Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(освітньо-кваліфікаційний рівень)

на тему

Використання ШІ в розробці алгоритмічної торгівлі крпиптовалютами

Виконав: студент 6 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Снитка І.В.
(прізвище та ініціали)
Керівник Кулик В. А.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Полтава – 2025 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки» на тему

«Використання ШІ в розробці алгоритмічної торгівлі криптовалютами»

Студента групи 601-ТН Снитки Ігоря Володимировича

Керівник роботи
д.е.н., проф. Кулик В.А.

Завідувач кафедри
кандидат фізико-математичних наук,
доцент Двірна О.А.

Полтава – 2025

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 83 с., 13 рисунків, 2 додатки, 26 джерел.

Об’єкт дослідження: алгоритмічна торгівля криптовалютами з використанням штучного інтелекту.

Мета роботи: розробка додатку для алгоритмічної торгівлі криптовалютами з інтеграцією модуля штучного інтелекту, що забезпечує аналіз ринкових даних і прогнозування цінових коливань.

Методи: машинне та глибоке навчання, архітектурне проєктування, робота з API криптовалютної біржі Binance, інтеграція з Telegram Bot API, тестування алгоритмів на основі історичних даних.

У роботі розроблено Telegram-бота, здатного виконувати алгоритмічну торгівлю криптовалютами на базі штучного інтелекту. Telegram-бот здійснює аналіз ринку в режимі реального часу, прогнозує зміни цін та надсилає рекомендації щодо купівлі або продажу активів, а також може виконувати торгові операції.

Ключові слова: алгоритмічна торгівля, криптовалюта, штучний інтелект, машинне навчання, Binance API, Telegram-бот.

ABSTRACT

Bachelor's qualification thesis: 83 pages, 13 figures, 2 appendices, 26 references.

Object of research: algorithmic cryptocurrency trading using artificial intelligence.

Purpose of the work: development of an application for algorithmic cryptocurrency trading with the integration of an artificial intelligence module that ensures market data analysis and price fluctuation forecasting.

Methods: machine learning and deep learning, architectural design, working with the Binance cryptocurrency exchange API, integration with Telegram Bot API, and testing algorithms based on historical data.

The work presents the development of a Telegram bot capable of performing algorithmic cryptocurrency trading based on artificial intelligence. The Telegram bot conducts real-time market analysis, forecasts price changes, sends recommendations on buying or selling assets, and can also execute trading operations.

Keywords: algorithmic trading, cryptocurrency, artificial intelligence, machine learning, Binance API, Telegram bot.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence (штучний інтелект)

ML – Machine Learning (машинне навчання)

DL – Deep Learning (глибоке навчання)

API – Application Programming Interface (інтерфейс програмування додатків)

Binance API – API для роботи з криптовалютною біржею Binance

Telegram Bot API – інструменти для створення та взаємодії з ботами у Telegram.

LSTM – Long Short-Term Memory архітектура рекурентних нейронних мереж для роботи з часовими рядами.

BTC – біткоїн

ETH – ефір

CSV – Comma-Separated Values формат файлів, у яких дані зберігаються у вигляді значень, розділених комами.

VS Code – Visual Studio Code інтегроване середовище розробки.

VENV – Virtual Environment віртуальне середовище для ізоляції залежностей Python.

ВСТУП

У сучасних умовах глобалізації фінансових ринків і стрімкого поширення цифрових активів особливої актуальності набувають інноваційні технології автоматизації торгових операцій. Криптовалютний ринок, який ще донедавна сприймався переважно як експериментальний або «нішевий» напрям, нині приваблює значну увагу з боку різних категорій учасників – від приватних інвесторів та дрібних трейдерів до великих інституційних гравців. У такому середовищі виникає потреба в якісних інструментах, що допоможуть опрацьовувати великі обсяги інформації й забезпечувати належний рівень швидкодії та надійності під час ухвалення торгових рішень.

Алгоритмічна торгівля вже давно зарекомендувала себе як ефективний засіб підвищення доходності, мінімізації ризиків і скорочення впливу людського чинника на результати торгівлі. Проте застосування класичних алгоритмічних методів у межах крипторинку стикається з певними складнощами. Волатильність багатьох цифрових активів у кілька разів перевищує аналогічні показники традиційних акцій чи валют. Додаткові труднощі виникають через фрагментованість ринку (існування великої кількості бірж), відсутність єдиної регуляторної бази й підвищений ризик шахрайства та кібератак. Все це потребує побудови складних моделей аналізу, здатних швидко адаптуватися до змін середовища.

У цьому контексті надзвичайно перспективним підходом є інтеграція алгоритмічних стратегій зі штучним інтелектом (ШІ). Використання методів машинного та глибинного навчання надає змогу виявляти приховані закономірності в динаміці ринкових цін, проводити швидку обробку великих масивів даних (як ринкових, так і новинних) та формувати прогнози більш високої точності. При цьому нейронні мережі та інші AI-алгоритми здатні самонавчатися, що особливо важливо з огляду на мінливість крипторинків. Завдяки такому підходу можна ефективно знаходити вразливості або нові можливості, що залишаються невидимими для традиційних торгових ботів.

Однак впровадження ШІ-технологій тягне за собою і низку викликів. Це, зокрема, забезпечення обчислювальної потужності для навчання та роботи моделей у режимі реального часу, правильне формулювання торгової стратегії та її параметрів, а також дотримання вимог безпеки та конфіденційності. Особливої уваги потребують питання правового регулювання – зокрема, щодо використання криптовалют і автоматизованих алгоритмів. Оскільки законодавча база у цій сфері часто ще не усталена, розробникам доводиться брати до уваги різноманітні норми й обмеження залежно від конкретної юрисдикції.

Мета даної дипломної роботи полягає в дослідженні можливостей і переваг використання ШІ в алгоритмічній торгівлі криптовалютами, а також у створенні прототипу Telegram-бота, який стане практичним прикладом інтеграції AI-модуля для ухвалення торгових рішень. У межах роботи буде здійснено теоретичний огляд основних концепцій алгоритмічного трейдингу та методів штучного інтелекту, проаналізовано сучасні інструменти для розробки й впровадження AI-рішень, а також проведено експерименти з тестування прототипу. Отримані результати дозволять оцінити ефективність застосування ШІ в межах алгоритмічної торгівлі криптовалютами, визначити потенційні обмеження та сформувати рекомендації щодо подальших напрямів розвитку.

Таким чином, актуальність порушеної теми зумовлена комплексом технічних, економічних та регуляторних чинників, які активно змінюють ринок цифрових валют. Вдосконалення системи алгоритмічної торгівлі за допомогою ШІ дає змогу своєчасно реагувати на коливання курсів, виявляти патерни в поведінці трейдерів і підвищувати загальну прибутковість операцій. Усе це сприяє поглибленню наукових досліджень і подальшому розвитку індустрії, створюючи фундамент для появи ще більш ефективних, автоматизованих і інтелектуальних торгових систем.

РОЗДІЛ 1.
АНАЛІТИЧНИЙ ОГЛЯД ТА ТЕОРЕТИЧНІ ОСНОВИ АЛГОРИТМІЧНОЇ ТОРГІВЛІ З ВИКОРИСТАННЯМ ШІ

1.1. Поняття та історичний розвиток алгоритмічної торгівлі

Алгоритмічна торгівля (algorithmic trading) – це процес автоматизованого здійснення біржових або позабіржових операцій на фінансових ринках за допомогою комп'ютерних програм і спеціальних математичних або статистичних моделей. Основна мета алгоритмічної торгівлі полягає в тому, щоб максимально зменшити вплив людських емоцій і суб'єктивних факторів на прийняття торгових рішень, а також підвищити швидкість реакції на ринкові зміни. У своїй сутності алгоритмічна торгівля є відповіддю на потребу в більш ефективних, структурованих та менш ризикованих методах здійснення угод, особливо в умовах високої волатильності або великого обсягу даних [1].

Зародження алгоритмічної торгівлі відбувалося в другій половині XX століття, коли комп'ютери почали активно використовуватися у фінансовій сфері. Перші згадки про автоматизовані системи ухвалення торгових рішень сягають 1970-х років, коли великі інвестиційні банки та фонди почали застосовувати програмні модулі для аналізу даних про рух цін і обсяги торгів. Однак тодішні можливості обчислювальної техніки були досить обмеженими, що впливало і на складність алгоритмів. Переважна більшість перших систем базувалася на простих індикаторах і статистичних розрахунках типу ковзних середніх, а вся ідея автоматизації полягала у швидкому виявленні арбітражних можливостей між різними ринками чи біржами [2].

У 1980-х роках разом із розвитком комп'ютерних технологій та розширенням доступу до ринкових даних відбувся якісний стрибок в алгоритмічній торгівлі. З'явилися складніші торгові стратегії, засновані на комбінуванні кількох аналітичних методів і технічних індикаторів. Банки й

великі хедж-фонди почали створювати власні дослідницькі підрозділи (quantitative research teams), що спеціалізувалися на розробці математичних моделей для прогнозування ринкової динаміки та автоматичного виставлення ордерів. Водночас суттєво зріс рівень конкуренції, що стимулювало постійний пошук нових підходів і способів оптимізації торгових алгоритмів.

Наступне потужне прискорення розвитку алгоритмічної торгівлі відбулося на межі 1990-х – 2000-х років, коли широкого поширення набув електронний трейдинг, а Інтернет став доступним інструментом для провайдерів ринкової інформації. Тоді ж стали популярними високочастотні стратегії (High-Frequency Trading, HFT), що дозволяли здійснювати велику кількість операцій за частки секунди. Принцип дії таких систем побудований на використанні найсучаснішого апаратного забезпечення та спеціальних надшвидких каналів зв'язку, аби скоротити час відправлення та виконання ордерів до мінімуму. Хоча HFT-стратегії зосереджені в основному на традиційних ринках (акції, облігації, товари), ці підходи значно вплинули й на розвиток усього сегмента алгоритмічного трейдингу, зумовлюючи появу нових технологічних рішень [3].

Зі зростанням популярності криптовалют з'явилося багато нових можливостей для алгоритмічної торгівлі у цифровому середовищі. Уперше криптотрейдинг став широко обговорюватися після стрімкого зростання курсу біткоїна у 2017 році, коли на ринку з'явилася значна кількість розрізнених платформ та великий приплив роздрібних інвесторів. Підвищена волатильність, відсутність централізованого регулювання, а також наявність великої кількості дрібних альткоїнів зі своїми унікальними характеристиками зробили цей ринок надзвичайно привабливим для алгоритмічних стратегій. На додачу, технологічна відкритість більшості бірж (через доступні публічні API) полегшила інтеграцію торгових ботів і сприяла появі широкої екосистеми професійних та напівпрофесійних трейдерів [4].

Таким чином, алгоритмічна торгівля пройшла еволюцію від простих програм для визначення арбітражних можливостей на кількох фондових біржах до складних систем зі штучним інтелектом, які працюють у різноманітних

ринкових середовищах, включно з криптовалютним. Сьогодні алгоритмічні трейдери забезпечують більшість угод на традиційних фондових ринках і поступово закріплюють свої позиції в сегменті цифрових активів. Усе це створює сприятливі умови для подальшого розвитку нових моделей, які об’єднують сучасні методи обробки даних, машинного навчання та глибинних нейронних мереж, відкриваючи ще більше можливостей для оптимізації торгової діяльності й прогнозування ринкових трендів.

1.2. Актуальність дослідження

У сучасному світі, криптовалюти стали одним із найбільш обговорюваних та інноваційних фінансових явищ. Вони не лише змінили уявлення про гроші, а й створили нові можливості для інвестування та торгівлі. Разом із цим, крипторинок характеризується низкою особливостей, які одночасно відкривають перспективи й створюють значні виклики. Висока волатильність, постійна доступність торгових майданчиків, стрімке зростання кількості цифрових активів і відсутність єдиного глобального регулювання формують унікальні умови для діяльності трейдерів і інвесторів. У таких умовах традиційні підходи до торгівлі стають малоефективними, що вимагає застосування нових методів і технологій [5].

Алгоритмічна торгівля, що базується на використанні комп'ютерних програм для автоматизації торгових процесів, уже довела свою ефективність на традиційних фінансових ринках. Однак специфіка криптовалютного середовища, зокрема його динамічність і мінливість, висуває особливі вимоги до цих алгоритмів. Прості моделі технічного аналізу або класичні стратегії, які успішно працюють на фондових ринках, не завжди можуть забезпечити необхідну точність і швидкість у контексті криптовалют. У відповідь на ці виклики все більша увага приділяється інтеграції штучного інтелекту (ШІ) у алгоритмічну торгівлю [6].

Застосування ШІ відкриває нові горизонти в аналізі ринкових даних і прийнятті рішень. Штучні нейронні мережі, методи машинного навчання та інші інструменти дозволяють не лише аналізувати історичні дані, але й враховувати зовнішні чинники, такі як новинний фон, активність у соціальних мережах або геополітичні події. Важливою перев-агою ШІ є його здатність до самонавчання та адаптації до швидкозмінних ринкових умов. Це дозволяє виявляти приховані закономірності, які залишаються недоступними для традиційних методів аналізу [6].

Особливо актуальним стає використання ШІ у вигляді інтегрованих рішень, наприклад, Telegram-ботів для алгоритмічної торгівлі. Такий підхід забезпечує не лише автоматизацію торгових процесів, а й високу оперативність у виконанні операцій. Telegram-боти, оснащені алгоритмами ШІ, можуть у реальному часі аналізувати ринкову ситуацію, формувати прогнози, приймати торгові рішення та негайно виконувати угоди. Крім того, вони є зручними для користувачів завдяки простоті інтеграції та доступності на будь-якому пристрої.

Таким чином, дослідження, спрямоване на розробку ефективного рішення для алгоритмічної торгівлі криптовалютами з використанням Telegram-бота на основі ШІ, є актуальним і своєчасним. Воно не лише відповідає сучасним технологічним тенденціям, але й може зробити вагомий внесок у підвищення ефективності торгівлі криптовалютами. Очікується, що результати цього дослідження будуть корисними як для трейдерів і інвесторів, так і для науковців, які займаються розробкою нових підходів до обробки ринкових даних [7].

1.3. Особливості та виклики торгівлі криптовалютами

Криптовалютний ринок суттєво відрізняється від традиційних фінансових ринків як своєю структурою, так і характером торгівлі. Щоб розуміти, яким чином він впливає на побудову алгоритмічних систем і застосування штучного

інтелекту, варто виділити низку його специфічних рис та пов’язаних із цим викликів.

Висока волатильність. Однією з найпомітніших особливостей криптовалют є надзвичайна мінливість їхніх курсів. Ціна може змінюватися на десятки відсотків упродовж кількох годин, що створює широкі можливості для отримання швидкого прибутку. Водночас це підвищує ризик значних втрат. Для алгоритмічних систем, зокрема й тих, які використовують ШІ, така волатильність є і перевагою (за рахунок великої кількості торгових можливостей), і слабким місцем (через підвищену складність адекватного прогнозування короткострокових трендів) [8].

Фрагментованість ринку. На відміну від традиційних бірж (NYSE, NASDAQ тощо), де обсяг торгів зосереджений на кількох головних площадках, криптовалютний простір поділений між сотнями бірж по всьому світі. Кожна з них має власний набір торговельних пар, різні правила лістингу й механізми формування ціни. Це може призводити до різниці у вартості того самого активу на різних біржах (арбітражні можливості), створюючи складнішу структуру ринку. Алгоритмічні системи мусять ураховувати цю фрагментованість, підключаючись до кількох торговельних платформ і обробляючи великі обсяги даних із різних джерел, щоби приймати найбільш інформовані рішення [9].

Цілодобовий доступ і нестандартні торговельні сесії. Традиційні фінансові ринки зазвичай мають фіксовані години роботи, тоді як криптовалютний ринок функціонує безперервно 24/7. Це вимагає від алгоритмічних систем працювати без вихідних і перерв, постійно моніторячи ринкову ситуацію та виконуючи трейди. ШІ-моделі при цьому повинні бути здатні швидко обробляти великі обсяги інформації, щоб вчасно реагувати на зміни курсів та новинні тригери.

Відсутність єдиного регуляторного середовища. У більшості країн законодавча база стосовно криптовалют ще остаточно не сформована або перебуває в стадії активних змін. У деяких юрисдикціях криптовалюти взагалі заборонені, в інших – частково легалізовані, а в окремих країнах існують лише загальні рекомендації без чітких норм. Така ситуація призводить до правової

невизначеності, ускладнюючи впровадження масштабних проєктів і підвищуючи потенційні ризики для учасників ринку. Для алгоритмічних систем це означає необхідність обачно обирати біржі, забезпечувати відповідність заходам KYC/AML та враховувати можливі законодавчі обмеження.

Різноманіття криптоактивів. Станом на сьогодні існують тисячі різних криптовалют та токенів, кожен із яких має власні особливості, історію цін та ринкову капіталізацію. Деякі з них утримують значну частку ринку (Bitcoin, Ethereum), тоді як інші залишаються малоліквідними альткоїнами з мінімальним торговим обсягом. В умовах такого розмаїття ШІ-моделям доводиться адаптувати свої стратегії залежно від показників ліквідності, глибини ринку та волатильності конкретного активу [9].

