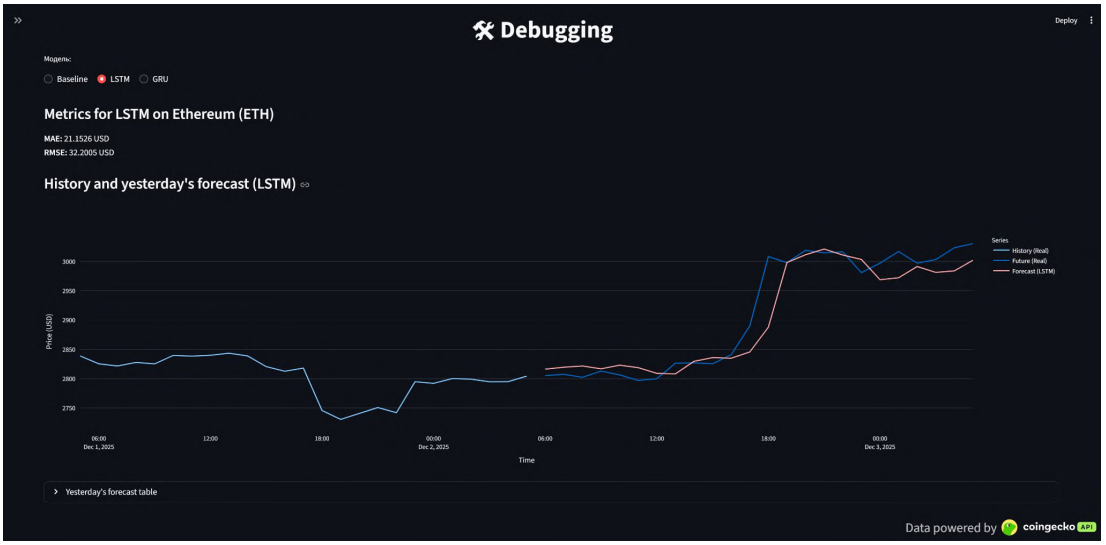


Рисунок 4.9 – Вікно налагодження, де відображаються фактичні та прогнозовані значення моделі

Розділ “Settings”: керування параметрами оновлення та періодом даних

У цьому підрозділі:

- змінювати період вибірки історичних даних (1 день, 7 днів, 30 днів, повна історія);
- увімкнути автоматичне оновлення даних;
- вказати інтервал оновлення (1, 5, 10 хвилин тощо).

Ці параметри дозволяють побудувати оновлювані графіки, які реагують на нові дані з DuckDB та API CoinGecko.

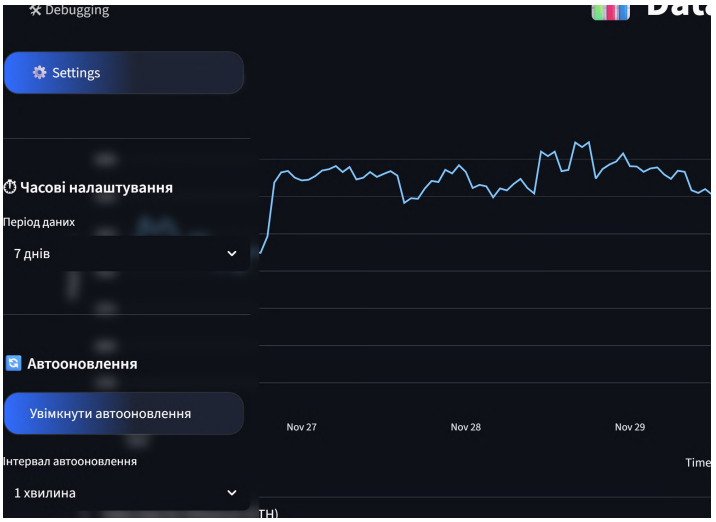


Рисунок 4.10 – Налаштування періоду даних та автооновлення

Вибір криптовалюти

У верхній частині бокового меню розміщений селектор криптовалюти, що визначає, для якого активу завантажуються історичні дані, ознаки, прогнози та лог-таблиці.

Після зміни активу інтерфейс повністю перебудовує всі графіки відповідно до нових даних.

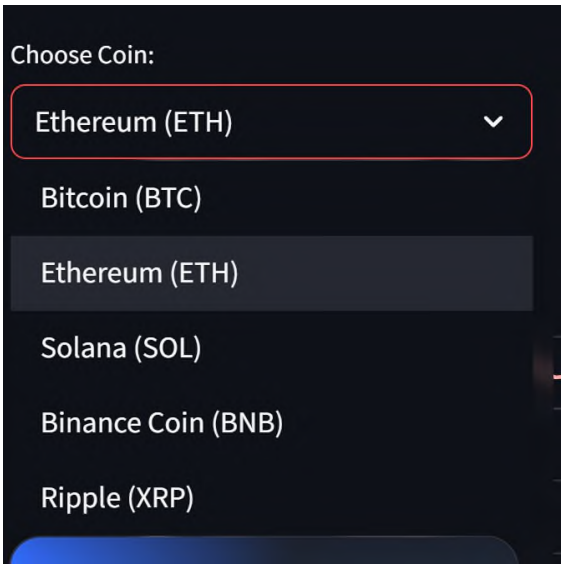


Рисунок 4.11 – Вибір криптовалюти у меню

Загальна оцінка зручності взаємодії

Структура інтерфейсу дозволяє користувачу:

- швидко перемикатися між модулями;
- отримувати як загальну оглядову інформацію (історія, тренди), так і поглиблений аналітичний матеріал (фічі, прогнози, метрики);
- працювати з даними в режимі реального часу без перезапуску застосунку;
- зручно здійснювати навігацію завдяки мінімалістичному дизайну та логічній структурі вкладок.

Висновки до розділу

У цьому розділі було розглянуто архітектуру розробленої програмної системи для аналізу та прогнозування ринку криптовалют, а також описано ключові технології та принципи реалізації її програмних модулів. Запропонована структура

проєкту побудована за модульним підходом, що забезпечує чіткий розподіл відповідальностей між компонентами та спрощує підтримку системи.

Було проаналізовано організацію робочого пайплайна, який включає збирання ринкових даних, їх збереження у базі DuckDB, формування ознак, генерацію прогнозів моделями LSTM та GRU, а також подальшу інтеграцію результатів у вебінтерфейсі. Особливу увагу приділено використанню локальної бази даних, що забезпечує високу продуктивність роботи з часовими рядами та дає змогу ефективно зберігати як історичні дані, так і накопичені прогнози.

Окремо було розглянуто особливості реалізації інтерфейсу Streamlit, який відповідає за візуалізацію ринкових даних та результатів прогнозування. Завдяки механізму автоматичного оновлення вмісту сторінки та зовнішнім задачам, що виконуються через Task Scheduler, система здатна працювати у режимі, наближеному до реального часу.

Таким чином, реалізовано програмне забезпечення забезпечує повний цикл аналізу даних – від їх отримання та обробки до формування прогнозів і їх відображення користувачу. Структура системи є гнучкою та розширювальною, що дозволяє надалі доповнювати її новими моделями, джерелами даних або функціональними можливостями.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Ідея проєкту полягає у створенні онлайн-платформи для аналізу та короткострокового прогнозування ринку криптовалют, яка поєднує автоматичне збирання ринкових даних, методи глибинного навчання та інтерактивний веб інтерфейс. На відміну від традиційних фінансових інструментів, що вимагають спеціалізованих знань або доступу до складної інфраструктури, запропонована система надає користувачу простий у використанні вебсервіс, де всі обчислення виконуються на сервері, а результати прогнозування доступні через зручну панель моніторингу.

Працюючи у режимі, наближеному до реального часу, система автоматично завантажує погодинні OHLCV-дані з відкритих API, обробляє їх на бекенді, формує ознаки для моделювання та генерує прогноз за допомогою нейронних мереж типу LSTM та GRU. Користувач взаємодіє лише з вебінтерфейсом, де відображаються історичні ринкові дані, результати моделювання та попередні прогнози. Усі процеси збирання даних, їх обробки, зберігання та виконання прогнозів повністю автоматизовані та відбуваються на стороні сервера, що дозволяє користувачу отримувати актуальну інформацію без необхідності встановлювати додаткове програмне забезпечення або мати спеціальні технічні навички.

Потенційними користувачами системи є приватні інвестори, трейдери-початківці, фінансові аналітики, студенти та викладачі у сфері фінансових технологій. Для інвесторів та трейдерів платформа є інструментом, який дозволяє швидко оцінити поточний стан ринку та отримати короткострокову прогнозну оцінку. Для навчальних та аналітичних цілей система може слугувати прикладом практичної реалізації моделей прогнозування фінансових часових рядів.

У контексті стартап-потенціалу проєкт може масштабуватися за рахунок підтримки кількох криптовалют, впровадження додаткових моделей машинного

навчання, розширення функціоналу аналітичних панелей, створення API-доступу до прогнозів та інтеграції з мобільними застосунками. Окремим напрямом розвитку є можливість формування сигналів або попереджень для користувачів на основі поведінки ринку.

Унікальність проєкту полягає у поєднанні автоматизованої серверної обробки ринкової інформації, сучасних моделей глибинного навчання та зручного вебінтерфейсу в одному продукті. Більшість існуючих рішень або працюють повністю на стороні користувача, або приховують внутрішню логіку та не дають можливості зрозуміти принципи роботи моделі. Запропонована система забезпечує прозорість алгоритмів, стабільність формування прогнозів та повну автоматизацію процесів, створюючи інструмент, який є одночасно доступним, технологічно прогресивним і придатним для подальшого розширення в рамках стартап-платформи.

5.2 Розроблення ринкової стратегії

Ринкова стратегія стартап-проєкту визначає принципи позиціонування продукту, особливості цільових сегментів користувачів та способи виходу на ринок. Оскільки система є онлайн-платформою, що працює на серверній інфраструктурі та забезпечує автоматичне формування прогнозів криптовалютної динаміки, стратегія її впровадження повинна враховувати як специфіку фінансового ринку, так і високий рівень конкуренції у сфері криптоаналітики.

Першим елементом ринкової стратегії є аналіз цільової аудиторії. Основними користувачами платформи можуть бути **приватні інвестори та трейдери-початківці**, які потребують простого інструмента для аналізу ринку без необхідності глибоких технічних і математичних знань. Для цієї групи важливими є доступність, зрозумілий інтерфейс та можливість отримувати прогнози у режимі реального часу.

Другою групою користувачів є **досвідчені трейдери та фінансові аналітики**, які можуть використовувати прогнозну модель як додаткове аналітичне джерело для прийняття рішень.

Третьою групою є **освітні та дослідницькі установи**, що використовують платформу для демонстрації роботи моделей машинного навчання у прикладних фінансових задачах.

Подальшим кроком є аналіз конкурентного середовища. На ринку існує значна кількість платформ для криптоаналітики – від комплексних сервісів типу TradingView до моделей прогнозування, що працюють за принципом платної підписки. Більшість таких рішень або не надають прозорості щодо алгоритмів прогнозування, або вимагають дорогих тарифів для розширеної аналітики. Створювана система має конкурентну перевагу завдяки поєднанню автоматизованої серверної обробки даних, використання класичних моделей глибинного навчання, прозорості алгоритмів та доступного вебінтерфейсу. Таким чином, продукт може стати оптимальним вибором для користувачів, яким потрібен інструмент без прихованих механізмів та складних бар'єрів входу.

З точки зору позиціонування, проєкт пропонується як онлайн-платформа короткострокового прогнозування криптовалют, що забезпечує регулярне оновлення прогнозів, автоматичне збирання ринкових даних і інтерактивну аналітику. Ключовим елементом позиціонування є простота та доступність: користувачу не потрібно налаштовувати власні моделі, підключати API чи підтримувати інфраструктуру – вся логіка виконується на сервері, а доступ до даних відбувається через вебінтерфейс.

Стратегія виходу на ринок передбачає поетапний підхід.

- На першому етапі формується MVP-версія платформи з підтримкою лише кількох популярних криптовалют, базових моделей прогнозування та основних графічних компонентів.

- На другому етапі система може бути розширена підтримкою додаткових валют, впровадженням нових моделей машинного навчання, аналітичних індикаторів та розширених налаштувань для досвідчених користувачів.
- На третьому етапі можливе створення мобільного застосунку, API-доступу до прогнозів або інтеграції з месенджерами для надсилання сигналів у реальному часі.

Цінова політика може передбачати гібридну модель: базовий доступ до прогнозів як безкоштовний продукт, а розширені функції – у форматі платної підписки. Це дозволяє залучити широку аудиторію, а згодом монетизувати сервіс за рахунок поглибленої аналітики, додаткових моделей або персоналізованих рекомендацій.

Таким чином, ринкова стратегія стартапу базується на поєднанні доступності, прозорості, технологічності та поступового масштабування функціональності. Завдяки цьому платформа має потенціал зайняти свою нішу серед продуктів для аналізу криптовалют, орієнтуючись як на масового користувача, так і на професійні сегменти.

5.3 Розроблення маркетингової програми

Маркетингова програма стартап-проєкту визначає комплекс заходів, спрямованих на популяризацію платформи, залучення користувачів та формування довгострокової цінності продукту на ринку фінансових технологій. Оскільки система є веборієнтованим сервісом із прогнозування криптовалютної динаміки, ключовим завданням маркетингу є демонстрація її практичної користі, надійності та прозорості алгоритмів для широкої аудиторії. Це вимагає комплексного підходу, що поєднує цифрові канали комунікації, освітні ініціативи та репутаційний маркетинг.

Основою маркетингової програми є формування впізнаваності продукту. На початковому етапі доцільно забезпечити публічну присутність платформи через створення офіційного вебсайту або лендингу, який міститиме опис ключового функціоналу, приклади прогнозів, інтерфейс панелі моніторингу та інструкції для

користувачів. Важливо донести сильні сторони системи – автоматичне оновлення даних, використання сучасних моделей глибинного навчання, прозорість алгоритмів та відсутність потреби у складних налаштуваннях. Для залучення перших користувачів ефективними можуть бути таргетовані рекламні кампанії у соціальних мережах, особливо у спільнотах, пов'язаних із криптовалютами та фінансовими інвестиціями.

Другим напрямом маркетингової програми є контент-стратегія. Регулярне створення інформаційних матеріалів – статей, відеоогляді, публікацій у блогах – дозволяє не лише збільшувати впізнаваність продукту, але й підкреслювати його експертність. Контент може включати пояснення принципів роботи LSTM та GRU, огляди ринкових тенденцій, приклади короткострокових прогнозів та порівняння різних моделей машинного навчання. Така стратегія формує довіру серед користувачів і сприяє органічному зростанню аудиторії.

Важливим елементом маркетингової програми є робота з освітніми та аналітичними спільнотами. Оскільки продукт має навчальний потенціал, доцільно пропонувати платформу як інструмент для університетських курсів з фінансів, економіки або машинного навчання. Співпраця з викладачами та науковими групами може сприяти підвищенню авторитету проєкту та популяризації його функціональності серед студентів, дослідників та молодих спеціалістів.

На етапі масштабування до маркетингової програми можуть бути включені партнерські колаборації з криптовалютними біржами, фінтех-компаніями або освітніми платформами. Такі партнерства не лише розширюють аудиторію, але й створюють додаткові можливості для монетизації, наприклад через API-доступ, інтеграцію прогнозних сигналів або впровадження аналітичних модулів у сторонні продукти.

Для утримання користувачів ключовим інструментом є формування спільноти. Важливо забезпечити наявність Telegram- або Discord-каналу, де публікуватимуться оновлення, результати прогнозів, аналітичні огляди та плани подальшого розвитку проєкту. Це дозволяє підтримувати активну взаємодію з аудиторією та отримувати

швидкий зворотний зв’язок, який є надзвичайно важливим для вдосконалення системи.

Таким чином, маркетингова програма поєднує створення експертного контенту, розвиток онлайн-присутності, партнерські ініціативи та вибудовування лояльної спільноти. Такий комплекс дій забезпечує збалансоване зростання користувацької бази, формує довіру до платформи та створює передумови для її подальшого розвитку як конкурентоспроможного стартап-продукту у сфері фінансових технологій.

5.4 Вимоги до технічного та програмного забезпечення

Для забезпечення стабільної роботи онлайн-платформи прогнозування криптовалютної динаміки необхідно визначити комплекс технічних і програмних вимог, які дозволяють реалізувати безперервне збирання даних, їх обробку, навчання моделей та формування прогнозів у режимі, наближеному до реального часу. Ці вимоги охоплюють серверну інфраструктуру, програмне забезпечення, бібліотеки машинного навчання та засоби автоматизації.

З технічної точки зору сервер, на якому розгортатиметься система, повинен мати достатню обчислювальну потужність для виконання регулярних прогнозних операцій. Оскільки моделі LSTM та GRU належать до класу рекурентних нейронних мереж і потребують більше ресурсів порівняно з класичними статистичними методами, доцільно використовувати сервер із багатоядерним процесором і достатнім обсягом оперативної пам’яті (від 8-16 ГБ залежно від інтенсивності оновлення). Для можливого масштабування або навчання складніших моделей може бути використаний сервер із підтримкою GPU, проте для поточного функціоналу достатньою є CPU-орієнтована конфігурація.

Система зберігання даних має забезпечувати швидкий доступ до історичних OHLCV-даних та результатів прогнозування. У рамках проєкту використовується DuckDB – аналітична вбудована СУБД, оптимізована для роботи з колонковими даними та локальними аналітичними запитами. Її перевагами є висока швидкодія,

відсутність потреби у складній конфігурації та можливість інтеграції з Python-скриптами. База даних зберігається на сервері, а взаємодія з нею відбувається через спеціалізовані модулі системи.

Програмні вимоги включають використання Python як основної мови розробки, а також низку бібліотек для роботи з даними, побудови графіків та реалізації моделей машинного навчання. До ключових інструментів належать: pandas та numpy для обробки таблиць, requests або aiohttp для збирання ринкових даних з API, duckdb для роботи з базою даних, PyTorch для реалізації та навчання моделей LSTM і GRU, scikit-learn для масштабування даних, а також streamlit та plotly для реалізації вебінтерфейсу та інтерактивної візуалізації.

Для автоматичного оновлення ринкових даних використовуються серверні задачі (job-и), які запускаються відповідно до визначеного розкладу. У локальному середовищі під час розробки для цього застосовувався Windows Task Scheduler, однак у разі реального серверного розгортання можуть бути використані cron-завдання або вбудовані планувальники у хмарних сервісах. Такий підхід гарантує регулярне оновлення історії цін та автоматичне генерування нових прогнозів без участі користувача.

Користувачеві не потрібно встановлювати будь-яке додаткове програмне забезпечення, оскільки доступ до системи здійснюється через браузер. Це робить платформу універсальною та доступною на будь-якому пристрої – ПК, ноутбуці або мобільному телефоні. Основні вимоги з боку клієнта зводяться до наявності сучасного браузера та стабільного інтернет-з’єднання.

Таким чином, вимоги до технічного та програмного забезпечення визначають архітектуру, необхідну для стабільної, надійної та масштабованої роботи онлайн-платформи. Використання сучасних бібліотек, оптимізованої аналітичної СУБД та автоматизованих механізмів оновлення даних забезпечує ефективність системи та можливість її подальшого розвитку в рамках стартап-проєкту.

Висновки до розділу

У цьому розділі було розглянуто стартап-компонент розробленої системи, що дозволяє позиціонувати її не лише як навчальний чи дослідницький інструмент, а як потенційно комерційний продукт у сфері фінансових технологій. Описано ідею проєкту як онлайн-платформи для аналізу та короткострокового прогнозування ринку криптовалют, у якій всі обчислення, обробка даних та формування прогнозів виконуються на серверній стороні, а користувач взаємодіє з результатами через вебінтерфейс.

Було визначено цільові сегменти аудиторії, серед яких приватні інвестори, трейдери-початківці, досвідчені аналітики та освітні установи. Розроблена ринкова стратегія враховує особливості конкурентного середовища та робить акцент на таких перевагах системи, як прозорість алгоритмів, використання моделей глибинного навчання, автоматизація процесів та доступний інтерфейс. Запропоновано поетапний підхід до виведення продукту на ринок – від створення MVP з базовим функціоналом до розширення можливостей платформи та потенційної інтеграції з іншими сервісами.

У межах маркетингової програми було окреслено основні напрями просування платформи: розвиток онлайн-присутності, контент-маркетинг, співпраця з тематичними спільнотами та освітніми закладами, формування користувацької спільноти та побудова партнерств із представниками фінтех-галузі. Такий підхід спрямований на підвищення впізнаваності продукту, формування довіри до його результатів та забезпечення поступового зростання аудиторії.

Окремо було проаналізовано вимоги до технічного та програмного забезпечення, необхідні для стабільної роботи платформи. Визначено базові характеристики серверної інфраструктури, обґрунтовано вибір DuckDB як засобу зберігання даних, Python та відповідних бібліотек для реалізації моделей, а також використання засобів автоматизації для регулярного оновлення ринкових даних і прогнозів. Підкреслено, що клієнтська частина не потребує спеціальних

налаштувань, оскільки доступ до платформи здійснюється через звичайний веббраузер.

Узагальнюючи, можна зробити висновок, що розроблений програмний продукт має не лише технічну, але й комерційну перспективу. Продумана ідея, ринкова стратегія, маркетингова програма та визначені вимоги до інфраструктури створюють основу для подальшого розвитку системи як конкурентоспроможної онлайн-платформи у сфері аналізу та прогнозування ринку криптовалют.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено інтелектуальну систему аналізу та короткострокового прогнозування ринку криптовалют, що поєднує сучасні методи машинного навчання, автоматизоване збирання даних та інтерактивний вебінтерфейс. У ході дослідження проведено комплексний огляд проблемної області, визначено особливості волатильності криптовалютних ринків та фактори, що впливають на точність прогнозування фінансових часових рядів.

На основі погодинних OHLCV-даних було виконано попередній аналіз, підготовку та масштабування даних, сформовано набір ознак для моделювання. Розглянуто математичні засади моделювання часових рядів і досліджено властивості кількох архітектур глибинного навчання – LSTM, GRU, CNN-LSTM та моделей з механізмом уваги. Проведений порівняльний аналіз показав, що для задач короткострокового прогнозування за умов високої волатильності найбільш стабільні результати демонструють мережі LSTM і GRU, що й визначило їх використання у програмній реалізації системи.

Розроблене програмне забезпечення включає серверну базу даних DuckDB, модулі автоматичного збирання ринкової інформації, інструменти для підготовки даних та модуль прогнозування, інтегрований зі стриманим і функціональним вебінтерфейсом на базі фреймворку Streamlit. Система працює у режимі регулярного оновлення, автоматично формуючи нові прогнозні значення та зберігаючи їх для подальшого аналізу. Такий підхід забезпечує безперервність роботи платформи та її актуальність для користувача.

У межах стартап-компоненти роботи було сформовано ринкову стратегію, визначено цільові сегменти користувачів, конкурентні переваги й можливості подальшого розвитку. Описано маркетингову програму та технічні вимоги, що дозволяють розгортати систему як комерційний вебсервіс. Це підтверджує потенціал проєкту не лише як навчального чи дослідницького інструмента, але й як реального продукту у сфері фінансових технологій.

Підсумовуючи, розроблена система успішно вирішує поставлену задачу прогнозування криптовалютної динаміки, забезпечуючи автоматизовану обробку великих обсягів ринкових даних, застосування сучасних моделей глибинного навчання та зручний інтерфейс для кінцевого користувача. Отримані результати підтверджують ефективність використаних методів і створюють основу для подальшого вдосконалення платформи, розширення її функціональності та інтеграції у реальні фінансово-аналітичні процеси.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. URL: https://bitcoin.org/bitcoin.pdf (дата звернення: 30.09.2025).
2. Houben R., Snyers A. Cryptocurrencies and Blockchain: Legal Context and Implications for Financial Crime, Money Laundering and Tax Evasion. European Parliament. Brussels, 2018. 98 p.
3. Siami-Namini S., Tavakoli N., Siami N. A Comparison of ARIMA and LSTM in Forecasting Time Series. 17th IEEE International Conference on Machine Learning and Applications (ICMLA). Orlando, 2018. P. 1394–1401.
4. Deng Y., Bao F., Kong Y., Ren Z., Dai Q. Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. IEEE Transactions on Neural Networks and Learning Systems. 2017. Vol. 28, № 3. P. 653–664.
5. Antonopoulos A. M. Mastering Bitcoin: Unlocking Digital Cryptocurrencies. 2nd ed. Sebastopol : O'Reilly Media, 2017. 408 p.
6. Gandal N., Hamrick J. T., Moore T., Oberman T. The Economics of Cryptocurrency Pump and Dump Schemes. Journal of Monetary Economics. 2021. Vol. 117. P. 349–365.
7. Phillips R. C., Gorse D. Social Media and Bitcoin: The Influence of Reddit Posts on Cryptocurrency Markets. Journal of Risk and Financial Management. 2018. Vol. 11, № 4. P. 56–71.
8. Wei W. The Impact of Tether Grants on Bitcoin. Economics Letters. 2018. Vol. 171. P. 19–22.
9. Corbet S., Lucey B., Urquhart A., Yarovaya L. Cryptocurrencies as a Financial Asset: A Systematic Analysis. International Review of Financial Analysis. 2019. Vol. 62. P. 182–199.
10. Fang F., Ventre C., Basios M. et al. Cryptocurrency Trading: A Comprehensive Survey. Financial Innovation. 2022. Vol. 8, № 1. P. 1–59.