Технологічні ризики та безпека. Криптовалютні біржі та гаманці доволі часто стають мішенню кібератак. Уразливості у смарт-контрактах, збої у роботі блокчейн-мереж, неправильне налаштування безпекових протоколів – все це становить додаткову загрозу для трейдерів. Алгоритмічні системи, що взаємодіють із біржовими API, мають бути ретельно захищені від можливих витоків ключів та скомпрометованого доступу до торгових операцій. Розробники алгоритмічного програмного забезпечення повинні регулярно оновлювати свої рішення, проводити аудит коду і впроваджувати сучасні засоби моніторингу та реагування на загрози [10].

Інформаційні та новинні чинники. Вартість криптовалют часто залежить від публікації новин, соціального та медійного фону, а також висловлювань публічних осіб. Заява відомої особи чи новина про державне регулювання можуть миттєво спричинити обвал чи стрімке зростання курсу. Для алгоритмічних систем цей чинник є особливо важливим, адже потребує розробки механізмів швидкого аналізу текстової інформації (наприклад, інструменти Natural Language Processing) і оперативного внесення змін до стратегії.

Таким чином, торгівля криптовалютами має низку принципових відмінностей від класичних фінансових ринків, що, з одного боку, відкриває нові можливості для трейдерів та інвесторів, а з іншого – накладає додаткові вимоги

до реалізації алгоритмічних систем і використання штучного інтелекту. Попри підвищені ризики та невизначеність, саме ці особливості роблять криптовалютний ринок привабливим майданчиком для впровадження інноваційних стратегій, які здатні швидко адаптуватися до умов, що постійно змінюються [11].

1.4. Основні методи та підходи застосування штучного інтелекту у фінансовій сфері

Штучний інтелект відіграє дедалі важливішу роль у фінансовій галузі, надаючи потужні інструменти для аналізу ринкових даних, прогнозування цін, управління ризиками й автоматизованого ухвалення рішень. Застосування ШІ набуло значного поширення не лише у класичному трейдингу акціями чи товарами, а й у швидкозростальному сегменті криптовалют. У контексті алгоритмічної торгівлі можна виділити кілька ключових методів і підходів ШІ, які знаходять найбільш практичне застосування [12].

Методи машинного навчання (Machine Learning, ML).

Регресійний аналіз (Linear Regression, Ridge, Lasso): використовується для приблизного визначення майбутніх цін активу, трендів і кореляцій між різними змінними.

Методи класифікації (Logistic Regression, Random Forest, Gradient Boosting): застосовуються для розпізнавання патернів, прогнозування зростання або падіння курсу, визначення «бичих» чи «ведмежих» ринкових фаз.

Методи кластеризації (K-Means, DBSCAN): дозволяють групувати активи чи часові проміжки за певними схожими характеристиками (волатильність, обсяг торгів тощо), що може бути корисно для побудови портфельних стратегій чи визначення ринкових сегментів.

Глибоке навчання (Deep Learning).

Нейронні мережі з різними архітектурами (MLP, CNN, RNN, LSTM тощо): дають змогу засвоювати складні залежності й обробляти як структуровані (цінові та об’ємні ряди), так і неструктуровані дані (новини, пости у соцмережах).

Рекурентні нейронні мережі (RNN/LSTM): ефективні для аналізу часових рядів і прогнозування динаміки курсу, оскільки враховують залежність між попередніми та поточними станами даних.

Автоенкодери, варіаційні автоенкодери (VAE) та генеративно-змагальні мережі (GAN): використовуються для виявлення прихованих закономірностей або аномалій у ринковій поведінці, моделювання руху цін, а також для синтетичного генерування даних для більш ефективного навчання.

Обробка природної мови (Natural Language Processing, NLP).

Семантичний аналіз (Sentiment Analysis): дозволяє визначити емоційне забарвлення публікацій у соціальних мережах, новинних статтях чи повідомленнях, що може впливати на настрої ринку та, відповідно, на курси криптовалют.

Ключові слова й тематичне моделювання (topic modeling): допомагають з’ясувати, які теми переважають в інформаційному просторі навколо криптовалют. Це дає змогу вчасно реагувати на потенційні загрози чи можливості.

Системи рекомендацій та чат-боти: у фінансовому середовищі можуть впроваджуватись для оперативного надання користувачам аналітики ринку, готових торгових сигналів чи персоналізованих порад.

Роботизовані системи прийняття рішень (Expert Systems, Rule-Based Systems).

Незважаючи на велику популярність машинного й глибокого навчання, класичні експертні системи та алгоритмічні модулі, що базуються на чітко прописаних правилах (правило IF-THEN), також залишаються релевантними. Їх часто використовують як частину більших гібридних систем, де ШІ генерує рекомендації, а набір експертних правил перевіряє, чи відповідають вони заданим критеріям безпеки й ризику [13].

Такі системи іноді є більш прозорими для користувачів і регуляторів, адже дозволяють чітко пояснити логіку ухвалення конкретного рішення.

Гібридні підходи.

Часто в одному рішенні об’єднуються кілька методів ШІ: наприклад, глибоке навчання для обробки цінових рядів і NLP-модуль для оцінки новинного фону. Це дає змогу отримати більш повний «портрет» ринку й підвищити точність прогнозів.

Поєднання ШІ з традиційною економіко-математичною моделлю (VAR, ARIMA, GARCH) може поліпшити стабільність і передбачуваність алгоритму, оскільки класичні методи аналізу часового ряду добре працюють за певних ринкових умов, а нейронні мережі можуть швидко адаптуватися до нетипових змін.

Управління ризиками та портфелем.

ШІ застосовується не тільки для пошуку оптимальних точок входу й виходу з угод, а й для більш комплексного управління портфелем, включно з балансуванням активів, розрахунком VaR (Value at Risk), аналізом кореляцій між різними криптовалютами та іншими ризиковими показниками [14].

Нейронні мережі та інші AI-алгоритми можуть працювати у режимі реального часу, постійно оновлюючи оцінки волатильності, ліквідності й тенденцій, що дозволяє миттєво коригувати позиції залежно від ринкових умов.

Загалом, застосування ШІ у фінансовій сфері швидко еволюціонує та має багатогранний характер. У криптовалютному трейдингу, як і на традиційних ринках, такі технології здатні не тільки спростити процедури збору та аналізу інформації, а й підвищити ефективність торгових стратегій, мінімізувати вплив емоційного фактору та покращити управління ризиками. Це робить штучний інтелект одним із ключових рушіїв розвитку сучасного фінансового сектора та відкриває нові перспективи для трейдерів, інвесторів і розробників.

1.5. Сучасний стан розвитку AI-систем в алгоритмічній торгівлі криптовалютами

На сьогоднішній день інтелектуальні алгоритми дедалі активніше проникають у сферу криптовалютного трейдингу, поєднуючи можливості великих обчислювальних потужностей з найновішими досягненнями в галузі машинного навчання та глибокого навчання. Значна кількість аналітичних платформ, таких як QuantConnect, 3Commas і Shrimpy, уже інтегрувала інструменти штучного інтелекту, надаючи трейдерам можливість швидко розгортати готові рішення або створювати власні моделі. При цьому зростає попит на хмарні обчислення, що дозволяють проводити навчання нейронних мереж на великих обсягах даних (Big Data) та оперативно запускати торгових ботів без необхідності підтримувати складну інфраструктуру на локальних серверах.

Однією з ключових тенденцій є активний розвиток спеціалізованого програмного забезпечення, котре спрощує інтеграцію з різними криптобіржами та надає готові модулі для AI-аналітики. Наприклад, багато майданчиків надають пісочниці (testnet), що дає змогу відпрацьовувати алгоритмічні стратегії на умовному балансі. Такі можливості дуже важливі для розробників, оскільки вони дозволяють тестувати ШІ-модулі, уникаючи ризиків реальних збитків. Наявність відкритих API більшості великих бірж (Binance, Coinbase, Bybit тощо) спростила створення кастомних ботів, а відкрита екосистема бібліотек машинного навчання (TensorFlow, PyTorch, scikit-learn) розширила доступ до сучасних методів обробки та аналізу даних [15].

Інституційні інвестори також активно виявляють інтерес до AI-рішень, особливо тих, що пропонують багатофакторний аналіз, прогнозування ринкових трендів, автоматизоване управління ризиками та адаптивне виконання ордерів. Такі технології забезпечують вищу швидкість реагування на зміни ринку, підвищуючи ефективність торгових стратегій. Утім, у контексті криптовалютного трейдингу залишається низка викликів, серед яких

нестабільність законодавчого поля, залежність від постійного доступу до високоякісних даних, необхідність захисту систем від кібератак і значні витрати на обчислювальні ресурси.

Окрім цього, активно розвиваються моделі генеративного штучного інтелекту, здатні аналізувати мільйони історичних даних для створення інноваційних стратегій. Гібридні системи, що об’єднують штучний інтелект із класичними фінансовими інструментами, стають основою для створення динамічних портфелів та автоматизованих інвестиційних платформ.

Попри виклики, ринок AI-рішень для алгоритмічної торгівлі криптовалютами демонструє стійкий тренд до зростання. Конкуренція серед стартапів і великих IT-корпорацій сприяє інтенсивним інноваціям, появі складних гібридних моделей і зручних для користувачів інструментів, які поступово змінюють не лише культуру трейдингу, а й формують нові підходи до інвестування та управління цифровими активами. Штучний інтелект стає невід'ємним компонентом сучасного фінансового середовища, відкриваючи нові горизонти для розвитку як індивідуальних трейдерів, так і великих інституцій.

1.6. Огляд нормативно-правового середовища

Застосування алгоритмічної торгівлі та штучного інтелекту у сфері криптовалют відбувається на тлі складного й неоднорідного правового поля. Законодавча невизначеність, різні підходи до регулювання криптоактивів у різних країнах та відсутність узгоджених міжнародних стандартів суттєво впливають на умови, в яких працюють розробники і користувачі таких рішень. Цей аспект має особливе значення, оскільки може визначати не лише легальність проведення торгових операцій, а й вимоги до захисту даних, ліцензування, виконання процедури KYC/AML тощо.

Світовий контекст.

Провідні юрисдикції (США, ЄС, Велика Британія): У цих регіонах питання правового статусу криптовалют і торгівлі ними наразі перебувають у стадії формування єдиних норм. У США регулюванням криптовалютної діяльності займається відразу кілька установ (SEC, CFTC, FinCEN), що створює мультиагентну систему з частково дубльованими вимогами. У ЄС триває впровадження регламенту Markets in Crypto-Assets (MiCA), який покликаний уніфікувати правила для учасників ринку та встановити мінімальні стандарти прозорості.

Країни Азії (Японія, Сінгапур, Південна Корея): Ці держави, зокрема Японія й Сінгапур, першими почали запроваджувати досить жорсткі, але водночас зрозумілі правила для криптовалютних бірж і компаній, що впроваджують фінтех-рішення на базі блокчейну. Такий підхід приваблює великих гравців, які зацікавлені в правовій стабільності.

Країни з обмеженнями та заборонами: Деякі країни (Китай, частково Росія, Туреччина тощо) запроваджують суворі обмеження на криптовалютну діяльність або повністю її забороняють. Це ускладнює роботу алгоритмічних трейдерів і розробників ШІ-систем, оскільки може бути заблоковано локальний доступ до бірж і сервісів [16].

Вітчизняне регулювання.

Правовий статус криптовалют: До недавнього часу законодавство України (як і багатьох інших пострадянських держав) не мало чітко сформульованих норм щодо криптовалют, визнаючи їх або «замінниками грошей», або «цифровим активом» без визначеного правового статусу. Нині ухвалюються перші кроки до регулювання ринку в рамках законів про «віртуальні активи», що передбачають певні вимоги до постачальників послуг, пов’язаних із криптовалютами [17].

Ліцензування та реєстрація: Якщо компанія або фізична особа хоче надавати послуги з обміну чи торгівлі криптовалютами, їй може знадобитися спеціальна реєстрація (якщо це передбачено законодавством). Також є вимоги

щодо фінансового моніторингу, ведення звітності та дотримання норм про запобігання відмиванню коштів (AML) і фінансуванню тероризму (KYC).

Оподаткування: Податкові органи багатьох країн поступово формулюють правила оподаткування операцій із криптовалютами. Зазвичай трейдинг криптоактивами прирівнюється до операцій із рухомим майном чи цінними паперами, тож прибуток може обкладатися податком на доходи фізичних осіб або на прибуток підприємства.

Захист прав споживачів і відповідальність: Поки що відсутнє чітке правове поле щодо того, хто й на яких підставах несе відповідальність за можливі збитки користувачів при використанні AI-ботів і алгоритмічних стратегій. Важливим аспектом є також захист даних користувачів і дотримання законодавства про конфіденційність.

Вплив нормативно-правового середовища на розробку алгоритмічних AI-систем.

Необхідність відповідати вимогам KYC/AML: У більшості країн оператори торгових ботів і біржі зобов’язані ідентифікувати клієнтів, особливо якщо йдеться про великі обсяги операцій. Це означає, що у процесі інтеграції бота з реальними акаунтами треба враховувати процедуру реєстрації та верифікації.

Правова визначеність та планування ризиків: Розробникам варто враховувати можливі зміни в законодавстві та аналізувати, як вони вплинуть на бізнес-модель і технологічну реалізацію продукту. Зокрема, необхідно відстежувати вимоги до ліцензування, до отримання згоди від користувача та до зберігання конфіденційних даних.

Регулювання AI та автоматизованих рішень: Хоча більшість законів про штучний інтелект перебувають у стадії проєктів чи рекомендацій, тенденція до посилення контролю AI-рішень очевидна. Можуть запроваджуватися вимоги щодо прозорості алгоритмів або запобігання дискримінації, що слід враховувати в довгостроковій перспективі.

Перспективи та рекомендації.

Глобальна тенденція до уніфікації: Зі зростанням масштабів крипторинку країни та міжнародні організації намагаються узгодити свої підходи, що призведе до поступового формування чіткішої нормативної бази. Це створить сприятливі умови для появи нових AI-рішень і збільшення прозорості ринку.

Необхідність консультацій із фахівцями у правовій сфері: Під час розробки та впровадження алгоритмічних систем із використанням ШІ необхідно залучати юристів і фахівців із комплаєнсу, аби вчасно врахувати усі регуляторні вимоги та мінімізувати правові ризики.

Фокус на безпеку та відповідальність: Попри недосконаліть законодавства, професійні учасники ринку прагнуть підвищувати рівень прозорості та надійності своїх сервісів. Безпека користувацьких даних, захист від шахрайства та стабільна робота алгоритмів є ключовими факторами успішної діяльності й довіри з боку клієнтів.

Отже, нормативно-правова база, в якій розгортається алгоритмічна торгівля криптовалютами із використанням ШІ, лишається фрагментованою й динамічною. Розробникам та користувачам таких систем слід уважно відстежувати міжнародні тенденції та законодавчі ініціативи, планувати архітектуру проєкту з урахуванням вимог KYC/AML, безпеки даних і можливих ліцензійних обмежень. Правильне розуміння та дотримання регуляторних норм не лише мінімізує правові ризики, а й закладає основу для сталого розвитку та довгострокової конкурентоспроможності AI-рішень у сфері крипто валют [18].

1.7. Вибір платформи для взаємодії користувача з ШІ

Платформа, на базі якої буде створено проект, визначає основу для його функціональності та способу взаємодії з користувачами. Для проекту, що стосується використання штучного інтелекту в розробці алгоритмічної торгівлі криптовалютами, платформа повинна забезпечувати зручність взаємодії, високу швидкість передачі даних та підтримку сучасних інструментів розробки.

Існують різні платформи, які можна використати для реалізації такого проекту, і кожна з них має свої переваги та особливості.

Месенджери (Telegram, Viber, Facebook): ці платформи надають потужні можливості для створення чат-ботів. Вони мають широку аудиторію користувачів і забезпечують зручну комунікацію за допомогою текстових повідомлень, кнопок та графічних елементів. Telegram, зокрема, вирізняється відкритою API, що дозволяє інтегрувати додаткові функції, які важливі для алгоритмічної торгівлі.

Мобільні платформи (iOS, Android): створення мобільного додатка дозволяє забезпечити взаємодію з користувачами через їхні смартфони. Це підвищує зручність доступу до функцій проекту, зокрема моніторингу ринку та отримання повідомлень про дії штучного інтелекту.

Веб-платформи: розробка веб-платформи відкриває можливості для створення інтерактивного інтерфейсу, доступного через браузер. Це рішення забезпечує гнучкість і масштабованість, які є важливими для складних систем алгоритмічної торгівлі.

Після аналізу різних платформ, було прийнято рішення використовувати Telegram як основну платформу для взаємодії користувачів із системою. Це рішення було прийнято з такими перевагами:

Telegram має значну кількість активних користувачів, кількість яких постійно збільшується, що дозволить залучити більше людей до використання віртуального фінансового помічника. На рис 1.1 наведено графік збільшення активних користувачів з 2014 по 2022 рік [19].