11. Hyndman R. J., Athanasopoulos G. Forecasting: Principles and Practice. 3rd ed. Melbourne : OTexts, 2021. 412 p.
12. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Vol. 9, № 8. P. 1735–1780.
13. Box G. E. P., Jenkins G. M., Reinsel G. C., Ljung G. M. Time Series Analysis: Forecasting and Control. 5th ed. Hoboken : Wiley, 2015. 712 p.
14. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, 2016. P. 785–794.
15. Vaswani A. et al. Attention Is All You Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.
16. CoinGecko. CoinGecko API Documentation. URL: https://www.coingecko.com/en/api/documentation (дата звернення: 01.10.2025).
17. PyTorch. PyTorch Documentation. URL:https://pytorch.org (дата звернення: 01.10.2025).
18. DuckDB. DuckDB Documentation. URL:https://duckdb.org (дата звернення: 01.10.2025)
19. Project’s GitHub Repository. URL:https://github.com/surjkee/crypto-forecasting

ДОДАТКИ

ДОДАТОК А Лістинг коду програми

app.py

```
import sys
import os

# ---- FIX IMPORT PATH ----
CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
PROJECT_ROOT = os.path.abspath(os.path.join(CURRENT_DIR, ".."))
sys.path.append(PROJECT_ROOT)
# -------------------------

import streamlit as st

from ui.components.sidebar import render_sidebar
from ui.tabs.debugging_tab import render_debugging_tab
from ui.tabs.data_tab import render_data_tab
from ui.tabs.forecast_tab import render_forecast_tab
from ui.tabs.features_tab import render_features_tab
from ui.components.footer import render_footer


def main():
    # --- Загальні налаштування сторінки ---
    st.set_page_config(
        page_title="Crypto Forecasting - Data",
        layout="wide",
    )
    if not st.session_state.get("developer_mode", True):
        st.markdown(
            """
            <style>
            [data-testid="stToolbar"] { display: none !important; }
            </style>
            """,
            unsafe_allow_html=True,
        )
    st.markdown(
        """
    <style>
        .stAppHeader {
            background: rgba(14, 12, 23, 0.0) !important;  /* темне скло */
        }
    </style>
    """,
        unsafe_allow_html=True,
    )
    # Глобальний сайдбар (монета + час + автооновлення)
    render_sidebar()

    active_tab = st.session_state.get("active_tab", "data")

    if active_tab == "data":
        render_data_tab()
    elif active_tab == "features":
        render_features_tab()
    elif active_tab == "forecast":
```

```
        render_forecast_tab()
    elif active_tab == "debug":
        render_debugging_tab()

    render_footer()

if __name__ == "__main__":
    main()
```

constants.py

```
TRACKED_COINS = [
    ("Bitcoin (BTC)", "bitcoin"),
    ("Ethereum (ETH)", "ethereum"),
    ("Solana (SOL)", "solana"),
    ("Binance Coin (BNB)", "binancecoin"),
    ("Ripple (XRP)", "ripple"),
]
```

.env.example

```
# Demo / local config template

# === API KEYS ===
COINGECKO_API_KEY=your_api_key_here

# === App defaults ===
DEFAULT_VS_CURRENCY=usd
HISTORY_DAYS_DEFAULT=60
HISTORY_INTERVAL=hourly
```

config/settings.py

```
from pathlib import Path
from pydantic_settings import BaseSettings, SettingsConfigDict
from pydantic import Field
from typing import List


# Корінь проєкту: .../Crypto-forecasting
PROJECT_ROOT = Path(__file__).resolve().parent.parent

# Директорія для даних
DATA_DIR = PROJECT_ROOT / "data"
DATA_DIR.mkdir(parents=True, exist_ok=True)

# Шлях до DuckDB
DUCKDB_PATH = DATA_DIR / "crypto.duckdb"


class Settings(BaseSettings):
    """
    Централізований конфіг проєкту.
    Значення беремо з .env або змінних середовища.
    """

    # === API keys ===
    coingecko_api_key: str = Field(..., alias="COINGECKO_API_KEY")

    # === App defaults ===
    default_vs_currency: str = Field("usd", alias="DEFAULT_VS_CURRENCY")
    history_days_default: int = Field(60, alias="HISTORY_DAYS_DEFAULT")
    history_interval: str = Field("hourly", alias="HISTORY_INTERVAL")
```

```
    #  Список монет, за якими ми тягнемо історію
    tracked_coins: List[str] = Field(
        default_factory=lambda: [
            "bitcoin",
            "ethereum",
            "solana",
            "binancecoin",
            "ripple",
        ],
        alias="TRACKED_COINS",
    )

    model_config = SettingsConfigDict(
        env_file=PROJECT_ROOT / ".env",
        env_file_encoding="utf-8",
        extra="ignore",
        case_sensitive=True,  # ігноруємо регістр в назвах змінних
    )


def get_settings() -> Settings:
    """
    Повертає новий інстанс Settings.

    ВАЖЛИВО:
    - ми не кешуємо settings, щоб у Streamlit зміни .env
      підхоплювалися при кожному rerun'і скрипта.
    """
    return Settings()


# Корисні "шорткати" для інших модулів, якщо треба
# (можемо використовувати їх або безпосередньо get_settings())
def get_default_vs_currency() -> str:
    return get_settings().default_vs_currency


def get_history_days_default() -> int:
    return get_settings().history_days_default


def get_history_interval() -> str:
    return get_settings().history_interval
```

data/db.py

```
from pathlib import Path
from typing import Optional
from datetime import datetime

import duckdb
import pandas as pd

from config.settings import PROJECT_ROOT, DATA_DIR, DUCKDB_PATH


def get_connection(read_only: bool = False) -> duckdb.DuckDBPyConnection:
    """
    Повертає підключення до DuckDB.
    """
```

```
    return duckdb.connect(str(DUCKDB_PATH))


def init_db() -> None:
    """
    Ініціалізація схеми БД (якщо таблиць ще немає).
    """
    con = get_connection()

    con.execute(
        """
        CREATE TABLE IF NOT EXISTS ohlcv_hourly (
            coin_id TEXT,
            vs_currency TEXT,
            ts TIMESTAMP,
            price DOUBLE,
            market_cap DOUBLE,
            volume DOUBLE
        )
        """
    )

    con.execute(
        """
        CREATE TABLE IF NOT EXISTS forecast_hourly (
            coin_id TEXT,
            vs_currency TEXT,
            ts_anchor TIMESTAMP,
            ts_forecast TIMESTAMP,
            model TEXT,
            y_pred DOUBLE,
            is_backfilled BOOLEAN DEFAULT FALSE,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        );
        """
    )


    con.close()


def load_ohlcv_hourly(
    coin_id: str,
    vs_currency: str,
    limit: Optional[int] = None,
) -> pd.DataFrame:
    """
    Завантажує історичні дані з DuckDB у вигляді pandas.DataFrame
    для заданої монети та валюти.

    :param coin_id: id монети (наприклад, "bitcoin")
    :param vs_currency: валюта ("usd")
    :param limit: опціонально – обмеження кількості останніх рядків
    """
    init_db()
    con = get_connection(read_only=True)

    if limit is not None:
        query = """
            SELECT *
```

```
            FROM ohlcv_hourly
            WHERE coin_id = ? AND vs_currency = ?
            ORDER BY ts DESC
            LIMIT ?
        """
        df = con.execute(query, [coin_id, vs_currency, limit]).df()
        # повертаємо у зростаючому порядку часу
        df = df.sort_values("ts").reset_index(drop=True)
    else:
        query = """
            SELECT *
            FROM ohlcv_hourly
            WHERE coin_id = ? AND vs_currency = ?
            ORDER BY ts
        """
        df = con.execute(query, [coin_id, vs_currency]).df()

    con.close()
    return df

def ensure_forecast_continuity(
    coin_id: str,
    vs_currency: str,
    model: str,
    max_backfill_hours: int = 48,  # наприклад, останні 2 дні
):
    """
    Гарантує, що для кожного фактичного ts у ВІКНІ останніх max_backfill_hours
    існує рядок у forecast_hourly.

    Якщо для певного ts у цьому вікні немає прогнозу – створюється запис:
      ts_anchor   = ts
      ts_forecast = ts
      y_pred      = фактична ціна
      is_backfilled = TRUE
    """
    init_db()

    # 1) Фактичні дані
    df_price = load_ohlcv_hourly(coin_id, vs_currency)
    if df_price.empty:
        return

    df_price = df_price.copy()
    df_price["ts_hour"] = df_price["ts"].dt.floor("h")
    df_price = (
        df_price.sort_values("ts_hour")
        .drop_duplicates(subset=["ts_hour"], keep="last")
        .reset_index(drop=True)
    )

    last_fact_ts = df_price["ts_hour"].max()
    # обмежуємо backfill лише останніми max_backfill_hours
    window_start_ts = last_fact_ts - pd.Timedelta(hours=max_backfill_hours)

    df_price_win = df_price[df_price["ts_hour"] >= window_start_ts].copy()
    if df_price_win.empty:
        return

    # 2) Прогнози у цьому ж вікні
```

```
    con = get_connection(read_only=True)
    df_fc = con.execute(
        """
        SELECT ts_forecast
        FROM forecast_hourly
        WHERE coin_id = ?
          AND vs_currency = ?
          AND model = ?
          AND ts_forecast >= ?
          AND ts_forecast <= ?
        ORDER BY ts_forecast
        """,
        [coin_id, vs_currency, model, window_start_ts, last_fact_ts],
    ).df()
    con.close()

    existing_ts = set(pd.to_datetime(df_fc["ts_forecast"])) if not df_fc.empty else set()

    # 3) Знаходимо "дірки" тільки в межах цього вікна і заповнюємо їх фактом
    for _, row in df_price_win.iterrows():
        ts_hour = row["ts_hour"]
        if ts_hour not in existing_ts:
            fact_price = float(row["price"])
            store_hourly_forecast(
                coin_id=coin_id,
                vs_currency=vs_currency,
                ts_anchor=ts_hour,
                ts_forecast=ts_hour,
                model=model,
                y_pred=fact_price,
                is_backfilled=True,
            )



def store_hourly_forecast(
    coin_id: str,
    vs_currency: str,
    ts_anchor,
    ts_forecast,
    model: str,
    y_pred: float,
    is_backfilled: bool = False,
):
    """
    Зберігає прогноз у forecast_hourly.

    Якщо для (coin_id, vs_currency, ts_forecast, model) вже є запис –
    видаляємо його і вставляємо новий (останній вважається актуальним).
    """
    init_db()
    con = get_connection()

    with con:
        con.execute(
            """
            DELETE FROM forecast_hourly
            WHERE coin_id = ?
              AND vs_currency = ?
```

```
            AND ts_forecast = ?
            AND model = ?
        """,
        [coin_id, vs_currency, ts_forecast, model],
    )

    con.execute(
        """
        INSERT INTO forecast_hourly (
            coin_id, vs_currency,
            ts_anchor, ts_forecast,
            model, y_pred,
            is_backfilled,
            created_at
        )
        VALUES (?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP)
        """,
        [
            coin_id,
            vs_currency,
            ts_anchor,
            ts_forecast,
            model,
            float(y_pred),
            bool(is_backfilled),
        ],
    )

    con.close()


def load_last_hourly_forecast(
    coin_id: str,
    vs_currency: str,
    model: str = "lstm_v0.4",
):
    con = get_connection()
    query = """
        SELECT *
        FROM forecast_hourly
        WHERE coin_id = ?
          AND vs_currency = ?
          AND model = ?
        ORDER BY ts_forecast DESC, created_at DESC
        LIMIT 1
    """
    df = con.execute(query, [coin_id, vs_currency, model]).df()

    con.close()

    return df

def load_hourly_forecasts(
    coin_id: str,
    vs_currency: str,
    limit: Optional[int] = None,
    model: str = "lstm_v0.4",
) -> pd.DataFrame:
    """
    Завантажує історичні прогнози t+1 з таблиці forecast_hourly
```

```
    для заданої монети та валюти.

    :param coin_id: id монети (наприклад, "bitcoin")
    :param vs_currency: валюта ("usd")
    :param limit: опціонально – обмеження кількості останніх прогнозів
    :param model: назва моделі (для фільтрації, за замовчуванням 'lstm_v0.4')
    """
    init_db()
    con = get_connection(read_only=True)

    if limit is not None:
        query = """
            SELECT *
            FROM forecast_hourly
            WHERE coin_id = ?
              AND vs_currency = ?
              AND model = ?
            ORDER BY ts_forecast DESC
            LIMIT ?
        """
        df = con.execute(query, [coin_id, vs_currency, model, limit]).df()
        # повертаємо у зростаючому порядку часу
        df = df.sort_values("ts_forecast").reset_index(drop=True)
    else:
        query = """
            SELECT *
            FROM forecast_hourly
            WHERE coin_id = ?
              AND vs_currency = ?
              AND model = ?
            ORDER BY ts_forecast
        """
        df = con.execute(query, [coin_id, vs_currency, model]).df()

    con.close()
    return df
```

features/transform.py

```
import pandas as pd
import numpy as np
from typing import Sequence


def add_returns(df: pd.DataFrame) -> pd.DataFrame:
    """
    Додає до DataFrame колонки з:
    - простими відсотковими змінами (return)
    - логарифмічними змінами (log_return)
    """
    df = df.copy()

    df["return"] = df["price"].pct_change()
    df["log_return"] = np.log(df["price"] / df["price"].shift(1))

    return df


def add_sma(df: pd.DataFrame, windows: Sequence[int]) -> pd.DataFrame:
    """
    Додає прості ковзні середні (SMA) для заданих вікон.
```

```
    Приклад: windows = [5, 10, 20]
    """
    df = df.copy()

    for w in windows:
        col = f"sma_{w}"
        df[col] = df["price"].rolling(window=w, min_periods=1).mean()

    return df


def add_ema(df: pd.DataFrame, windows: Sequence[int]) -> pd.DataFrame:
    """
    Додає експоненційні ковзні середні (EMA).
    """
    df = df.copy()

    for w in windows:
        col = f"ema_{w}"
        df[col] = df["price"].ewm(span=w, adjust=False, min_periods=1).mean()

    return df


def add_volatility(df: pd.DataFrame, window: int = 24) -> pd.DataFrame:
    """
    Додає ковзну волатильність (стандартне відхилення)
    за вікном window (у кількості спостережень).
    За замовчуванням: 24 → добова волатильність для погодинних даних.
    """
    df = df.copy()

    # Використаємо log_return для оцінки волатильності
    if "log_return" not in df.columns:
        df = add_returns(df)

    df[f"volatility_{window}"] = (
        df["log_return"].rolling(window=window, min_periods=1).std()
    )

    return df


def build_feature_frame(
    df: pd.DataFrame,
    sma_windows: Sequence[int] = (5, 20, 50),
    ema_windows: Sequence[int] = (10, 50),
    volatility_window: int = 24,
) -> pd.DataFrame:
    """
    Повний пайплайн побудови фіч:
    - returns, log_returns
    - SMA
    - EMA
    - rolling volatility

    Повертає новий DataFrame з усіма колонками.
    """
    feat_df = df.copy()
    feat_df = add_returns(feat_df)
```

```
    feat_df = add_sma(feat_df, sma_windows)
    feat_df = add_ema(feat_df, ema_windows)
    feat_df = add_volatility(feat_df, volatility_window)

    return feat_df
```

ingest/coingecko_client.py

```
import sys
import os

# ---- FIX IMPORT PATH ----
CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
PROJECT_ROOT = os.path.abspath(os.path.join(CURRENT_DIR, ".."))
sys.path.append(PROJECT_ROOT)
# -------------------------

import requests
from typing import Any, Dict, List

from config.settings import get_settings


class CoinGeckoClient:
    """
    Клієнт для роботи з CoinGecko API, що використовує Settings.
    """

    def __init__(
        self,
        base_url: str | None = None,
        timeout: int | None = None,
        api_key: str | None = None,
    ) -> None:
        settings = get_settings()

        # Якщо параметри не передано – беремо з Settings
        self.base_url = (base_url or "https://api.coingecko.com/api/v3").rstrip("/")
        self.timeout = timeout or 10
        self.api_key = api_key or settings.coingecko_api_key

        if not self.api_key:
            raise RuntimeError(
                "CoinGecko API key is empty. "
                "Перевір .env (COINGECKO_API_KEY) і налаштування Settings."
            )

    def _get(self, path: str, params: Dict[str, Any] | None = None) -> Any:
        url = f"{self.base_url}/{path.lstrip('/')}"

        if params is None:
            params = {}

        #  Додаємо API key як query-параметр (рекомендований спосіб для Demo API)
        if self.api_key and "x_cg_demo_api_key" not in params:
            params["x_cg_demo_api_key"] = self.api_key

        #  І паралельно кладемо в headers (так теж дозволено)
        headers: Dict[str, str] = {}
        if self.api_key:
            headers["x-cg-demo-api-key"] = self.api_key
```

```
    response = requests.get(url, params=params, timeout=self.timeout, headers=headers)

    #  Тимчасове debug-логування при помилках
    if response.status_code != 200:
        try:
            err_json = response.json()
        except Exception:
            err_json = response.text

        print("=== CoinGecko DEBUG ===")
        print("URL:", response.url)
        print("Status code:", response.status_code)
        print("Response:", err_json)
        print("API key prefix:", repr(self.api_key[:6]) + "..." if self.api_key else "<empty>")
        print("API key length:", len(self.api_key) if self.api_key else 0)
        print("========================")

        response.raise_for_status()

    return response.json()


# -------- API METHODS --------

def ping(self) -> Dict[str, Any]:
    """Перевірка доступності API."""
    return self._get("/ping")

def get_top_coins(
    self,
    vs_currency: str | None = None,
    per_page: int = 50,
    page: int = 1,
) -> List[Dict[str, Any]]:
    settings = get_settings()
    vs_currency = vs_currency or settings.default_vs_currency

    params = {
        "vs_currency": vs_currency,
        "order": "market_cap_desc",
        "per_page": per_page,
        "page": page,
        "sparkline": False,
    }
    return self._get("/coins/markets", params=params)

def get_market_chart(
    self,
    coin_id: str,
    vs_currency: str | None = None,
    days: int | None = None,
    interval: str | None = None,
) -> Dict[str, Any]:
    settings = get_settings()

    vs_currency = vs_currency or settings.default_vs_currency
    days = days or settings.history_days_default
    interval = interval or settings.history_interval
```

```
        # --- Нормалізація інтервалу під обмеження CoinGecko ---
        # Якщо хочемо погодинні дані, але не маємо Enterprise:
        # - для days 2–90: просто не вказуємо interval → CoinGecko сам дає hourly
        # - для days < 2: піднімаємо до 2, і теж без interval
        interval_param: str | None = interval

        if interval == "hourly":
            interval_param = None  # не передаємо interval в API

        params: Dict[str, Any] = {
            "vs_currency": vs_currency,
            "days": days,
        }
        if interval_param is not None:
            params["interval"] = interval_param

        return self._get(f"/coins/{coin_id}/market_chart", params=params)
```

models/baseline/naive.py

```
from typing import Tuple
import pandas as pd
import numpy as np


def naive_constant_forecast(
    history: pd.DataFrame,
    horizon_hours: int = 24,
) -> Tuple[pd.DataFrame, pd.Series]:
    """
    Дуже простий baseline:

    - беремо останню відому ціну з history['price']
    - прогнозуємо її як константу на наступні horizon_hours годин

    Повертає:
    - DataFrame з колонками ['ts', 'y_pred']
    - окремо Series з останньою історичною ціною (для зручності, але можемо не використовувати)
    """
    if history.empty:
        raise ValueError("History is empty, cannot build forecast")

    last_ts = history["ts"].max()
    last_price = history["price"].iloc[-1]

    # Генеруємо горизонт (тут просто рівномірно по годинах)
    ts_future = pd.date_range(
        start=last_ts + pd.Timedelta(hours=1),
        periods=horizon_hours,
        freq="h",
    )

    y_pred = np.full(shape=horizon_hours, fill_value=last_price, dtype=float)

    df_forecast = pd.DataFrame({"ts": ts_future, "y_pred": y_pred})

    return df_forecast, pd.Series([last_price], index=["last_price"])
```

models/lstm/config.py

```
from dataclasses import dataclass
```

```
from pathlib import Path

PROJECT_ROOT = Path(__file__).resolve().parents[2]
ARTIFACTS_DIR = PROJECT_ROOT / "models" / "artifacts"
ARTIFACTS_DIR.mkdir(parents=True, exist_ok=True)


@dataclass
class LSTMConfig:
    # базові параметри з нашого Colab-патерну
    window_size: int = 48          # довжина history-вікна
    train_ratio: float = 0.8       # частка train

    hidden_size: int = 256
    num_layers: int = 2
    dropout: float = 0.2

    batch_size: int = 64
    num_epochs: int = 200
    learning_rate: float = 3e-4

    target_col: str = "price"

     # Нові параметри:
    early_stopping_patience: int = 40   # скільки епох чекаємо без покращення
    early_stopping_min_delta: float = 1e-5

    lr_reduce_factor: float = 0.5       # у скільки разів зменшувати lr
    lr_reduce_patience: int = 15         # скільки епох без покращення до зменшення lr
    lr_min: float = 1e-6

    def artifact_path(self, coin_id: str, vs_currency: str) -> Path:
        fname = f"lstm_{coin_id}_{vs_currency}.pt"
        return ARTIFACTS_DIR / fname
```

models/lstm/dataset.py

```
from typing import List, Tuple

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import torch
from torch.utils.data import Dataset, DataLoader

from features.transform import build_feature_frame
from models.lstm.config import LSTMConfig


class SequenceDataset(Dataset):
    def __init__(
        self,
        data_scaled: np.ndarray,
        window_size: int,
        target_col_idx: int,
    ) -> None:
        """
        data_scaled: (N, num_features) – вже відмасштабований масив
        window_size: довжина вікна
        target_col_idx: індекс колонки-цілі в data_scaled
        """
```

```
        self.data_scaled = data_scaled
        self.window_size = window_size
        self.target_col_idx = target_col_idx

    def __len__(self) -> int:
        # останній y – на позиції i, де i = len - 1
        # перший доступний i = window_size
        return len(self.data_scaled) - self.window_size

    def __getitem__(self, idx: int):
        x_start = idx
        x_end = idx + self.window_size
        y_idx = x_end  # next step after window

        x = self.data_scaled[x_start:x_end, :]   # (window_size, num_features)
        y = self.data_scaled[y_idx, self.target_col_idx]  # scalar

        x_tensor = torch.tensor(x, dtype=torch.float32)
        y_tensor = torch.tensor([y], dtype=torch.float32)  # (1,)

        return x_tensor, y_tensor


def build_model_frame(df_raw: pd.DataFrame) -> pd.DataFrame:
    """
    Створює df_model для моделі: price + числові фічі з build_feature_frame.

    ВАЖЛИВО:
    - беремо тільки числові колонки (float/int),
    - 'price' ставимо першою,
    - ігноруємо будь-які string-поля (типу coin_id, symbol, vs_currency).
    """
    df_feat = build_feature_frame(df_raw)

    # Вибираємо тільки числові колонки
    numeric_cols = df_feat.select_dtypes(include=["number", "float", "int"]).columns.tolist()

    # 'ts' – не фіча, цю колонку тримаємо окремо
    numeric_cols = [c for c in numeric_cols if c != "ts"]

    if "price" not in numeric_cols:
        raise ValueError("Очікується колонка 'price' у фреймі з фічами (numeric_cols).")

    # price спочатку, потім решта числових фіч
    other_cols = [c for c in numeric_cols if c != "price"]
    ordered_cols = ["price"] + other_cols

    df_model = df_feat[["ts"] + ordered_cols].copy()
    return df_model



def prepare_datasets_and_scaler(
    df_raw: pd.DataFrame,
    config: LSTMConfig,
) -> Tuple[
    DataLoader, DataLoader, MinMaxScaler, List[str], int, np.ndarray, np.ndarray
]:
    """
    Готує train/test DataLoader'и, скейлер і службову інформацію.
```

```
Повертає:
- train_loader
- test_loader
- scaler (MinMaxScaler по всіх фічах)
- feature_cols (список колонок без ts)
- target_col_idx (індекс 'price' в цих фічах)
- train_scaled (масив для можливого аналізу)
- test_scaled (масив для можливого аналізу)
"""
df_model = build_model_frame(df_raw)

# Беремо тільки фічі (без ts)
feature_cols = [c for c in df_model.columns if c != "ts"]
target_col_idx = feature_cols.index(config.target_col)

#  Критично: чистимо NaN перед скейлінгом
df_model_clean = df_model.dropna(subset=feature_cols).reset_index(drop=True)

if len(df_model_clean) <= config.window_size + 1:
    raise RuntimeError(
        f"Замало даних після dropna: {len(df_model_clean)} рядків, "
        f"а window_size={config.window_size}. "
        "Спробуй або зменшити window_size, або збільшити історію."
    )

values = df_model_clean[feature_cols].values.astype(np.float32)

n_total = len(df_model_clean)
split_idx = int(n_total * config.train_ratio)

if split_idx <= config.window_size:
    raise RuntimeError(
        f"split_idx={split_idx} <= window_size={config.window_size}. "
        "Збільште обсяг даних або змініть train_ratio/window_size."
    )

# Як у Colab:
# train = [0 : split_idx]
# test  = [split_idx - window : end] (з нахльостом)
train_values = values[:split_idx]
test_values = values[split_idx - config.window_size :]

scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_values)
test_scaled = scaler.transform(test_values)

train_dataset = SequenceDataset(
    train_scaled,
    window_size=config.window_size,
    target_col_idx=target_col_idx,
)
test_dataset = SequenceDataset(
    test_scaled,
    window_size=config.window_size,
    target_col_idx=target_col_idx,
)

train_loader = DataLoader(
    train_dataset,
```

```
        batch_size=config.batch_size,
        shuffle=True,
        drop_last=True,
    )
    test_loader = DataLoader(
        test_dataset,
        batch_size=config.batch_size,
        shuffle=False,
        drop_last=False,
    )

    return (
        train_loader,
        test_loader,
        scaler,
        feature_cols,
        target_col_idx,
        train_scaled,
        test_scaled,
    )
```