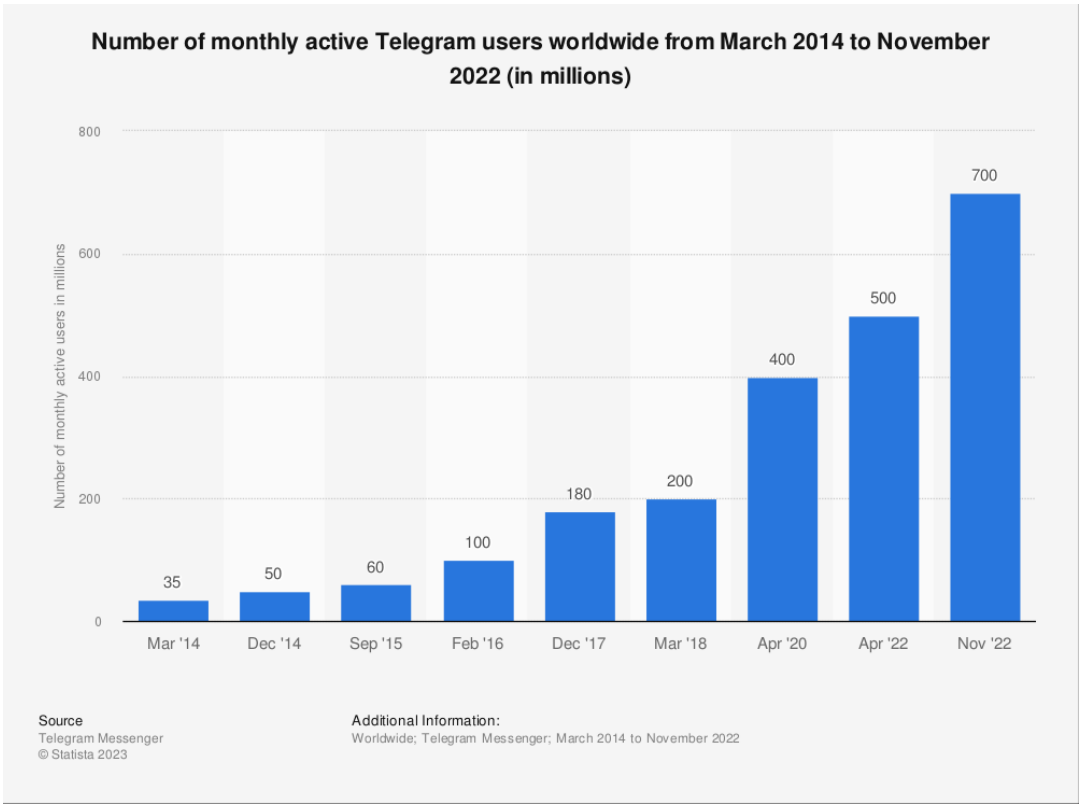


Рисунок 1.1 – Графік росту користувачів Telegram

Вбудована підтримка ботів. Telegram надає широкий набір інструментів для створення ботів через Bot API. Це спрощує інтеграцію складних функцій, зокрема інструментів для алгоритмічної торгівлі.

Простота використання. Інтерфейс Telegram є інтуїтивно зрозумілим для користувачів, що забезпечує легкий доступ до функцій системи, таких як отримання сигналів, сповіщень та аналітики ринку.

Безпека та шифрування. Telegram відомий своїми високими стандартами безпеки, що є критично важливим для роботи з фінансовими даними користувачів.

1.8. Постановка задачі

З урахуванням проведеного аналітичного огляду та виявлених особливостей алгоритмічної торгівлі криптовалютами із залученням штучного

інтелекту, можна сформулювати основні завдання, розв’язання яких передбачене в межах цієї дипломної роботи:

Дослідити та узагальнити теоретичні підходи до алгоритмічної торгівлі з використанням ШІ

Проаналізувати еволюцію алгоритмічної торгівлі, а також її відмінності у сфері криптовалют, зважаючи на підвищену волатильність та фрагментованість ринку.

Розглянути сучасні методи штучного інтелекту та визначити, які саме з них можуть бути найбільш ефективними в алгоритмічному трейдингу.

Обґрунтувати вибір технологічного стеку для розробки системи алгоритмічної торгівлі

Визначити оптимальні мови програмування та фреймворки, що дозволять реалізувати модель ШІ та взаємодіяти з криптовалютними біржами (через API).

Обґрунтувати доцільність використання Telegram-бота як інтерфейсу для користувача, враховуючи вимоги до зручності, безпеки й можливості масштабування.

Розглянути варіанти побудови архітектури проєкту з урахуванням особливостей обробки ринкових даних у режимі реального часу та проведення торгових операцій.

Розробити прототип Telegram-бота із вбудованим AI-модулем для алгоритмічної торгівлі

Спроєктувати структуру системи, включно з компонентами для збору та обробки даних, модулем для навчання та використання моделі ШІ, а також механізмом виконання торгових операцій.

Забезпечити можливість підключення до обраних криптобірж, належний рівень безпеки зберігання ключів доступу та виконання ордерів.

Реалізувати базовий функціонал обробки запитів користувача, надання рекомендацій щодо угод (купівля, продаж, утримання) та автоматизованого відкриття/закриття позицій.

Здійснити тестування та оцінити ефективність створеної системи

Провести серію тестів (модульне, інтеграційне, системне) для виявлення й усунення помилок у роботі бота та AI-модуля.

Виконати backtesting на історичних даних і paper trading (або тестування в пісочниці на криптобіржі) для оцінки точності прогнозів та результатів торгових стратегій.

Порівняти показники прибутковості та рівень ризиків із базовими стратегіями, що не використовують ШІ.

Надати рекомендації та визначити перспективи подальшого розвитку

Проаналізувати результати тестування й окреслити можливі шляхи підвищення точності прогнозів, пришвидшення обробки даних та розширення функціоналу бота.

Розглянути питання масштабування системи на інші торговельні платформи, а також впровадження додаткових інструментів ШІ (наприклад, NLP-аналізу новин).

Окреслити, як саме зміни в нормативно-правовому середовищі можуть вплинути на розвиток такого проєкту та які юридичні аспекти слід враховувати на подальших етапах.

Таким чином, головна мета дипломної роботи полягає в дослідженні ефективності використання штучного інтелекту для алгоритмічної торгівлі криптовалютами та розробці прототипу Telegram-бота, здатного здійснювати автоматичний аналіз ринку, формувати торгові рекомендації та виконувати операції за допомогою інтеграції з біржами. Виконання поставлених завдань дозволить перевірити доцільність таких рішень, виявити сильні й слабкі сторони AI-стратегії та дати практичні рекомендації щодо розвитку інтелектуальних торгових систем у динамічному криптовалютному середовищі.

РОЗДІЛ 2.
ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТАРІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ АЛГОРИТМІЧНОЇ ТОРГІВЛІ НА БАЗІ ШІ

2.1. Вибір мови програмування та бібліотек

Створення алгоритмічної системи торгівлі на базі штучного інтелекту потребує ретельного вибору мови програмування й супутніх бібліотек. Від цього залежатиме зручність реалізації торгових логік, інтеграція з біржовими API, продуктивність та простота масштабування.

Python найпопулярніша мова в галузі аналізу даних та машинного навчання; має багату екосистему бібліотек (NumPy, pandas, scikit-learn, TensorFlow, PyTorch тощо). Переваги: простота синтаксису, висока швидкість розробки прототипів, активна спільнота розробників, легка інтеграція з API криптобірж (через бібліотеку CCXT). Недоліки: нижча швидкодія порівняно з C++ або Java, що може бути критично для високочастотної торгівлі [20].

C++ використовується для систем із високими вимогами до швидкодії та мінімальної затримки, зокрема у високочастотному трейдингу (HFT). Переваги: ефективне використання пам’яті, швидке виконання коду. Недоліки: вищий поріг входу, складніша розробка й підтримка коду, відносно невелика кількість спеціалізованих бібліотек для AI.

Java часто використовується в корпоративних і високонавантажених системах, де потрібна масштабованість і стійкість. Переваги: велика екосистема для роботи з Big Data (Hadoop, Spark), є бібліотеки для глибинного навчання (DeepLearning4j). Недоліки: громіздкий синтаксис і складніше прототипування AI-рішень.

З огляду на потреби швидкої розробки, наявність широкого спектра AI-бібліотек і простоту інтеграції з криптовалютними біржами, оптимальним вибором для цього проєкту є Python. Його гнучкість та популярність у галузі

машинного навчання дозволять швидко створити й протестувати прототип алгоритмічної системи, а також у разі необхідності оперативно вносити зміни до торгової логіки.

Для задач, що потребують надвисокої продуктивності (наприклад, HFT-стратегії з мінімальними затримками), розробник може розглянути C++. Проте в даному проєкті робиться акцент на дослідницьку, аналітичну та алгоритмічну складову, тож переваги Python переважають можливі недоліки, зокрема й з точки зору майбутнього масштабування.

Оскільки проект передбачає створення Telegram-бота, постає питання щодо вибору бібліотеки, яка забезпечуватиме зручний доступ до Telegram Bot API.

python-telegram-bot. Одна з найстаріших і найбільш перевірених бібліотек для створення Telegram-ботів. Має хорошу документацію, підтримує асинхронні виклики (через asyncio) у новіших версіях.

telebot (pyTelegramBotAPI). Популярна бібліотека із простим синтаксисом, легка у використанні для невеликих ботів. Однак для складніших рішень може бути менш гнучкою.

Aiogram. Асинхронна бібліотека, яка одразу створювалася з урахуванням можливостей asyncio. Забезпечує високу продуктивність та гнучку архітектуру. Активно розвивається і має велику спільноту.

З огляду на те, що в алгоритмічній торгівлі надзвичайно важлива швидкодія і масштабованість, а також комфортна робота з асинхронними запитами (зокрема для обміну даними з біржами), найкращим рішенням буде бібліотека Aiogram. Вона поєднує зручну структуру, гнучку обробку команд і повідомлень, а також високу продуктивність [21].

Поряд із бібліотеками для машинного навчання, проект потребує засобів для ефективної обробки масивів даних, зокрема історичних часових рядів, і для побудови наочних графіків.

NumPy. Базова бібліотека Python для роботи з багатовимірними масивами й векторами. Забезпечує базові операції лінійної алгебри, швидкі обчислення на рівні C/Fortran. Є фундаментом для багатьох інших бібліотек.

Pandas. Потужний інструмент для аналізу та структурування даних із використанням таких структур, як DataFrame. Зручно працює з часовими рядами, що є критично важливим у фінансовому аналізі.

Matplotlib. Класична бібліотека для побудови графіків і діаграм. Дозволяє відображати лінійні графіки, свічкові графіки, гістограми, scatter plots тощо, що потрібно для візуалізації ринкових даних і результатів роботи алгоритму.

Усі вищезгадані бібліотеки швидко встановлюються та активно підтримуються спільнотою. Вони стануть основою для підготовки, виведення статистики та графічного відображення результатів під час розробки й тестування AI-модуля.

2.2. Вибір середовища для розробки

Після визначення мови програмування та ключових бібліотек постає питання про вибір середовища розробки (Integrated Development Environment, IDE) або редактора коду, що вплине на зручність написання, налагодження та тестування проєкту.

Visual Studio Code (VS Code) – це легкий та гнучкий редактор коду від Microsoft, який завдяки розширенням може підтримувати будь-які мови програмування, зокрема Python. Він працює кросплатформено, має швидку реакцію інтерфейсу, інтегрований термінал і тісну взаємодію з системами контролю версій (Git). Для Python передбачене офіційне розширення, що забезпечує автозаповнення, відлагодження, роботу з віртуальними середовищами й Jupyter Notebook. Це робить VS Code комфортним для проєктів, де планується використання декількох бібліотек (Aiogram, NumPy, pandas, matplotlib, TensorFlow/PyTorch тощо).

PyCharm від JetBrains орієнтований винятково на розробку на Python і пропонує розширені інструменти для рефакторингу, налагодження, а також окремі наукові модулі (Scientific Mode). Він корисний для складних проєктів, але може бути надмірним, якщо потрібне універсальне та ресурсозберігаюче рішення. Крім того, повна версія PyCharm є платною, що обмежує його доступність.

Jupyter Notebook / JupyterLab – веб-орієнтоване середовище, зручне в першу чергу для інтерактивних експериментів і швидкого прототипування моделей машинного навчання. Воно дозволяє крок за кроком запускати блоки коду й одразу відображати графіки, таблиці та результати. Утім, для розробки повноцінної структури коду й багатомодульних проєктів Jupyter часто виявляється незручним.

Sublime Text – швидкий та простий редактор, що підходить для базових сценаріїв і редагування скриптів. Він не є спеціалізованим Python-середовищем, тому більшість функцій (автозаповнення, відлагодження тощо) потребують додаткових плагінів.

Оскільки в межах проєкту планується написання повноцінного алгоритмічного торгового модуля зі складними логіками, взаємодія з біржовими API, реалізація бота в Telegram та інтеграція моделей машинного навчання, найоптимальнішим вибором вважається Visual Studio Code. Він забезпечує високу гнучкість завдяки екосистемі розширень, містить інструменти для зручного відлагодження й версіонування та не потребує значних ресурсів [22].

2.3. Інструменти машинного навчання та глибокого навчання

У межах створення алгоритмічної системи торгівлі криптовалютами особливу увагу слід приділити вибору бібліотек для машинного навчання й глибокого навчання. Вони дадуть змогу розробити та навчити модель, здатну аналізувати часові ряди, виявляти закономірності у динаміці ринку та формувати

прогнози руху цін. Крім того, ці інструменти необхідні для подальшої інтеграції моделей у Telegram-бот і автоматичного ухвалення рішень.

Серед багатьох фреймворків, які використовуються в машинному навчанні, найвідомішими є TensorFlow і PyTorch. TensorFlow зручний для розподіленого навчання й має широку базу прикладів, але його синтаксис може здаватися складним на початкових етапах роботи. PyTorch вирізняється динамічними обчислювальними графами та більш інтуїтивним підходом до відлагодження коду, а також вважається одним із найкращих варіантів для наукових досліджень, швидкого прототипування й гнучких експериментів із нейронними мережами. У цій роботі обрано саме PyTorch як головний інструмент для глибинного навчання [23].

PyTorch має вбудовані засоби для роботи з обчисленнями на графічних процесорах (GPU), що особливо корисно при обробці великих обсягів ринкових даних або під час навчання складних моделей (наприклад, мереж LSTM чи Transformer, розроблених для прогнозування часових рядів). Його динамічна структура обчислень дає змогу швидко відстежувати помилки й виконувати коригування, що прискорює розробку та експерименти над моделлю. Спільнота PyTorch пропонує багато прикладів відкритого коду, включно зі спеціалізованими реалізаціями RNN, LSTM, GRU та інших архітектур, які підходять для прогнозування цін або визначення торгових сигналів [24].

Окрім PyTorch для глибинного навчання, у проєкті можуть бути використані класичні методи машинного навчання зі scikit-learn, де є реалізації лінійних, логістичних і регресійних моделей, дерев рішень та інших алгоритмів (Random Forest, Gradient Boosting). Ці методи інколи ефективні для проміжних задач – наприклад, коли необхідно швидко виділити ключові патерни, провести базовий аналіз факторів або створити початкові прогностичні моделі, які можна порівняти з більш складними нейронними мережами. Також scikit-learn зручний для трансформації та нормалізації даних (наприклад, використання StandardScaler чи MinMaxScaler), налаштування крос-валідації та оцінювання моделі за різноманітними метриками.

Інтеграція PyTorch із бібліотеками NumPy та pandas дає змогу безпосередньо передавати масиви та DataFrame у тензорний формат, а також виконувати передобробку даних у звичному середовищі. Це особливо важливо для задачі збору й «очищення» ринкової історії, коли необхідно працювати зі складними часовими рядами або комбінувати дані з різних джерел (кількох бірж, новин, соціальних мереж тощо). У процесі навчання моделі все це допомагає швидко готувати потрібні підвибірки або створювати набори змінних (features), які визначатимуть логіку торгівлі.

Зважаючи на високу волатильність криптовалют, рекомендується спочатку протестувати кілька архітектур нейронних мереж, що спеціалізуються на роботі з часовими рядами. Поширеними підходами є LSTM та GRU для обліку послідовних залежностей, а також згорткові мережі (CNN) для виявлення складних патернів у сигналах ціни чи обсягу. Деякі розробники успішно застосовують трансформер-архітектури (як-от Informer чи Temporal Fusion Transformer), які ефективно працюють із довгими послідовностями. PyTorch дає можливість швидко реалізовувати або адаптувати подібні складні архітектури, застосовуючи вбудовані модулі (torch.nn) та інструменти оптимізації (torch.optim).

Завершальним етапом стане перехід до режиму інференсу: навчена модель інтегрується з Telegram-ботом, отримує оновлені ринкові дані, робить прогноз і формує рішення щодо купівлі чи продажу активу. Цей процес можна запускати періодично або в реальному часі – усе залежить від обраної стратегії. У будь-якому разі PyTorch дає змогу завантажити модель у пам’ять і викликати функцію передбачення (predict) навіть на малопотужних машинах або в контейнерізованому середовищі.

Таким чином, PyTorch виступає ядром глибинного навчання в проєкті, забезпечуючи гнучке середовище для розробки та випробування різноманітних нейронних архітектур. У поєднанні з бібліотеками scikit-learn, pandas і NumPy це створює єдину екосистему, де можна організувати повний цикл роботи з даними, побудови моделей і їх інтеграції в алгоритмічну торгівлю.

2.4. Платформи та API для взаємодії з біржами криптовалют

Для повноцінної реалізації алгоритмічної торгівлі з використанням штучного інтелекту необхідно забезпечити взаємодію з криптовалютними біржами, отримувати актуальні ринкові дані та мати змогу відкривати й закривати позиції. Сучасні торговельні платформи зазвичай надають публічні API, що охоплюють як REST-запити, так і WebSocket-підключення для отримання потокових даних у реальному часі. Серед найпопулярніших бірж варто відзначити Binance, Coinbase, Bybit, KuCoin, OKX та Kraken, які пропонують широкий вибір торгових пар і мають достатню ліквідність, особливо для найбільш відомих цифрових активів на кшталт Bitcoin чи Ethereum.