models/lstm/inference.py

```
from datetime import timedelta

import numpy as np
import pandas as pd
import torch

from config.settings import get_settings

from data.db import load_ohlcv_hourly
from data.db import store_hourly_forecast
from data.db import ensure_forecast_continuity

from features.transform import build_feature_frame

from models.lstm.config import LSTMConfig
from models.lstm.model import LSTMForecastModel
from models.lstm.train import _inverse_scale_target


def _to_device():
    return torch.device("cuda" if torch.cuda.is_available() else "cpu")


def load_lstm_checkpoint(coin_id: str, vs_currency: str | None = None):
    settings = get_settings()
    vs_currency = vs_currency or settings.default_vs_currency
    cfg = LSTMConfig()
    path = cfg.artifact_path(coin_id, vs_currency)

    checkpoint = torch.load(path, map_location="cpu", weights_only=False)

    feature_cols = checkpoint["feature_cols"]
    target_col_idx = checkpoint["target_col_idx"]
    scaler = checkpoint["scaler"]
    cfg_loaded = cfg  # можна пізніше зчитати з checkpoint["config"]

    model = LSTMForecastModel(
        input_size=len(feature_cols),
```

```
        hidden_size=cfg_loaded.hidden_size,
        num_layers=cfg_loaded.num_layers,
        dropout=cfg_loaded.dropout,
    )
    model.load_state_dict(checkpoint["state_dict"])
    model.eval()

    return model, scaler, feature_cols, target_col_idx, cfg_loaded

def forecast_from_history(
    coin_id: str,
    history_df: pd.DataFrame,
    vs_currency: str | None = None,
    horizon_hours: int = 24,
) -> pd.DataFrame:
    """
    Рекурсивний прогноз LSTM на horizon_hours кроків вперед,
    використовуючи довільну історію (history_df) як "світ до тепер".

    Використовується в Debugging-режимі, коли ми "відрізаємо" дані на рівні 'вчора'.
    """
    settings = get_settings()
    vs_currency = vs_currency or settings.default_vs_currency

    model, scaler, feature_cols, target_col_idx, cfg = load_lstm_checkpoint(
        coin_id, vs_currency
    )
    device = _to_device()
    model.to(device)

    # Будуємо фічі так само, як при тренуванні
    df_feat = build_feature_frame(history_df)
    df_model = df_feat[["ts"] + feature_cols].copy()

    # Чистимо NaN як у train-пайплайні
    df_model = df_model.dropna(subset=feature_cols).reset_index(drop=True)
    if len(df_model) < cfg.window_size:
        raise RuntimeError(
            f"Замало даних в history_df після dropna: {len(df_model)} рядків, "
            f"потрібно хоча б window_size={cfg.window_size}."
        )

    values = df_model[feature_cols].values.astype(np.float32)
    scaled = scaler.transform(values)

    window = scaled[-cfg.window_size :, :]  # останнє вікно
    last_ts = df_model["ts"].max()

    preds_scaled = []
    ts_future = []

    with torch.no_grad():
        for step in range(horizon_hours):
            x = torch.tensor(window[None, :, :], dtype=torch.float32, device=device)
            y_scaled = model(x).cpu().numpy()[0, 0]

            preds_scaled.append(y_scaled)

            # оновлюємо вікно: зсуваємо і додаємо новий рядок
            new_row = np.zeros((window.shape[1],), dtype=np.float32)
```

```
            new_row[target_col_idx] = y_scaled

            window = np.vstack([window[1:], new_row])
            ts_future.append(last_ts + timedelta(hours=step + 1))

    preds_scaled_arr = np.array(preds_scaled, dtype=np.float32)

    from models.lstm.train import _inverse_scale_target

    y_pred = _inverse_scale_target(
        scaler, feature_cols, target_col_idx, preds_scaled_arr
    )

    df_forecast = pd.DataFrame(
        {
            "ts": ts_future,
            "y_pred": y_pred,
        }
    )
    return df_forecast


def forecast_next_t1_and_store(
    coin_id: str,
    vs_currency: str | None = None,
    model_name: str = "lstm_v1.0",
):
    """
    1) Завантажує останні дані (OHLCV)
    2) Будує фічі
    3) Беремо останнє історичне вікно (window_size)
    4) Проганяємо через LSTM -> отримуємо y_pred(t+1)
    5) Зберігаємо результат у forecast_hourly
    """
    settings = get_settings()
    vs_currency = vs_currency or settings.default_vs_currency

    ensure_forecast_continuity(coin_id, vs_currency, model_name)

    # 1) Завантажуємо сирі дані
    df_raw = load_ohlcv_hourly(coin_id, vs_currency)
    if df_raw.empty:
        raise RuntimeError(
            f"Немає OHLCV для {coin_id} ({vs_currency}). Спершу запусти jobs.fetch_history."
        )

    # 2) Готуємо фічі
    df_feat = build_feature_frame(df_raw)
    cfg = LSTMConfig()

    # Колонки фіч – усі, крім ts
    feature_cols = [c for c in df_feat.columns if c not in ("ts",)]
    if cfg.target_col not in feature_cols:
        raise RuntimeError("В df_feat немає 'price' як таргета!")

    df_model = df_feat[["ts"] + feature_cols].dropna(subset=feature_cols).reset_index(drop=True)
    if df_model.empty:
        raise RuntimeError("Після dropna df_model порожній.")

    # 3) Завантажуємо модель + scaler + параметри
```

```
try:
    model, scaler, trained_feature_cols, target_col_idx, cfg_loaded = load_lstm_checkpoint(
        coin_id, vs_currency
    )
except Exception as e:
    raise RuntimeError(f"Не вдалося завантажити LSTM-модель: {e}")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# Перевіряємо фічі (в тренованій моделі порядок може бути іншим)
missing = [c for c in trained_feature_cols if c not in df_model.columns]
if missing:
    raise RuntimeError(
        "У df_model не вистачає фіч, на яких тренувалась модель:\n" + ", ".join(missing)
    )

# Перебудовуємо df_model відповідно до порядку фіч моделі
df_model = df_model[["ts"] + trained_feature_cols]

# Масштабуємо всі фічі (але без fit!)
values = df_model[trained_feature_cols].values.astype(np.float32)
scaled = scaler.transform(values)

# 4) Витягуємо останнє вікно
if len(df_model) < cfg.window_size:
    raise RuntimeError(
        f"Замало даних: {len(df_model)} рядків, потрібно {cfg.window_size}."
    )

window_scaled = scaled[-cfg.window_size :, :]  # (W, F)
x = torch.tensor(window_scaled[None, :, :], dtype=torch.float32, device=device)

# Прогноз (в scaled-space)
with torch.no_grad():
    y_scaled = model(x).cpu().numpy()[0, 0]

# 5) Інверсія масштабу
y_pred = _inverse_scale_target(
    scaler,
    trained_feature_cols,
    target_col_idx,
    np.array([y_scaled]),
)[0]

# Часові мітки
ts_anchor = df_model["ts"].max()
ts_forecast = ts_anchor + timedelta(hours=1)

# 6) Зберігаємо в БД
store_hourly_forecast(
    coin_id=coin_id,
    vs_currency=vs_currency,
    ts_anchor=ts_anchor,
    ts_forecast=ts_forecast,
    model=model_name,
    y_pred=float(y_pred),
    is_backfilled=False,
)
```

```
    return {
        "coin_id": coin_id,
        "vs_currency": vs_currency,
        "ts_anchor": ts_anchor,
        "ts_forecast": ts_forecast,
        "y_pred": y_pred,
    }
```

models/lstm/model.py

```
import torch
from torch import nn


class LSTMForecastModel(nn.Module):
    """
    Простий LSTM → Linear для one-step ahead прогнозу.
    Формат вхідних даних: (batch, seq_len, input_size)
    Вихід: (batch, 1)
    """

    def __init__(
        self,
        input_size: int,
        hidden_size: int,
        num_layers: int = 2,
        dropout: float = 0.1,
    ) -> None:
        super().__init__()

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0.0,
        )

        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        # x: (batch, seq_len, input_size)
        out, _ = self.lstm(x)      # out: (batch, seq_len, hidden_size)
        last_hidden = out[:, -1, :]  # беремо останній timestep
        y_hat = self.fc(last_hidden) # (batch, 1)
        return y_hat
```