Binance є однією з наймасштабніших платформ з огляду на кількість доступних монет, добовий обсяг торгів і наявність ф'ючерсного ринку, а також пропонує Testnet для перевірки алгоритмів без ризику втрати реальних коштів. Саме Binance API в даному проєкті обрано для інтеграції торгового модуля: це зумовлено широким вибором активів, надійністю виконання ордерів, великою ліквідністю й достатньою кількістю інструментів для торгівлі (спот, ф'ючерси, маржинальні операції). Coinbase популярна в США та має чітке регуляторне підґрунтя, що може бути важливо для великих інвесторів. Bybit приваблює трейдерів похідними інструментами, KuCoin і OKX пропонують багато альткоїнів, а Kraken відомий більш консервативним підходом і надійністю [25].

Для взаємодії з біржею потрібне вивчення специфікації її REST API та (за потреби) WebSocket-протоколу, аби в реальному часі отримувати котирування, інформацію про ордери й біржовий стакан. У Python зручно використовувати універсальну бібліотеку CCXT, яка уніфікує роботу з багатьма біржами та дає змогу отримувати дані, відкривати й закривати позиції без необхідності вивчати кожен індивідуальний API. CCXT підтримує основні режими торгівлі (спот, ф'ючерси, маржинальну торгівлю), а також містить методи для отримання історичних даних, які можна застосовувати для бектесту. Якщо ж потрібен максимальний контроль і додаткові функції, розробник може звертатися

безпосередньо до офіційної документації Binance або будь-якої іншої платформи.

Найбільш поширений спосіб підключення передбачає створення власного облікового запису на біржі та генерування API-ключів, що надають доступ до торгових операцій. Задля безпеки варто обмежувати права ключів лише тими функціями, які справді потрібні: зокрема, доступом на читання ринкових даних і здійснення конкретних типів ордерів. Для критично важливих операцій або при роботі з великим депозитом рекомендують додатково вмикати двофакторну аутентифікацію, білий список IP-адрес і інші методи захисту. У тестовому режимі, завдяки наявності sandbox- або testnet-середовища (зокрема в Binance Futures Testnet), можна перевіряти роботу Telegram-бота з алгоритмічним модулем, не ризикуючи реальними коштами.

Оскільки алгоритмічна система має працювати цілодобово та своєчасно отримувати оновлення про стан ринку, найкраще використовувати або постійні WebSocket-підключення, або періодичні REST-запити з мінімальним інтервалом опитування. WebSocket-протоколи дають змогу оперативно отримувати повідомлення про зміну ціни, обсягу ордерів у стакані чи статус ордера, що істотно пришвидшує реакцію системи порівняно з «пулінгом». Проте деякі біржі обмежують кількість одночасних WebSocket-з’єднань або вимагають повторної аутентифікації через певний час, тому логіку перепідключення також слід передбачити в коді.

Загалом Binance API дає можливість реалізувати всі необхідні сценарії: від побудови базових торгових стратегій на спотовому ринку до створення складних алгоритмів ф’ючерсної чи маржинальної торгівлі. Завдяки великій кількості існуючих прикладів відкритого коду та активній спільноті на форумах і GitHub, інтеграція з Binance доволі проста навіть для порівняно невеликого проєкту, а можливість працювати в тестовій мережі значно знижує поріг входу й дає змогу відшліфувати торгові логіки без ризику фінансових втрат [26].

2.5. Архітектурне планування проекту

Архітектура системи алгоритмічної торгівлі на базі штучного інтелекту та Telegram-бота формується навколо кількох ключових компонентів, що взаємодіють між собою в єдиному робочому середовищі. У загальному вигляді проект складається з модуля збору та обробки даних, модуля штучного інтелекту (моделей машинного та глибинного навчання), шару бізнес-логіки для прийняття рішень, Telegram-бота для взаємодії з користувачем та інтеграційного шару з біржою (Binance API).

На першому етапі система отримує ринкові дані з Binance (через REST або WebSocket), які можуть включати актуальні котирування, обсяги торгів, статуси ордерів і глибину ринку. Ці дані передаються в модуль попередньої обробки, де відбуваються фільтрація, агрегація та перетворення у формат, зручний для аналізу. На цьому рівні застосовується бібліотека pandas для структурування часових рядів, а також NumPy для ефективних обчислень. Якщо потрібно накопичувати дані тривалий час або працювати з великою історією, вони можуть зберігатися в локальній чи хмарній базі даних (наприклад, PostgreSQL або InfluxDB).

Другий етап передбачає роботу з моделлю на базі PyTorch. Ця модель може бути рекурентною нейронною мережею (LSTM чи GRU), згортковою (CNN) або будь-якою іншою архітектурою, здатною моделювати ринкові патерни. Після завантаження та ініціалізації модель отримує нові дані про ринок, виконує прогноз щодо майбутньої динаміки ціни чи генерує рекомендацію (купівля, продаж, утримання). Результат передається у шар бізнес-логіки, де перевіряються додаткові умови (наприклад, ризик-менеджмент, ліміти на розмір позиції, умови зупинення торгівлі в разі різких коливань). Якщо всі критерії дотримані, система ініціює торгові операції через API-бібліотеку (CCXT або офіційну Binance API).

Telegram-бот, написаний за допомогою Aiogram, забезпечує інтерактивну взаємодію між системою та користувачем. При запиті від користувача бот може

надавати статистику про поточні угоди, показувати графіки (створені за допомогою matplotlib) або пропонувати змінити налаштування торговельної стратегії (наприклад, розмір лота чи частоту виконання угод). Бот також може надсилати повідомлення про виконані угоди, сповіщати про досягнення стоп-лосів чи тейк-профітів і реагувати на аварійні ситуації (збої в мережі, відмови з боку біржі, помилки моделі ШІ). Для зменшення затримок та кращої масштабованості кожен компонент (модуль збору даних, PyTorch-модель, Telegram-бот) може бути винесений в окремий процес або мікросервіс, який обмінюється інформацією через чергу повідомлень (RabbitMQ, Redis) чи інші механізми IPC.

Така багаторівнева архітектура дає змогу легко розширювати функціонал проєкту: додавати нові торговельні стратегії, підключати додаткові біржі або впроваджувати альтернативні AI-моделі. Крім того, у випадку значного навантаження окремі частини системи можна розгорнути на різних серверах або у хмарному середовищі (AWS, Azure, GCP), забезпечивши збалансоване використання ресурсів.

2.6. Засоби розгортання та інтеграції Telegram-бота з AI-модулем

Зручне й безпечне розгортання системи алгоритмічної торгівлі передбачає низку технічних рішень, які спрощують роботу з ботом, надають можливість швидко оновлювати або масштабувати компоненти та гарантують надійність у режимі реального часу. У цьому контексті особливу увагу слід приділити налаштуванню віртуальних середовищ, інтеграції з PyTorch-моделлю.

Найпростішим підходом до розгортання є використання віртуальних середовищ (virtualenv, Anaconda) на локальному або віддаленому сервері. Це дозволяє ізолювати версії бібліотек Python (включно з PyTorch, Aiogram, pandas тощо), уникаючи конфліктів і спрощуючи подальше оновлення пакетів. Після налаштування середовища можна запускати скрипт бота та AI-модуля у

фоновому режимі за допомогою сервісу systemd або менеджера процесів на кшталт PM2.

Однак для більш структурованої організації часто застосовують контейнеризацію, зокрема Docker. Використання Docker має низку переваг:

Чітке визначення середовища виконання через Dockerfile, що включає версії Python, бібліотек і системних залежностей.

Легке перенесення контейнера на різні платформи або хмарні сервіси, такі як AWS, Azure, GCP.

Можливість масштабувати бота та AI-модуль окремо, якщо це потрібно (наприклад, запустити кілька копій Telegram-бота чи окремо тренувати й розгорнути AI-модель).

Під час інтеграції PyTorch-моделі з ботом варто визначити спосіб виклику моделі: або модель «підвантажується» в той самий процес, у якому працює бот, або ж модель запускається окремим мікросервісом (наприклад, за допомогою Flask/FastAPI), і бот звертається до нього через HTTP-запити. Другий варіант дає змогу розділити логіку взаємодії з користувачем і роботу інференсу, а також може спростити оновлення або заміну нейронної мережі.

Для безперебійного функціонування критично важливо забезпечити багаторівневу перевірку:

Перевірка стану Telegram-бота: якщо бот перестає відповідати (через збій у мережі або зависання процесу), система має перезапустити контейнер або процес.

Логування та моніторинг: логи мають зберігатися у централізованому сховищі чи сервісі (наприклад, ELK Stack – Elasticsearch, Logstash, Kibana), щоб можна було швидко переглянути історію дій бота й AI-модуля, виявляти помилки та шукати закономірності у збоях.

Нагляд за продуктивністю: зокрема, використанням CPU/GPU, часом відповіді PyTorch-модуля, кількістю успішних викликів до Binance API. Це може бути організовано через Prometheus/Grafana чи подібні інструменти.

Створення Dockerfile, де вказують базовий образ (наприклад, python:3.9-slim), встановлюють PyTorch, Aiogram, бібліотеку ccxt та інші залежності, копіюють файли проєкту, налаштовують робочу директорію.

Формування docker-compose.yml (за потреби), щоб одночасно підняти кілька контейнерів – для Telegram-бота.

Конфігурація середовища через файли .env, у яких зберігають змінні оточення: токени, ключі та інші чутливі дані.

Розгортання на сервері (локальному чи хмарному): запуск docker-compose up -d, перевірка коректності мережевих налаштувань, можливе додавання зворотного проксі (nginx) для маршрутизації запитів.

Тестування та моніторинг: після запуску перевіряють, чи бот відповідає в Telegram, чи може успішно викликати AI-модуль, чи відбуваються торгові операції на тестовій біржі, чи з’являється відповідна інформація у логах і графіках моніторингу.

При потребі швидко оновлювати модель PyTorch (наприклад, у разі донавчання чи зміни архітектури мережі) можна викласти новий контейнер або замінити лише окремий компонент системи. Це мінімізує час простою, що дуже важливо для автоматизованих трейдингових систем. Якщо на рівні біржі чи Telegram API змінюються протоколи, досить внести зміни у вихідний код і знову перевипустити контейнер.

Зрештою, описаний підхід із контейнеризацією, належним зберіганням секретів та налагодженим моніторингом дозволяє створити відмовостійку, безпечну й масштабовану систему. Telegram-бот інтегрований з AI-модулем через виклики усередині одного контейнера чи окремого сервісу, а взаємодія з біржею відбувається безпосередньо через ccxt або офіційний Binance API. Така архітектура дає змогу швидко випускати оновлення, контролювати етапи експлуатації й аналізувати вплив змін на прибутковість торгових стратегій.

РОЗДІЛ 3.
РОЗРОБКА TELEGRAM-БОТА ДЛЯ АЛГОРИТМІЧНОЇ ТОРГІВЛІ НА ОСНОВІ ШІ

3.1. Архітектура та логіка взаємодії компонентів

Для реалізації алгоритмічної торгівлі з використанням ШІ було розроблено комплексну систему, в якій кожен компонент виконує чітко визначені завдання та взаємодіє із рештою елементів через стандартизовані інтерфейси. У загальному вигляді система складається з Telegram-бота, сервера з вбудованою моделлю на основі штучного інтелекту, брокерської шини RabbitMQ, а також сховища проміжних даних і модуля інтеграції з Binance API (рис. 3.1).

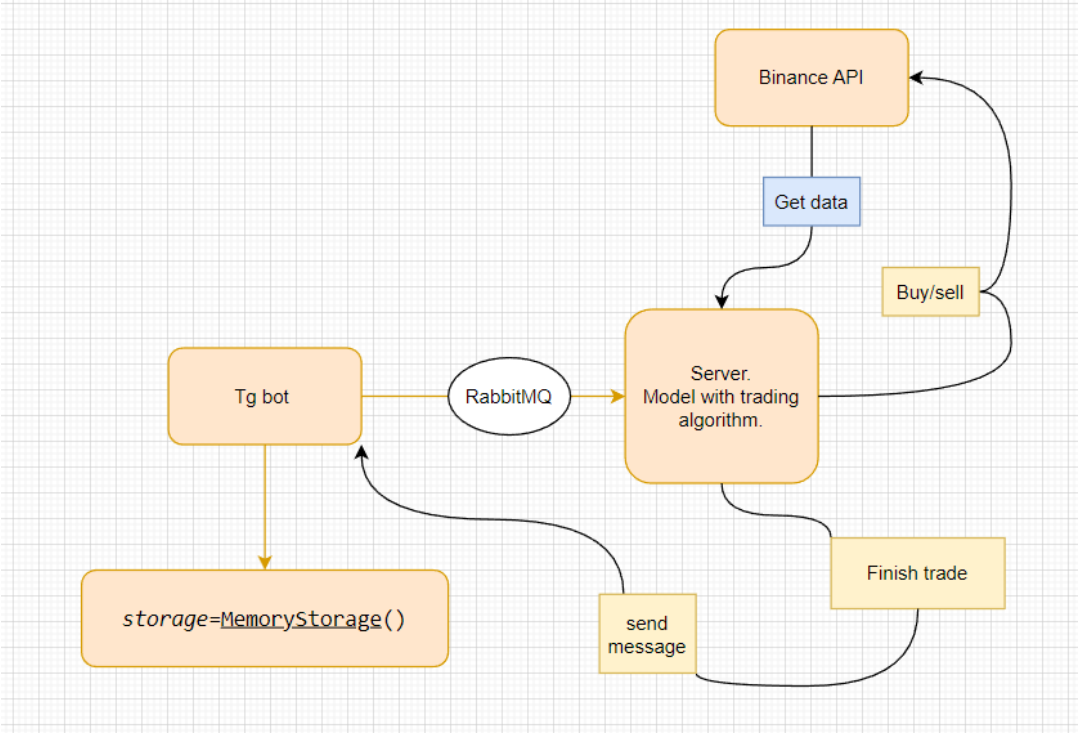


Рисунок 3.1 – Загальна схема проекту

Telegram-бот виступає основною точкою взаємодії з користувачем. Він приймає повідомлення, формує запити та надсилає їх на сервер для подальшої обробки. Користувач може відправляти команди, переглядати історію торгів, перевіряти статус поточних угод тощо.

Використовується Telegram Bot API для приймання та відправлення повідомлень.

Задіяне MemoryStorage (вбудоване сховище) для зберігання проміжних даних про поточну сесію користувача (наприклад, налаштування бота, вибрані пари для торгів, останні команди).

RabbitMQ. Після отримання повідомлень від користувача, бот через чергу RabbitMQ пересилає завдання на сервер. Використання брокера повідомлень забезпечує:

Асиметричний обмін даними: бот не блокується в очікуванні відповіді сервера, що покращує швидкодію та масштабованість.

Надійність передачі: RabbitMQ гарантує доставлення повідомлень навіть у випадку часткових збоїв чи пікового навантаження.

Прогноз та прийняття рішень: модель ШІ на основі обраної архітектури (наприклад, нейронна мережа чи інший метод машинного навчання) визначає найбільш ймовірні рухи ринку та видає рекомендації щодо купівлі чи продажу.

Логіка торгів: після отримання сигналу від моделі сервер формує запит на біржу Binance для виконання операції (купівля/продаж).

Зворотний зв’язок: інформація про успішність операції або статус заявки повертається сервером в RabbitMQ, а далі – до Telegram-бота.

Binance API слугує джерелом даних про поточні курси криптовалют та точкою доступу для виконання торгових операцій:

Отримання ринкових даних у реальному часі (Order Book, поточні ціни, обсяг торгів).

Здійснення операцій купівлі та продажу з урахуванням усіх налаштувань облікового запису користувача (баланс, доступні пари тощо).

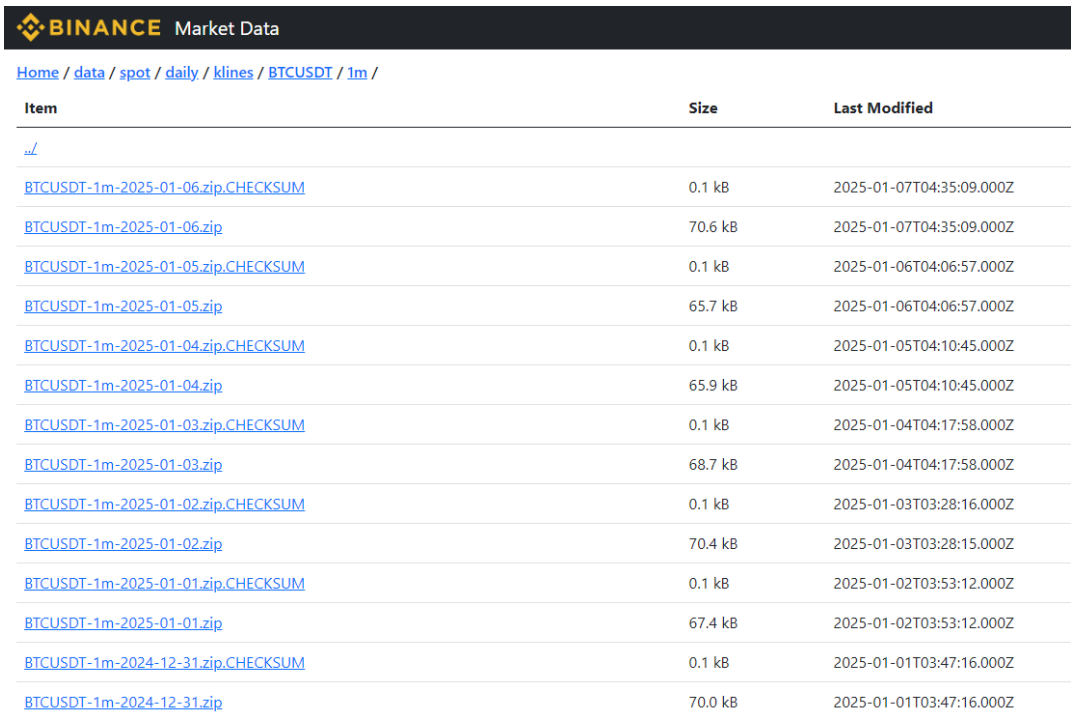
Після успішного або невдалого виконання угоди сервер формує підсумковий звіт (фінальну ціну, отриманий прибуток чи збиток, підтвердження від Binance) і надсилає його в чергу RabbitMQ. Звідти бот обробляє відповідь та відправляє користувачеві повідомлення про результат.

Для зберігання проміжних даних (наприклад, настройки користувача, параметри запиту, останні стани торгових сесій) використовується MemoryStorage. Він полегшує керування сесією та мінімізує затрати на роботу з базою даних, проте є проміжним рішенням.

Таким чином, архітектура проекту поєднує простоту розробки (Telegram-бот та MemoryStorage) з гнучкістю та надійністю (RabbitMQ та інтеграція з Binance API). Центральним елементом залишається сервер з вбудованою ШІ-моделлю, яка виконує прогнозування та визначає стратегію торгів.

3.2. Збір та попередня обробка даних

Перш ніж перейти до побудови ШІ-моделі, варто сформувати якісний датасет, що міститиме історичні котирування та підготовлені ознаки (features). Для цього обрано пару BTC/USDT за червень 2022 року, а дані завантажено з офіційного сервісу Binance Data Portal. Завантажений файл у форматі CSV містить інформацію про ціну відкриття, найвищу та найнижчу ціну, ціну закриття, обсяг торгів і деякі додаткові поля (рис. 3.2).


BINANCE Market Data

Home / data / spot / daily / klines / BTCUSDT / 1m /

Item	Size	Last Modified
../		
BTCUSDT-1m-2025-01-06.zip.CHECKSUM	0.1 kB	2025-01-07T04:35:09.000Z
BTCUSDT-1m-2025-01-06.zip	70.6 kB	2025-01-07T04:35:09.000Z
BTCUSDT-1m-2025-01-05.zip.CHECKSUM	0.1 kB	2025-01-06T04:06:57.000Z
BTCUSDT-1m-2025-01-05.zip	65.7 kB	2025-01-06T04:06:57.000Z
BTCUSDT-1m-2025-01-04.zip.CHECKSUM	0.1 kB	2025-01-05T04:10:45.000Z
BTCUSDT-1m-2025-01-04.zip	65.9 kB	2025-01-05T04:10:45.000Z
BTCUSDT-1m-2025-01-03.zip.CHECKSUM	0.1 kB	2025-01-04T04:17:58.000Z
BTCUSDT-1m-2025-01-03.zip	68.7 kB	2025-01-04T04:17:58.000Z
BTCUSDT-1m-2025-01-02.zip.CHECKSUM	0.1 kB	2025-01-03T03:28:16.000Z
BTCUSDT-1m-2025-01-02.zip	70.4 kB	2025-01-03T03:28:15.000Z
BTCUSDT-1m-2025-01-01.zip.CHECKSUM	0.1 kB	2025-01-02T03:53:12.000Z
BTCUSDT-1m-2025-01-01.zip	67.4 kB	2025-01-02T03:53:12.000Z
BTCUSDT-1m-2024-12-31.zip.CHECKSUM	0.1 kB	2025-01-01T03:47:16.000Z
BTCUSDT-1m-2024-12-31.zip	70.0 kB	2025-01-01T03:47:16.000Z

Рисунок 3.2 – Завантаження файлу історичних даних

Початковий CSV-файл має кілька зайвих стовпчиків, що можуть не знадобитися під час моделювання. Для зручності було здійснено перейменування ключових полів (timestamp, Open, High, Low, Close, Volume, Count) та видалено колонки, які не використовуватимуться у подальшому аналізі. У бібліотеці pandas ця операція виконується засобами для перейменування та вилучення стовпчиків.

Для усунення некоректних записів або пропущених значень реалізовано очищення даних, що передбачає видалення рядків із пропусками. Це допомагає уникнути потенційних помилок у роботі моделі та підвищує точність аналізу. У pandas таку перевірку й видалення часто здійснюють за допомогою функцій dropna.

Оскільки в межах експерименту була необхідність позначити потенційний прибутковий рух ціни, створено нову колонку, що сигналізує про досягнення цільового рівня. У прикладі кожен рядок отримує значення “1”, якщо різниця між найвищою ціною (High) та ціною відкриття (Open) перевищує певний поріг. В іншому випадку фіксується “0”. Здійснено це за допомогою функції apply, яка аналізує кожен рядок і порівнює потрібні поля.

Для більш зручної роботи з часом виконано перетворення мілісекунд у більш придатні одиниці вимірювання. Далі колонку з часовим штампом виставлено як індекс таблиці. Це забезпечує зручний доступ до рядків за часовим інтервалом і полегшує можливу візуалізацію динаміки ринкових показників.

Насамкінець скориговані дані збережено в новий CSV-файл, де вже містяться ключові стовпчики, цільова змінна та очищені часові позначки. Саме цей файл у подальшому передаватиметься навчальним алгоритмам, що аналізуватимуть залежності між змінами ринкової ціни, обсягом торгів і визначеною ціллю.

Дотримання структурованого підходу до обробки даних дає змогу гарантувати високу якість вхідного датасету та покращити результати ШІ-моделі в алгоритмічній торгівлі.

3.3. Реєстрація binance API

Для інтеграції з біржою Binance потрібно створити ключі доступу (API Key та Secret Key), за допомогою яких бот зможе здійснювати запити на виконання торгових операцій.

Увійти до облікового запису Binance. Перейти до розділу керування API. У налаштуваннях акаунта є розділ «Управління API» (API Management) (рис. 3.3). Тут знаходиться список усіх наявних ключів і кнопка для створення нового.

Після натискання кнопки для створення API, задаємо назву ключа.

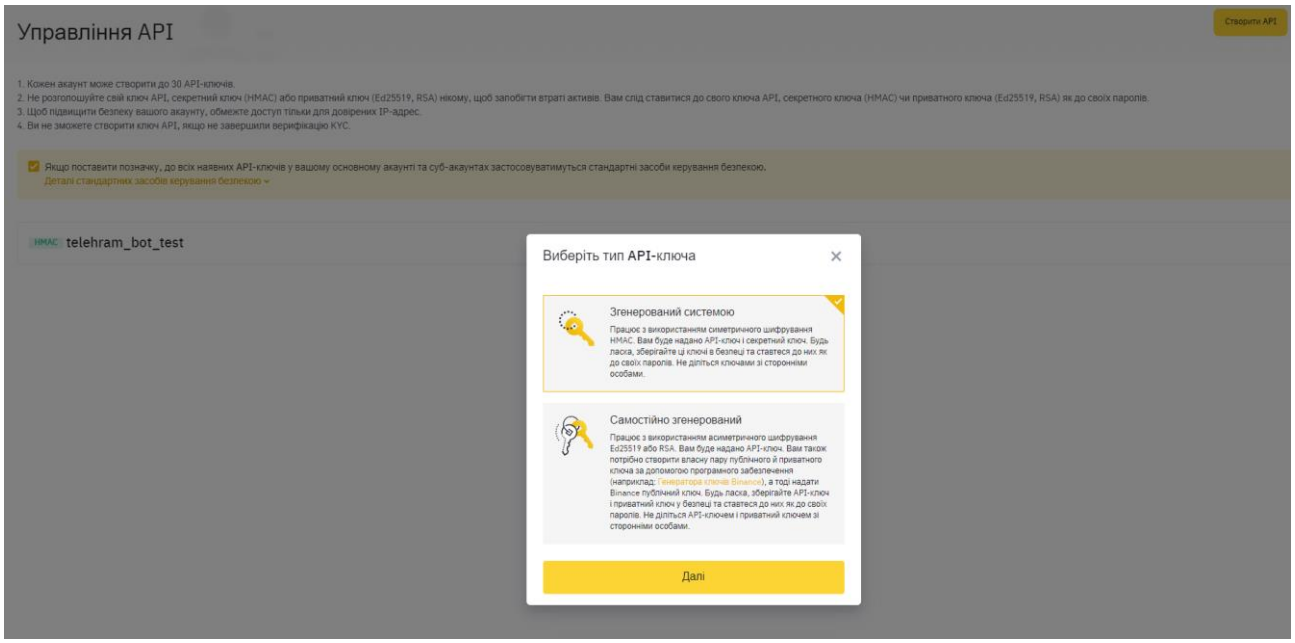


Рисунок 3.3 – Створення API Binance

Налаштувати рівень прав доступу. Серед варіантів доступу, які можна встановити для ключа, є:

Enable Reading (читання даних про рахунок та ринки);

Enable Spot & Margin Trading (дозвіл виконувати торгові операції);

Enable Withdrawals (зняття коштів). Для нашого бота зазвичай вистачить дозволів на читання та спот-торгівлю. Активізація зняття коштів потребує додаткових заходів безпеки.

Завершення процесу і збереження ключів. Після успішного створення API-ключа, обов'язково потрібно скопіювати та надійно зберегти як сам ключ (API Key), так і секретний ключ (Secret Key). Другий не можна буде переглянути повторно (рис. 3.4).

Рисунок 3.4 – Дані про Binance API

Створений ключ використовується в коді Telegram-бота, де він додається як параметр для ініціалізації підключення до Binance API. Це дає змогу боту отримувати дані про ринок та відправляти запити на торгівлю в автоматичному режимі.

3.4. Модель прогнозування та прийняття рішень

Код всіх методів та функцій наведений в додатку В.

Для побудови та навчання ШІ-моделі, здатної передбачати можливі рухи ринку та надавати сигнали для торгів, використалась бібліотека PyTorch Lightning.

Перед безпосереднім навчанням мережі вхідні дані було доповнено додатковими ознаками, що мають поліпшити результати прогнозу. Серед них – розрахунок технічних індикаторів (CCI, ROC, SMA, EWMA, Bollinger Bands, Force Index, EVM). Усі ці метрики додаються як окремі стовпчики до початкового датасету, що містить історію цін та обсяги торгів (рис. 3.4).

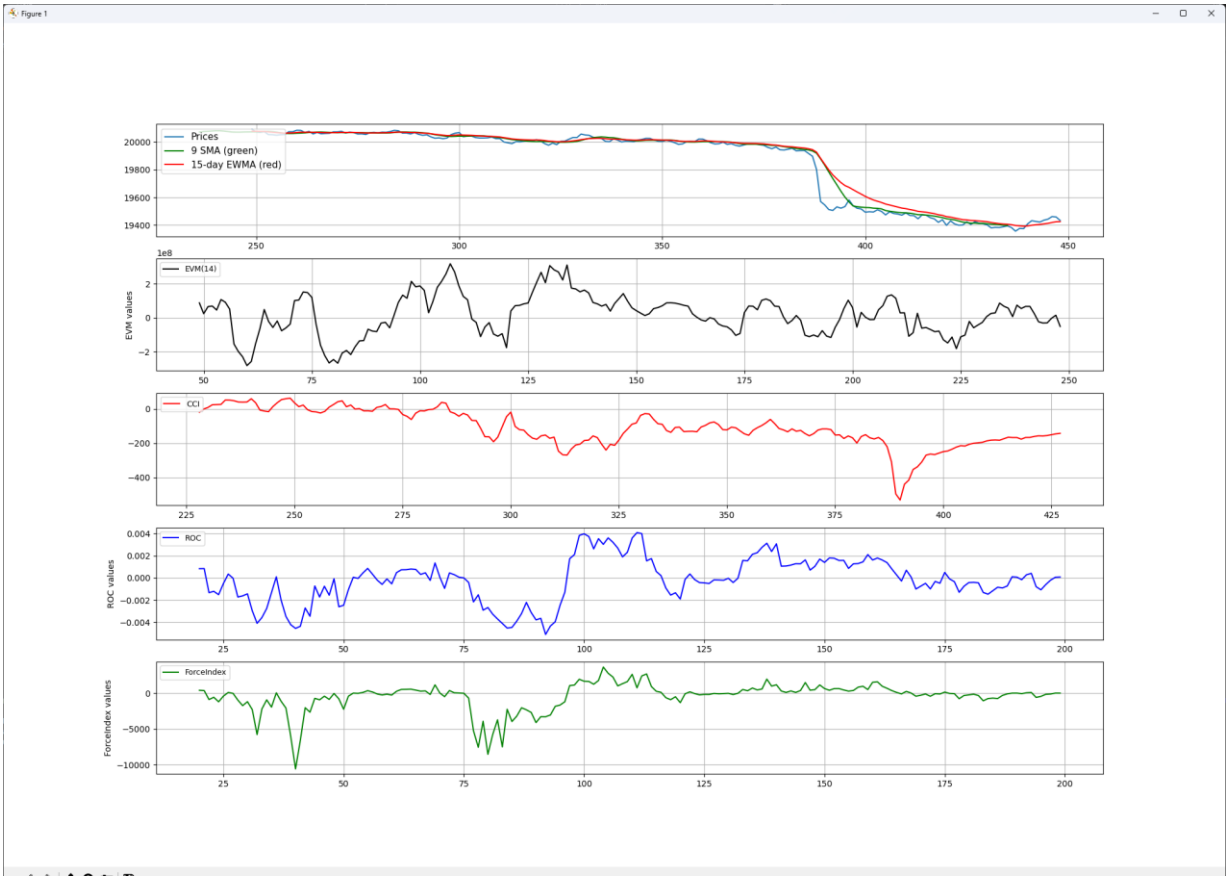


Рисунок 3.5 – Візуалізація даних для навчання моделі

Створення вікон (sequence): з загального набору спостережень формуються фрагменти певної довжини 250 рядків. До вихідного сигналу (таргета) відноситься останній рядок кожного фрагмента.

Нормалізація даних: для узгодженості масштабів різних індикаторів застосовано просту нормалізацію шляхом ділення кожної колонки на її норму.

Кодування цільової змінної: якщо таргет представлено у вигляді категорій (0/1 або декілька класів), його перетворюють за допомогою LabelEncoder на числові значення.

Архітектура LSTM-моделі. Модель було реалізовано на базі рекурентної нейронної мережі з типом шарів LSTM (Long Short-Term Memory). Подібна архітектура ефективно обробляє часові послідовності та вловлює залежності між віддаленими точками у часовому ряду. Основні компоненти:

LSTM-шари: один або кілька шарів, що обробляють послідовні фрагменти завдовжки 250 кроків та формують вектор прихованого стану.

Вихідний шар (Linear): приймає останній прихований стан та перетворює його в прогноз для конкретних класів «ціна зросте» чи «ціна впаде».

Функція втрат, CrossEntropyLoss: визначає, наскільки вірними є передбачення порівняно з реальними мітками, і скеровує процес оновлення ваг.

Організація навчання в PyTorch Lightning.

DataModule: для зручності було створено спеціальний модуль, що керує завантаженням наборів даних для тренування, валідації та тестування.

SurfacePredictor (LightningModule): центральний клас, у якому визначено логіку прямих проходжень (forward) та обчислення похибки (training_step, validation_step, test_step). Також тут налаштовується оптимізатор і задається швидкість навчання.

Логування та збереження: у процесі навчання виконуються вимірювання точності (accuracy) та функції втрат (loss). Кращі результати зберігаються за допомогою ModelCheckpoint, а TensorBoardLogger дає змогу візуально відстежувати перебіг навчання.

Оскільки тренування LSTM-мережі на великих обсягах даних може вимагати значної обчислювальної потужності, у проєкті перевіряється наявність GPU, CUDA-сумісної відеокарти. Якщо така є, модель виконується саме на GPU, що пришвидшує обчислення. Також можна стежити за завантаженням відеокарти та використанням пам’яті, аби обирати оптимальний розмір batch, кількість епох і рівень деталізації даних.. Якщо з GPU виникають обмеження, тренування можуть виконуватися на процесорі (CPU), але це збільшить загальний час навчання.

Після налаштування усіх компонентів запускається цикл навчання. Набір параметрів – кількість епох для тестування було обрано 10, але в подальшому модель буде тренуватись 500 епох. В процесі тренування відбуваються: послідовна подача пакетів даних через мережу. Обчислення і поширення похибки назад (backpropagation), оновлення ваг. Контроль за рівнем валідаційної похибки на окремій вибірці. Збереження кращих ваг у файл, якщо показники покращуються.

Після завершення навчання модель перевіряють на тестовій вибірці, щоб оцінити її узагальнювальну здатність. З результатів тестування можна побачити точність (accuracy) та інші метрики, які визначено у процесі конфігурації. Потім збережені ваги моделі можна завантажувати назад у мережу для подальшого використання у виробничому середовищі, тобто в реальному часі у Telegram-боті.