models/lstm/train.py

```
from pathlib import Path
from typing import Dict, Any

import numpy as np
import torch
from torch import nn
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import ReduceLROnPlateau  #  новий імпорт

from config.settings import get_settings
from data.db import load_ohlcv_hourly
```

```
from models.lstm.config import LSTMConfig
from models.lstm.dataset import prepare_datasets_and_scaler
from models.lstm.model import LSTMForecastModel


def _to_device():
    return torch.device("cuda" if torch.cuda.is_available() else "cpu")


def _train_one_epoch(
    model: nn.Module,
    loader: DataLoader,
    criterion: nn.Module,
    optimizer: torch.optim.Optimizer,
    device: torch.device,
) -> float:
    model.train()
    total_loss = 0.0
    n_batches = 0

    for x_batch, y_batch in loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        optimizer.zero_grad()
        y_pred = model(x_batch).squeeze(-1)   # (batch,)
        y_true = y_batch.squeeze(-1)          # (batch,)

        loss = criterion(y_pred, y_true)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        n_batches += 1

    return total_loss / max(n_batches, 1)


def _evaluate(
    model: nn.Module,
    loader: DataLoader,
    criterion: nn.Module,
    device: torch.device,
) -> float:
    model.eval()
    total_loss = 0.0
    n_batches = 0

    with torch.no_grad():
        for x_batch, y_batch in loader:
            x_batch = x_batch.to(device)
            y_batch = y_batch.to(device)

            y_pred = model(x_batch).squeeze(-1)
            y_true = y_batch.squeeze(-1)

            loss = criterion(y_pred, y_true)

            total_loss += loss.item()
            n_batches += 1
```

```
    return total_loss / max(n_batches, 1)


def _inverse_scale_target(
    scaler,
    feature_cols,
    target_col_idx: int,
    y_scaled: np.ndarray,
) -> np.ndarray:
    """
    Переносимо y_scaled (N,) назад в оригінальний масштаб через scaler.inverse_transform.

    Робимо трюк:
    - створюємо нульову матрицю (N, num_features)
    - в колонку target_col_idx кладемо y_scaled
    - ганяємо через inverse_transform
    - беремо ту ж саму колонку
    """
    num_features = len(feature_cols)
    tmp = np.zeros((len(y_scaled), num_features), dtype=np.float32)
    tmp[:, target_col_idx] = y_scaled

    inv = scaler.inverse_transform(tmp)
    y_inv = inv[:, target_col_idx]
    return y_inv


def train_lstm_for_coin(
    coin_id: str,
    vs_currency: str = None,
    config: LSTMConfig | None = None,
) -> Dict[str, Any]:
    """
    Повний цикл:
    - завантажує дані з DuckDB
    - будує фічі
    - готує датасети
    - тренує LSTM
    - рахує тестові MAE/RMSE в оригінальному масштабі ціни
    - зберігає чекпоінт
    """
    settings = get_settings()
    vs_currency = vs_currency or settings.default_vs_currency
    cfg = config or LSTMConfig()

    print(f" Завантажуємо дані для {coin_id} ({vs_currency})...")
    df_raw = load_ohlcv_hourly(coin_id, vs_currency)

    if df_raw.empty:
        raise RuntimeError(
            f"У DuckDB немає даних для монети '{coin_id}' ({vs_currency}). "
            f"Спочатку запусти jobs.fetch_history."
        )

    print(" Готуємо датасети та скейлер...")
    (
        train_loader,
        test_loader,
        scaler,
```

```
    feature_cols,
    target_col_idx,
    train_scaled,
    test_scaled,
) = prepare_datasets_and_scaler(df_raw, cfg)

input_size = len(feature_cols)
device = _to_device()
print(f" Пристрій: {device}, input_size={input_size}")

model = LSTMForecastModel(
    input_size=input_size,
    hidden_size=cfg.hidden_size,
    num_layers=cfg.num_layers,
    dropout=cfg.dropout,
).to(device)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=cfg.learning_rate)

#  Шедулер для зменшення lr, коли val_loss виходить на плато
scheduler = ReduceLROnPlateau(
    optimizer,
    mode="min",
    factor=getattr(cfg, "lr_reduce_factor", 0.5),
    patience=getattr(cfg, "lr_reduce_patience", 5),
    min_lr=getattr(cfg, "lr_min", 1e-5),
)

best_val_loss = float("inf")
best_state_dict = None
no_improve_epochs = 0
early_patience = getattr(cfg, "early_stopping_patience", 15)
early_min_delta = getattr(cfg, "early_stopping_min_delta", 1e-4)

# --- Тренування ---
print(" Починаємо тренування LSTM...")
for epoch in range(1, cfg.num_epochs + 1):
    train_loss = _train_one_epoch(model, train_loader, criterion, optimizer, device)
    val_loss = _evaluate(model, test_loader, criterion, device)

    # оновлюємо lr відносно val_loss
    scheduler.step(val_loss)

    print(
        f"[Epoch {epoch:03d}/{cfg.num_epochs}] "
        f"train_loss={train_loss:.6f}, val_loss={val_loss:.6f}"
    )

    # --- EarlyStopping логіка ---
    if val_loss + early_min_delta < best_val_loss:
        best_val_loss = val_loss
        best_state_dict = model.state_dict()
        no_improve_epochs = 0
    else:
        no_improve_epochs += 1

    if no_improve_epochs >= early_patience:
        print(
            f" EarlyStopping: не було покращення val_loss "
```

```
            f"{no_improve_epochs} епох поспіль. Зупиняємося на епосі {epoch}."
        )
        break

# Після циклу – відкатитися до найкращої версії моделі
if best_state_dict is not None:
    model.load_state_dict(best_state_dict)
    print(f" Відновлено ваги з найкращим val_loss={best_val_loss:.6f}")
else:
    print(" best_state_dict порожній – зберігаємо модель з останньої епохи.")

# --- Оцінка на тесті в оригінальному масштабі ---
print(" Оцінюємо модель на тесті...")
model.eval()

y_true_scaled_list = []
y_pred_scaled_list = []

with torch.no_grad():
    for x_batch, y_batch in test_loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        y_pred = model(x_batch).squeeze(-1)
        y_true = y_batch.squeeze(-1)

        y_true_scaled_list.append(y_true.cpu().numpy())
        y_pred_scaled_list.append(y_pred.cpu().numpy())

y_true_scaled = np.concatenate(y_true_scaled_list, axis=0)
y_pred_scaled = np.concatenate(y_pred_scaled_list, axis=0)

y_true = _inverse_scale_target(scaler, feature_cols, target_col_idx, y_true_scaled)
y_pred = _inverse_scale_target(scaler, feature_cols, target_col_idx, y_pred_scaled)

mae = np.mean(np.abs(y_true - y_pred))
rmse = np.sqrt(np.mean((y_true - y_pred) ** 2))

print(f" Test MAE  = {mae:.4f} {vs_currency.upper()}")
print(f" Test RMSE = {rmse:.4f} {vs_currency.upper()}")

# --- Зберігаємо чекпоінт ---
artifact_path: Path = cfg.artifact_path(coin_id, vs_currency)
checkpoint = {
    "config": cfg.__dict__,
    "feature_cols": feature_cols,
    "target_col_idx": target_col_idx,
    "state_dict": model.state_dict(),
    "scaler": scaler,
}

torch.save(checkpoint, artifact_path)
print(f" Модель збережено в: {artifact_path}")

return {
    "artifact_path": artifact_path,
    "mae": float(mae),
    "rmse": float(rmse),
    "config": cfg,
}
```

ui/tabs/data_tab.py

```
import pandas as pd
import plotly.express as px
import streamlit as st

from config.settings import get_settings
from data.db import load_ohlcv_hourly

@st.cache_data(show_spinner=False)
def load_market_data(coin_id: str, vs_currency: str) -> pd.DataFrame:
    """
    Завантажує ринкові дані з DuckDB для обраної монети та валюти.
    Кешується Streamlit-ом.
    """
    df = load_ohlcv_hourly(coin_id, vs_currency)
    return df


def render_data_tab():
    settings = get_settings()

    # --- Вибір валюти (поки лише одна, але можна розширити) ---
    vs_currency = settings.default_vs_currency

    # --- Вибір монети ---
    selected_coin_id = st.session_state.get("selected_coin_id", "bitcoin")
    selected_label = st.session_state.get("selected_coin_label", "Bitcoin (BTC)")

    st.markdown("""
<style>
        /* Remove blank space at top and bottom */
        .block-container {
            padding-top: 0rem;
            padding-bottom: 0rem;
        }
</style>
""", unsafe_allow_html=True)

    st.markdown(
    """
    <h1 style="text-align: center; margin-top: 0;">
        Data
    </h1>
    """,
    unsafe_allow_html=True
)



    # --- Завантажуємо дані з DuckDB ---
    df = load_market_data(selected_coin_id, vs_currency)

    if df.empty:
        st.warning(
            "У базі немає даних для цієї монети. "
            "Спочатку запусти job для завантаження історії:\n\n"
            "`python -m jobs.fetch_history`"
        )
        return
```

```
    # Обмежуємо діапазон дат згідно з глобальним налаштуванням
    time_range_hours = st.session_state.get("time_range_hours")

    if time_range_hours is not None and not df.empty:
        cutoff_ts = df["ts"].max() - pd.Timedelta(hours=time_range_hours)
        df = df[df["ts"] >= cutoff_ts]

    # --- Базовий графік ціни ---
    fig_price = px.line(
        df,
        x="ts",
        y="price",
        title=f"Price {selected_label} - {vs_currency.upper()}",
    )
    fig_price.update_layout(
        xaxis_title="Time",
        yaxis_title=f"Price ({vs_currency.upper()})",
        height=500,
    )

    st.plotly_chart(fig_price, width="stretch")

    # --- Опціонально: додаткова інформація / таблиця ---
    with st.expander(f"Table Data for {selected_label}"):
        st.dataframe(
            df.sort_values("ts", ascending=False),
            width="stretch",
            height=400,
        )
```

ui/tabs/features_tab.py

```
import pandas as pd
import plotly.express as px
import streamlit as st

from config.settings import get_settings
from features.transform import build_feature_frame
from ui.constants import TRACKED_COINS
from ui.tabs.data_tab import load_market_data


@st.cache_data(show_spinner=False)
def compute_features(
    coin_id: str,
    vs_currency: str,
) -> pd.DataFrame:
    """
    Обчислює технічні фічі поверх історичних даних.
    """
    df = load_market_data(coin_id, vs_currency)
    if df.empty:
        return df

    feat_df = build_feature_frame(df)
    return feat_df


def render_features_tab():
    settings = get_settings()
```

```
    vs_currency = settings.default_vs_currency

    st.markdown("""
<style>
        /* Remove blank space at top and bottom */
        .block-container {
            padding-top: 0rem;
            padding-bottom: 0rem;
        }
</style>
""", unsafe_allow_html=True)

    st.markdown(
    """
    <h1 style="text-align: center; margin-top: 0;">
        Features
    </h1>
    """,
    unsafe_allow_html=True
)
    # Використовуємо глобально обрану монету з session_state
    labels = [label for label, _ in TRACKED_COINS]
    ids = [cid for _, cid in TRACKED_COINS]

    default_index = ids.index("bitcoin") if "bitcoin" in ids else 0
    default_label = labels[default_index]
    default_id = ids[default_index]

    selected_coin_id = st.session_state.get("selected_coin_id", default_id)
    selected_label = st.session_state.get("selected_coin_label", default_label)

    # Завантажуємо дані + фічі
    df_raw = load_market_data(selected_coin_id, vs_currency)
    if df_raw.empty:
        st.warning(
            "У базі немає даних для цієї монети. "
            "Спочатку запусти job для завантаження історії:\n\n"
            "`python -m jobs.fetch_history`"
        )
        return

    df_feat = compute_features(selected_coin_id, vs_currency)

    # --- 2x2 grid з графіками ---
    st.subheader(f"Metrics for {selected_label}")

    row1_col1, row1_col2 = st.columns(2)
    row2_col1, row2_col2 = st.columns(2)

    # 1) Ціна + SMA
    with row1_col1:
        st.markdown("**Price + SMA**")
        fig_price = px.line(
            df_feat,
            x="ts",
            y=["price", "sma_5", "sma_20", "sma_50"],
            labels={"value": f"Price / SMA ({vs_currency.upper()})", "ts": "Time"},
        )
        fig_price.update_layout(height=350, legend_title_text="Серія")
        st.plotly_chart(fig_price, width="stretch")
```

```
    # 2) Returns
    with row1_col2:
        st.markdown("**Returns**")
        fig_ret = px.line(
            df_feat,
            x="ts",
            y="return",
            labels={"return": "Return", "ts": "Time"},
        )
        fig_ret.update_layout(height=350)
        st.plotly_chart(fig_ret, width="stretch")

    # 3) Волатильність
    vol_col = "volatility_24"
    with row2_col1:
        st.markdown("**Volatility (24 points)**")
        if vol_col in df_feat.columns:
            fig_vol = px.line(
                df_feat,
                x="ts",
                y=vol_col,
                labels={vol_col: "Volatility (σ)", "ts": "Time"},
            )
            fig_vol.update_layout(height=350)
            st.plotly_chart(fig_vol, width="stretch")
        else:
            st.info("Volatility not calculated yet.")

    # 4) Об’єм
    with row2_col2:
        st.markdown("**Volume**")
        fig_volm = px.line(
            df_feat,
            x="ts",
            y="volume",
            labels={"volume": "Volume", "ts": "Time"},
        )
        fig_volm.update_layout(height=350)
        st.plotly_chart(fig_volm, width="stretch")

    with st.expander("Features Table"):
        st.dataframe(
            df_feat.sort_values("ts", ascending=False),
            width="stretch",
            height=400,
        )
```