Таким чином, реалізована LSTM-модель на базі PyTorch Lightning дає змогу автоматично обробляти часові дані, аналізувати кілька технічних індикаторів і передбачати ймовірний рух ціни. Це створює основу для швидкого прийняття торгових рішень, інтегрованих у боті.

3.5. Реєстрація та налаштування Telegram Bot API

Для реалізації віртуального помічника з алгоритмічної торгівлі необхідно спочатку створити Telegram-бота й отримати до нього унікальний API-токен. Це дасть змогу боту взаємодіяти з користувачами через офіційну інфраструктуру Telegram.

Для початку у пошуку Telegram знаходимо офіційного бота BotFather. Він відповідає за реєстрацію й налаштування усіх ботів у Telegram Bot API.

Надсилаємо BotFather команду /newbot. Вказуємо ім’я (Name) бота. Вказуємо юзернейм (Username), що має закінчуватися на ...bot (Duptrending601TNBot).

Після підтвердження бот буде створено, а у відповідь прийде повідомлення з унікальним API-токеном (рис. 3.5).

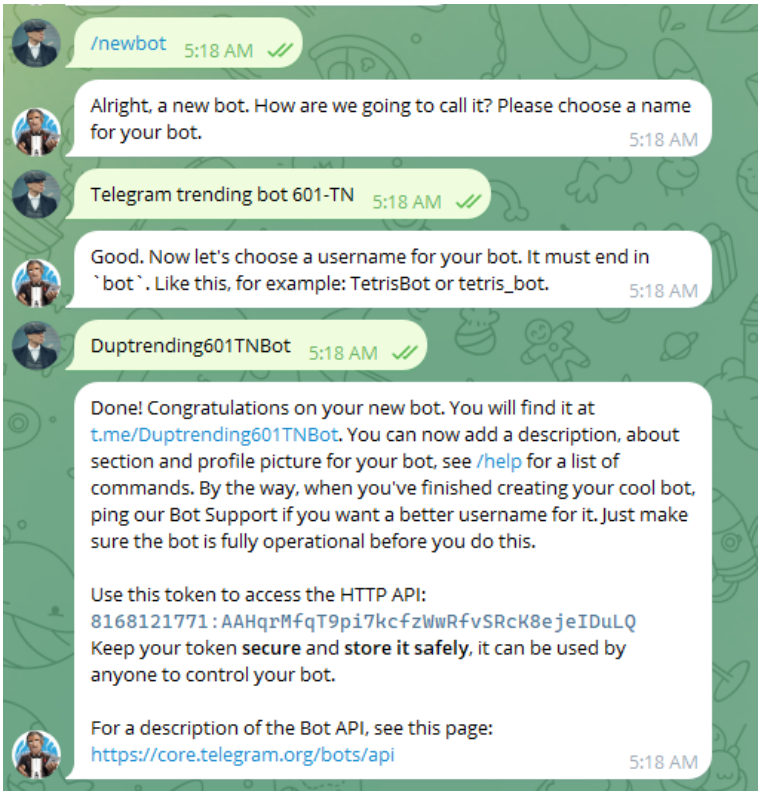


Рисунок 3.5 – Створення телеграм бота

Надання боту додаткових налаштувань. За допомогою додаткових команд у BotFather (табл. 3.1).

Таблиця 3.1 – Команди для налаштування телеграм бота

Команда	Опис команди
/token	Встановлює токен доступу до бота
/setname	Змінює ім'я бота
/setdescription	Змінює опис бота
/setcommands	Змінює список команд бота
/setuserpic	Змінює фотографію бота
/deletebot	Видаляє бота
/revokebot	Скасовує токен доступу до бота
/cancel	Скасовує поточну дію
/help	Показує довідку по командах

Створений у такий спосіб Telegram-бот забезпечує базову точку доступу до всієї логіки алгоритмічної торгівлі. Надалі його буде інтегровано з серверною частиною, модулями штучного інтелекту та торговими інтерфейсами Binance API, щоб реалізувати повноцінного віртуального асистента для аналізу ринку й автоматичного здійснення угод.

3.6. Реалізація Telegram-бота: інтерфейс, виклики до ШІ-моделі, обробка запитів

Код всіх методів та функцій наведений в додатку В.

Цей код буде об’єднувати Telegram-бота, інтеграцію з Binance API та використання натренованої моделі для алгоритмічної торгівлі.

Для початку створимо обробник команди /start який надсилає привітальне повідомлення та відображає головне меню.

Перегляд балансу. Використовуючи API Binance, бот отримує баланс акаунта та відображає його у зручному форматі. У разі помилки користувач отримує відповідне повідомлення.

Покупка та продаж монет. При виборі опції купівлі чи продажу бот пропонує список монет (наприклад, BTC, ETH, USDT). Для цього використовуються InlineKeyboard з кнопками для швидкого вибору.

Після вибору монети генерується команда /buy або /sell із відповідною назвою монети.

Перегляд топ-валют. Бот використовує метод get_ticker() Binance API для отримання інформації про топ криптовалют за обсягом торгів.

Аналітика. Бот надає базову аналітику ринку, включно з поточною волатильністю, трендами та рекомендаціями щодо торгівлі, що дає змогу користувачу краще розумітися на ринку.

Налаштування**.** Ця функція дає змогу налаштувати бота під свої потреби, керувати сповіщення, налаштувати мову, та ін.

Оскільки основний акцент робиться на прийнятті торгових рішень на основі прогнозів ШІ-моделі та їх автоматичній реалізації через Binance API.

Для початку розглянемо використання черги повідомлень RabbitMQ, щоб приймати віддалені запити на запуск торгівельних стратегій. Загальна логіка взаємодії з RabbitMQ передбачає:

Встановлення з'єднання з брокером повідомлень. Реалізовано через асинхронну бібліотеку, що дозволяє «слухати» потрібну чергу, а також витримувати можливі затримки.

Оголошення черги. Якщо черга з потрібною назвою ще не існує, вона буде створена із заданою стратегією збереження повідомлень (durable = True).

Циклічне отримання повідомлень. Кожне повідомлення містить в собі JSON із параметрами торгівлі (API-ключі, торговий символ, кількість угод).

Обробку та розбір вмісту. Зчитуються необхідні поля (api_key, api_secret, symbol, user_id, num_trades, а також додатковий прапорець stop, що вказує на призупинення торгів).

Виклик торгової логіки. Якщо параметр stop дорівнює False, запускається спеціальна асинхронна функція торгівлі, яка далі виконує увесь необхідний цикл дій.

Після успішної обробки користувацького запиту (повідомлення), при виникненні будь-яких помилок або успішному завершенні сценарію, з'єднання з RabbitMQ закривається.

Наступним кроком є модуль підготовки даних. Він відповідає за:

Збір історичних даних з біржі. Застосунок звертається до Binance, щоб отримати котирування криптовалют за певний період у вигляді свічок.

Формування початкового DataFrame. Отримані котирування перетворюються у таблицю з усіма потрібними колонками, як-от Open, High, Low, Close, Volume, тощо.

Розрахунок індикаторів. Для моделі необхідно підготувати різноманітні технічні індикатори (наприклад, CCI, EVM, SMA, EMA, Bollinger Bands, Rate of

Change та ін.). На основі історичних цін і об’ємів торгів застосовуються формули для обчислення зазначених індикаторів, які додаються в DataFrame.

Формування цільового стовпця. За визначеною логікою позначається, чи в найближчому проміжку часу відбувався підйом ціни достатній для того, щоб вважати операцію успішною. Це формує «цільову змінну» (Target).

Нормалізація та енкодування даних. Після розрахунків індикаторів дані можуть бути нормалізовані за певними осями (наприклад, скейлінг або Z-нормалізація), а цільовий стовпець перетворений у категорії (0 або 1).

Створення послідовностей. Оскільки використовується рекурентна модель (LSTM), дані розбиваються на невеликі послідовності фіксованої довжини (наприклад, 250 рядків), де кожна послідовність має мітку (0 чи 1).

Завдяки такій обробці формується готовий набір вхідних і цільових даних, які можна зручно передавати в модель для передбачення.

Для реалізації інтелектуальної частини використано фреймворк глибинного навчання. Структура складових модуля така:

Створення Dataset та DataModule.

Dataset призначений для зберігання послідовностей і відповідних міток (класів). Кожен елемент містить набір індикаторів за N послідовних рядків та одну «цільову» мітку.

DataModule відповідає за поділ даних на навчальну, валідаційну і тестову вибірки (якщо це передбачено) та формує DataLoader-и. Для алгоритму, який застосовується в реальному часі, здебільшого використовується лише тестовий або inference-набір (наприклад, для оперативної перевірки передбачення).

Рекурентна модель (LSTM).

Архітектура містить рекурентний шар (LSTM) із заданою кількістю шарів і нейронів (n_hidden, n_layers), а також шар лінійної трансформації, що зводить вихід LSTM до потрібної кількості класів.

Деякі проміжні параметри, такі як dropout або ініціалізація, допомагають боротися з перенавчанням і стабілізують навчання.

Модуль для передбачення.

Містить виклики до основної моделі.

Застосовує функцію втрат (наприклад, кросентропію) та виконує обчислення ймовірностей для кожного класу.

У режимі тестування (або inference) прогноз класифікується як 0 чи 1 за максимальною ймовірністю.

Завантаження збережених ваг.

Після попереднього навчання в окремому скрипті збережені ваги моделі (файл із розширенням .pth).

Під час торгової сесії ці ваги завантажуються, і модель переходить у режим eval. Це дає змогу використовувати її для формування сигнальних рішень (купівля або утримання).

Загалом ці кроки дозволяють зібрати готову до роботи модель: LSTM, яка обчислює ймовірність зростання ціни та слугує основою для прийняття торгових рішень.

Після того, як модель вивела свій прогноз, у програмі виконується низка дій: авторизація на Binance. Для цього використовуються надані користувачем api_key та api_secret.

Отримання поточних даних. Безпосередньо перед ухваленням рішення програма отримує актуальні котирування (наприклад, останні свічки з біржі).

Аналіз передбачення. Якщо модель прогнозує, що ціна зростатиме, застосунок формує ордер на купівлю. Якщо ж необхідно зафіксувати прибуток або обмежити збитки, програма виконує продаж.

Встановлення умов та рівнів Take Profit / Stop Loss. У коді реалізована серія перевірок, які порівнюють поточну ціну з ціною відкриття ордера. Якщо ціна досягає певного порогу («+N доларів»), виконується продаж. Так само контролюється «-N доларів» для запобігання великим збиткам.

Безперервний цикл торгів. Логіка побудована таким чином, що виконання торгів повторюється до досягнення заданої кількості успішно відкритих і закритих угод (num_trades). По завершенню цикл може зупинятися або знову запускатися за запитом RabbitMQ.

Для оперативного інформування користувача про стан торгівлі застосунок надсилає повідомлення у Telegram. Основні етапи:

Отримання token і chat_id. У коді ці змінні зберігаються заздалегідь або отримуються з вхідних параметрів. token відповідає за доступ до бота, а chat_id (ідентифікатор користувача) потрібен для надсилання повідомлень конкретному користувачеві.

Надсилання повідомлень. При кожній покупці чи продажу, зміні балансу або виникненні помилки створюється HTTP-запит до API Telegram, де вказується текст повідомлення. Таким чином, користувач відразу бачить на смартфоні інформацію про свіжу операцію чи поточний баланс.

Статуси та звіти. У тексті повідомлень виводиться, зокрема:

Дані про символ і кількість активів для торгівлі.

Баланс до та після операцій.

Відсоток прибутковості від стартового балансу.

Деталі завершення кожної угоди й кількість залишкових угод, які ще планується виконати.

Завдяки такій структурі застосунок може працювати в безперервному режимі, відстежуючи ринкові зміни, виконуючи операції купівлі/продажу та передаючи їх статус у Telegram.

РОЗДІЛ 4.
ТЕСТУВАННЯ, ОЦІНКА ЕФЕКТИВНОСТІ ТА ОГЛЯД ФУНКЦІОНАЛУ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Результати реалізації продукту для торгівлі крипто валютою з використанням ШІ

Під час розробки було створено Telegram-бот, який автоматизує алгоритмічну торгівлю криптовалютами за допомогою штучного інтелекту. Бот забезпечує інтеграцію з біржею Binance через API, надає функції перегляду балансу, аналітики криптовалют і здійснення торгових операцій. У цьому розділі представлено результати роботи бота.

Після першого відкриття торгового борту, з назвою «Telegram trending bot 601-TN». В телеграмі можна бачити коротку інформацію про застосунок (рис. 4.1).

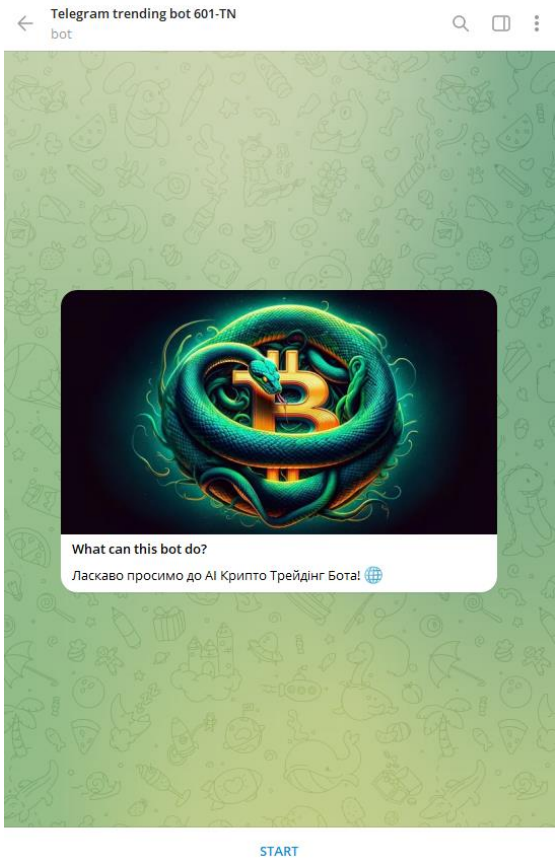


Рисунок 4.1 – Стартова сторінка телеграм-бота

Для початку введемо команду «/start» для початку роботи з телеграм ботом. Бот автоматично запитує API Key та API Secret, які потрібні для підключення акаунта Binance до системи. У вітальному повідомленні детально описано, для чого ці ключі потрібні, а також надано інструкції, що допоможуть користувачу правильно виконати дії. Інтерфейс забезпечує зручність користування, надаючи покрокові вказівки. (рис. 4.2).

Рисунок 4.2 – Привітання чат-бота та запит на введення даних для авторизації

Після введення API Key та API Secret користувач отримує повідомлення про успішне підключення акаунта до Telegram-бота. На екрані з'являється список основних функцій, які можна активувати через кнопки інтерфейсу, такі як «Баланс», «Купити монету», «Придати», «Торгувати BTC за допомогою ШІ»,

«Топ валют», «Налаштування» та «Аналітика». Інтерактивні кнопки дозволяють швидко переходити до відповідних функцій без додаткових команд. (рис. 4.3).

Рисунок 4.3 – Повідомлення про успішну авторизацію

Для перегляду балансу користувач може натиснути на кнопку «Баланс» після чого, бот виведе актуальну інформацію про баланс на акаунті, зокрема кількість наявних коштів у різних криптовалютах, таких як BNB, ETH, BTC, USDT, USDC. Цей функціонал дозволяє користувачу швидко оцінити стан своїх активів без необхідності заходити на інші платформи. (рис. 4.4).

Рисунок 4.4 – Вивід активного балансу

Також, можна отримати актуальний список найбільш популярних валют за оборотом за допомогою кнопки «Топ валют». Бот виводить інформацію про назви валют і їх обсяги в еквіваленті до USDT, що дозволяє трейдерам оцінити ринкову активність і визначити перспективні активи для торгівлі. (рис. 4.5).

Рисунок 4.5 – Вивід найбільш популярних валют

І головним функціоналом даного боту є автоматичний аналіз ринок і виконання угод з криптовалютною парою BTC/USDT: купівлю та продаж. Процес супроводжується докладними повідомленнями, включаючи інформацію про ціну покупки/продажу, кількість активів та підсумковий баланс. Завдяки прозорому відображенню дій користувач отримує повну картину роботи бота та впевненість у коректності виконання операцій.