ui/tabs/forecast_tab.py

```
import pandas as pd
import plotly.express as px
import streamlit as st

from config.settings import get_settings
from data.db import load_ohlcv_hourly, load_hourly_forecasts
from ui.constants import TRACKED_COINS
```

```
def render_forecast_tab():
    settings = get_settings()
    vs_currency = settings.default_vs_currency

    st.markdown("""
<style>
        /* Remove blank space at top and bottom */
        .block-container {
            padding-top: 0rem;
            padding-bottom: 0rem;
        }
</style>
""", unsafe_allow_html=True)

    st.markdown(
    """
    <h1 style="text-align: center; margin-top: 0;">
        Forecast
    </h1>
    """,
    unsafe_allow_html=True
)

    # --- Використовуємо глобально обрану монету ---
    labels = [label for label, _ in TRACKED_COINS]
    ids = [cid for _, cid in TRACKED_COINS]

    default_index = ids.index("bitcoin") if "bitcoin" in ids else 0
    default_label = labels[default_index]
    default_id = ids[default_index]

    selected_coin_id = st.session_state.get("selected_coin_id", default_id)
    selected_label = st.session_state.get("selected_coin_label", default_label)

    # --- Завантажуємо фактичні дані ---
    df_price = load_ohlcv_hourly(selected_coin_id, vs_currency)

    if df_price.empty:
        st.warning(
            "У базі немає фактичних даних для цієї монети. "
            "Спочатку запусти:\n\n"
            "`python -m jobs.fetch_history`"
        )
        return

    # --- Завантажуємо прогнози ---
    df_fc = load_hourly_forecasts(
        selected_coin_id,
        vs_currency,
        limit=200,
        model="lstm_v1.0",
    )

    df_fc_gru = load_hourly_forecasts(
        selected_coin_id,
        vs_currency,
        limit=200,
        model="gru_v1.0",
    )
```

```
if df_fc.empty:
    st.warning(
        "У таблиці forecast_hourly немає прогнозів для цієї монети.\n\n"
        "Спочатку натренуй модель та запусти:\n\n"
        "`python -m jobs.train_lstm_all`\n"
        "`python -m jobs.run_forecast`"
    )
    return

# --- Обмеження по часовому діапазону ---
time_range_hours = st.session_state.get("time_range_hours")

if time_range_hours is not None:
    # Для цін
    if not df_price.empty:
        cutoff_price = df_price["ts"].max() - pd.Timedelta(hours=time_range_hours)
        df_price = df_price[df_price["ts"] >= cutoff_price]

    # Для LSTM-прогнозів
    if not df_fc.empty:
        cutoff_fc = df_fc["ts_forecast"].max() - pd.Timedelta(hours=time_range_hours)
        df_fc = df_fc[df_fc["ts_forecast"] >= cutoff_fc]

    # Для GRU-прогнозів (якщо є)
    if df_fc_gru is not None and not df_fc_gru.empty:
        cutoff_fc_gru = df_fc_gru["ts_forecast"].max() - pd.Timedelta(hours=time_range_hours)
        df_fc_gru = df_fc_gru[df_fc_gru["ts_forecast"] >= cutoff_fc_gru]


# --- Останній факт та останній прогноз ---
last_fact = df_price.iloc[-1]
last_fc = df_fc.iloc[-1]

last_price = float(last_fact["price"])
last_price_ts = last_fact["ts"]

y_pred = float(last_fc["y_pred"])
ts_forecast = last_fc["ts_forecast"]
ts_anchor = last_fc["ts_anchor"]

delta_abs = y_pred - last_price
delta_pct = (delta_abs / last_price) * 100 if last_price != 0 else 0.0

col_fact, col_fc = st.columns(2)



# --- Графік: фактична ціна + прогнози t+1 ---
# Беремо глобальний часовий діапазон із sidebar
time_range_hours = st.session_state.get("time_range_hours")

if df_price.empty:
    st.info("Немає цінових даних для побудови графіка")
    return

ts_max = df_price["ts"].max()

if time_range_hours is None:
    # "Увесь період" – показуємо всі доступні дані
```

```
    ts_min = df_price["ts"].min()
else:
    ts_min = ts_max - pd.Timedelta(hours=time_range_hours)

# --- Фактична ціна ---
df_price_plot = df_price[df_price["ts"] >= ts_min].copy()
df_price_plot["series"] = "Real"
df_price_plot["ts_plot"] = df_price_plot["ts"]

df_plot_all = df_price_plot[["ts_plot", "price", "series"]].copy()

# LSTM
df_fc_plot = df_fc[df_fc["ts_forecast"] >= ts_min].copy()
if not df_fc_plot.empty:
    df_fc_plot["series"] = "LSTM"
    df_fc_plot["ts_plot"] = df_fc_plot["ts_forecast"]
    df_fc_plot = df_fc_plot.rename(columns={"y_pred": "price"})
    df_plot_all = pd.concat(
        [df_fc_plot[["ts_plot", "price", "series"]], df_plot_all],
        ignore_index=True,
    )

# GRU
df_fc_gru_plot = df_fc_gru[df_fc_gru["ts_forecast"] >= ts_min].copy()
if not df_fc_gru_plot.empty:
    df_fc_gru_plot["series"] = "GRU"
    df_fc_gru_plot["ts_plot"] = df_fc_gru_plot["ts_forecast"]
    df_fc_gru_plot = df_fc_gru_plot.rename(columns={"y_pred": "price"})
    df_plot_all = pd.concat(
        [df_fc_gru_plot[["ts_plot", "price", "series"]], df_plot_all],
        ignore_index=True,
    )

# Малюємо
fig = px.line(
    df_plot_all,
    x="ts_plot",
    y="price",
    color="series",
    title=f"Price {selected_label} - {vs_currency.upper()}",
    labels={
        "ts_plot": "Time",
        "price": f"Price ({vs_currency.upper()})",
        "series": "Series",
    },
)
fig.update_layout(height=500)

st.plotly_chart(fig, width="stretch")

with st.expander("Show extended metrics"):
    st.subheader("Last Real Price")
    st.write(f"**Time:** {last_price_ts}")
    st.write(f"**Price:** {last_price:,.2f} {vs_currency.upper()}")

    st.subheader("Last Prediction LSTM")
    st.write(f"**ts_anchor (Last knows Price):** {ts_anchor}")
    st.write(f"**ts_forecast (t+1):** {ts_forecast}")
    st.write(f"**Prediction:** {y_pred:,.2f} {vs_currency.upper()}")
    st.write(
```

```
                f"**Δ to last price:** "
                f"{delta_abs:+.2f} {vs_currency.upper()} "
                f"({delta_pct:+.2f}%)"
                )
            if not df_fc_gru.empty:
                last_gru = df_fc_gru.iloc[-1]
                y_pred_gru = float(last_gru["y_pred"])
                ts_forecast_gru = last_gru["ts_forecast"]

                st.subheader("Last Prediction GRU")
                st.write(f"**ts_forecast (t+1):** {ts_forecast_gru}")
                st.write(f"**Forecast:** {y_pred_gru:,.2f} {vs_currency.upper()}")

                st.markdown("---")

        # --- Таблиця прогнозів ---
        with st.expander("Forecast Logs LSTM"):
            if df_fc.empty:
                st.info("No LSTM Forecasts for this coin")
            else:
                st.dataframe(
                    df_fc.sort_values("ts_forecast", ascending=False),
                    width="stretch",
                    height=400,
                )
        with st.expander("Forecast Logs GRU"):
            if df_fc_gru.empty:
                st.info("No GRU Forecasts for this coin")
            else:
                st.dataframe(
                    df_fc_gru.sort_values("ts_forecast", ascending=False),
                    width="stretch",
                    height=400,
                )
```