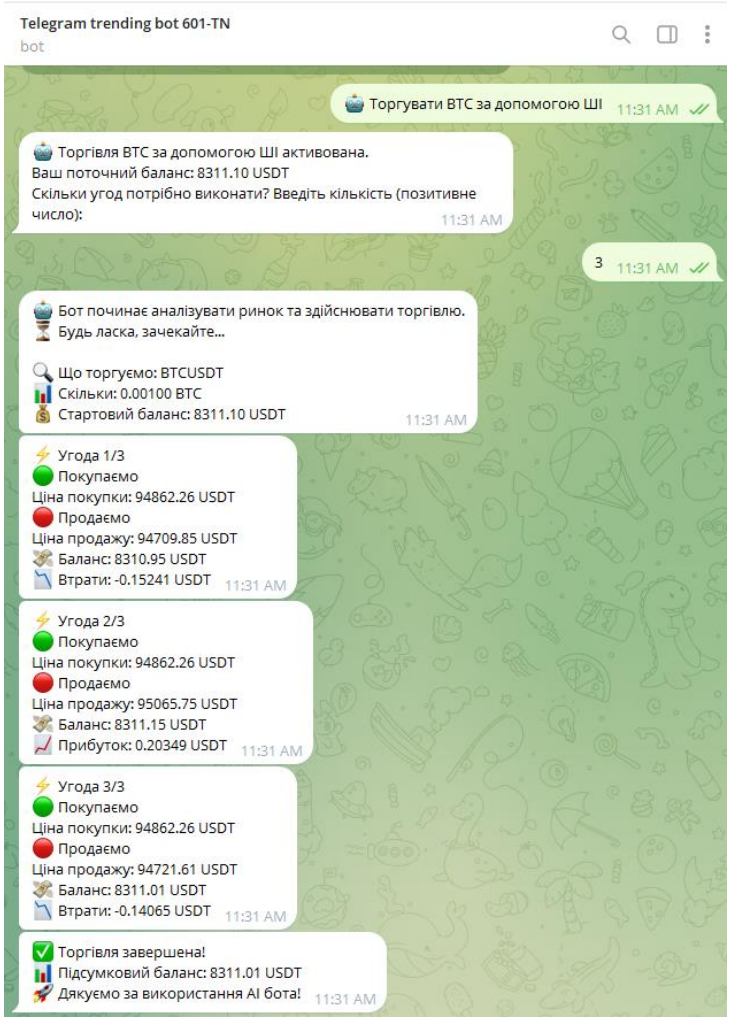


Рисунок 4.6 – Процес торгів

Після завершення торгівлі бот повідомляє про виконані угоди та підсумковий баланс. Це забезпечує прозорість і зручність для користувача.

Результати демонструють ефективність розробленого продукту та підтверджують його функціональність у реальних умовах. Бот є зручним інструментом як для початківців, так і для досвідчених трейдерів.

4.2. Тестування

Для перевірки роботи системи було обрано ручний метод тестування, який забезпечує найбільшу точність при перевірці взаємодії з користувачем та функціоналу. Ручне тестування дозволило детально перевірити кожен елемент системи, виявити можливі помилки та неточності.

Для початку створемо тест план по якому будемо тестувати додаток.

1. Реєстрація нового користувача з валідними даними (таблиця А.1): введення коректних API Key та API Secret. Перевірка повідомлення про успішне підключення.
2. Реєстрація нового користувача з невалідними даними (таблиця А.2): введення некоректних API Key та API Secret. Перевірка обробки помилок та відповідного повідомлення.
3. Перегляд балансу (таблиця А.3): запит балансу після підключення до Binance API. Перевірка коректного відображення даних.
4. Купити монету з валідними параметрами (таблиця А.4): виконання операції купівлі BTC/USDT. Перевірка коректного збереження даних про угоду.
5. Купити монету з невалідними параметрами (таблиця А.5): спроба здійснити операцію з некоректними параметрами. Перевірка повідомлення про помилку.
6. Скасування транзакції (таблиця А.6): скасування операції на стадії вибору параметрів. Перевірка повідомлення про скасування.
7. Аналітика ринку (таблиця А.7): Запит топових криптовалют. Перевірка коректного відображення даних.

Ручне тестування підтвердило функціональність і стабільність системи, дозволяючи ідентифікувати та усунути проблему, а саме відсутність скасування транзакції на стадії вибору параметрів.

4.3. Введення в екстлуатацію

Розроблений Telegram-бот може бути запущений як на хмарному хостингу, так і на локальному сервері. Для введення в експлуатацію я обрав, легкий та дешевий спосіб, а саме запуск на локальному сервері, для якого необхідно виконати такі кроки:

Створити віртуальне середовища. Виконати команду python -m venv venv для створення ізольованого середовища розробки.

Активувати середовище за допомогою команди venv\Scripts\activate (Windows) або source venv/bin/activate (Linux/Mac).

У терміналі середовища виконати команду pip install -r requirements.txt для встановлення всіх необхідних бібліотек.

Завантаження вже навченої моделі на локальний сервер. Або при необхідності скориставшись скриптами data preprocessing.py та lstm.py для тренування нової моделі.

Виконати команду python bot.py для запуску основного скрипта.

При необхідності Telegram-бот також може бути розгорнутий на хмарному сервері (наприклад, AWS, Heroku чи Google Cloud) для забезпечення доступності 24/7. Перед запуском на хмарі слід адаптувати налаштування, включаючи змінні оточення та автоматизацію процесу розгортання.

Введення бота в експлуатацію відкриває можливості для автоматизованої торгівлі та забезпечує ефективне управління криптовалютними активами користувачів.

4.4. Можливості масштабування та подальшого розвитку проєкту

Створена система автоматизації торгівлі, має низку можливостей для масштабування та подальшого вдосконалення.

Інтеграція з іншими біржами. Окрім Binance можна додати підтримку Coinbase, KuCoin, Kraken або інших популярних платформ. Це дозволить трейдерам диверсифікувати ризики та ефективніше використовувати різні ринкові можливості.

Додаткові індикатори та стратегії. У систему можна додати інші алгоритми технічного аналізу, індикатори та фільтри для покращення точності сигналів купівлі/продажу. Зокрема більш просунуті методи обробки часових рядів, включно з моделями типу Transformer, або нейронні мережі, що враховують новини та сентимент (NLP-аналіз).

Керування ризиками та автоматичне коригування позицій. Додавання спеціального модуля ризик-менеджменту з розрахунком максимальної кількості угод, оптимальних лімітів та управлінням капіталом забезпечить стабільніший результат торгів.

Використання методів ансамблю (поєднання кількох моделей) здатне зменшити рівень помилок і знизити ймовірність «хибнопозитивних» сигналів.

Розширення Telegram-інтерфейсу та інтеграція з іншими каналами. Інтеграція з іншими месенджерами (Facebook Messenger, Slack тощо) або вебдодатком розширить аудиторію користувачів.

Можна створити окремий інформаційний канал (канал сигналів) або онлайн-панель управління, де будуть доступні статистичні графіки, налаштування ордерів, історія угод тощо.

Таким чином, поточний проєкт має великі перспективи розвитку як у плані внутрішнього масштабування.

ВИСНОВКИ

В рамках проведеного дослідження розроблено та продемонстровано додаток системи алгоритмічної торгівлі криптовалютами з використанням штучного інтелекту, що інтегрований у Telegram-бот. У роботі послідовно проаналізовано актуальність застосування ШІ для торгівлі, розглянуто існуючі підходи та інструменти, а також реалізовано робоче рішення, яке включає повний цикл від попередньої обробки даних до прийняття торгових рішень і викликів до біржі.

Поєднання програмного забезпечення для опрацювання торгових сигналів (AI-модель) із зручним інтерфейсом у вигляді Telegram-бота.

Створення повноцінного інструменту, здатного працювати з високочастотними ринковими даними в режимі реального часу, виконувати обчислювально складні завдання з аналізу та формування сигналів.

Реалізація можливості гнучко налаштовувати торгові стратегії, а також інформувати трейдера про ключові події за допомогою повідомлень у Telegram.

Отримані результати демонструють, що запропонований підхід може бути успішно використаний для автоматизації торгівлі у високоризиковому криптовалютному середовищі. Водночас проєкт залишається відкритим для оптимізації та розширення: застосування удосконалених архітектур нейронних мереж, інтеграції більшої кількості бірж і інструментів, впровадження продуманого ризик-менеджменту та налаштування комплексної системи моніторингу.

Таким чином, виконана робота підтверджує життєздатність і перспективність алгоритмічних рішень на базі ШІ для криптовалютних ринків. Запропонована система є гнучкою платформою, яку можна масштабувати й налаштовувати під різні стратегії й умови. Подальший розвиток проєкту створить умови для більшої стабільності, адаптивності та зручності використання, відкриваючи можливість розширення функціоналу та залучення більш широкого кола зацікавлених користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритмічна торгівля [Електронний ресурс]. – Режим доступу: https://www.investopedia.com/terms/a/algorithmictrading.asp (дата звернення: 10.09.2024).

2. Зародження алгоритмічної торгівлі: історія і перспективи [Електронний ресурс]. – Режим доступу: https://www.tradingview.com/ideas/algorithmictrading (дата звернення: 10.01.2024).

3. High-Frequency Trading (HFT): основи та переваги [Електронний ресурс]. – Режим доступу: https://www.bloomberg.com/high-frequency-trading (дата звернення: 10.09.2024).

4. Інтеграція API для торгових ботів у криптовалютній сфері [Електронний ресурс]. – Режим доступу: https://www.binance.com/en/api (дата звернення: 10.09.2024).

5. Волатильність криптовалютного ринку: можливості та виклики [Електронний ресурс]. – Режим доступу: https://www.coindesk.com/volatility-crypto-market (дата звернення: 10.09.2024).

6. Інтеграція штучного інтелекту в алгоритмічну торгівлю [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/topics/artificial-intelligence-trading (дата звернення: 12.09.2024).

7. Застосування нейронних мереж у фінансовому аналізі [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/neural-networks-in-finance (дата звернення: 13.09.2024).

8. Особливості волатильності криптовалютного ринку [Електронний ресурс]. – Режим доступу: https://www.investopedia.com/cryptocurrency-volatility (дата звернення: 13.09.2024).

9. Різноманіття криптоактивів і їхній вплив на торгівлю [Електронний ресурс]. – Режим доступу: https://www.coinmarketcap.com/crypto-assets (дата звернення: 17.09.2024).

10. Кібербезпека у криптовалютній торгівлі: рекомендації [Електронний ресурс]. – Режим доступу: https://www.binance.com/crypto-security (дата звернення: 18.09.2024).

11. Інструменти Natural Language Processing для аналізу новин [Електронний ресурс]. – Режим доступу: https://www.towardsdatascience.com/nlp-in-finance (дата звернення: 18.09.2024).

12. Методи машинного навчання у фінансовій галузі [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/machine-learning-in-finance (дата звернення: 20.09.2024).

13. Глибоке навчання для трейдингу: архітектури нейронних мереж [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/deep-learning-for-trading (дата звернення: 23.09.2024).

14. Інструменти для аналізу ризиків у криптовалютах [Електронний ресурс]. – Режим доступу: https://www.coinmarketcap.com/risk-analysis-tools (дата звернення: 23.09.2024).

15. Бібліотеки машинного навчання для фінансового аналізу (TensorFlow, PyTorch, scikit-learn) [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/resources (дата звернення: 23.09.2024).

16. Глобальні тенденції у регулюванні штучного інтелекту [Електронний ресурс]. – Режим доступу: https://www.oecd.org/ai-policy (дата звернення: 25.09.2024).

17. Ліцензування та реєстрація криптокомпаній в Україні [Електронний ресурс]. – Режим доступу: https://www.nkrzi.gov.ua/virtual-assets-ukraine (дата звернення: 25.09.2024).

18. Захист прав споживачів у криптовалютній сфері [Електронний ресурс]. – Режим доступу: https://www.coindesk.com/consumer-protection-crypto (дата звернення: 26.09.2024).

19. Telegram global MAU 2022 | статистика [Електронний ресурс]. – Режим доступу: https://www.statista.com/statistics/234038/telegram-messenger-mau-users/(дата звернення: 05.09.2024).

20. Python: офіційна документація [Електронний ресурс]. – Режим доступу: https://docs.python.org/3 (дата звернення: 30.09.2024).

21. Огляд бібліотек для створення Telegram-ботів [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api (дата звернення: 02.10.2024).

22. Порівняння середовищ розробки для Python [Електронний ресурс]. – Режим доступу: https://realpython.com/python-ides-code-editors-guide (дата звернення: 12.10.2024).

23. PyTorch: офіційна документація [Електронний ресурс]. – Режим доступу: https://pytorch.org/docs (дата звернення: 12.10.2024).

24. Розробка часових рядів з LSTM та GRU у PyTorch [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/time-series-lstm-pytorch (дата звернення: 15.10.2024).

25. Binance API Documentation [Електронний ресурс]. – Режим доступу: https://binance-docs.github.io/apidocs (дата звернення: 16.10.2024).

26. Використання WebSocket для отримання ринкових даних [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 18.10.2024).

ДОДАТОК А
ТЕСТ-КЕЙСИ

Таблиця А.1 – Реєстрація нового користувача з валідними даними

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм бота та ввести команду «/start»	Відображається привітальне повідомлення та запит електронної адреси
2	Ввести валідні API Key та API Secret	Відображається повідомлення про успішне підключення
Результат	Пройдено	

Таблиця А.2 – Реєстрація нового користувача з невалідними даними

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм бота та ввести команду «/start»	Відображається привітальне повідомлення та запит електронної адреси
2	Ввести не валідні API Key та API Secret	Відображається повідомлення про помилку
Результат	Пройдено	

Таблиця А.3 – Перегляд балансу

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм бота та натиснути кнопку «Баланс»	Виводиться інформацію про баланс на акаунті
Результат	Пройдено	

Таблиця А.4 – Купити монету з валідними параметрами

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм та натиснути кнопку «Купити монету»	Відображається повідомлення про вибір монет для покупок
2	Обрати монету, та ввести суму (формату 0.00 використовуючи "." або ",")	Відображається повідомлення про успішне розміщення ордеру
Результат	Пройдено	

Таблиця А.5 – Купити монету з невалідними параметрами

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм та натиснути кнопку «Купити монету»	Відображається повідомлення про вибір монет для покупок
2	Обрати монету, та ввести ввести некоректну суму (наприклад, ввести букви замість чисел)	Відображається повідомлення про невірно введені дані
Результат	Пройдено	

Таблиця А.6 – Скасування транзакції

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм та натиснути кнопку «Торгувати BTC за допомогою ШІ»	Відображається повідомлення про вибір параметрів для торгів
2	Натиснути кнопку «Скасувати»	Відображається повідомлення про скасування операції
Результат	Пройдено	

Таблиця А.7 – Аналітика ринку

№	Кроки виконання	Очікуваний результат
1	Відкрити телеграм бота та та натиснути кнопку «Топ валют»	Виводиться список найпопулярніших валют за обсягом торгів
Результат	Пройдено	