ui/tabs/debugging_tab.py

```
# ui/tabs/debugging.py

import numpy as np
import pandas as pd
import plotly.express as px
import streamlit as st
import torch

from config.settings import get_settings
from data.db import load_ohlcv_hourly
from features.transform import build_feature_frame

from models.baseline import naive_constant_forecast
from models.lstm.inference import load_lstm_checkpoint
from models.lstm.train import _inverse_scale_target
from models.gru.inference import load_gru_checkpoint
from models.gru.config import GRUConfig


from ui.constants import TRACKED_COINS
```

```
def render_debugging_tab():
    settings = get_settings()
    vs_currency = settings.default_vs_currency

    st.markdown("""
<style>
        /* Remove blank space at top and bottom */
        .block-container {
            padding-top: 0rem;
            padding-bottom: 0rem;
        }
</style>
""", unsafe_allow_html=True)

    st.markdown(
    """
    <h1 style="text-align: center; margin-top: 0;">
        Debugging
    </h1>
    """,
    unsafe_allow_html=True
)
    # Використовуємо глобально обрану монету + локальний вибір моделі
    labels = [label for label, _ in TRACKED_COINS]
    ids = [cid for _, cid in TRACKED_COINS]
    default_index = ids.index("bitcoin") if "bitcoin" in ids else 0
    default_label = labels[default_index]
    default_id = ids[default_index]

    selected_coin_id = st.session_state.get("selected_coin_id", default_id)
    selected_label = st.session_state.get("selected_coin_label", default_label)


    model_choice = st.radio(
        "Модель:",
        options=["Baseline", "LSTM", "GRU"],
        horizontal=True,
        key="debug_model_choice",
    )

    # Завантажуємо дані з DuckDB
    df_raw = load_ohlcv_hourly(selected_coin_id, vs_currency)

    if df_raw.empty:
        st.warning(
            "У базі немає даних для цієї монети. "
            "Спочатку запусти job для завантаження історії:\n\n"
            "`python -m jobs.fetch_history`"
        )
        return

    # Нормалізуємо timestamps до цілої години
    df_raw = df_raw.copy()
    df_raw["ts_hour"] = df_raw["ts"].dt.floor("h")

    # Один запис на годину
    df_hourly = (
        df_raw.sort_values("ts_hour")
        .drop_duplicates(subset=["ts_hour"], keep="last")
        .reset_index(drop=True)
```

```
)

if len(df_hourly) < 24 * 3:
    st.warning(
        "Замало даних для адекватного backtest'у (потрібно хоча б 3 дні "
        "з погодинними даними). Спробуй завантажити більший інтервал історії."
    )
    return

max_hour = df_hourly["ts_hour"].max()
anchor_hour = max_hour - pd.Timedelta(hours=24)

# Історія до anchor_hour (включно)
df_history = df_hourly[df_hourly["ts_hour"] <= anchor_hour].copy()

# Факт на 24 години після anchor_hour
df_future_true = df_hourly[
    (df_hourly["ts_hour"] > anchor_hour)
    & (df_hourly["ts_hour"] <= anchor_hour + pd.Timedelta(hours=24))
].copy()

if len(df_future_true) < 1:
    st.warning(
        "Не вдалося знайти дані після 'вчора' для побудови backtest'у. "
        "Можливо, історія ще не повна."
    )
    return

# ---------- BASELINE ВАРІАНТ (як був) ----------
if model_choice == "Baseline":
    hist_for_model = df_history.sort_values("ts_hour").copy()
    hist_for_model["ts"] = hist_for_model["ts_hour"]

    try:
        df_forecast, _ = naive_constant_forecast(
            history=hist_for_model,
            horizon_hours=len(df_future_true),
        )
    except Exception as e:
        st.error(f"Помилка під час побудови baseline-прогнозу: {e}")
        return

    model_name = "Baseline (naive constant)"

# ---------- LSTM: teacher forcing 1-step backtest на 'вчора' ----------
elif model_choice == "LSTM":
    try:
        # вантажимо модель + scaler + список фіч
        model, scaler, feature_cols, target_col_idx, cfg = load_lstm_checkpoint(
            selected_coin_id, vs_currency
        )
    except FileNotFoundError:
        st.error(
            "Не знайдено збережену LSTM-модель для цієї монети.\n\n"
            "Спочатку натренуй її командою:\n\n"
            f"`python -m jobs.train_lstm --coin_id {selected_coin_id}`"
        )
        return
    except Exception as e:
        st.error(f"Помилка при завантаженні LSTM-моделі: {e}")
```

```
    return

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# Для фіч: використовуємо ts_hour як еталонний час
df_feat_input = df_hourly.copy()
df_feat_input["ts"] = df_feat_input["ts_hour"]

df_feat = build_feature_frame(df_feat_input)

# df_model: тільки ts + ті фічі, на яких навчалась модель
missing = [c for c in feature_cols if c not in df_feat.columns]
if missing:
    st.error(
        "В поточному фреймі фіч не вистачає колонок, "
        "на яких навчалась модель:\n\n"
        + ", ".join(missing)
    )
    return

df_model = df_feat[["ts"] + feature_cols].copy()
df_model = df_model.dropna(subset=feature_cols).reset_index(drop=True)

if len(df_model) <= cfg.window_size + 1:
    st.error(
        f"Замало даних для побудови вікон: {len(df_model)} рядків після dropna, "
        f"потрібно хоча б window_size={cfg.window_size}."
    )
    return

# Масштабуємо всі фічі тим самим scaler'ом, що був на train
values = df_model[feature_cols].values.astype(np.float32)
scaled_all = scaler.transform(values)

# Створюємо мапу ts -> індекс у df_model
ts_series = df_model["ts"]
ts_to_idx = {ts: idx for idx, ts in enumerate(ts_series)}

# Цільові точки прогнозу: (anchor, anchor + 24h], з кроком 1 година
ts_start = anchor_hour + pd.Timedelta(hours=1)   # anchor+1
ts_end = anchor_hour + pd.Timedelta(hours=len(df_future_true))

target_ts_list = []
target_indices = []

for ts in ts_series:
    if ts_start <= ts <= ts_end:
        idx = ts_to_idx[ts]
        if idx >= cfg.window_size:
            target_ts_list.append(ts)
            target_indices.append(idx)

if not target_indices:
    st.error(
        "Не вдалося знайти достатньо точок для побудови вікон LSTM "
        "на 'вчорашній' добі. Можливо, замало даних після dropna."
    )
    return
```

```
    # One-step ahead прогнози з teacher forcing:
    # для кожного t_pred беремо реальне вікно [t_pred-window_size .. t_pred-1]
    preds_scaled = []

    with torch.no_grad():
        for idx in target_indices:
            window_scaled = scaled_all[idx - cfg.window_size : idx, :]  # (W, F)
            x = torch.tensor(
                window_scaled[None, :, :],
                dtype=torch.float32,
                device=device,
            )
            y_scaled = model(x).cpu().numpy()[0, 0]
            preds_scaled.append(y_scaled)

    preds_scaled_arr = np.array(preds_scaled, dtype=np.float32)

    # Інверсія масштабу для таргета
    y_pred = _inverse_scale_target(
        scaler,
        feature_cols,
        target_col_idx,
        preds_scaled_arr,
    )

    # Реальні ціни (таргет) на ці самі моменти часу
    y_true = (
        df_model.loc[target_indices, cfg.target_col]
        .to_numpy(dtype=float)
    )

    df_forecast = pd.DataFrame(
        {
            "ts": target_ts_list,
            "y_pred": y_pred,
        }
    )

    # Для коректного мерджу з df_future_true працюємо через ts_hour
    df_forecast["ts_hour"] = df_forecast["ts"].dt.floor("h")

    model_name = "LSTM"

# ---------- GRU: teacher forcing 1-step backtest ----------
elif model_choice == "GRU":
    try:
        model, scaler, feature_cols, target_col_idx, cfg = load_gru_checkpoint(
            selected_coin_id, vs_currency
        )
    except FileNotFoundError:
        st.error(
            "Не знайдено збережену GRU-модель для цієї монети.\n\n"
            "Спочатку натренуй її командою:\n\n"
            f"`python -m jobs.train_gru --coin_id {selected_coin_id}`"
        )
        return
    except Exception as e:
        st.error(f"Помилка при завантаженні GRU-моделі: {e}")
        return
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

df_feat_input = df_hourly.copy()
df_feat_input["ts"] = df_feat_input["ts_hour"]
df_feat = build_feature_frame(df_feat_input)

missing = [c for c in feature_cols if c not in df_feat.columns]
if missing:
    st.error(
        "В поточному фреймі фіч не вистачає колонок, "
        "на яких навчалась GRU:\n\n" + ", ".join(missing)
    )
    return

df_model = df_feat[["ts"] + feature_cols].dropna().reset_index(drop=True)

values = df_model[feature_cols].values.astype(np.float32)
scaled_all = scaler.transform(values)

ts_series = df_model["ts"]
ts_to_idx = {ts: idx for idx, ts in enumerate(ts_series)}

ts_start = anchor_hour + pd.Timedelta(hours=1)
ts_end = anchor_hour + pd.Timedelta(hours=len(df_future_true))

target_ts_list = []
target_indices = []

for ts in ts_series:
    if ts_start <= ts <= ts_end:
        idx = ts_to_idx[ts]
        if idx >= cfg.window_size:
            target_ts_list.append(ts)
            target_indices.append(idx)

preds_scaled = []

with torch.no_grad():
    for idx in target_indices:
        window_scaled = scaled_all[idx - cfg.window_size : idx, :]
        x = torch.tensor(
            window_scaled[None, :, :], dtype=torch.float32, device=device
        )
        y_scaled = model(x).cpu().numpy()[0, 0]
        preds_scaled.append(y_scaled)

preds_scaled_arr = np.array(preds_scaled, dtype=np.float32)

y_pred = _inverse_scale_target(
    scaler, feature_cols, target_col_idx, preds_scaled_arr
)

y_true = df_model.loc[target_indices, cfg.target_col].to_numpy(float)

df_forecast = pd.DataFrame(
    {"ts": target_ts_list, "y_pred": y_pred}
)
```

```
    df_forecast["ts_hour"] = df_forecast["ts"].dt.floor("h")

    model_name = "GRU"


# ---------- Спільна частина: метрики, графік, таблиця ----------

# Нормалізуємо час у прогнозі та об'єднуємо по ts_hour
if model_choice == "Baseline":
    df_forecast = df_forecast.copy()
    df_forecast["ts_hour"] = df_forecast["ts"].dt.floor("h")

df_merged = pd.merge(
    df_future_true[["ts_hour", "price"]],
    df_forecast[["ts_hour", "y_pred"]],
    on="ts_hour",
    how="inner",
)

if df_merged.empty:
    st.warning(
        "Не вдалося зіставити фактичні та прогнозні значення по годинах. "
        "Перевір, чи дані мають погодинну частоту."
    )
    return

# Метрики
y_true_merge = df_merged["price"]
y_pred_merge = df_merged["y_pred"]

mae = (y_true_merge - y_pred_merge).abs().mean()
rmse = ((y_true_merge - y_pred_merge) ** 2).mean() ** 0.5

st.subheader(f"Metrics for {model_name} on {selected_label}")
st.write(
    f"**MAE:** {mae:.4f} {vs_currency.upper()}  \n"
    f"**RMSE:** {rmse:.4f} {vs_currency.upper()}"
)

# Графік
ctx_hours = 24
ts_min_plot = anchor_hour - pd.Timedelta(hours=ctx_hours)

df_plot_hist = df_hourly[
    (df_hourly["ts_hour"] >= ts_min_plot) & (df_hourly["ts_hour"] <= anchor_hour)
].copy()
df_plot_hist["series"] = "History (Real)"
df_plot_hist["ts_plot"] = df_plot_hist["ts_hour"]

df_plot_future = df_future_true.copy()
df_plot_future["series"] = "Future (Real)"
df_plot_future["ts_plot"] = df_plot_future["ts_hour"]

df_plot_forecast = df_forecast.copy()
df_plot_forecast["series"] = f"Forecast ({model_name})"
df_plot_forecast["ts_plot"] = df_plot_forecast["ts_hour"]
df_plot_forecast = df_plot_forecast.rename(columns={"y_pred": "price"})

df_plot_actual = pd.concat(
    [
```

```
            df_plot_hist[["ts_plot", "price", "series"]],
            df_plot_future[["ts_plot", "price", "series"]],
        ],
        ignore_index=True,
    )

    df_plot_all = pd.concat(
        [
            df_plot_actual,
            df_plot_forecast[["ts_plot", "price", "series"]],
        ],
        ignore_index=True,
    )

    st.subheader(f"History and yesterday's forecast ({model_name})")

    fig = px.line(
        df_plot_all,
        x="ts_plot",
        y="price",
        color="series",
        labels={
            "ts_plot": "Time",
            "price": f"Price ({vs_currency.upper()})",
            "series": "Series",
        },
    )
    fig.update_layout(height=500)

    st.plotly_chart(fig, width="stretch")

    with st.expander("Yesterday's forecast table"):
        st.dataframe(
            df_merged.sort_values("ts_hour"),
            width="stretch",
            height=400,
        )
```

ui/components/sidebar.py

```
# ui/components/sidebar.py

import streamlit as st
from streamlit_autorefresh import st_autorefresh

from ui.constants import TRACKED_COINS

# ключі та лейбли вкладок
NAV_ITEMS = [
    ("data", "Data"),
    ("features", "Features"),
    ("forecast", "Forecast"),
    ("debug", "Debugging"),
]


def render_sidebar() -> None:
    """
    Малює лівий сайдбар і оновлює st.session_state:
    - active_tab
    - selected_coin_id / selected_coin_label
```

```
- time_range_hours
- auto_refresh_enabled / автооновлення інтервал
"""


with st.sidebar:

    # ---------- ВИБІР МОНЕТИ ----------
    coin_labels = [label for label, _ in TRACKED_COINS]
    coin_ids = [cid for _, cid in TRACKED_COINS]

    # дефолт – bitcoin, якщо є
    default_coin_id = "bitcoin" if "bitcoin" in coin_ids else coin_ids[0]
    default_coin_label = coin_labels[coin_ids.index(default_coin_id)]

    current_coin_id = st.session_state.get("selected_coin_id", default_coin_id)
    if current_coin_id not in coin_ids:
        current_coin_id = default_coin_id

    current_index = coin_ids.index(current_coin_id)

    selected_coin_label = st.selectbox(
        "Choose Coin:",
        options=coin_labels,
        index=current_index,
        key="sidebar_coin_select",
    )
    selected_coin_id = coin_ids[coin_labels.index(selected_coin_label)]

    st.session_state["selected_coin_id"] = selected_coin_id
    st.session_state["selected_coin_label"] = selected_coin_label

    st.markdown("---")

    # ---------- НАВІГАЦІЯ ПО РОЗДІЛАХ ----------
    if "active_tab" not in st.session_state:
        st.session_state["active_tab"] = "data"

    nav_keys = [k for k, _ in NAV_ITEMS]
    nav_labels = [lbl for _, lbl in NAV_ITEMS]

    current_tab_key = st.session_state.get("active_tab", "data")
    if current_tab_key not in nav_keys:
        current_tab_key = "data"

    default_index = nav_keys.index(current_tab_key)

    selected_label = st.radio(
        "Розділ",
        options=nav_labels,
        index=default_index,
        key="nav_tab",
        label_visibility="collapsed",
    )

    # оновлюємо active_tab
    selected_key = nav_keys[nav_labels.index(selected_label)]
    st.session_state["active_tab"] = selected_key

    # Кнопка-перемикач налаштувань
```

```
settings_open = st.checkbox(
    "Settings",
    key="sidebar_settings_open",
    value=True,
)

st.markdown("---")

if settings_open:
    # ---------- ЧАСОВІ НАЛАШТУВАННЯ ----------
    st.subheader("Часові налаштування")

    time_range_label = st.selectbox(
        "Період даних",
        options=[
            "24 години",
            "3 дні",
            "7 днів",
            "30 днів",
            "Увесь період",
        ],
        index=2,  # дефолт – 7 днів
        key="time_range_label",
    )

    time_range_hours_map = {
        "24 години": 24,
        "3 дні": 24 * 3,
        "7 днів": 24 * 7,
        "30 днів": 24 * 30,
        "Увесь період": None,
    }
    st.session_state["time_range_hours"] = time_range_hours_map[time_range_label]

    st.markdown("---")


    # ---------- АВТООНОВЛЕННЯ ----------
    st.subheader("Автооновлення")

    auto_refresh_enabled = st.checkbox(
        "Увімкнути автооновлення",
        value=False,
        key="auto_refresh_enabled",
    )

    refresh_interval_label = st.selectbox(
        "Інтервал автооновлення",
        options=["30 секунд", "1 хвилина", "5 хвилин"],
        index=1,
        key="auto_refresh_interval_label",
    )

    if auto_refresh_enabled:
        interval_ms_map = {
            "30 секунд": 30_000,
            "1 хвилина": 60_000,
            "5 хвилин": 5 * 60_000,
        }
        interval_ms = interval_ms_map[refresh_interval_label]
```

```
        st_autorefresh(interval=interval_ms, key="global_autorefresh")
```

ui/components/footer.py

```
import base64
import streamlit as st
from pathlib import Path


def load_svg_base64(path: str) -> str:
    """Завантажити локальний SVG і повернути base64 data-uri."""
    svg_path = Path(path)
    if not svg_path.exists():
        raise FileNotFoundError(f"SVG file not found: {path}")

    data = svg_path.read_text(encoding="utf-8")
    return base64.b64encode(data.encode("utf-8")).decode()


def render_footer(
    svg_path: str = "ui/assets/coingecko-dark.svg",
    text: str = "Data powered by",
    height: int = 32,
):
    """Рендер футера з фіксованим положенням і SVG логотипом."""
    svg_base64 = load_svg_base64(svg_path)

    st.markdown(
        f"""
<style>
/* Додатковий відступ знизу, щоб контент не "влізав" під footer */
[data-testid="stMain"] {{
    padding-bottom: 3.5rem;
}}

.crypto-footer {{
    position: fixed;
    bottom: 0;
    left: 0;
    width: 100%;
    padding: 0.4rem 1.4rem;

    display: flex;
    justify-content: flex-end;
    align-items: center;
    gap: 0.55rem;

    background-color: rgba(255, 255, 255, 0.0);

   /* background: rgba(10, 12, 20, 0.15);
    backdrop-filter: blur(12px);
    border-top: 1px solid rgba(255, 255, 255, 0.08);*/

    font-size: 1.5rem;
    color: rgba(255,255,255,0.75);

    z-index: 999;
}}
</style>
```

```
<div class="crypto-footer">
    <span>{text}</span>
    <img src="{svg_base64}" height="{height}">
</div>
""",
        unsafe_allow_html=True,
    )
```