ДОДАТОК В
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

```
import io
import json
import requests
import functools
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import yfinance

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.nn import functional as F
from torch.autograd import Variable

import pytorch_lightning as pl
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping
from pytorch_lightning.loggers import TensorBoardLogger
from torchmetrics.functional import accuracy

use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")

torch.manual_seed(0)
if use_cuda:
    torch.cuda.manual_seed(0)
np.random.seed(0)

def CCI(data, ndays):
    TP = (data['High'] + data['Low'] + data['Close']) / 3
    CCI_serie = pd.Series(
        (TP - TP.rolling(ndays).mean()) / (0.015 * TP.rolling(ndays).std()),
        name='CCI'
    )
    data = data.join(CCI_serie)
    return data

def EVM(data, ndays):
```

```
    dm = ((data['High'] + data['Low'])/2) - ((data['High'].shift(1) +
data['Low'].shift(1))/2)
    br = (data['Volume'] / 100000000) / (data['High'] - data['Low'])
    evm_serie = dm / br
    EVM_MA = pd.Series(evm_serie.rolling(ndays).mean(), name='EVM')
    data = data.join(EVM_MA)
    return data

def SMA(data, ndays):
    SMA_serie = pd.Series(data['Close'].rolling(ndays).mean(), name='SMA')
    data = data.join(SMA_serie)
    return data

def EWMA(data, ndays):
    EMA = pd.Series(
        data['Close'].ewm(span=ndays, min_periods=ndays - 1).mean(),
        name='EWMA_' + str(ndays)
    )
    data = data.join(EMA)
    return data

def BBANDS(data, window):
    MA = data['Close'].rolling(window).mean()
    SD = data['Close'].rolling(window).std()
    data['UpperBB'] = MA + (2 * SD)
    data['LowerBB'] = MA - (2 * SD)
    return data

def ForceIndex(data, ndays):
    FI = pd.Series(data['Close'].diff(ndays) * data['Volume'], name='ForceIndex')
    data = data.join(FI)
    return data

def ROC(data, n):
    N = data['Close'].diff(n)
    D = data['Close'].shift(n)
    roc_serie = pd.Series(N / D, name='Rate of Change')
    data = data.join(roc_serie)
    return data


df = pd.read_csv('train_2022_Jun.csv')
print("Форма даних:", df.shape)
print("Стовпці:", df.columns)

data = df.copy()

n = 20
NIFTY_ROC = ROC(data, n)
ROC_serie = NIFTY_ROC['Rate of Change']
```

```
data = NIFTY_ROC

n = 50
NIFTY_BBANDS = BBANDS(data, n)
data = NIFTY_BBANDS

n = 20
AAPL_ForceIndex = ForceIndex(data, n)
ForceIndex_serie = AAPL_ForceIndex['ForceIndex']
data = AAPL_ForceIndex


n = 14
AAPL_EVM = EVM(data, n)
AAPL_EVM = AAPL_EVM.dropna()
EVM_serie = AAPL_EVM['EVM']
data = AAPL_EVM

# 60*3 = 180
n = 60*3
NIFTY_CCI = CCI(data, n)
NIFTY_CCI = NIFTY_CCI.dropna()
CCI_serie = NIFTY_CCI['CCI']
data = NIFTY_CCI

n = 9
SMA_NIFTY = SMA(data, n)
SMA_NIFTY = SMA_NIFTY.dropna()
SMA_serie = SMA_NIFTY['SMA']
data = SMA_NIFTY

ew = 15
EWMA_NIFTY = EWMA(data, ew)
EWMA_NIFTY = EWMA_NIFTY.dropna()
EWMA_serie = EWMA_NIFTY[f'EWMA_{ew}']
data = EWMA_NIFTY

fig = plt.figure(figsize=(20, 15))

ax = fig.add_subplot(5, 1, 1)
plt.plot(data['Close'][:200], label='Prices')
plt.plot(SMA_serie[:200], 'g', label='9 SMA (green)')
plt.plot(EWMA_serie[:200], 'r', label='15-day EWMA (red)')
plt.legend(loc=2, prop={'size': 11})
plt.grid(True)

bx = fig.add_subplot(5, 1, 2)
plt.plot(EVM_serie[:200], 'k', linestyle='-', label='EVM(14)')
plt.legend(loc=2, prop={'size': 9})
plt.ylabel('EVM values')
```

```
plt.grid(True)

cx = fig.add_subplot(5, 1, 3)
plt.plot(CCI_serie[:200], 'r', linestyle='-', label='CCI')
plt.legend(loc=2, prop={'size': 9.5})
plt.grid(True)

dx = fig.add_subplot(5, 1, 4)
plt.plot(ROC_serie[:200], 'b', linestyle='-', label='ROC')
plt.legend(loc=2, prop={'size': 9})
plt.ylabel('ROC values')
plt.grid(True)

ex = fig.add_subplot(5, 1, 5)
plt.plot(ForceIndex_serie[:200], 'g', linestyle='-', label='ForceIndex')
plt.legend(loc=2, prop={'size': 9})
plt.ylabel('ForceIndex values')
plt.grid(True)

plt.show()

data = data.dropna()
print("Форма даних після видалення NaN:", data.shape)

if 'timestamp' in data.columns:
    data = data.set_index('timestamp')

target_col = 'Target'
features = [col for col in data.columns if col not in [target_col]]
print("Ознаки:", features)

X_all = data[features].copy()
y_all = data[target_col].copy()

X_all = pd.DataFrame(preprocessing.normalize(X_all, axis=0), columns=X_all.columns,
index=X_all.index)

label_encoder = LabelEncoder()
y_all_enc = label_encoder.fit_transform(y_all)

y_all_enc = pd.Series(y_all_enc, index=X_all.index, name='label')

print("Унікальні класи:", label_encoder.classes_)

seq_length = 250
sequences = []
indexes = X_all.index.tolist()
for i in range(len(X_all) - seq_length):
    x_window = X_all.iloc[i : i+seq_length]
    y_label = y_all_enc.iloc[i + seq_length - 1]
```

```
    sequences.append((x_window, y_label))

print("Загальна кількість сформованих послідовностей:", len(sequences))

# Поділ train/test
train_sequences, test_sequences = train_test_split(sequences, test_size=0.2,
random_state=42)
print("Кількість train-послідовностей:", len(train_sequences))
print("Кількість test-послідовностей:", len(test_sequences))

class SurfaceDataset(Dataset):
    def __init__(self, sequences):
        super().__init__()
        self.sequences = sequences

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, idx):
        x_window, y_label = self.sequences[idx]
        return {
            "sequence": torch.tensor(x_window.values, dtype=torch.float32),
            "label": torch.tensor(y_label, dtype=torch.long)
        }

class SurfaceDataModule(pl.LightningDataModule):
    def __init__(self, train_seq, test_seq, batch_size=128):
        super().__init__()
        self.train_seq = train_seq
        self.test_seq = test_seq
        self.batch_size = batch_size

    def setup(self, stage=None):
        self.train_dataset = SurfaceDataset(self.train_seq)
        self.test_dataset  = SurfaceDataset(self.test_seq)

    def train_dataloader(self):
        return DataLoader(self.train_dataset, batch_size=self.batch_size,
shuffle=True, drop_last=True)

    def val_dataloader(self):
        return DataLoader(self.test_dataset, batch_size=self.batch_size,
shuffle=False, drop_last=True)

    def test_dataloader(self):
        return DataLoader(self.test_dataset, batch_size=self.batch_size,
shuffle=False, drop_last=True)

# DataModule
BATCH_SIZE = 128
```

```
data_module = SurfaceDataModule(train_sequences, test_sequences, BATCH_SIZE)

class SequenceModel(nn.Module):
    def __init__(self, n_features, n_classes, n_hidden=128, n_layers=2):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size=n_features,
            hidden_size=n_hidden,
            num_layers=n_layers,
            batch_first=True,
            dropout=0.3
        )
        self.linear = nn.Linear(n_hidden, n_classes)

    def forward(self, x):
        # x shape: [batch_size, seq_length, n_features]
        _, (hidden, _) = self.lstm(x)
        out = hidden[-1]
        out = self.linear(out)
        return out

class SurfacePredictor(pl.LightningModule):
    def __init__(self, n_features, n_classes):
        super().__init__()
        self.model = SequenceModel(n_features, n_classes)
        self.criterion = nn.CrossEntropyLoss()

    def forward(self, x, labels=None):
        logits = self.model(x)
        loss = None
        if labels is not None:
            loss = self.criterion(logits, labels)
        return loss, logits

    def training_step(self, batch, batch_idx):
        sequences = batch["sequence"]
        labels    = batch["label"]
        loss, outputs = self.forward(sequences, labels)
        predictions = torch.argmax(outputs, dim=1)


        step_accuracy = accuracy(
            predictions,
            labels,
            task="multiclass",
            num_classes=self.model.linear.out_features
        )

        self.log("train_loss", loss, prog_bar=True, logger=True)
        self.log("train_accuracy", step_accuracy, prog_bar=True, logger=True)
```

```
        return {"loss": loss, "accuracy": step_accuracy}

    def validation_step(self, batch, batch_idx):
        sequences = batch["sequence"]
        labels    = batch["label"]
        loss, outputs = self.forward(sequences, labels)
        predictions = torch.argmax(outputs, dim=1)

        step_accuracy = accuracy(
            predictions,
            labels,
            task="multiclass",
            num_classes=self.model.linear.out_features
        )

        self.log("val_loss", loss, prog_bar=True, logger=True)
        self.log("val_accuracy", step_accuracy, prog_bar=True, logger=True)

        return {"loss": loss, "accuracy": step_accuracy}

    def test_step(self, batch, batch_idx):
        sequences = batch["sequence"]
        labels    = batch["label"]
        loss, outputs = self.forward(sequences, labels)
        predictions = torch.argmax(outputs, dim=1)

        step_accuracy = accuracy(
            predictions,
            labels,
            task="multiclass",
            num_classes=self.model.linear.out_features
        )

        self.log("test_loss", loss, prog_bar=True, logger=True)
        self.log("test_accuracy", step_accuracy, prog_bar=True, logger=True)

        return {"loss": loss, "accuracy": step_accuracy}

    def configure_optimizers(self):
        return optim.Adam(self.model.parameters(), lr=1e-3)

model = SurfacePredictor(
    n_features=len(features),
    n_classes=len(label_encoder.classes_)
)

checkpoint_callback = ModelCheckpoint(
    dirpath="checkpoints",
    filename="best-checkpoint",
    save_top_k=1,
```

```
    verbose=True,
    monitor="val_loss",
    mode="min"
)

logger = TensorBoardLogger("lightning_logs", name="surface_predictor")

trainer = pl.Trainer(
    max_epochs=10,
    accelerator='gpu' if use_cuda else 'cpu',
    devices=1,
    logger=logger,
    callbacks=[checkpoint_callback],
    enable_checkpointing=True
)

trainer.fit(model, data_module)

torch.save(model.state_dict(), "model_weights.pth")

print("=== Навчання завершено. Ваги збережені в model_weights.pth ===")

trainer.test(model, datamodule=data_module)

model_loaded = SurfacePredictor(
    n_features=len(features),
    n_classes=len(label_encoder.classes_)
)
model_loaded.load_state_dict(torch.load("model_weights.pth"))
model_loaded.eval()

print("Завантажена модель готова до використання.")
```

```
async def main() -> None:
    try:
        connection = await connect(os.environ['AMQP_URL'], timeout=60*60*24)
    except Exception:
        logging.exception("connection not open")

    channel = await connection.channel()

    exchange = channel.default_exchange

    queue = await channel.declare_queue("rpc_queue", durable=True)
```

```
    logging.info(" [x] Awaiting RPC requests")

    async with queue.iterator() as qiterator:
        message: AbstractIncomingMessage
        async for message in qiterator:
            try:
                async with message.process(requeue=True):
                    inputJson = message.body.decode("UTF-8")
                    api_key = json.loads(inputJson).pop("api_key")
                    api_secret = json.loads(inputJson).pop("api_secret")
                    SYMBOL = json.loads(inputJson).pop("symbol")
                    id = json.loads(inputJson).pop("user_id")
                    num_trades = json.loads(inputJson).pop("num_trades")
                    stop = json.loads(inputJson).pop("stop")
                    if stop == False:
                        await trading(symbol=SYMBOL, api_key=api_key,
api_secret=api_secret, id =id,num_trades = num_trades)
            except Exception:
                logging.exception("Processing error for message %r", message)
            try:
                await asyncio.Future()
            finally:
                await connection.close()


def dataset():
    def CCI(data, ndays):
        TP = (data['High'] + data['Low'] + data['Close']) / 3
        CCI = pd.Series((TP - TP.rolling(ndays).mean()) / (0.015 *
TP.rolling(ndays).std()), name = 'CCI')
        data = data.join(CCI)
        return data

    def EVM(data, ndays):
        dm = ((data['High'] + data['Low'])/2) - ((data['High'].shift(1) +
data['Low'].shift(1))/2)
        br = (data['Volume'] / 100000000) / ((data['High'] - data['Low']))
        EVM = dm / br
        EVM_MA = pd.Series(EVM.rolling(ndays).mean(), name = 'EVM')
        data = data.join(EVM_MA)
        return data

    def SMA(data, ndays):
        SMA = pd.Series(data['Close'].rolling(ndays).mean(), name = 'SMA')
        data = data.join(SMA)
        return data

    def EWMA(data, ndays):
        EMA = pd.Series(data['Close'].ewm(span = ndays, min_periods = ndays -
1).mean(),
```

```
                        name = 'EWMA_' + str(ndays))
        data = data.join(EMA)
        return data

    def BBANDS(data, window):
        MA = data.Close.rolling(window).mean()
        SD = data.Close.rolling(window).std()
        data['UpperBB'] = MA + (2 * SD)
        data['LowerBB'] = MA - (2 * SD)
        return data

    def ForceIndex(data, ndays):
        FI = pd.Series(data['Close'].diff(ndays) * data['Volume'], name =
'ForceIndex')
        data = data.join(FI)
        return data

    def ROC(data,n):
        N = data['Close'].diff(n)
        D = data['Close'].shift(n)
        ROC = pd.Series(N/D,name='Rate of Change')
        data = data.join(ROC)
        return data

    cl = Spot()
    r = cl.klines('BTCUSDT', '1m', limit =  500)
    df = DataFrame(r).iloc[:, :9]
    df = df.drop([6, 7], axis = 1)
    df.columns = ('timestamp', 'Open', 'High', 'Low', 'Close', 'Volume', 'Count')
    data = pd.DataFrame(df)
    data['Open'] = data['Open'].astype(object).astype(float)
    data['High'] = data['High'].astype(object).astype(float)
    data['Low'] = data['Low'].astype(object).astype(float)
    data['Close'] = data['Close'].astype(object).astype(float)
    data['Volume'] = data['Volume'].astype(object).astype(float)
    data['Count'] = data['Count'].astype(object).astype(float)

    BATCH_SIZE = 1
    seq_length = 250
    df = data
    df['Target']= 0

    for i in range (len(df['Open'])):
        open_price= df['Open'][i]
        max = df['High'][i:i+15:].max()
        if max >= open_price + 25:
            df['Target'][i] = 1
        else:
            df['Target'][i] = 0
    n = 20
```

```
NIFTY_ROC = ROC(df,n)
ROC = NIFTY_ROC['Rate of Change']
df = NIFTY_ROC

n = 50
NIFTY_BBANDS = BBANDS(df, n)
df = NIFTY_BBANDS

n = 20
AAPL_ForceIndex = ForceIndex(df,n)
ForceIndex = AAPL_ForceIndex['ForceIndex']
df = AAPL_ForceIndex

# Compute the 14-day Ease of Movement for AAPL
n = 14
AAPL_EVM = EVM(df, n)
AAPL_EVM  = AAPL_EVM.dropna()
EVM = AAPL_EVM['EVM']
df = AAPL_EVM

n = 60*3
NIFTY_CCI = CCI(df, n)
NIFTY_CCI = NIFTY_CCI.dropna()
CCI = NIFTY_CCI['CCI']
df = NIFTY_CCI

n = 9
SMA_NIFTY = SMA(df,n)
SMA_NIFTY = SMA_NIFTY.dropna()
SMA = SMA_NIFTY['SMA']
df = SMA_NIFTY

ew = 15
EWMA_NIFTY = EWMA(df,ew)
EWMA_NIFTY = EWMA_NIFTY.dropna()
EWMA = EWMA_NIFTY['EWMA_'  + str(ew)]
df = EWMA_NIFTY

df_train = df
df_train = df_train.set_index("timestamp")

target = 'Target'
drops = ['timestamp']
features = [f for f in df_train.columns if f not in drops + [target]]

X_train = df_train[features]
y_train = df_train[target]
d = preprocessing.normalize(X_train, axis=0)
X_train = pd.DataFrame(d, columns=X_train.columns)
```

```
    label_encoder = LabelEncoder()
    encoded_labels = label_encoder.fit_transform(y_train)
    y_train["label"] = encoded_labels

    sequences =[]
    for idx in range(len(X_train) - seq_length):
        x = X_train[idx:idx + seq_length]
        y = y_train[idx:idx + seq_length].iloc[0]
        sequences.append((x, y))
    return sequences, features, label_encoder

class SurfaceDataset(Dataset):
    def __init__(self, sequences):
        super().__init__()
        self.sequences = sequences

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, idx):
        sequence, label = self.sequences[idx]
        return dict(
            sequence = torch.Tensor(sequence.values),
            label = torch.tensor(label).long()
        )

class SurfaceDataModule(pl.LightningDataModule):
    def __init__(self, test_sequences, batch_size):
        super().__init__()
        self.test_sequences = test_sequences
        self.batch_size = batch_size

    def setup(self, stage=None):
        self.test_dataset = SurfaceDataset(self.test_sequences)

    def test_dataloader(self):
        return DataLoader(
            self.test_dataset,
            batch_size = self.batch_size,
            shuffle = False,
            drop_last = True
        )

class SequenceModel(pl.LightningModule):
    def __init__(self, n_features, n_classes, n_hidden=400, n_layers=2):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size = n_features,
            hidden_size = n_hidden,
            batch_first = True,
```

```
            num_layers = n_layers,
            dropout = 0.3
        )
        self.liner1 = nn.Linear(n_hidden, n_classes)

    def forward(self, x):
        _, (hidden, _) = self.lstm(x)
        out = hidden[-1]
        return out

async def trading(symbol, api_key, api_secret, id, num_trades):
    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda:0" if use_cuda else "cpu")

    torch.manual_seed(0)
    torch.cuda.manual_seed(0)
    np.random.seed(0)
    id=str(id)
    BATCH_SIZE = 1
    SYMBOL = symbol
    INTERVAL = '1m'
    LIMIT = '200'
    QNTY = decimal.Decimal('0.00340')
    client = Client(api_key, api_secret)
    test_sequences, features, label_encoder = dataset()


    def send(text):
        url =
'https://api.telegram.org/bot'+token+'/sendMessage?chat_id='+id+'&text='+text+''
        resp = requests.get(url)
        r = resp.json()
        return r

    def get_data():
        url =
'https://api.binance.com/api/v3/klines?symbol={}&interval={}&limit={}'.format(SYMBO
L, INTERVAL, LIMIT)
        res = requests.get(url)
        return_data = []
        for each in res.json():
            return_data.append(float(each[4]))
        return np.array(return_data)

    def place_order(order_type):
        if(order_type == 'buy'):
            client.create_order(symbol=SYMBOL, side='buy', type='MARKET', quantity=
QNTY)
        else:
```

```
            client.create_order(symbol=SYMBOL, side='sell', type='MARKET',
quantity= QNTY)

    class SurfacePredictor(pl.LightningModule):
        def __init__(self, n_features, n_classes):
            super().__init__()
            self.model = SequenceModel(n_features, n_classes)
            self.criterion = nn.CrossEntropyLoss()

        def forward(self, x, labels=None):
            output = self.model(x)
            if labels is not None:
                output = Variable(torch.randn(BATCH_SIZE,
len(label_encoder.classes_)).float(), requires_grad=True).to(device)
            return output

        def test_step(self, batch, batch_idx):
            global predictions
            sequences = batch["sequence"]
            labels = batch["label"]
            outputs = self.forward(sequences, labels)
            predictions = torch.argmax(outputs, dim=1).cpu().numpy()
            print(predictions)

    def condition_buy(price_current, price_buy, position):
        global number_of_trades, sell_state, buy, sell
        if price_buy + 1 + position > price_current >= price_buy + position:
            sell_state = position
            return 1
        elif (sell_state == position) and (price_current < price_buy + position):
            place_order('sell')
            send(f'Продємо. +{position}: {(100*price_current)/price_buy}\n Ціна
продажу: {price_current}')
            number_of_trades[position] += 1
            buy = False
            sell = True
            sell_state = 0
            return 1
        else: return 0
```